WEB SERVICE TESTING FOR DOMAIN SPECIFIC WEB SERVICE DISCOVERY FRAMEWORK

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SELMA UTKU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

FEBRUARY 2012

Approval of the thesis:

WEB SERVICE TESTING FOR DOMAIN SPECIFIC WEB SERVICE DISCOVERY FRAMEWORK

submitted by **SELMA UTKU** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

ABSTRACT

WEB SERVICE TESTING FOR DOMAIN SPECIFIC WEB SERVICE DISCOVERY FRAMEWORK

Utku, Selma
M.Sc., Department of Computer Engineering
Supervisor : Assoc. Prof. Dr. Pınar Şenkul

February 2012, 70 pages

The reliability of web services is important for both users and other service providers, with which they are in interaction. Thus, to guarantee reliability of the web services that are invoked and integrated at runtime, automatic testing of web services is needed.

In web service testing, different test cases for web services are generated. The most important issue is to generate the most appropriate value for input parameters of web services at runtime. In this thesis, we developed a method for automatic web service testing that uses semantics dependency-based and data mutation-based techniques to analyze web services and generate different test cases. Thus, we both check whether the services function correctly by generating appropriate input values from different data sources and check robustness of web services by generating random and error-prone data inputs. With respect to the behaviors of web services, the test values are calculated and saved to the database for each web service.

Keywords: web service, semantic web service discovery, testing of web services, semantic dependency analysis, mutation analysis

ÖZ

ALANA ÖZGÜ WEB SERVİS KEŞİF SİSTEMLERİNDE WEB SERVİSLERİN TEST EDİLMESİ

Utku, Selma
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Pınar Şenkul

Şubat 2012, 70 sayfa

Etkileşim içinde oldukları web servislerin güvenilirlikleri, kullanıcılar ve diğer servis sağlayıcılar açısından oldukça önemlidir. Bu nedenle, çalışma zamanı sırasında çağrılan ve entegre edilen servislerin, güvenilirliklerini garanti etmek için otomatik olarak test edilmeleri gerekmektedir.

Web servislerin test edilmesi işleminde, farklı test senaryoları üretilir. Bu konuda en önemli husus ise, servislerin girdi parametreleri için en uygun değerin üretilmesidir. Bu tezde, web servislerin otomatik olarak test edilmesi için bir yöntem geliştirilmiştir. Bu yöntemde, semantik bağımlılık tabanlı ve veri mutasyon tabanlı teknikler kullanılarak, web servislerin analizi ve değişik test senaryolarının üretilmesi sağlanmaktadır. Böylece, çeşitli veri kaynakları kullanılarak, web servisler için en uygun parametre değerlerinin üretilmesi sağlanır ve beklenilen davranışları sergileyip sergilemediği kontrol edilir. Bunun yanısıra özel girdi değerleri üretilerek, web servislerin dayanıklılıkları ve hataya ne kadar açık oldukları da tespit edilir. Elde edilen sonuçlar doğrultusunda her bir web servis için test puanları hesaplanır ve sisteme kaydedilir.

Anahtar Kelimeler: web servis, semantik web servis keşfi, web servis testi, semantik bağımlılık analizi, mutasyon analizi

To my lovely family

ACKNOWLEDGMENTS

I would like to express my deepest sense of gratitude to my advisor Dr. Pınar Şenkul for her encouragement, supervision, advice and proper guidance throughout the development of this thesis study. I also wish to thank my committee members who were generous with their expertise and precious time.

My deepest gratitude goes to my family for their love and support throughout my life. I have no suitable word that can fully describe my mother, Ayşe Utku's everlasting love to me. I am indebted to my father, Hüseyin Utku for his care and love. My sisters Belgin and Bilge, my brother Bekir, thanks for their love, trust, understanding and every kind of support not only throughout my thesis but also throughout my life.

I wish to thank ASELSAN A.Ş. for giving me the opportunity of continuing my education. I would like to express my special appreciation to my colleagues and seniors in my department for their support.

I would like to thank to my supportive friends and all people who have helped and inspired me during my thesis study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Web services provide interaction between different distributed applications and the use of web services becomes one of the most preferable technologies by software developers and users. Nowadays, the number of published web services has been increasing and this rapid development brings some problems. There is a high number of public web services on the Web and finding the most appropriate web service that is needed is a hard task for users. For this, services must be filtered or ordered with respect to some criteria.

Another important problem stems from the dynamic nature of the Web. There are many web services that are published and are becoming outdated very rapidly and there is no control for their reliability. Therefore the testing of web services is needed before deployment. In addition, web services are published, integrated and invoked at runtime. For this reason, traditional offline and manual testing processes cannot satisfy the requirements of web service testing. The dynamically selected web services to be included in a software application has to be tested without human interaction [5].

Web service testing focuses on how to guarantee desired properties in web service such as correctness, fault-tolerance, deadlock avoidance, reachability, liveness [6]. To test a web service, different test cases are generated. The most important issue is to generate the most appropriate value for input parameters of web services automatically and it is very time-consuming process.

There are two types of web services; atomic web service and complex web service. Atomic web service provides a request/response type of functionality and does not support transactions. Complex web services provide business-to-business collaborations and business process management [7]. In addition, different high-level modeling languages such as WSDL, BPEL and OWL-S can be used for defining web services. In the literature, there are several studies for testing of atomic or complex web services and they are based on one of the above description languages [2], [3], [5], [8].

In this thesis, we present a method for automatic testing of web services. It is based on two techniques. The first one is mutation analysis. In this technique, a web service is tested by using random and specified values in different ranges that are set according to the parameter types. The second technique analyzes a web service semantically and generates different test cases in the light of the results of the analysis. After the web services are invoked by using the input parameters that are generated by using these two methods, several results are obtained. The test results are obtained by analyzing these results in two dimensions. In the first dimension, the execution results are checked. If an exception occurs during the execution of the web service request, the web service is considered unsuccessful, otherwise, it is considered successful. The second dimension is based on checking whether the web service returns different results and not the same results on all test cases execution.

The proposed method is implemented in a domain specific web service discovery system, namely DSWSD-S, Domain Specific Web Service Discovery with Semantics [9], [10]. The testing module in DSWSD-S is in charge of checking whether the discovered services function correctly.

1.1 Motivation

There are many web services that are published and finding the most appropriate web services that are needed is a hard task for users. DSWSD-S System enables the users to search for web services easily. The found web service should satisfy the requirements of the users and their reliability should be checked. Therefore, the main motivation of this thesis is to test whether the web services, which are crawled by DSWSD-S System actually perform the advertised tasks. In the testing process, web services are invoked by using different input parameter values. Web services in the system are atomic and in order to describe their interfaces for service consumers, they use WSDL files, which contain poor information about the web services. In the literature, the testing studies that focus on atomic web services use different methods to generate input values and examine the invocation results for testing of web services. While some of them just control the robustness of the web services, some of them make syntactic dependency analysis of the web services to generate different values for input parameters. These approaches in the literature were inadequate in order to design the testing module of DSWSD-S System. For this reason, we developed the techniques presented in this thesis and implemented the testing module, which can both check the robustness of web services by generating random and error-prone data inputs and can check whether the services function correctly by generating appropriate input values from different data sources.

1.2 Contributions

The main contributions of this thesis are as follows:

- A novel web service testing technique is generated. It is realized within a web service discovery system, namely DSWSD-S. Testing of the web services discovered by DSWSD-S System is automatically performed.
- A tool with a graphical user interface is developed, in order to enable the users to run the testing algorithm step by step on the discovered web services.
- Data mutation-based test cases are generated.
- Semantic dependency analysis-based test cases are generated.
- The DSWSD-S ontology instances and WordNet ontology instances are used in generation of test cases.
- Interactions of the testing module with other modules in DSWSD-S system are presented.

1.3 Thesis Organization

This thesis is organized as follows:

Chapter 2 presents the related work on web service testing and input data generation algorithms.

Chapter 3 describes the overall architecture of Domain Specific Web Service Discovery with Semantics (DSWSD-S) System and and the web service testing module within the system. In addition, a general overview of the architecture for web service testing module and designed graphical user interface are given in the rest of Chapter 3.

In Chapter 4, the proposed web service testing algorithm is explained in detail.

Chapter 5 presents the case studies and evaluation of results and the statistical data about the experiments is given in this chapter.

In Chapter 6, discussion and conclusion are presented. Some future work topics are suggested.

CHAPTER 2

RELATED WORK

Web service testing is a problem-rich area, where there are several research works that concentrate on different aspects of the problem and are based on different methods. In Section 2.1, the studies, which aim to test the robustness of web services are mentioned. Section 2.2 explains the studies that concentrate on WSDL-based web service testing. In Section 2.3, we present studies that use mutation testing method. In Section 2.4, a related work on web service sampling is described.

Beside studies that are summarized in the following sections, there are also several studies that propose methods using semantic model OWL-S for test data generation [5], [8], [11]. Since these studies are not directly related with this thesis, we very briefly summarize them as follows. Wang et al. [5] propose a Petri-Net based testing approach. A Petri-Net model is generated from the OWL-S process model and the test data are generated using ontology reasoning. Zhu [8] proposes an ontology of software testing using semantic web services. Bai et al. [11] also propose an approach for OWL-S semantic services. In their approach, a testing ontology model that describes test relations, concepts and serves as a contract is introduced and test data partitions are generated from ontology information.

2.1 Robustness Testing of Web Service

Before using the web service by service consumers, testing is needed to guarantee the correctness and robustness of web services. The study of Martin et al. [2] is one of studies that emphasize on robustness testing of web services using WSDL. It presents a framework to generate and execute test cases automatically and also generates the service consumer code to invoke web services. The overview of the framework is given in Figure 2.1.

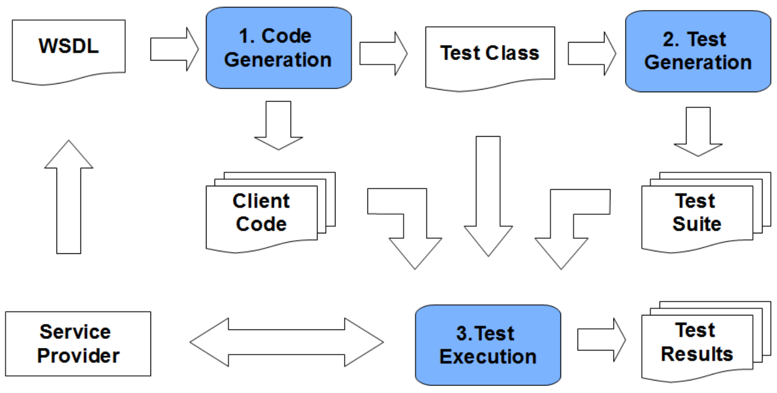


Figure 2.1: The Overview of Framework Proposed by Martin et al. [2]

In the code generation step, Axis [12] is used in order to generate client code from the WSDL file, which is provided by service provider and describes the web service interfaces. WSDL2Java class of Axis is very useful to easily implement this step. To generate tests, JUnit generation tools are used. However, a wrapper class for the code, which is generated by WSDL2Java should be manually implemented to use in unit-test generation tool. In test generation step, JCrasher [13], which is a unit-test generation tool is used in order to automatically generate JUnit tests. JCrasher generates -1, 0 and 1 for arguments with the integer type and it can generate method sequences that create values for those arguments with non-primitive types [2]. Thus, a robustness testing can be performed by causing that an undeclared runtime exception occurs in invocation of the web services. As last step, generated JUnit tests are executed and the obtained results are collected.

In this study, only robustness testing is performed. However, it is not adequate for testing of web service. In our approach, in addition to checking robustness of test web services, it is checked the behaviors of a web service when it is invoked by using the valid values. Also, our approach generates various random and error-prone data inputs for all parameter types in order to check robustness and reliability of web services.

2.2 WSDL-Based Web Service Testing

Web services are provided with a description files to describe their interfaces for service consumers. As the description files, the web service providers usually use an XML-formatted document called a Web Services Description Language (WSDL) document. Therefore, the traditional test methods can not be used in testing of web service. Some of the studies that deal with web service testing uses WSDL-based test case generation methods.

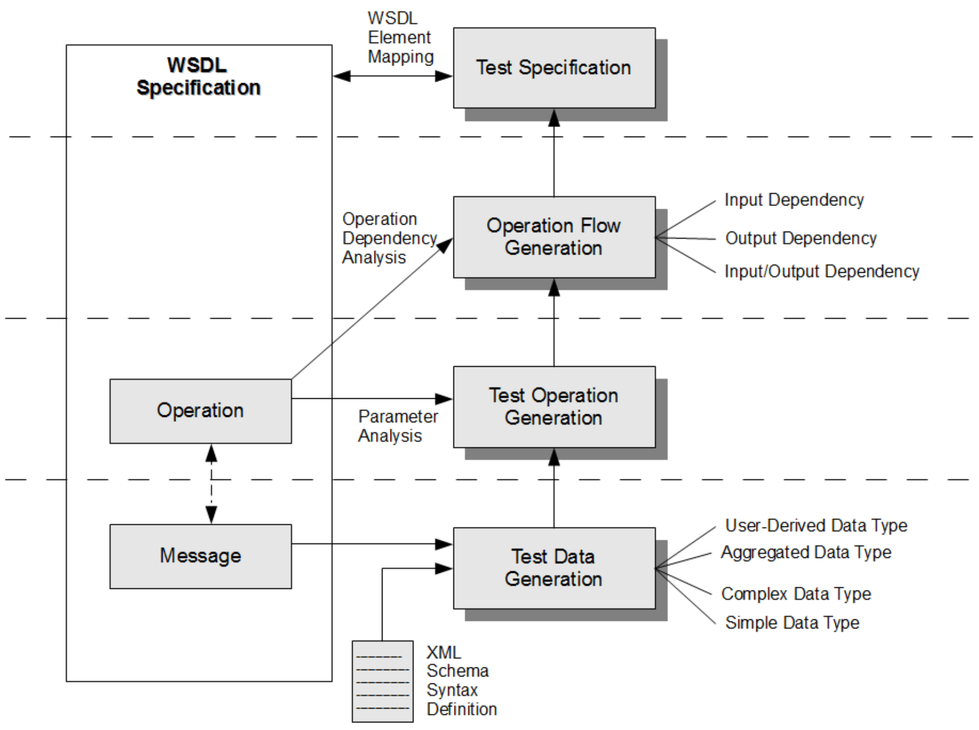


Figure 2.2: Test Case Generation Levels Proposed by Bai et al. [3]

Bai et al. [3] proposes an approach about WSDL-based test generation and test case documentation to provide reusability of generated test cases. Test cases are generated from four levels: test data generation, individual test operation generation, operation flow generation, and test specification generation [3]. Test data is generated by analysis WSDL message definitions. In individual test operation generation level, input parameters of web services are analyzed and test operation are generated. In operation flow generation level, the sequence of the web services are determined by analysis of dependencies between the web services. In this approach, three dependencies are used; input dependency, input/output dependency and output dependency. WS1 is input/output dependent on WS2 if at least one of the input parameters of WS1 has the same type with the output message of WS2. If two web services both share their input message, they are input dependent. Two web services are output dependent if they have the same output type [3]. In test specification generation level, test cases that are generated from other levels are recorded in XML-based files called Service Test Specification to reuse the test cases. The test generation levels are depicted in Figure 2.2.

In this approach, the dependency relations between the web services are syntactically analyzed and the obtained dependency results are used for operation flow generation. Our approach supports both syntactically and semantically dependency analysis and the analysis results are used for input parameter value generation.

2.3 Mutation-Based Web Service Testing

Siblini et al. [1] propose a mutation testing method to test web services. In this work, mutant operators to the WSDL document of web services are defined and mutated web service interfaces are generated. With each modification, a new version of test case is created and it is called *mutant*. There are two types of mutants. If the output of the mutant is different from the output of the original version, the mutant is killed and it is called *dead mutants*. If the outputs of test cases generated with mutant and the original version are same and it is called *equivalent mutant*. The mutation score is calculated by using Equation 2.1 and the main goal is to improve the mutation score to 1.0, indicating all mutants have been detected [1].

$$MutationScore = \frac{numberOfDeadMutants}{numberOfTotalMutants - numberOfEquivalentMutants} \tag{2.1}$$

Mutant operators are applied to input parameters, output parameter of web services and the data types that are defined in the WSDL document. The aim of this approach is finding errors relevant to both the WSDL interface and the logic of web service programming. The mutation operators are grouped; Switch group, Special group and Occurence group.

In mutant operators of the Switch group, the sequence of an element is replaced and by this way, many various WSDL can be created. For each variation of mutant WSDL document, replacement of one element with another one in the operation definition is performed. As an example, we can handle a web service having 2 input parameters I1, I2 and one output parameter O. One mutant replaces the sequence of I1 by the sequence of I2 and another mutant replaces I1 by O.

In mutant operators of the Special group, the value of an element is modified. For instance, the value of input parameters are set to boundary values, null values or the next value in the same domain of the input parameters. Each modification can cause a different type of error.

Mutant operators of the group Occurrence delete or add an occurrence of an element. The proposed mutation operators in all groups are listed in Table 2.1.

Table 2.1: WSDL Mutation Operators [1]

Mutation operator	**Group**	**Description**
STCE (SwitchTypesComplexTypeElement)	Switch	Switch elements of the same type in the complexType element
STCA (SwitchTypesComplexTypeAttribute)	Switch	Switch attributes of the same type in the complexType element
OTCE (OccurrenceTypesComplexTypeElement)	Occurrence	Add or delete occurrence of an element in the complexType
OTCA (OccurrenceTypesComplexTypeAttribute)	Occurrence	Add or delete an optional attribute in the complexType element
STEN (SpecialTypesElementNil)	Special	Set the nil attribute to true in the complexType element
STSE (SwitchTypesSimpleTypeElement)	Switch	Switch elements of the same data type in the SimpleType element
STSA (SwitchTypesSimpleTypeAttribute)	Switch	Switch attributes of the same data type in the SimpleType element
SMP (SwitchMessagesPart)	Switch	Switch parts of the same element type in the Message element
SPM (SwitchPortTypeMessage)	Switch	Switch messages of same type in the operation element that is defined in a PortType element, which has as operation of type request-response or solicit-response

In this approach, mutant operators are defined and applied to the WSDL document of web services. However, in our approach, data mutation is applied for input parameters. The input parameter values are modified by using different data groups in order to generate test cases. Like the approach proposed by Siblini et al. [1], switch mutant operator is defined. However, switching of generated values of input parameters is performed in our approach and obtaining different output values are expected when this operation is applied to input parameters.

2.4 Web Service Sampling

Typical information source for web services is service descriptions. As service providers generally focus on the implementation of the web services, they generally do not provide rich service descriptions and the lack of rich service descriptions is one of the challenges for web service users. Therefore, gathering additional information about web services becomes important. However, gathering such information manually is a hard and time consuming task. In order to solve this problem, methods that automatically gather extra information about the web services and enrich the web service descriptions are required.

AbuJarour et al. [4] propose an approach to generate annotations for web services, e.g., valid input parameters, examples of expected outputs and tags, by sampling invocations of web services automatically. The generated annotations are integrated to web forms to help service consumers for actual service invocations. In this approach, in order to generate valid parameters, various resources such as random values, outputs of other web services that are provided by the same web service provider and different providers, external data sources, e.g., WordNet, DBpedia, Freebase are used. The overview of the approach that is proposed by AbuJarour et al. is as shown in Figure 2.3.

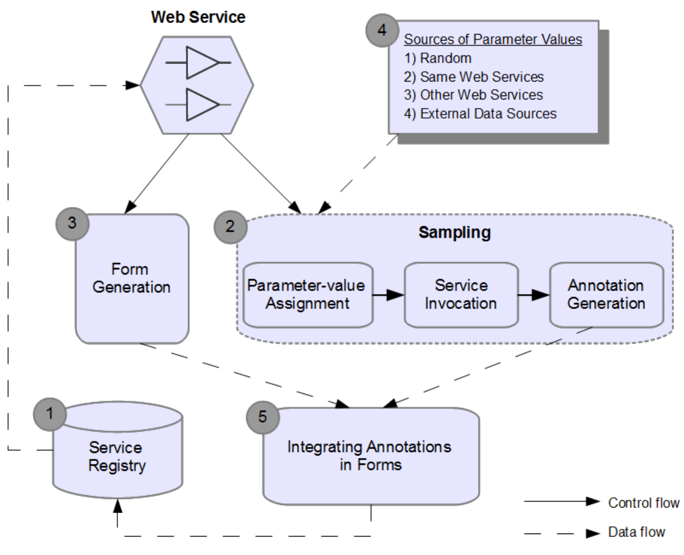
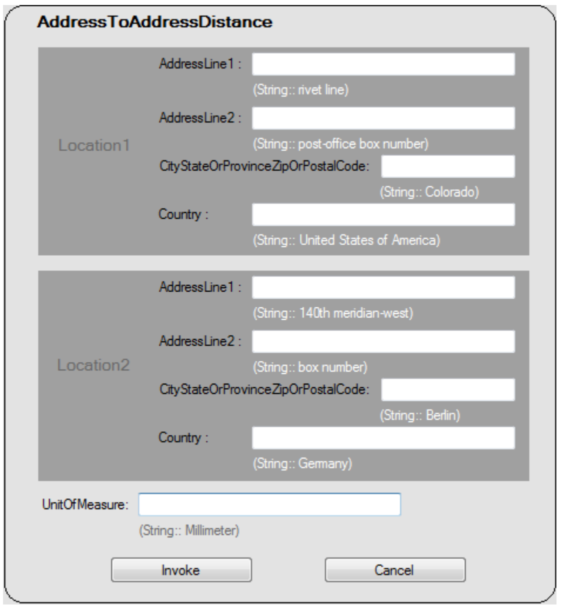


Figure 2.3: The Overview of AbuJarour's Approach [4]

Public web services are collected from Web and they are stored in a service registry. For each web service in the registry, sampling steps are applied one by one.

The first step, which is also one of the most important steps of sampling, is value assignment to input parameters. Firstly, some text processing is applied on the name of input parameters. These processes are stopword removal, identification of common types and data type deduction. The input parameter whose name includes stopwords used for authentication, user identification and accounting are determined and ignored in next processes. `UserID`, `keyID` and `password` are some of these stopwords. As the second process, name of input parameters that are commonly used in web services are determined. A set of values for common data is manually created and value of the input parameters are assigned from this set. `City`, `state` and `postcode` are some examples for the common parameters. Some of parameters used for numeric or boolean data can be defined as String and to identify their actual type, string analysis is performed under some assumptions. The parameters whose name starts with a verb like `allowNameEdit`, `hasStock`, etc. have boolean type. The parameters whose name includes the term ”number” like `pageNumber`, `houseNumber`, etc. have numeric type. For such parameters, random values can be assigned. If the name and type of parameters are not suitable for assigning value with mentioned methods, the relations between input and output parameters of the web services of same provider are analyzed and if there is any matched output parameter, the values of these outputs are utilized in assignment value to input parameters. Otherwise, the outputs of web services of other providers are retrieved. However, in this case, the domain is very important. Although the input parameters and output parameters have the same name, they can have different semantics due to the domain of web services. For the input parameters that can not be assigned values, the last of value assignment methods is applied. In this method, the external sources like WordNet, DBpedia, Freebase are used. By using WordNet, the synset that is best matched with name of input parameter are tried to find and its instance is assigned to value of input parameter.

After the value assignment to input parameters, web service is invoked automatically. As last step of sampling, annotations are generated by parsing the obtained information after each successful invocation. In parallel to the mentioned steps, a schema-based web form are generated. After completing the steps labeled 2 and 3, the generated annotations and the web form are integrated to enable the users to invoke the web service easily. An example of schema-based web form, which is generated and integrated with the annotations is given in Figure 2.4.


AddressToAddressDistance

Location1

AddressLine1 :
(String:: rivet line)

AddressLine2 :
(String:: post-office box number)

CityStateOrProvinceZipOrPostalCode:
(String:: Colorado)

Country :
(String:: United States of America)

Location2

AddressLine1 :
(String:: 140th meridian-west)

AddressLine2 :
(String:: box number)

CityStateOrProvinceZipOrPostalCode:
(String:: Berlin)

Country :
(String:: Germany)

UnitOfMeasure:
(String:: Millimeter)

Figure 2.4: A Schema-Based and Annotated Web Form by Sampling [4]

The real invocations are made by the users and from each new invocation, additional annotations can be obtained and saved to registry for the other automatic invocation.

CHAPTER 3

WEB SERVICE TESTING IN A WEB SERVICE DISCOVERY SYSTEM

The web service testing approach that is presented in this thesis is implemented as part of Domain Specific Web Service Discovery with Semantics (DSWSD-S) System [9], [10]. In this chapter, DSWSD-S System and the web service testing module within this system are described. Firstly, in Section 3.1, the overall architecture of DSWSD-S System and its layers are presented. In Section 3.2, internal architecture of web service testing module and the steps of the proposed approach are given in details. Finally, Section 3.3 presents the graphical user interface of the application that is designed to perform the proposed web service testing approach.

3.1 Overall Architecture of DSWSD-S System

DSWSD-S System is a service discovery system that uses domain-specific web service discovery sub-systems. This system consists of different domains and for each domain, a subsystem is built. Each subsystem is specialized for an ontology and it crawls over the web, discovers and indexes web services semantically according to its own ontology. The aim of DSWSD-S System is to provide following capabilities:

- Ability to handle the increasing number of web services and domains
- Providing service discovery with syntactic and semantic matching capabilities
- Keeping the web services up-to-date

- Providing both syntactic and semantic web service queries
- Providing automated quality of service calculation
- Providing automated service testing

There are many web services that are published and most of them are not registered to any of the business registries. Therefore, the management of these web services and finding the most appropriate web services that users need is a hard task. This system aims to provide some facilities to solve these problems. The system analyzes the web services by associating with the ontology and groups these associated web services. Each group is handled as a domain-specific web service discovery node.

This system consists of two layers; Domain Specific Discovery Layer and Domain Specific Crawler layer. The overall architecture of the system is given in Figure 3.1.

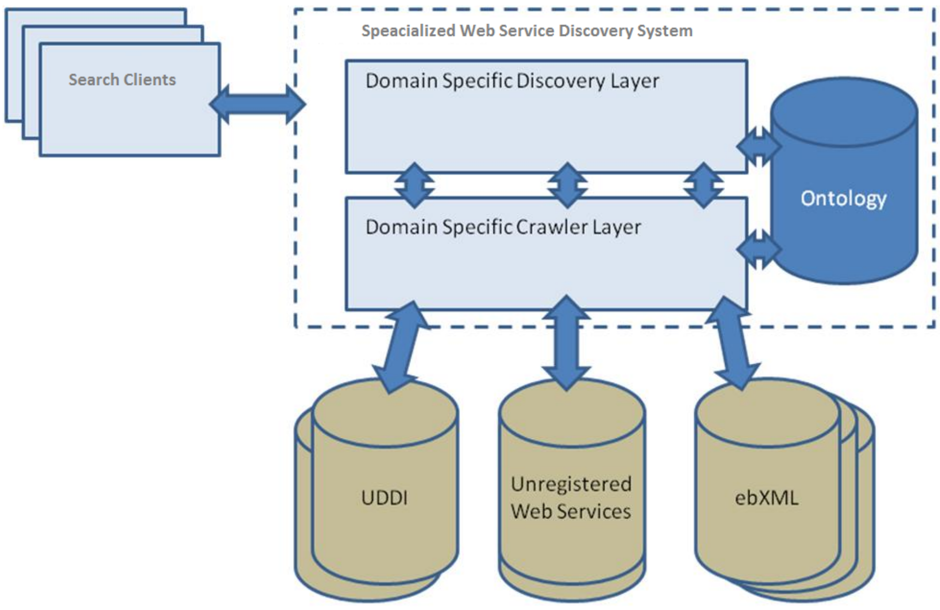

Search Clients
Speacialized Web Service Discovery System
Domain Specific Discovery Layer
Domain Specific Crawler Layer
Ontology
UDDI
Unregistered Web Services
ebXML

Figure 3.1: DSWSD-S System Architecture

In Domain Specific Crawler layer, each subsystem contains some business units that are responsible for the specific processes to fulfill the system requirements. The architecture of Domain Specific Crawler layer is depicted in Figure 3.2. The main responsibility of this layer is to generate web service database according to its own ontology. In generation of local database, the first process is web service address acquisition and the purpose is to collect the

WSDL files of the web services. From the obtained WSDLs, the context of web services are analyzed and controlled whether it is related to the ontology of the subsystem. The next process is validation of the web services that are related to its own ontology. Each web service is invoked with simple appropriate input parameters. Thus, it is checked whether the web service is alive or not. The validated web services are passed to annotation module. The responsibility of the annotation module is to check the compatibility of the service with the domain ontology at hand and to annotate it if the service is found to be relevant. In service annotation process, the name, input and output parameters of web services are compared with the ontology terms by using syntactic and semantic matching. According to the comparison results, relation of web services to the ontology are determined. The next process is the service testing and it is the responsibility of testing module. This module and the testing process constitute the scope of this thesis and they are explained in detail in Chapters 4 and 5. After these processes, in order to find the web services that have the best quality, QoS of the relevant web services are calculated.

Crawler module periodically searches the web and discovers the web services that are associated with the ontology. The local databases are updated by inserting new acquired web services and removing the web services that lost the functionality. Thus, it supports obtaining the up-to-date status (working properly / not functioning) of web service.

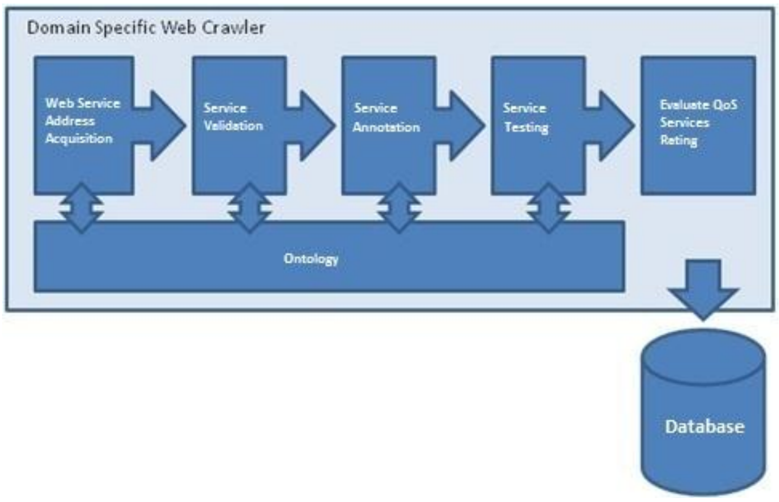

Domain Specific Web Crawler
Web Service Address Acquisition
Service Validation
Service Annotation
Service Testing
Evaluate QoS Services Rating
Ontology
Database

Figure 3.2: Domain-Specific Crawler Layer Architecture

The main responsibility of Domain-Specific Service Discovery layer is to provide a user interface for the user to query the ontology specific web service database. The designed graphical user interface allows the user to specify the keyword(s) to search, the domain information that is interested in and set the service quality criteria for the desired services. Discovery layer passes the inputs gathered from the user to syntactic and semantic matching engine by transforming them into an S-USQL query sentence. It decides the crawler that will be searched according to the specified inputs of the user and the web services that fulfill the given requirements are picked from the database of the corresponding crawler. Finally, list of obtained web services is shown in the graphical user interface as search results.

3.2 Overall Architecture of Testing Module

In DSWSD-S, testing module is responsible for the automatic testing of web services and the proposed approach for web service testing is implemented within this module. Figure 3.3 shows the overall architecture of the proposed approach for web service testing:

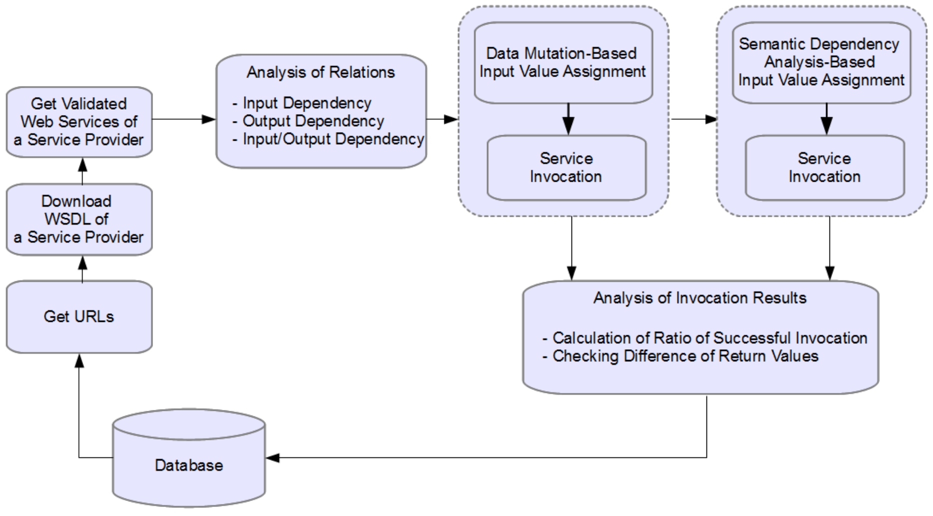

Get Validated Web Services of a Service Provider
Analysis of Relations
- Input Dependency
- Output Dependency
- Input/Output Dependency
Data Mutation-Based Input Value Assignment
Service Invocation
Semantic Dependency Analysis-Based Input Value Assignment
Service Invocation
Download WSDL of a Service Provider
Get URLs
Analysis of Invocation Results
- Calculation of Ratio of Successful Invocation
- Checking Difference of Return Values
Database

Figure 3.3: The Architecture of Web Service Testing Module

Web service testing process starts with getting the URLs of service providers from the database. After that, for each service provider, the following steps are implemented. The initial step is to download the WSDL document that describes the web services of the service provider and to save it to local storage. By analyzing the the content of WSDL documents, information as to which web services are provided and how to invoke are obtained. For each provided web service, the validation process is performed in order to check whether it is alive or not and the set of the validated web services that are provided are acquired. On this set, the analysis process of the relations of web services with each other are performed, so that these relations can be used as input dependency relation, output dependency relation and input/output dependency relation.

The next step is test case generation with data mutation-based method for each web service. By this method, the values of input parameters are generated according to the parameters that the service description contains. With the generated input parameters, the web service is invoked and the invocation result is saved for analysis.

After the invocations of the all validated web services by using data mutation-based method are completed, test case generation with semantic dependency analysis based-method for each web service is performed. For each web service, the value assignment of input parameters of web service is performed according to the method and the web service is invoked with the generated input parameters. As in the previous step, for each invocation, the results are saved.

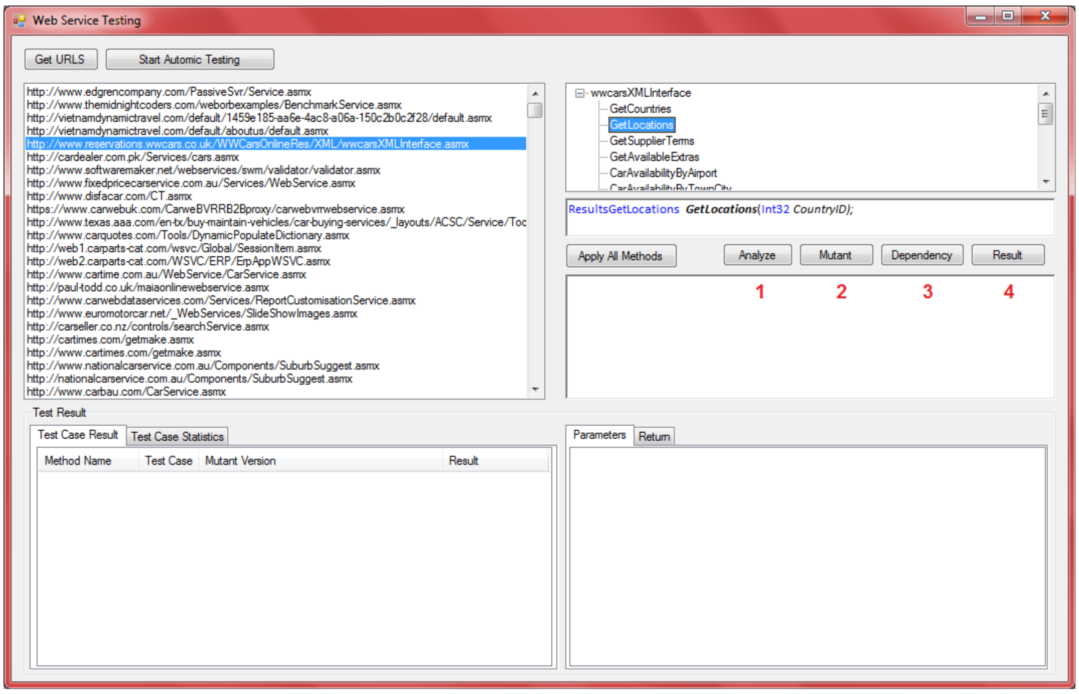
Finally, for each web service, the final test result is calculated by analyzing the results of all invocations and it is saved to the database.

3.3 Graphical User Interface

For realization of the proposed approach, an application is designed. This application provides a graphical user interface to test the web services automatically and enable the users to run the algorithm step by step for the web services of each service provider. A snapshot of the graphical user interface is as shown in Figure 3.4.

The user can list the all URLs of the service provider in database by clicking on ”Get URLs”. The upper left part of the form is separated to view the URLs. If one of them is selected, the descriptions of validated web services of the service provider are picked from database and viewed in upper right part of the form. The description of the selected web service, including the service name, the input parameters and output parameter can also be viewed.

The steps of the testing algorithm; dependency analysis of web services, mutant based-test case generation, semantic dependency based-test case generation and analysis the invocation results of web service will be mentioned in following sections in details. The user can perform



```
http://www.edgrencompany.com/PassiveSvr/Service.asmx
http://www.themidnightcoders.com/weborbexamples/BenchmarkService.asmx
http://vietnamdynamictravel.com/default/1459e185-aa6e-4ac8-a06a-150c2b0c2f28/default.asmx
http://vietnamdynamictravel.com/default/aboutus/default.asmx
http://www.reservations.wwcars.co.uk/WWCarsOnlineRes/XML/wwcarsXMLInterface.asmx
http://cardealer.com.pk/Services/cars.asmx
http://www.softwaremaker.net/webservices/swm/validator/validator.asmx
http://www.fixedpricecarservice.com.au/Services/WebService.asmx
http://www.disfacar.com/CT.asmx
https://www.carwebuk.com/CarweBVRRB2Bproxy/carwebvrrwebservice.asmx
http://www.texas.aaa.com/en-tx/buy-maintain-vehicles/car-buying-services/_layouts/ACSC/Service/Toc
http://www.carquotes.com/Tools/DynamicPopulateDictionary.asmx
http://web1.carparts-cat.com/wsvc/Global/SessionItem.asmx
http://web2.carparts-cat.com/WSVC/ERP/ErpAppWSVC.asmx
http://www.cartime.com.au/WebService/CarService.asmx
http://paul-todd.co.uk/maiaonlinewebservice.asmx
http://www.carwebdataservices.com/Services/ReportCustomisationService.asmx
http://www.euromotorcar.net/_WebServices/SlideShowImages.asmx
http://carseller.co.nz/controls/searchService.asmx
http://cartimes.com/getmake.asmx
http://www.cartimes.com/getmake.asmx
http://www.nationalcarservice.com.au/Components/SuburbSuggest.asmx
http://nationalcarservice.com.au/Components/SuburbSuggest.asmx
http://www.carbau.com/CarService.asmx
```

```
wwcarsXMLInterface
    GetCountries
    GetLocations
    GetSupplierTerms
    GetAvailableExtras
    CarAvailabilityByAirport
    CarAvailabilityByTownCity
```

```
ResultsGetLocations GetLocations(Int32 CountryID);
```

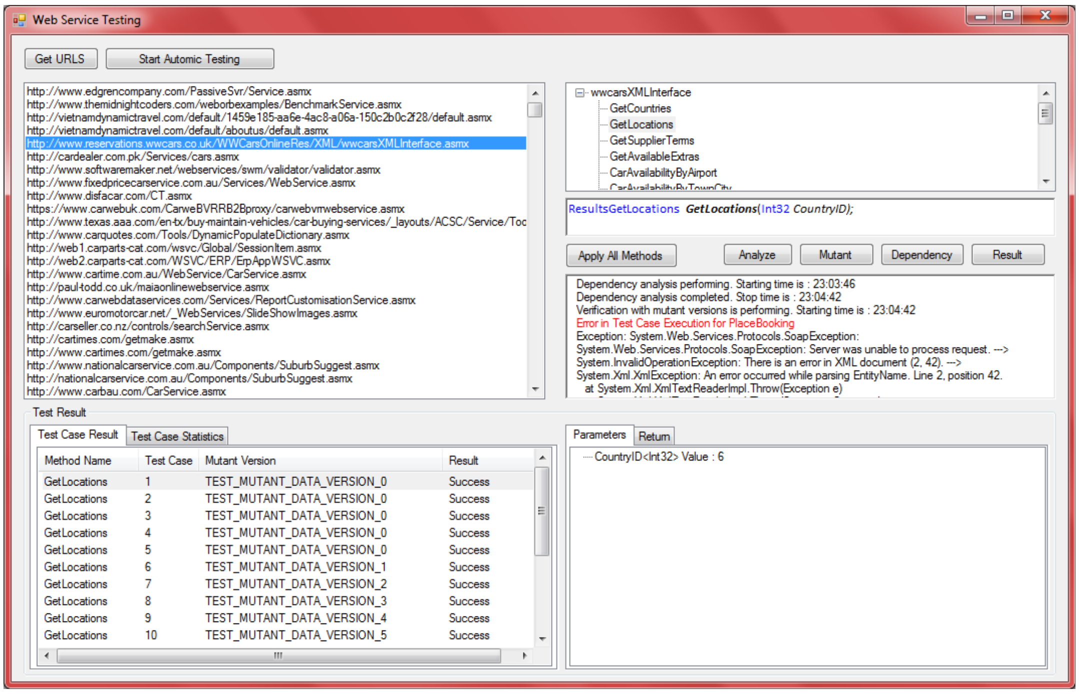
Method Name	Test Case	Mutant Version	Result

Figure 3.4: The Graphical User Interface of Application - I

each step for all web services one by one by through the associated buttons or all steps for all web services are performed sequentially by clicking the ”Applied All Methods” button.

The middle left part of the window is separated for the execution logs. The logs give the user some information about the execution status like start and end time of the steps, and the occurred exceptions in invocation of the web service.

The generated test cases with the assigned value of input parameters and the return values, the invocation results and the statistical information about the tests can be viewed in bottom parts of the window as shown in Figure 3.5 and Figure 3.6.



```
http://www.edgrencompany.com/PassiveSvr/Service.asmx
http://www.themidnightcoders.com/weborbexamples/BenchmarkService.asmx
http://vietnamdynamictravel.com/default/1459e185-aa6e-4ac8-a06a-150c2b0c2f28/default.asmx
http://vietnamdynamictravel.com/default/aboutus/default.asmx
http://www.reservations.wwcars.co.uk/WWCarsOnlineRes/XML/wwcarsXMLInterface.asmx
http://cardealer.com.pk/Services/cars.asmx
http://www.softwaremaker.net/webservices/swm/validator/validator.asmx
http://www.fixedpricecarservice.com.au/Services/WebService.asmx
http://www.disfacar.com/CT.asmx
https://www.carwebuk.com/CarweBVRRB2Bproxy/carwebvrrwebservice.asmx
http://www.texas.aaa.com/en-tx/buy-maintain-vehicles/car-buying-services/_layouts/ACSC/Service/Toc
http://www.carquotes.com/Tools/DynamicPopulateDictionary.asmx
http://web1.carparts-cat.com/wsvc/Global/SessionItem.asmx
http://web2.carparts-cat.com/WSVC/ERP/ErpAppWSVC.asmx
http://www.cartime.com.au/WebService/CarService.asmx
http://paul-todd.co.uk/maiaonlinewebservice.asmx
http://www.carwebdataservices.com/Services/ReportCustomisationService.asmx
http://www.euromotorcar.net/_WebServices/SlideShowImages.asmx
http://carseller.co.nz/controls/searchService.asmx
http://cartimes.com/getmake.asmx
http://www.cartimes.com/getmake.asmx
http://www.nationalcarservice.com.au/Components/SuburbSuggest.asmx
http://nationalcarservice.com.au/Components/SuburbSuggest.asmx
http://www.carbau.com/CarService.asmx
```

```
wwcarsXMLInterface
    GetCountries
    GetLocations
    GetSupplierTerms
    GetAvailableExtras
    CarAvailabilityByAirport
```

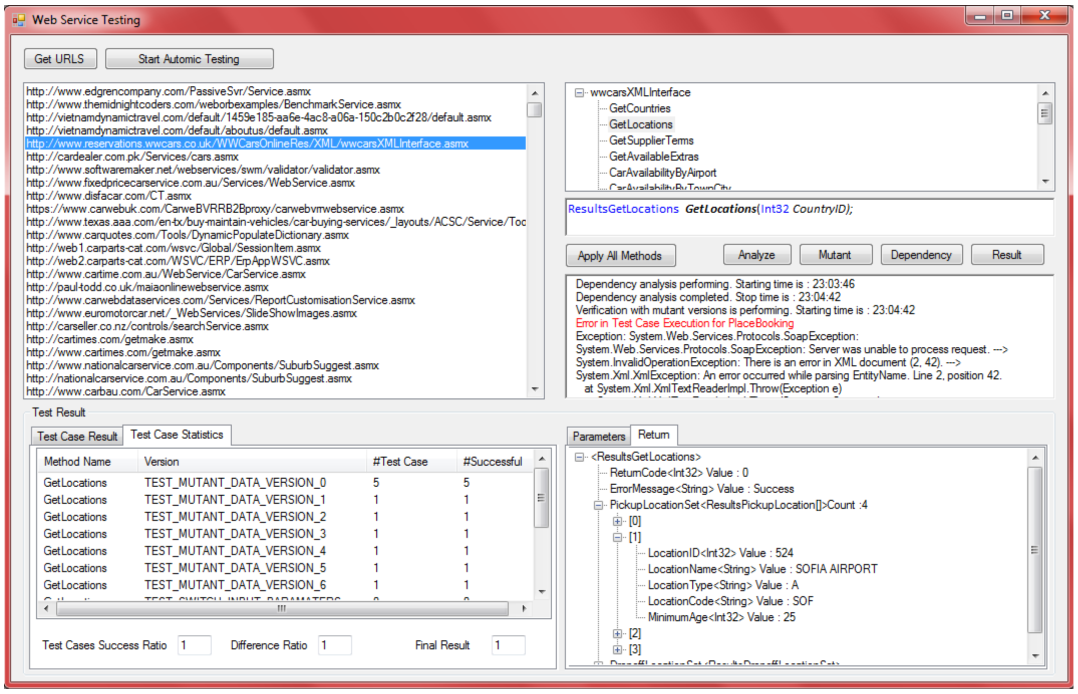
```
ResultsGetLocations  GetLocations(Int32 CountryID);
```

```
Dependency analysis performing. Starting time is : 23:03:46
Dependency analysis completed. Stop time is : 23:04:42
Verification with mutant versions is performing. Starting time is : 23:04:42
Error in Test Case Execution for PlaceBooking
Exception: System.Web.Services.Protocols.SoapException:
System.Web.Services.Protocols.SoapException: Server was unable to process request. --->
System.InvalidOperationException: There is an error in XML document (2, 42). --->
System.Xml.XmlException: An error occurred while parsing EntityName. Line 2, position 42.
  at System.Xml.XmlTextReaderImpl.Throw(Exception e)
```

Method Name	Test Case	Mutant Version	Result
GetLocations	1	TEST_MUTANT_DATA_VERSION_0	Success
GetLocations	2	TEST_MUTANT_DATA_VERSION_0	Success
GetLocations	3	TEST_MUTANT_DATA_VERSION_0	Success
GetLocations	4	TEST_MUTANT_DATA_VERSION_0	Success
GetLocations	5	TEST_MUTANT_DATA_VERSION_0	Success
GetLocations	6	TEST_MUTANT_DATA_VERSION_1	Success
GetLocations	7	TEST_MUTANT_DATA_VERSION_2	Success
GetLocations	8	TEST_MUTANT_DATA_VERSION_3	Success
GetLocations	9	TEST_MUTANT_DATA_VERSION_4	Success
GetLocations	10	TEST_MUTANT_DATA_VERSION_5	Success

```
CountryID<Int32> Value : 6
```

Figure 3.5: The Graphical User Interface of Application - II



```
http://www.edgrencompany.com/PassiveSvr/Service.asmx
http://www.themidnightcoders.com/weborbexamples/BenchmarkService.asmx
http://vietnamdynamictravel.com/default/1459e185-aa6e-4ac8-a06a-150c2b0c2f28/default.asmx
http://vietnamdynamictravel.com/default/aboutus/default.asmx
http://www.reservations.wwcars.co.uk/WWCarsOnlineRes/XML/wwcarsXMLInterface.asmx
http://cardealer.com.pk/Services/cars.asmx
http://www.softwaremaker.net/webservices/swm/validator/validator.asmx
http://www.fixedpricecarservice.com.au/Services/WebService.asmx
http://www.disfacar.com/CT.asmx
https://www.carwebuk.com/CarweBVRRB2Bproxy/carwebvrrwebservice.asmx
http://www.texas.aaa.com/en-tx/buy-maintain-vehicles/car-buying-services/_layouts/ACSC/Service/Toc
http://www.carquotes.com/Tools/DynamicPopulateDictionary.asmx
http://web1.carparts-cat.com/wsvc/Global/SessionItem.asmx
http://web2.carparts-cat.com/WSVC/ERP/ErpAppWSVC.asmx
http://www.cartime.com.au/WebService/CarService.asmx
http://paul-todd.co.uk/maiaonlinewebservice.asmx
http://www.carwebdataservices.com/Services/ReportCustomisationService.asmx
http://www.euromotorcar.net/_WebServices/SlideShowImages.asmx
http://carseller.co.nz/controls/searchService.asmx
http://cartimes.com/getmake.asmx
http://www.cartimes.com/getmake.asmx
http://www.nationalcarservice.com.au/Components/SuburbSuggest.asmx
http://nationalcarservice.com.au/Components/SuburbSuggest.asmx
http://www.carbau.com/CarService.asmx
```

```
wwcarsXMLInterface
    GetCountries
    GetLocations
    GetSupplierTerms
    GetAvailableExtras
    CarAvailabilityByAirport
```

```
ResultsGetLocations  GetLocations(Int32 CountryID);
```

```
Dependency analysis performing. Starting time is : 23:03:46
Dependency analysis completed. Stop time is : 23:04:42
Verification with mutant versions is performing. Starting time is : 23:04:42
Error in Test Case Execution for PlaceBooking
Exception: System.Web.Services.Protocols.SoapException:
System.Web.Services.Protocols.SoapException: Server was unable to process request. --->
System.InvalidOperationException: There is an error in XML document (2, 42). --->
System.Xml.XmlException: An error occurred while parsing EntityName. Line 2, position 42.
  at System.Xml.XmlTextReaderImpl.Throw(Exception e)
```

Method Name	Version	#Test Case	#Successful
GetLocations	TEST_MUTANT_DATA_VERSION_0	5	5
GetLocations	TEST_MUTANT_DATA_VERSION_1	1	1
GetLocations	TEST_MUTANT_DATA_VERSION_2	1	1
GetLocations	TEST_MUTANT_DATA_VERSION_3	1	1
GetLocations	TEST_MUTANT_DATA_VERSION_4	1	1
GetLocations	TEST_MUTANT_DATA_VERSION_5	1	1
GetLocations	TEST_MUTANT_DATA_VERSION_6	1	1

Test Cases Success Ratio: 1 Difference Ratio: 1 Final Result: 1

```
<ResultsGetLocations>
    ReturnCode<Int32> Value : 0
    ErrorMessage<String> Value : Success
    PickupLocationSet<ResultsPickupLocation[]>Count :4
        [0]
        [1]
            LocationID<Int32> Value : 524
            LocationName<String> Value : SOFIA AIRPORT
            LocationType<String> Value : A
            LocationCode<String> Value : SOF
            MinimumAge<Int32> Value : 25
        [2]
        [3]
```

Figure 3.6: The Graphical User Interface of Application - III

CHAPTER 4

AUTOMATIC WEB SERVICE TESTING ALGORITHM

The proposed work in this thesis is based on atomic web services that are specified in WSDL. Each service provider has a WSDL document to specify the information about the provided web services. This document contains the names of the web services, the attributes of input and output parameters of the web services and also user defined types. By using this document, information about all services can be obtained.

Unfortunately, the obtained information is not sufficient to test the web services. The specification that is written in WSDL are available. However, the source code and behavioral information of web service are not available. Therefore, only black-box testing can be performed for validation and testing of web service.

The definition for web service validation and web service testing can be given as follows. *Web service validation* is the process of checking whether the web service is still alive and accessible or not by invoking the web service by simple appropriate parameter values. Whereas, *web service testing* is the task of checking whether a web service functions as it should be.

Both processes require generation of input values for test cases. Firstly, the types of the input and output parameters of a given web service should be identified. There are a set of fundamental types for parameters, such as boolean, character, integer and floating point number. There are also user defined enumeration types to constrain the sets of values of a parameter. By using these fundamental types, different complex types can be constructed, such as array types, reference type, class type or data structures such as list or stack. To analyze and generate a value for the parameter with one of the fundamental types is more straightforward than with complex types. In order to generate test data of complex data types, the generator recursively analyzes the structure of the data type until it reaches the fundamental type.

The second step is to assign the value to each input parameter of web service. This step is different for validation and testing process due to the aim of the processes.

The aim of the validation process is to check whether the web service is still alive and accessible or not by invoking the web service. Therefore, setting simple default values to input parameters is sufficient for validation process. The default values used for fundamental parameter types are as shown in Table 4.1.

Table 4.1: Default Values of Web Service Parameters for Validation

Parameter Type	**Parameter Value**
Int64 or Int32 or Int16 or Byte	1
Double or Decimal or Single	1.0
Boolean	true
String	”text”
DateTime	Current Date and Time
Enum	First value of the type of the parameter

After assigning input parameter values, web services are invoked and the responses are analyzed in order to check whether the service is still active.

The aim of the service testing process is to determine whether the appropriate outputs are obtained for the given different inputs. In order to test a web service, it is expected that no exception is taken when the web service is invoked by using the generated input values. Another important point in testing process is to check whether the service gives different outputs for different input values. If a service always returns the same value in spite of different input parameters, it is considered that it is a test service or this service has no implementation. Such services are failed in test.

In the testing process, the most important issue is generating the most appropriate input values for test cases. In this work, two different methods are used for generating the input values. The first method uses dependency analysis to generate different candidate value for parameter. In the second method, different random values in the range of parameter’s type and specific values are generated. This method also uses external ontology sources for generation of input values.

4.1 Semantic Dependency Analysis-Based Test Case Generation Method

A service provider can provide multiple services and some of them have interaction with the other ones. In such a situation, for testing atomic web services, test cases can be generated by using the return values of other web services.

Semantic dependency analysis contains three types of dependency; input dependency, output dependency and input/output dependency.

- **input dependent**: A web service WS1 is "input dependent" on WS2 if and only if WS1 and WS2 share at least one input parameter that has the same type and same name.
- **output dependent**: A web service WS1 is "output dependent" on WS2 if and only if output parameters of WS1 and WS2 have the same type and same name.
- **input/output dependent**: A web service WS1 is "input/output dependent" on WS2 if and only if at least one input parameter of WS1 has the same type and similar name with at least one field of output parameter of WS2.

In this work, especially, semantic input/output dependencies are used. The generation of the values for input parameters of each web service that are provided by the same service provider is a very time consuming process. To deal with this problem, the input dependency and output dependency are also used. However, the output and input dependency analysis is performed just syntactically by comparing the types of input or output parameters of the web services.

Initially, input dependencies, output dependencies and the input/output dependencies between all validated services of each service provider are analyzed and according to this analysis result, the test case generation is performed. A web service may have no input parameter and also may return no value. For each validated web service with at least one input parameter, the dependencies with the other validated web services that are provided by the same service provider and return a value except void are analyzed. For each validated web service, the following steps are performed in the analysis phase.

Firstly, the number of input parameters and type of each one of them are obtained. After that, in order to find the web services whose return parameter contains at least one field having the same type with the analyzed input parameter, for each input parameter, the return parameters of other validated web services are retrieved. The dependency results of the analyzed input parameter are kept in a list.

If a web service with dependency is found, all fields of output parameters having the same type with the analyzed input parameter are taken and the names of the fields are compared with the name of input parameter and the similarity matching degree between them is calculated.

Nevertheless, the fields of output parameters whose type is the same as the type of the analyzed input parameter can have no name. In this situation, the calculation of the matching degree is impossible for these fields. The following alternatives are tried for matching degree calculation.

If the name of the output parameter is available although its type is not matched with the input parameters, the matching degree of the name of the output parameter with the name of input parameter is calculated by applying the same method, which is used for the name of field of output parameters. The similarity result and the name of output parameter and its web service are recorded to dependency list of analyzed input parameter. In the worst case, both the output parameter and each field of it do not have any name although itself or some of fields have the same type as the analyzed input parameter. In this situation, the name of the web service whose output parameter is analyzed is used in calculation of similarity degree.

In the matching degree calculation phase, the functions of word matching library [14], which are extended from WordNet matching library are used. It performs both syntactic and semantic matching. While finding matches, each term in the first parameter is compared with the all of the terms in second parameters. In addition, these terms should conform to some special conditions.

If the name of the parameters to be compared is composed of a set of words, in order to extract the words, the following steps are performed for both input and output terms.

1. The term is split into tokens from capital cases or digits that are applicable to the case [14].
2. Special characters like fg():;.,!”? are removed from each tokens.
3. Stop words such as ”get”, ”set”, ”list”, ”by”, ”id”, ”code”, ”and”, ”is”, etc. are removed from token list.

4. The tokens in token list are combined by adding a space characters after each token and a full string from tokens are obtained.

After each step is performed for both input term and output terms, the obtained results can be fed into similarity function as an argument.

Matching each word of the first parameter term is performed by comparing all of words in the second parameter term one by one. If the first parameter has *m* words and the second parameter has *n* words, totally *m* x *n* comparisons are carried on. According to these word comparisons, similarity of the words are calculated. In order to calculate similarity of each pair of words, we need to find the synonyms of each word from different senses. Each synonym can have different derivation hierarchies. Synonyms of each word and its derivation hierarchies are obtained from WordNet. The hierarchy elements of first word are compared with the hierarchy elements of second word. Then, the nearest common parent element and the words' derivation hierarchy levels are determined. In the light of this information, the similarity of words is calculated by using Equation 4.1 [14].

$$similarity\left(term1_{word_i}, term2_{word_j}\right) = \frac{2 * derivation\_order\_of\_nearest\_level}{derivation\_level\_of\_sense1 + derivation\_level\_of\_sense2} \tag{4.1}$$

After calculating all similarities, the similarity of words is decided as maximum of these similarity values by using Equation 4.2 [14].

$$similarity\left(term_1, term2_{word_j}\right) = \mathbf{max}_i(similarity(term1_{word_i}, term2_{word_j})) \tag{4.2}$$

After these processes, we obtain a similarity degree array with length of *n*. However, we desire to get a single value as the similarity degree of whole output term to whole input term. Therefore, the average value of these similarity degrees is calculated for the final similarity value (S1) by using Equation 4.3.

$$similarity\,(term_1, term_2) = \frac{1}{n}\sum_{j=1}^{n} similarity(term_1, term2_{word_j}) \tag{4.3}$$

We devise also another method to calculate the similarity degree between input and output terms. In this method, the distance between the input term and each ontology terms and the distance between each output term and each ontology terms are calculated. These distances are used to determine the ontological places of the terms. By calculating the average of the differences of these distance values with each ontology term, the similarity value (S2) between input term and output term is calculated by using Equation 4.4.

$$ontSim(term_1, term_2) = 1 - \frac{1}{N}\sum_{k=1}^{N}|similarity(term_1, ontTerm_k) - similarity(term_2, ontTerm_k)| \tag{4.4}$$

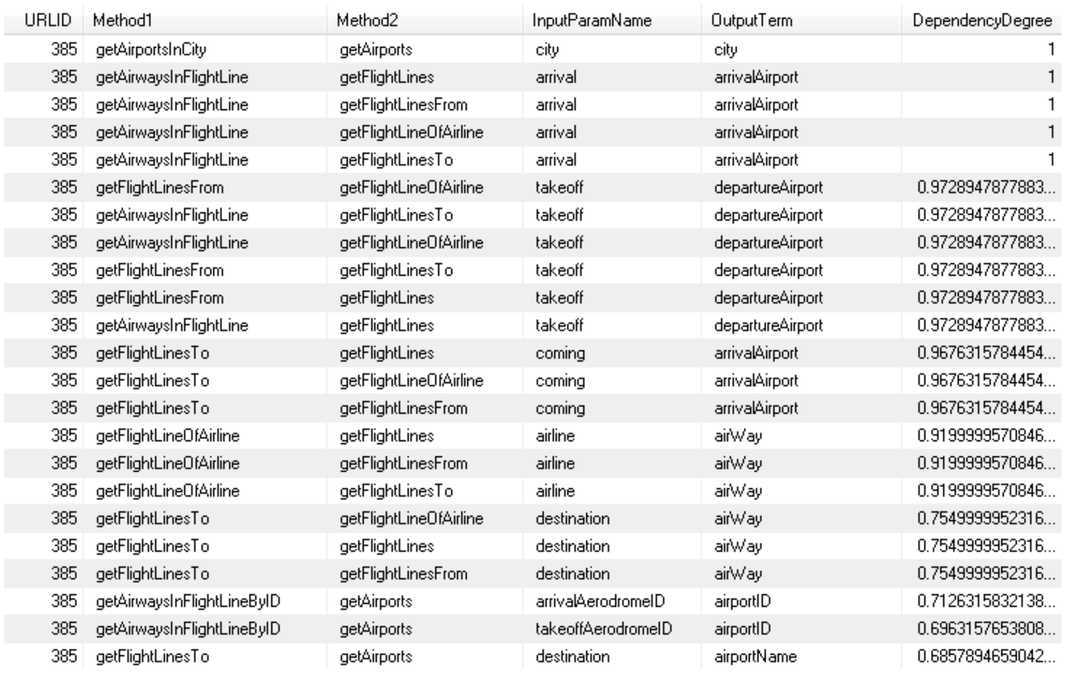
where N is the count of the ontology terms.

As described above, similarity values from two different ways are obtained. These values are named as S1 and S2. Final similarity value is calculated by using Equation 4.5.

$$finalSimilarity(term_1, term_2) = \frac{S1 + S2}{2} \tag{4.5}$$

The final similarity value with the name of the output term and the web service whose output parameter is analyzed are recorded to input/output dependency list of analyzed input parameter. A sample list of input/output dependency relations of web services is shown in Figure 4.1.

The overall algorithm of dependency analysis for a web service that are mentioned above is as follows.



URLID	Method1	Method2	InputParamName	OutputTerm	DependencyDegree
385	getAirportsInCity	getAirports	city	city	1
385	getAirwaysInFlightLine	getFlightLines	arrival	arrivalAirport	1
385	getAirwaysInFlightLine	getFlightLinesFrom	arrival	arrivalAirport	1
385	getAirwaysInFlightLine	getFlightLineOfAirline	arrival	arrivalAirport	1
385	getAirwaysInFlightLine	getFlightLinesTo	arrival	arrivalAirport	1
385	getFlightLinesFrom	getFlightLineOfAirline	takeoff	departureAirport	0.9728947877883...
385	getAirwaysInFlightLine	getFlightLinesTo	takeoff	departureAirport	0.9728947877883...
385	getAirwaysInFlightLine	getFlightLineOfAirline	takeoff	departureAirport	0.9728947877883...
385	getFlightLinesFrom	getFlightLinesTo	takeoff	departureAirport	0.9728947877883...
385	getFlightLinesFrom	getFlightLines	takeoff	departureAirport	0.9728947877883...
385	getAirwaysInFlightLine	getFlightLines	takeoff	departureAirport	0.9728947877883...
385	getFlightLinesTo	getFlightLines	coming	arrivalAirport	0.9676315784454...
385	getFlightLinesTo	getFlightLineOfAirline	coming	arrivalAirport	0.9676315784454...
385	getFlightLinesTo	getFlightLinesFrom	coming	arrivalAirport	0.9676315784454...
385	getFlightLineOfAirline	getFlightLines	airline	airWay	0.9199999570846...
385	getFlightLineOfAirline	getFlightLinesFrom	airline	airWay	0.9199999570846...
385	getFlightLineOfAirline	getFlightLinesTo	airline	airWay	0.9199999570846...
385	getFlightLinesTo	getFlightLineOfAirline	destination	airWay	0.7549999952316...
385	getFlightLinesTo	getFlightLines	destination	airWay	0.7549999952316...
385	getFlightLinesTo	getFlightLinesFrom	destination	airWay	0.7549999952316...
385	getAirwaysInFlightLineByID	getAirports	arrivalAerodromeID	airportID	0.7126315832138...
385	getAirwaysInFlightLineByID	getAirports	takeoffAerodromeID	airportID	0.6963157653808...
385	getFlightLinesTo	getAirports	destination	airportName	0.6857894659042...

Figure 4.1: Input/Output Dependency Relations of Web Services

Algorithm 4.1.1 `AnalyzeInputOutputDependency`

```
1: get input parameters of web service
2: for each $inputParam \in$ input parameter list do
3:   $term_1 \leftarrow$ name of $inputParam$
4:   for each $WS \in$ list of other web service provided by the same service provider do
5:     $maxSimilarity \leftarrow 0$
6:     $dependentField \leftarrow null$
7:     $outputParam \leftarrow$ output parameter of $WS$
8:     if $outputParam \neq$ void then
9:       for each $field \in$ field list of $outputParam$ do
10:        if type of $inputParam$ = type of $outputParam$ then
11:          if name of $field \neq null$ then
12:            $term_2 \leftarrow$ name of $field$
13:          else
14:            if name of $outputParam \neq null$ then
15:              $term_2 \leftarrow$ name of $outputParam$
16:            else
17:              $term_2 \leftarrow$ name of $WS$
18:            end if
19:          end if
20:          $S1 \leftarrow similarity(term_1, term_2)$
21:          $S2 \leftarrow ontSim(term_1, term_2)$
22:          $finalSimilarity \leftarrow (S1 + S2)/2$
23:          if $finalSimilarity > maxSimilarity$ then
24:            $maxSimilarity \leftarrow finalSimilarity$
25:            $dependentField \leftarrow field$
26:          end if
27:        end if
28:      end for
29:      insert $< WS, field, maxSimilarity >$ to similarity list of $inputParam$
30:    end if
31:  end for
32: end for
```

In input dependency analysis, other web services that have at least one common input parameter with the web service under analysis are searched. The input parameters must have both the same name and the same type to accept the web services as input dependent.

In output dependency analysis, other web services that share the output parameters with the web service in analysis phase are searched. The obtained web services are accepted as output dependent with the web service in analysis. With the method mentioned above, for each parameter of all web services that are provided by the handled service provider, the dependency analysis results are obtained. After these processes, by utilizing the final analysis results, different values for the input parameters and also different test cases are generated for each web service.

4.2 Data Mutation-Based Test Case Generation Method

In software engineering, the purpose of the mutation testing is to help the tester develop effective tests or locate weaknesses in the test data used for the program or in sections of the code that are seldom or never accessed during execution [15]. The proposed method is inspired from mutation testing methods, however the aim is quite different. The aim of mutation testing is to measure test adequacy. On the other hand, the aim of data mutation in the proposed method is to generate test case. In traditional mutation testing, mutation operators are used to transform the program under test. In contrast, this method is applied for generating random and error-prone data inputs.

In mutation-based test case generation, a parameter can have different values in the range of its type domain. In this method, the various random and error-prone data input values are generated and they are grouped. When we use the generated values in each range for input parameter, we expect that invoked web service presents different behaviors.

We mention that types of input parameters can be fundamental or complex type. In case of having complex type, the generator recursively analyzes the structure of the input parameter type until it reaches the fundamental type. For some of types like Int64, Int32, Int16, Byte, Decimal, Double, Single, Double, Boolean, Enum and DateTime where the range is limited, conventional-based method can be used easily in order to generate value. When compared to these types, generating a valid value for String typed input parameters is a challenging issue. Therefore, different methods should be used. For this reason, we also use ontology-based test case generation method for input parameter whose type is String.

However Numeric and DateTime input parameters can be defined with String type. For such input parameters, we make some checking to detect the actual types of them and by using conventional-based test case generation method, valid values are generated and converted to String. If the name of the parameter contains ”id”, ”number” or ”count”, it is handled as Byte type. If the name of the parameter contains ”DateTime” or ”Time”, it is handled as DateTime type. If the name of the parameter contains ”Date”, it is handled as DateTime type and the value is generated by using ”yyyy-MM-dd” format.

4.2.1 Conventional-Based Test Case Generation Method

In this method, various values in the range of its type are generated and the generated values are grouped. For each group, the range of parameter value is also constrained. In Mutant Version 0, positive values are generated for input parameters. When the web service is invoked by using these values, it is expected that invoked web service returns a proper value. In Mutant Version 1, 2, 3, 4, 5 and 6, error-prone input values are generated.

For numeric types, in Mutant Version 1, input parameter is set to 0. By this way, it can be checked whether the web service prevents division by zero error or pointer address error.

In Mutant Version 2, the numeric input parameter is set to -1 and thus it is checked whether unsigned to signed conversion error, out-of-bounds memory access error, signed/unsigned mismatch warning in comparison is prevented by web service. Incorrect sign conversions can lead to undefined behavior and the web service can be crashed. In the web service, the input parameter can be assigned other variables. The developers often define the integer variable as Int32, however signed/unsigned property of type is important. Some variables can be used to define ”index”, ”size”, ”count” and such variables should have signed type and to assign -1 to such variable can be lead the error.

In Mutant Version 3 and 4, boundary values are generated. In Mutant Version 5 and 6, very high and low values are generated randomly. Thus, the fault resistance of the web service is checked.

For each type, the values that can be generated are different. Therefore, the mutant groups are constructed differently with respect to types. For numeric types, the mutant groups are constructed as shown in Table 4.2. The mutant groups shown in Table 4.3 are for DateTime type. For Enum type, the mutant groups are constructed as shown in Table 4.4 and Table 4.5 is for Boolean type.

Table 4.2: Mutant Groups for Numeric Types

Mutant Version 0	In this version, a value in range 1-100 is randomly generated.
Mutant Version 1	In this version, the value of parameter is set to 0
Mutant Version 2	In this version, the value of parameter is set to -1
Mutant Version 3	In this version, the value of parameter is set to the maximum value of its range.
Mutant Version 4	In this version, the value of parameter is set to the minimum value of its range.
Mutant Version 5	In this version, a positive value between 1000 and the maximum value is randomly generated.
Mutant Version 6	In this version, a negative values between -1000 and the minimum value is randomly generated.

Table 4.3: Mutant Groups for DateTime Type

Mutant Version 0	In this version, a time value between time of Today and time, 31.12.Today.Year 23:59:59 is randomly generated.
Mutant Version 1	In this version, the value of parameter is set to time, 1.1.1 1:1:1
Mutant Version 2	In this version, the value of parameter is set to time, 31.12.Today.Year 23:59:59
Mutant Version 3	In this version, the value of parameter is set to the maximum time value of its range.
Mutant Version 4	In this version, the value of parameter is set to the minimum time value of its range.
Mutant Version 5	In this version, a time value between time of Today and the maximum time value is randomly generated.
Mutant Version 6	In this version, a time value between time of Today and the minimum time value is randomly generated.

Table 4.4: Mutant Groups for Enum Type

Mutant Version 0	In this version, an integer representing one of the values in the enum is generated.
Mutant Version 1 and 2	In this version, the value of parameter is set to first value in the enum.
Mutant Version 3 and 5	In this version, the value of parameter is set to upper value in the enum.
Mutant Version 4 and 6	In this version, the value of parameter is set to the lower value in the enum.

As another mutant version for the test case generation, switching the values of input parameters with the same type is performed. As source test cases, we use the test cases that are

Table 4.5: Mutant Groups for Boolean Type

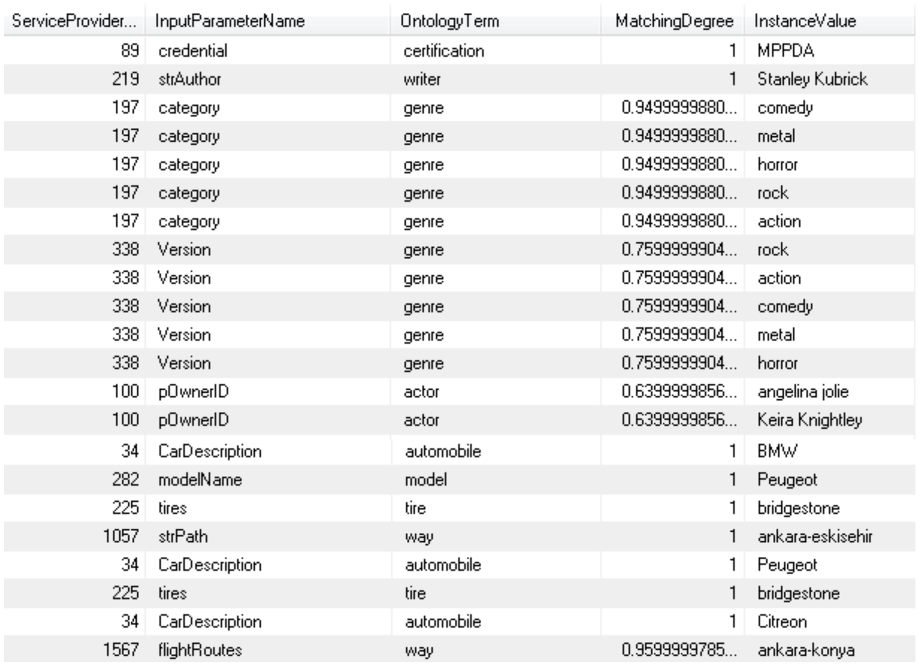
Mutant Version 0, 1 and 2	In this version, the value of parameter is set to true.
Mutant Version 3, 4, 5 and 6	In this version, the value of parameter is set to false.

generated in version 0 and provide that the web services are invoked successfully. The values of input parameters having the same type are randomly mixed and new test cases are generated by this way.

4.2.2 Ontology-Based Test Case Generation Method

This method is used in order to generate valid values for String typed textual input parameters. In DSWSD-S System, each subnode has its own ontology. Having such an ontology is useful for extracting the domain instances for parameters. If they are available, the instances can be directly taken from the ontology or the ontology can be populated with instances by using public resources.

Some of the obtained DSWSD-S ontology instances for input parameter are listed in Figure 4.2.



ServiceProvider...	InputParameterName	OntologyTerm	MatchingDegree	InstanceValue
89	credential	certification	1	MPPDA
219	strAuthor	writer	1	Stanley Kubrick
197	category	genre	0.9499999880...	comedy
197	category	genre	0.9499999880...	metal
197	category	genre	0.9499999880...	horror
197	category	genre	0.9499999880...	rock
197	category	genre	0.9499999880...	action
338	Version	genre	0.7599999904...	rock
338	Version	genre	0.7599999904...	action
338	Version	genre	0.7599999904...	comedy
338	Version	genre	0.7599999904...	metal
338	Version	genre	0.7599999904...	horror
100	pOwnerID	actor	0.6399999856...	angelina jolie
100	pOwnerID	actor	0.6399999856...	Keira Knightley
34	CarDescription	automobile	1	BMW
282	modelName	model	1	Peugeot
225	tires	tire	1	bridgestone
1057	strPath	way	1	ankara-eskisehir
34	CarDescription	automobile	1	Peugeot
225	tires	tire	1	bridgestone
34	CarDescription	automobile	1	Citreon
1567	flightRoutes	way	0.9599999785...	ankara-konya

Figure 4.2: The DSWSD-S Ontology Instances for Input Parameter

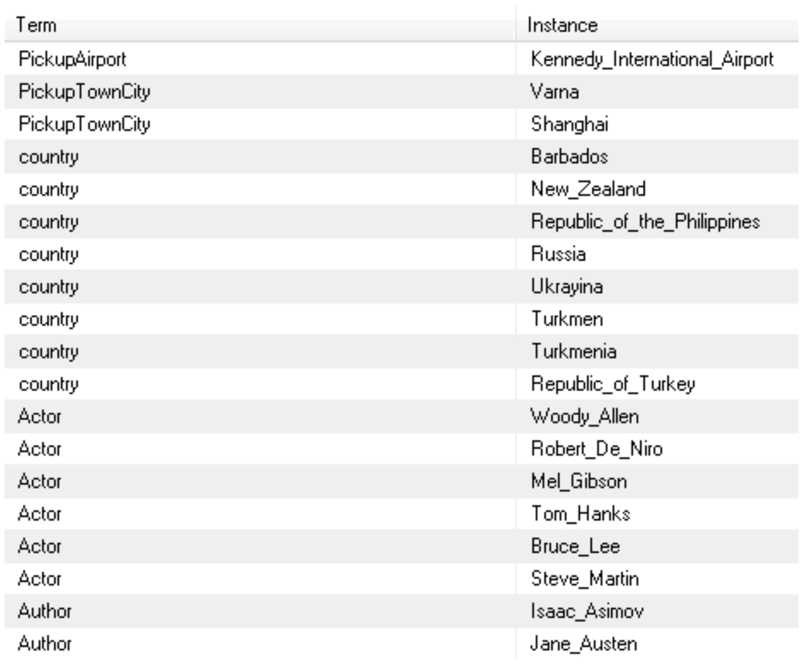
In ontology version, DSWSD-S ontology is used as the external resource and the following steps are performed in order to generate the values of input parameters.

As the first step, the name of input parameters are semantically compared with the ontology terms in order to find the similarity value and ontological position of the input parameters. We obtain the ontology terms with matching degree above the threshold value. The domain instances of these ontology terms are used for generating the values of input parameters.

As another version, WordNet [16], is used for generation of input parameter value. In WordNet, nouns are organized into hierarchies based on different relations between synsets. Two of these relations are more meaningful for us in process of generation input value.

Hyponym: The specific term used to designate a member of a class. **X** is a hyponym of **Y** if **X** is a (kind of) **Y** [16].

Instance: A proper noun that refers to a particular, unique referent (as distinguished from nouns that refer to classes). This is a specific form of hyponym [16].



Term	Instance
PickupAirport	Kennedy_International_Airport
PickupTownCity	Varna
PickupTownCity	Shanghai
country	Barbados
country	New_Zealand
country	Republic_of_the_Philippines
country	Russia
country	Ukrayina
country	Turkmen
country	Turkmenia
country	Republic_of_Turkey
Actor	Woody_Allen
Actor	Robert_De_Niro
Actor	Mel_Gibson
Actor	Tom_Hanks
Actor	Bruce_Lee
Actor	Steve_Martin
Author	Isaac_Asimov
Author	Jane_Austen

Figure 4.3: The WordNet Instances for Input Parameter

We check whether there is any instance for the ontology term corresponding to the name of input parameter. If it has at least one instance, a random one of them is used for input parameter value in test case generation. Otherwise, the hyponym terms of the name of input parameter are searched and if any term is obtained, the instances of the obtained hyponym terms are searched. If it has at least one instance, a random one of them is assigned to input parameter as value. Some of the obtained WordNet instances for input parameter are listed in Figure 4.3.

4.3 Test Case Generation and Execution

In this phase, the overall process of test case generation is described. For all of validated web services, different test cases based on both semantic analysis operation and data mutations are generated.

Firstly, the test case generation based on data mutant analysis is performed. By using each version of mutant value generation mentioned in the previous section, different values are generated for each input parameter. Web service is invoked with these input parameters and return value is obtained.

In Mutant Version 0, 5 different values in the given range are generated for each input parameter. By setting the input parameters to these generated values, the test case is prepared for the web service. The next step is test case execution. In this phase, web services are invoked and the results are checked. If an exception occurs during the execution of the web service request, then the web service is accepted as unsuccessful. This result is recorded to statistics of testing of web service. On the other hand, it is expected that the web services that cause no exception have different return values. Currently, a web service pass the test if it produces different outputs to different inputs. However, the obtained result value will be further analyzed in detail.

The similar steps are followed in the other mutant versions. In Mutant Version 1, 2, 3, 4, 5 and 6, just one test case is generated. To test the web service successfully, it is expected that no exception is occurred. In Mutant Version 5 and 6, the possibility of being failed is higher than the other versions because the web services might not handle the values in these ranges. As mentioned for Mutant Version 0, the return values are analyzed in detail and the test results are recorded.

For mutant version in which the values of input parameters from ontology instances are obtained, one test case is generated in which the values of input parameters from WordNet instances are obtained.

As the last mutation version, input parameter switching is performed. The test cases that are generated in mutant version 0 and that provide successful test result are used again in switching version, if the parameters are suitable.

As the second phase, test case generation based on dependency analysis is performed. If the web service has no input parameter, it is invoked without generating any input value and the invocation result is analyzed. If it has at least one input parameter, the value set is generated for each input parameter by performing the following steps.

Firstly, the dependency list is analyzed. If it has no dependency with the return parameters of the other web services, considering its type, a random value is generated by using version 0 in the mutant-based method. Otherwise, by starting with the first web service in the dependency list, the test cases of web services is analyzed to get the appropriate value from its return value.

The dependency relation list contains the information as to which web service's output parameter has dependency and its dependency degree. The list is sorted by dependency degree, therefore the analysis operation is started with the first one. If the first web service in the list has no successful test case, it is not used for generation and the analysis is continued with the next web service in the dependency list. In case of failure, these steps are recursively performed until a web service in the dependency list has successfully executed test case is.

The dependency relation list contains also the name of the field with the same type as the input parameter for which it is to be used to generate a value. By utilizing this data, the value is acquired from the return value of the web service, whose successful test case is determined. Therefore the output value is analyzed to find this field and get its value. If this field is the element of an array, one of the values in list is selected randomly. The acquired value for input parameter can be equal to null, 0 or empty string "". In this case, this value is not preferred as input value and therefore, the next web service in dependency list is tried. When an appropriate dependency is not obtained from the dependency list, a random value is generated by using 'version 0' in the mutant-based method.

After the values are generated for all input parameters of web services in testing phase, it is ready to be invoked. After this state, the following steps are the same as the testing process in which the mutation method is used. The services are invoked and the return values are analyzed to decide whether it passes the test. As the final process, the test results are recorded.

As an important point for performance, before the test case generation, the input dependencies are checked from the earlier test results. If a web service with input dependency is successfully tested before, the values of the input parameters of this web service can be again used for the web service in test phase.

The algorithm of test case generation and execution that are described above is given in Algorithm 4.3.2 and 4.3.3.

Algorithm 4.3.2 `GenerateAndInvokeTestCase`

```
1:  for $i \leftarrow 0, 6$ do
2:     if $i \geq 0$ then
3:        $countTestCase \leftarrow 1$
4:     else
5:        $countTestCase \leftarrow 5$
6:     end if
7:     for $j = 1 \rightarrow countTestCase$ do
8:        $T \leftarrow$ create a new test case
9:        for each $p \in$ input parameter list do
10:          generate value in the range of mutant version $i$
11:          assign generated value to $p$
12:       end for
13:       InvokeTestCase($T$)
14:       insert $T$ to test case list
15:    end for
16: end for
17: $T \leftarrow$ create a new test case
18: for each $p \in$ input parameter list do
19:    calculate the similarities between the name of $p$ and ontology terms
20:    $ontTerm \leftarrow$ ontology term, which provides highest similarity
21:    get ontology instances of $ontTerm$ from DSWSD database
22:    assign one of found ontology instances to $p$
23: end for
```

```
24: InvokeTestCase($T$)
25: insert $T$ to test case list
26: $T \leftarrow$ create a new test case
27: for each $p \in$ input parameter list do
28:    get WordNet instances of similar WordNet ontology term
29:    assign one of found WordNet instances to $p$
30: end for
31: InvokeTestCase($T$)
32: insert $T$ to test case list
33: if at least two parameters have the same type then
34:    $successfulTestCaseList \leftarrow$ the list of test cases, which provide $SUCCESSFUL$ result
35:    for each $p \in successfulTestCaseList$ do
36:       $T \leftarrow$ create a new test case
37:       switch the values of parameters, which have the same type randomly
38:       assign switched values to parameters
39:       InvokeTestCase($T$)
40:       insert $T$ to test case list
41:    end for
42: end if
43: i $\leftarrow$ 0
44: while $i \leq 5$ do
45:    for each $p \in$ input parameter list do
46:       if length of input/output dependent web service list > 0 then
47:          for each $DW \in$ input/output dependency list do
48:             $DST \leftarrow$ list of $DW$'s successful test cases, which are generated with dependency ver-
                sion
49:             for each $sourceTestCase \in DST$ do
50:                if return value of $sourceTestCase \neq$ null then
51:                   get the value of dependent field of return value of $sourceTestCase$
52:                   assign this value to $p$
53:                   break
54:                end if
55:             end for
56:             if the value of p is assigned then
57:                break
58:             end if
59:          end for
```

```
60:         end if
61:     end for
62:     InvokeTestCase(T)
63:     insert $T$ to test case list
64:     i ← i+1
65: end while
```

Algorithm 4.3.3 `InvokeTestCase( testCase )`

```
1: execute the new test case
2: if exception occurs then
3:     result of testCase ← $UNSUCCESSFUL$
4: else
5:     result of testCase ← $SUCCESSFUL$
6:     save the return value of $T$
7: end if
```

CHAPTER 5

CASE STUDIES AND EVALUATION

This chapter consists of two parts. The first part is about the analysis method. The second part presents the experimental results of the proposed algorithm and some statistical information.

5.1 Analysis Method for Test Results

As we described in previous sections, web service invocation results are obtained by calling a web service by using the input parameters that are generated with data mutation based methods and semantic dependency analysis based method.

The test results are generated by examining the invocation results in two dimensions. The first one is based on checking the execution results. An exception can occur during the execution of the web service request. In this case, the web service is accepted as unsuccessful. This result will be recorded to database of web service test statistics. On the other hand, if it causes no exception, it is accepted as successful. The second analysis dimension is examining the difference of invocation results for various test cases.

The calculation of final test value of a web service is summarized in Table 5.1 and Table 5.2. Table 5.1 shows how to calculate the ratio of the successful test cases that are generated by using each data generation method. First column presents number of the successful test cases and the first row presents number of the total test cases for each data generation method. In Table 5.2, the terms that is given in Table 5.1 are described.

Table 5.1: The Calculation of Ratio of Successful Test Cases

/	N_{MT0}	N_{MT1}	N_{MT2}	N_{MT3}	N_{MT4}	N_{MT5}	N_{MT6}	N_{OT}	N_{WT}	N_{SIT}	N_{DT}
N_{SMT0}	R_0										
N_{SMT1}		R_1									
N_{SMT2}			R_2								
N_{SMT3}				R_3							
N_{SMT4}					R_4						
N_{SMT5}						R_5					
N_{SMT6}							R_6				
N_{SOT}								R_7			
N_{SWT}									R_8		
N_{SSIT}										R_9	
N_{SDT}											R_{10}

Table 5.2: Definitions of Evaluation Terms

Term	**Definition**
N_{SMT0}	The number of successful test cases in which the input parameters are generated by mutant version 0.
N_{MT0}	The number of total test cases in which the input parameters are generated by mutant version 0.
R_0	The ratio of successful test cases using mutant version 0.
N_{SMT1}	The number of successful test cases in which the input parameters are generated by mutant version 1.
N_{MT1}	The number of total test cases in which the input parameters are generated by mutant version 1.
R_1	The ratio of successful test cases using mutant version 1.
N_{SMT2}	The number of successful test cases in which the input parameters are generated by mutant version 2.
N_{MT2}	The number of total test cases in which the input parameters are generated by mutant version 2.
R_2	The ratio of successful test cases using mutant version 2.
N_{SMT3}	The number of successful test cases in which the input parameters are generated by mutant version 3.
N_{MT3}	The number of total test cases in which the input parameters are generated by mutant version 3.
R_3	The ratio of successful test cases using mutant version 3.

Table 5.2 – Definitions of Evaluation Terms (continued)

Term	**Definition**
N_{SMT4}	The number of successful test cases in which the input parameters are generated by mutant version 4.
N_{MT4}	The number of total test cases in which the input parameters are generated by mutant version 4.
R_4	The ratio of successful test cases using mutant version 4.
N_{SMT5}	The number of successful test cases in which the input parameters are generated by mutant version 5.
N_{MT5}	The number of total test cases in which the input parameters are generated by mutant version 5.
R_5	The ratio of successful test cases using mutant version 5.
N_{SMT6}	The number of successful test cases in which the input parameters are generated by mutant version 6.
N_{MT6}	The number of total test cases in which the input parameters are generated by mutant version 6.
R_6	The ratio of successful test cases using mutant version 6.
N_{SOT}	The number of successful test cases in which the input parameters are generated by using ontology instances.
N_{OT}	The number of total test cases in which the input parameters are generated by using ontology instances.
R_7	The ratio of successful test cases using ontology instances.
N_{SWT}	The number of successful test cases in which the input parameters are generated by using WordNet instances.
N_{WT}	The number of total test cases in which the input parameters are generated by using WordNet instances.
R_8	The ratio of successful test cases using WordNet instances.
N_{SSIT}	The number of successful test cases in which the input parameters are generated based on mutation, switching the values of input parameters with the same type.
N_{SIT}	The number of total test cases generated by using switching method.
R_9	The ratio of successful test cases using input parameter switching method.
N_{SDT}	The number of successful test cases in which the input parameters are generated by using semantic dependency analysis-based method.

Table 5.2 – Definitions of Evaluation Terms (continued)

Term	Definition
N_{DT}	The number of total test cases in which the input parameters are generated by using semantic dependency analysis-based method.
R_{10}	The ratio of successful test cases generated by using semantic dependency analysis-based method.

If the web service has just one parameter, the test case generation by using switching method is not performed. Therefore, N_{SIT} is equal to 0 and in this case, the calculation is not applied. In the previous sections, we mentioned that the input parameters are generated by switching the input parameters of the source test cases. As the source test cases, we use the test cases that are generated in the version 0 and provide successful result when the web services are invoked. Therefore, it can be said that N_{SIT} is equal to N_{SMT0}.

To calculate the overall ratio of successful test cases generated by using the input parameter generation methods (VR_{RST}), Equation 5.1 and 5.2 are used.

If N_{SIT} is equal to 0,

$$VR_{RST} = (R_0 + R_1 + R_2 + R_3 + R_4 + R_5 + R_6 + R_7 + R_8 + R_{10})/10 \tag{5.1}$$

Else

$$VR_{RST} = (R_0 + R_1 + R_2 + R_3 + R_4 + R_5 + R_6 + R_7 + R_8 + R_9 + R_{10})/11 \tag{5.2}$$

In testing process, the second dimension is based on checking whether the service gives different outputs for different input values. If a service always returns the same value in spite of different input parameters, it is considered as a test service or this service has no implementation. If the same output value is obtained after all successful execution of test cases, the web services is failed for this point.

$$D_{ST} = 0 \tag{5.3}$$

where D_{ST} is the result of difference checking for all successful test cases. Otherwise, it is accepted as successful in test $D_{ST} = 1$.

The other difference checking is for the input parameter switching method. It is expected that different return values are obtained after the executions of the test cases that use the switching method and the source test cases whose input parameters are switched to generate the new test case. For each test case using input parameter switching method, if the return values of itself and source test case are different, it is accepted as successful in this checking process. Otherwise it is failed in test.

For the ratio of successful test cases that are generated by using the input parameter switching method and give different return value from its source test case (R_{DSST}), Equation 5.4 is used.

$$R_{DSST} = N_{DSSIT}/N_{SSIT} \tag{5.4}$$

where N_{DSSIT} is the number of successful test cases that are generated by using the input parameter switching method and gives, different return value from its source test case and N_{SSIT} is the number of successful test cases that are generated by using the input parameter switching method.

If there is no successful test case using switching input method (N_{SSIT} is equal to 0), this calculation is not applied.

The test result of these two difference checking process are calculated separately.

If N_{SSIT} is not equal to 0,

$$VR_{DST} = (D_{ST} + R_{DSST})/2 \tag{5.5}$$

Else

$$VR_{DST} = D_{ST} \tag{5.6}$$

By using the obtained results of the web services for two points mentioned above, the final test result is calculated. In testing of a web service, its successful invocation is more important than its returning different values. The web services that throw exception when they are invoked are never preferred by the service consumers. Therefore, the overall success result of the test cases is more weighted than the differences of obtained outputs. Equation 5.7 is used for the final test result.

$$VR_{final} = (VR_{RST} * 0.7) + (VR_{DST} * 0.3) \tag{5.7}$$

5.2 Experiments

5.2.1 Experiments with Web Services in DSWSD-S System

In DSWSD-S System, a set of nodes on different domains have been constructed. Each node crawls for and collects the web services related with its own domain. In our experiments, the proposed method is tested on the collected web services in the nodes of ”Car”, ”Aviation”, ”Film” and ”Sports” domains.

For the web services in ”Car” domain, the statistics about the collected web services are presented in Table 5.3.

Table 5.3: The Statistics about Experiments for ”Car” Domain

Total number of web service providers	904
Total number of validated web services	5713
Total number of input parameters	15284
Total number of string input parameters	9966
Total number of string input parameters whose value can be assigned from WordNet instances	1426
Average number of services per service provider	6.3
Average number of input parameter per web service	2.7

The number of the web services that are provided by the same provider are so varying that one service provider has only one web service whereas the other one has 54 web services. Some of them has no input parameter, on the other hand, there is a web service having 50 input parameters. There are both the input parameters with very complex types and with fundamental types. While some web services have no output parameter and it means that it returns ”void”, some of them return complex typed results.

For the web services in ”Aviation” domain, the statistics about the collected services are presented in Table 5.4.

For the web services in ”Film” domain, the statistics about the collected services are presented

Table 5.4: The Statistics about Experiments for ”Aviation” Domain

Total number of web service providers	33
Total number of validated web services	137
Total number of input parameters	163
Total number of string input parameters	105
Total number of string input parameters whose value can be assigned from WordNet instances	10
Average number of services per service provider	4.2
Average number of input parameter per web service	1.2

in Table 5.5.

Table 5.5: The Statistics about Experiments for ”Film” Domain

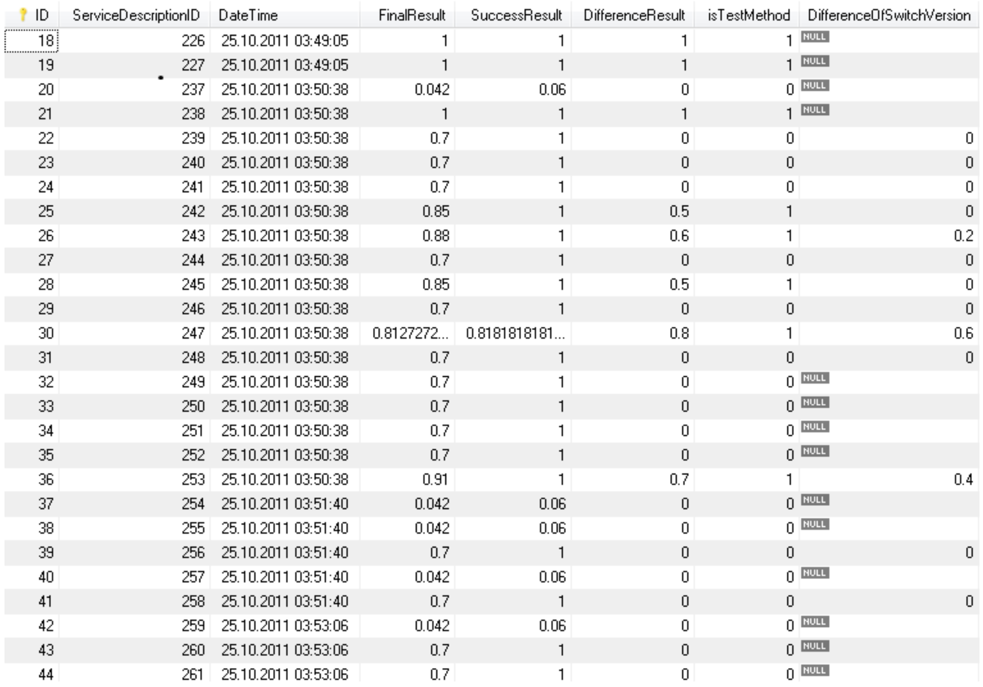
Total number of web service providers	60
Total number of validated web services	187
Total number of input parameters	233
Total number of string input parameters	162
Total number of string input parameters whose value can be assigned from WordNet instances	3
Average number of services per service provider	3.1
Average number of input parameter per web service	1.2

For the web services in ”Sports” domain, the statistics about the collected services are presented in Table 5.6.

Table 5.6: The Statistics about Experiments for ”Sports” Domain

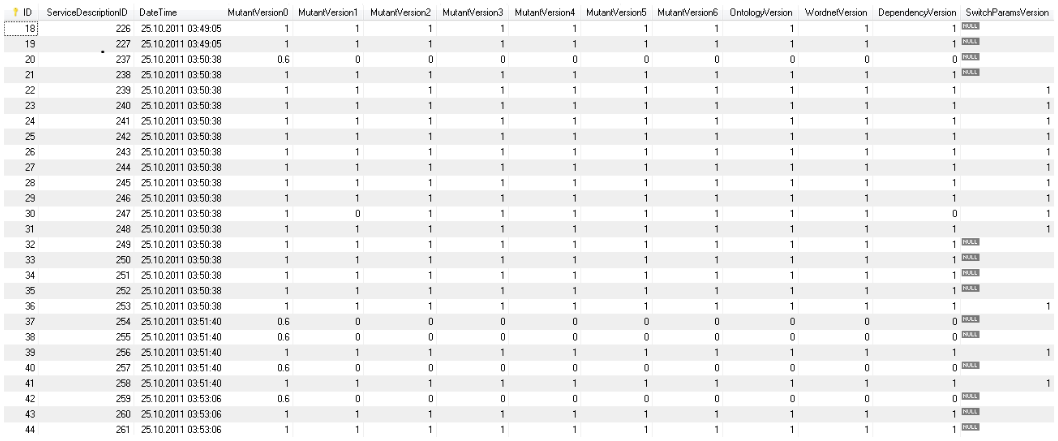
Total number of web service providers	60
Total number of validated web services	187
Total number of input parameters	218
Total number of string input parameters	148
Total number of string input parameters whose value can be assigned from WordNet instances	3
Average number of services per service provider	3.1
Average number of input parameter per web service	1.2

In the previous sections, it is mentioned that the test results of web services are obtained by using data mutation based methods and semantic dependency analysis based method. These results are recorded to database in details. For the web services in ”Car” domain, the obtained results are as shown in Figure 5.1 and Figure 5.2. Figure 5.3 and Figure 5.4 show the test results of the web services in ”Aviation” domain. In Figure 5.5 and Figure 5.6, the results of the web services in ”Film” domain are presented. Figure 5.7 and Figure 5.8 show the test results of the web services in ”Sports” domain.



ID	ServiceDescriptionID	DateTime	FinalResult	SuccessResult	DifferenceResult	isTestMethod	DifferenceOfSwitchVersion
18	226	25.10.2011 03:49:05	1	1	1	1	NULL
19	227	25.10.2011 03:49:05	1	1	1	1	NULL
20	237	25.10.2011 03:50:38	0.042	0.06	0	0	NULL
21	238	25.10.2011 03:50:38	1	1	1	1	NULL
22	239	25.10.2011 03:50:38	0.7	1	0	0	0
23	240	25.10.2011 03:50:38	0.7	1	0	0	0
24	241	25.10.2011 03:50:38	0.7	1	0	0	0
25	242	25.10.2011 03:50:38	0.85	1	0.5	1	0
26	243	25.10.2011 03:50:38	0.88	1	0.6	1	0.2
27	244	25.10.2011 03:50:38	0.7	1	0	0	0
28	245	25.10.2011 03:50:38	0.85	1	0.5	1	0
29	246	25.10.2011 03:50:38	0.7	1	0	0	0
30	247	25.10.2011 03:50:38	0.8127272...	0.8181818181...	0.8	1	0.6
31	248	25.10.2011 03:50:38	0.7	1	0	0	0
32	249	25.10.2011 03:50:38	0.7	1	0	0	NULL
33	250	25.10.2011 03:50:38	0.7	1	0	0	NULL
34	251	25.10.2011 03:50:38	0.7	1	0	0	NULL
35	252	25.10.2011 03:50:38	0.7	1	0	0	NULL
36	253	25.10.2011 03:50:38	0.91	1	0.7	1	0.4
37	254	25.10.2011 03:51:40	0.042	0.06	0	0	NULL
38	255	25.10.2011 03:51:40	0.042	0.06	0	0	NULL
39	256	25.10.2011 03:51:40	0.7	1	0	0	0
40	257	25.10.2011 03:51:40	0.042	0.06	0	0	NULL
41	258	25.10.2011 03:51:40	0.7	1	0	0	0
42	259	25.10.2011 03:53:06	0.042	0.06	0	0	NULL
43	260	25.10.2011 03:53:06	0.7	1	0	0	NULL
44	261	25.10.2011 03:53:06	0.7	1	0	0	NULL

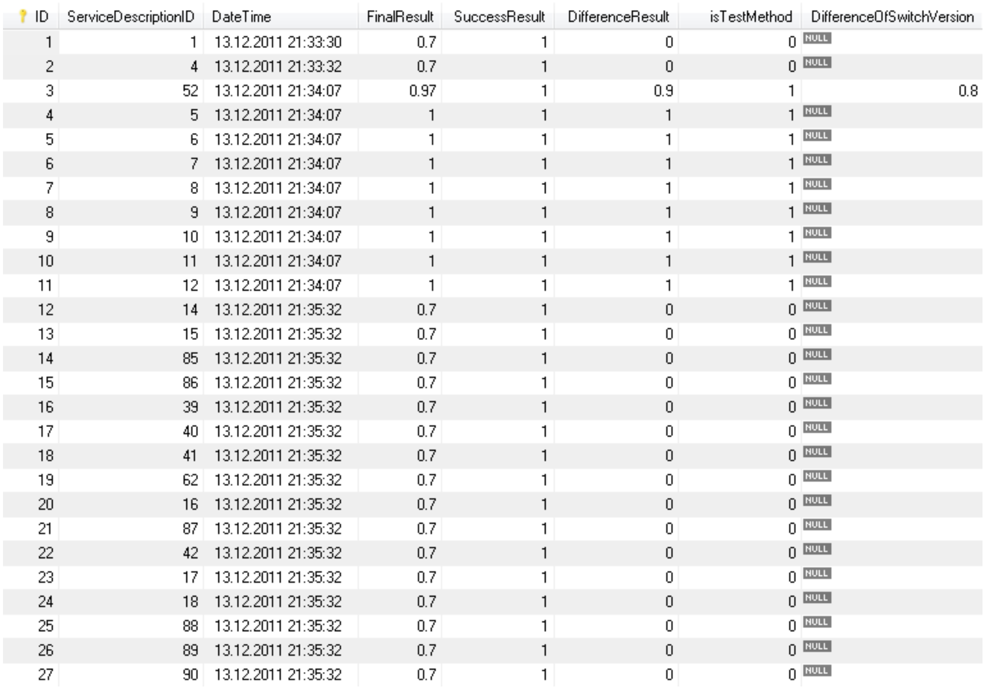
Figure 5.1: The Test Results of Web Services in ”Car” Domain - I



ID	ServiceDescriptionID	DateTime	MutantVersion0	MutantVersion1	MutantVersion2	MutantVersion3	MutantVersion4	MutantVersion5	MutantVersion6	OntologyVersion	WordnetVersion	DependencyVersion	SwitchParamsVersion
18	226	25.10.2011 03:49:05	1	1	1	1	1	1	1	1	1	1	NULL
19	227	25.10.2011 03:49:05	1	1	1	1	1	1	1	1	1	1	NULL
20	237	25.10.2011 03:50:38	0.6	0	0	0	0	0	0	0	0	0	NULL
21	238	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	NULL
22	239	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	1
23	240	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	1
24	241	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	1
25	242	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	1
26	243	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	1
27	244	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	1
28	245	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	1
29	246	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	1
30	247	25.10.2011 03:50:38	1	0	1	1	1	1	1	1	1	0	1
31	248	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	1
32	249	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	NULL
33	250	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	NULL
34	251	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	NULL
35	252	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	NULL
36	253	25.10.2011 03:50:38	1	1	1	1	1	1	1	1	1	1	1
37	254	25.10.2011 03:51:40	0.6	0	0	0	0	0	0	0	0	0	NULL
38	255	25.10.2011 03:51:40	0.6	0	0	0	0	0	0	0	0	0	NULL
39	256	25.10.2011 03:51:40	1	1	1	1	1	1	1	1	1	1	1
40	257	25.10.2011 03:51:40	0.6	0	0	0	0	0	0	0	0	0	NULL
41	258	25.10.2011 03:51:40	1	1	1	1	1	1	1	1	1	1	1
42	259	25.10.2011 03:53:06	0.6	0	0	0	0	0	0	0	0	0	NULL
43	260	25.10.2011 03:53:06	1	1	1	1	1	1	1	1	1	1	NULL
44	261	25.10.2011 03:53:06	1	1	1	1	1	1	1	1	1	1	NULL

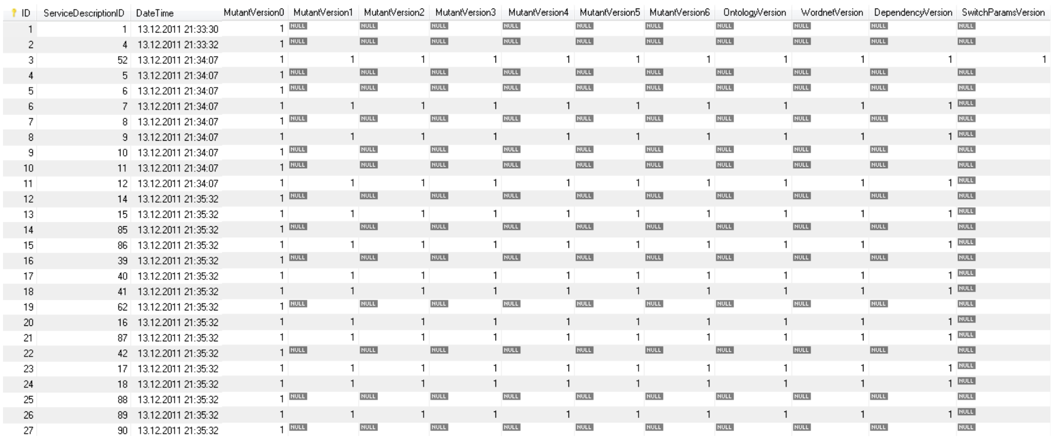
Figure 5.2: The Test Results of Web Services in ”Car” Domain- II

In the test results, *final result* column shows to which degree a web service is tested success-



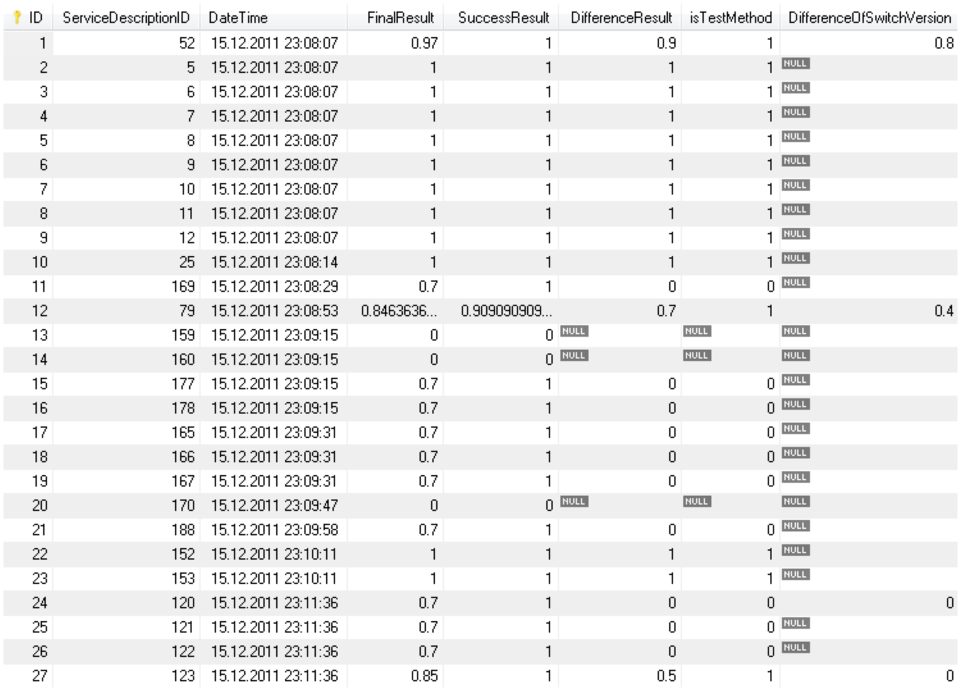
ID	ServiceDescriptionID	DateTime	FinalResult	SuccessResult	DifferenceResult	isTestMethod	DifferenceOfSwitchVersion
1	1	13.12.2011 21:33:30	0.7	1	0	0	NULL
2	4	13.12.2011 21:33:32	0.7	1	0	0	NULL
3	52	13.12.2011 21:34:07	0.97	1	0.9	1	0.8
4	5	13.12.2011 21:34:07	1	1	1	1	NULL
5	6	13.12.2011 21:34:07	1	1	1	1	NULL
6	7	13.12.2011 21:34:07	1	1	1	1	NULL
7	8	13.12.2011 21:34:07	1	1	1	1	NULL
8	9	13.12.2011 21:34:07	1	1	1	1	NULL
9	10	13.12.2011 21:34:07	1	1	1	1	NULL
10	11	13.12.2011 21:34:07	1	1	1	1	NULL
11	12	13.12.2011 21:34:07	1	1	1	1	NULL
12	14	13.12.2011 21:35:32	0.7	1	0	0	NULL
13	15	13.12.2011 21:35:32	0.7	1	0	0	NULL
14	85	13.12.2011 21:35:32	0.7	1	0	0	NULL
15	86	13.12.2011 21:35:32	0.7	1	0	0	NULL
16	39	13.12.2011 21:35:32	0.7	1	0	0	NULL
17	40	13.12.2011 21:35:32	0.7	1	0	0	NULL
18	41	13.12.2011 21:35:32	0.7	1	0	0	NULL
19	62	13.12.2011 21:35:32	0.7	1	0	0	NULL
20	16	13.12.2011 21:35:32	0.7	1	0	0	NULL
21	87	13.12.2011 21:35:32	0.7	1	0	0	NULL
22	42	13.12.2011 21:35:32	0.7	1	0	0	NULL
23	17	13.12.2011 21:35:32	0.7	1	0	0	NULL
24	18	13.12.2011 21:35:32	0.7	1	0	0	NULL
25	88	13.12.2011 21:35:32	0.7	1	0	0	NULL
26	89	13.12.2011 21:35:32	0.7	1	0	0	NULL
27	90	13.12.2011 21:35:32	0.7	1	0	0	NULL

Figure 5.3: The Test Results of Web Services in ”Aviation” Domain - I



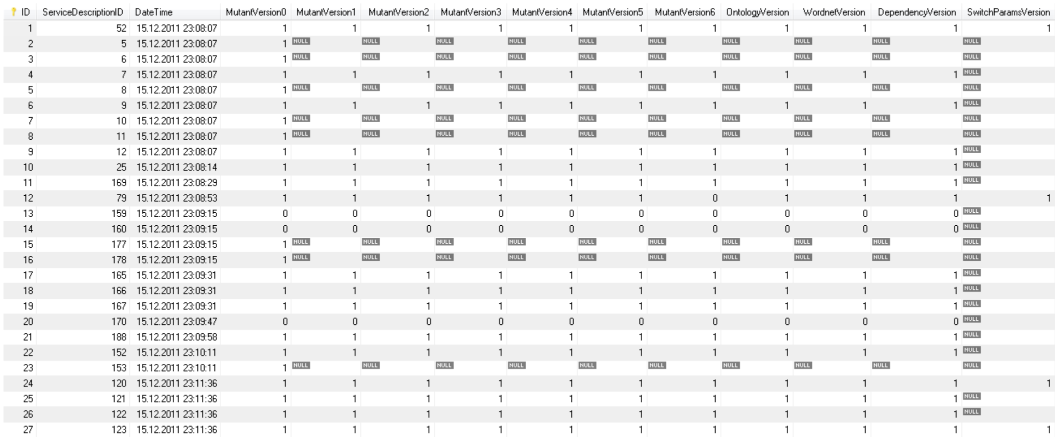
ID	ServiceDescriptionID	DateTime	MutantVersion0	MutantVersion1	MutantVersion2	MutantVersion3	MutantVersion4	MutantVersion5	MutantVersion6	OntologyVersion	WordnetVersion	DependencyVersion	SwitchParamsVersion
1	1	13.12.2011 21:33:30	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2	4	13.12.2011 21:33:32	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	52	13.12.2011 21:34:07	1	1	1	1	1	1	1	1	1	1	1
4	5	13.12.2011 21:34:07	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
5	6	13.12.2011 21:34:07	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
6	7	13.12.2011 21:34:07	1	1	1	1	1	1	1	1	1	1	NULL
7	8	13.12.2011 21:34:07	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
8	9	13.12.2011 21:34:07	1	1	1	1	1	1	1	1	1	1	NULL
9	10	13.12.2011 21:34:07	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
10	11	13.12.2011 21:34:07	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
11	12	13.12.2011 21:34:07	1	1	1	1	1	1	1	1	1	1	NULL
12	14	13.12.2011 21:35:32	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
13	15	13.12.2011 21:35:32	1	1	1	1	1	1	1	1	1	1	NULL
14	85	13.12.2011 21:35:32	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
15	86	13.12.2011 21:35:32	1	1	1	1	1	1	1	1	1	1	NULL
16	39	13.12.2011 21:35:32	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
17	40	13.12.2011 21:35:32	1	1	1	1	1	1	1	1	1	1	NULL
18	41	13.12.2011 21:35:32	1	1	1	1	1	1	1	1	1	1	NULL
19	62	13.12.2011 21:35:32	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
20	16	13.12.2011 21:35:32	1	1	1	1	1	1	1	1	1	1	NULL
21	87	13.12.2011 21:35:32	1	1	1	1	1	1	1	1	1	1	NULL
22	42	13.12.2011 21:35:32	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
23	17	13.12.2011 21:35:32	1	1	1	1	1	1	1	1	1	1	NULL
24	18	13.12.2011 21:35:32	1	1	1	1	1	1	1	1	1	1	NULL
25	88	13.12.2011 21:35:32	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
26	89	13.12.2011 21:35:32	1	1	1	1	1	1	1	1	1	1	NULL
27	90	13.12.2011 21:35:32	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 5.4: The Test Results of Web Services in ”Aviation” Domain - II



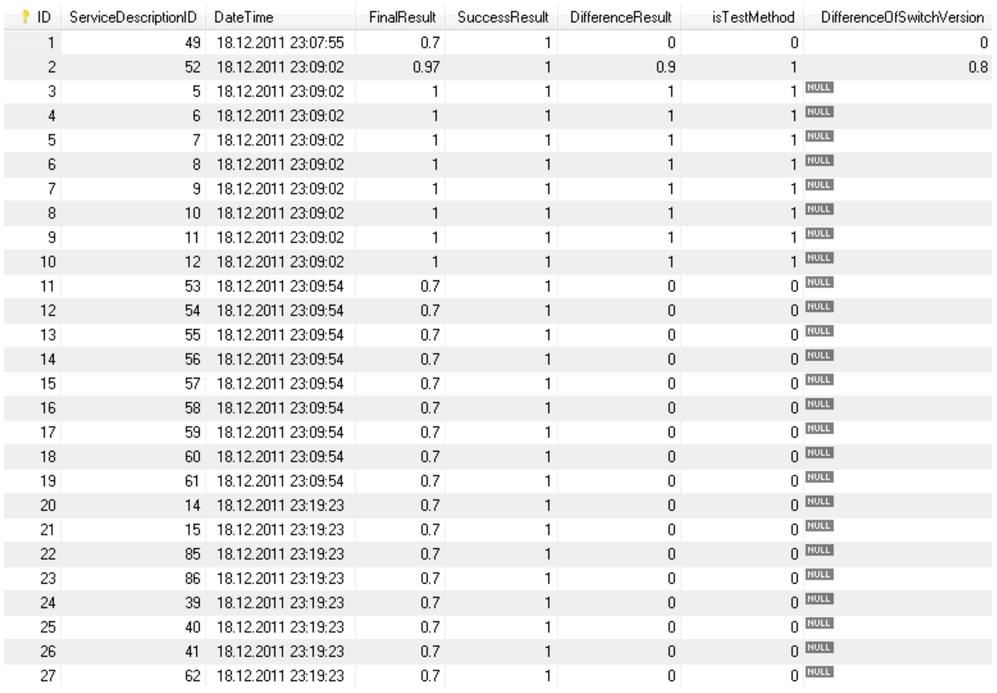
ID	ServiceDescriptionID	DateTime	FinalResult	SuccessResult	DifferenceResult	isTestMethod	DifferenceOfSwitchVersion
1	52	15.12.2011 23:08:07	0.97	1	0.9	1	0.8
2	5	15.12.2011 23:08:07	1	1	1	1	NULL
3	6	15.12.2011 23:08:07	1	1	1	1	NULL
4	7	15.12.2011 23:08:07	1	1	1	1	NULL
5	8	15.12.2011 23:08:07	1	1	1	1	NULL
6	9	15.12.2011 23:08:07	1	1	1	1	NULL
7	10	15.12.2011 23:08:07	1	1	1	1	NULL
8	11	15.12.2011 23:08:07	1	1	1	1	NULL
9	12	15.12.2011 23:08:07	1	1	1	1	NULL
10	25	15.12.2011 23:08:14	1	1	1	1	NULL
11	169	15.12.2011 23:08:29	0.7	1	0	0	NULL
12	79	15.12.2011 23:08:53	0.8463636...	0.909090909...	0.7	1	0.4
13	159	15.12.2011 23:09:15	0	0	NULL	NULL	NULL
14	160	15.12.2011 23:09:15	0	0	NULL	NULL	NULL
15	177	15.12.2011 23:09:15	0.7	1	0	0	NULL
16	178	15.12.2011 23:09:15	0.7	1	0	0	NULL
17	165	15.12.2011 23:09:31	0.7	1	0	0	NULL
18	166	15.12.2011 23:09:31	0.7	1	0	0	NULL
19	167	15.12.2011 23:09:31	0.7	1	0	0	NULL
20	170	15.12.2011 23:09:47	0	0	NULL	NULL	NULL
21	188	15.12.2011 23:09:58	0.7	1	0	0	NULL
22	152	15.12.2011 23:10:11	1	1	1	1	NULL
23	153	15.12.2011 23:10:11	1	1	1	1	NULL
24	120	15.12.2011 23:11:36	0.7	1	0	0	0
25	121	15.12.2011 23:11:36	0.7	1	0	0	NULL
26	122	15.12.2011 23:11:36	0.7	1	0	0	NULL
27	123	15.12.2011 23:11:36	0.85	1	0.5	1	0

Figure 5.5: The Test Results of Web Services in ”Film” Domain - I



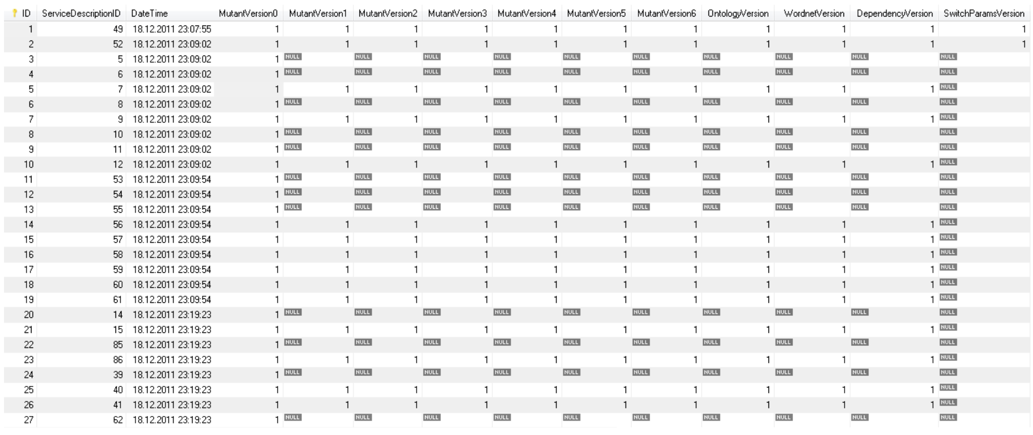
ID	ServiceDescriptionID	DateTime	MutantVersion0	MutantVersion1	MutantVersion2	MutantVersion3	MutantVersion4	MutantVersion5	MutantVersion6	OntologyVersion	WordnetVersion	DependencyVersion	SwitchParamsVersion
1	52	15.12.2011 23:08:07	1	1	1	1	1	1	1	1	1	1	1
2	5	15.12.2011 23:08:07	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	6	15.12.2011 23:08:07	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
4	7	15.12.2011 23:08:07	1	1	1	1	1	1	1	1	1	1	NULL
5	8	15.12.2011 23:08:07	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
6	9	15.12.2011 23:08:07	1	1	1	1	1	1	1	1	1	1	NULL
7	10	15.12.2011 23:08:07	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
8	11	15.12.2011 23:08:07	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
9	12	15.12.2011 23:08:07	1	1	1	1	1	1	1	1	1	1	NULL
10	25	15.12.2011 23:08:14	1	1	1	1	1	1	1	1	1	1	NULL
11	169	15.12.2011 23:08:29	1	1	1	1	1	1	1	1	1	1	NULL
12	79	15.12.2011 23:08:53	1	1	1	1	1	1	0	1	1	1	1
13	159	15.12.2011 23:09:15	0	0	0	0	0	0	0	0	0	0	NULL
14	160	15.12.2011 23:09:15	0	0	0	0	0	0	0	0	0	0	NULL
15	177	15.12.2011 23:09:15	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
16	178	15.12.2011 23:09:15	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
17	165	15.12.2011 23:09:31	1	1	1	1	1	1	1	1	1	1	NULL
18	166	15.12.2011 23:09:31	1	1	1	1	1	1	1	1	1	1	NULL
19	167	15.12.2011 23:09:31	1	1	1	1	1	1	1	1	1	1	NULL
20	170	15.12.2011 23:09:47	0	0	0	0	0	0	0	0	0	0	NULL
21	188	15.12.2011 23:09:58	1	1	1	1	1	1	1	1	1	1	NULL
22	152	15.12.2011 23:10:11	1	1	1	1	1	1	1	1	1	1	NULL
23	153	15.12.2011 23:10:11	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
24	120	15.12.2011 23:11:36	1	1	1	1	1	1	1	1	1	1	1
25	121	15.12.2011 23:11:36	1	1	1	1	1	1	1	1	1	1	NULL
26	122	15.12.2011 23:11:36	1	1	1	1	1	1	1	1	1	1	NULL
27	123	15.12.2011 23:11:36	1	1	1	1	1	1	1	1	1	1	1

Figure 5.6: The Test Results of Web Services in ”Film” Domain - II



ID	ServiceDescriptionID	DateTime	FinalResult	SuccessResult	DifferenceResult	isTestMethod	DifferenceOfSwitchVersion
1	49	18.12.2011 23:07:55	0.7	1	0	0	0
2	52	18.12.2011 23:09:02	0.97	1	0.9	1	0.8
3	5	18.12.2011 23:09:02	1	1	1	1	NULL
4	6	18.12.2011 23:09:02	1	1	1	1	NULL
5	7	18.12.2011 23:09:02	1	1	1	1	NULL
6	8	18.12.2011 23:09:02	1	1	1	1	NULL
7	9	18.12.2011 23:09:02	1	1	1	1	NULL
8	10	18.12.2011 23:09:02	1	1	1	1	NULL
9	11	18.12.2011 23:09:02	1	1	1	1	NULL
10	12	18.12.2011 23:09:02	1	1	1	1	NULL
11	53	18.12.2011 23:09:54	0.7	1	0	0	NULL
12	54	18.12.2011 23:09:54	0.7	1	0	0	NULL
13	55	18.12.2011 23:09:54	0.7	1	0	0	NULL
14	56	18.12.2011 23:09:54	0.7	1	0	0	NULL
15	57	18.12.2011 23:09:54	0.7	1	0	0	NULL
16	58	18.12.2011 23:09:54	0.7	1	0	0	NULL
17	59	18.12.2011 23:09:54	0.7	1	0	0	NULL
18	60	18.12.2011 23:09:54	0.7	1	0	0	NULL
19	61	18.12.2011 23:09:54	0.7	1	0	0	NULL
20	14	18.12.2011 23:19:23	0.7	1	0	0	NULL
21	15	18.12.2011 23:19:23	0.7	1	0	0	NULL
22	85	18.12.2011 23:19:23	0.7	1	0	0	NULL
23	86	18.12.2011 23:19:23	0.7	1	0	0	NULL
24	39	18.12.2011 23:19:23	0.7	1	0	0	NULL
25	40	18.12.2011 23:19:23	0.7	1	0	0	NULL
26	41	18.12.2011 23:19:23	0.7	1	0	0	NULL
27	62	18.12.2011 23:19:23	0.7	1	0	0	NULL

Figure 5.7: The Test Results of Web Services in ”Sports” Domain - I



ID	ServiceDescriptionID	DateTime	MutantVersion0	MutantVersion1	MutantVersion2	MutantVersion3	MutantVersion4	MutantVersion5	MutantVersion6	OntologyVersion	WordnetVersion	DependencyVersion	SwitchParamsVersion
1	49	18.12.2011 23:07:55	1	1	1	1	1	1	1	1	1	1	1
2	52	18.12.2011 23:09:02	1	1	1	1	1	1	1	1	1	1	1
3	5	18.12.2011 23:09:02	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
4	6	18.12.2011 23:09:02	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
5	7	18.12.2011 23:09:02	1	1	1	1	1	1	1	1	1	1	NULL
6	8	18.12.2011 23:09:02	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
7	9	18.12.2011 23:09:02	1	1	1	1	1	1	1	1	1	1	NULL
8	10	18.12.2011 23:09:02	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
9	11	18.12.2011 23:09:02	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
10	12	18.12.2011 23:09:02	1	1	1	1	1	1	1	1	1	1	NULL
11	53	18.12.2011 23:09:54	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
12	54	18.12.2011 23:09:54	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
13	55	18.12.2011 23:09:54	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
14	56	18.12.2011 23:09:54	1	1	1	1	1	1	1	1	1	1	NULL
15	57	18.12.2011 23:09:54	1	1	1	1	1	1	1	1	1	1	NULL
16	58	18.12.2011 23:09:54	1	1	1	1	1	1	1	1	1	1	NULL
17	59	18.12.2011 23:09:54	1	1	1	1	1	1	1	1	1	1	NULL
18	60	18.12.2011 23:09:54	1	1	1	1	1	1	1	1	1	1	NULL
19	61	18.12.2011 23:09:54	1	1	1	1	1	1	1	1	1	1	NULL
20	14	18.12.2011 23:19:23	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
21	15	18.12.2011 23:19:23	1	1	1	1	1	1	1	1	1	1	NULL
22	85	18.12.2011 23:19:23	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
23	86	18.12.2011 23:19:23	1	1	1	1	1	1	1	1	1	1	NULL
24	39	18.12.2011 23:19:23	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
25	40	18.12.2011 23:19:23	1	1	1	1	1	1	1	1	1	1	NULL
26	41	18.12.2011 23:19:23	1	1	1	1	1	1	1	1	1	1	NULL
27	62	18.12.2011 23:19:23	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 5.8: The Test Results of Web Services in ”Sports” Domain - II

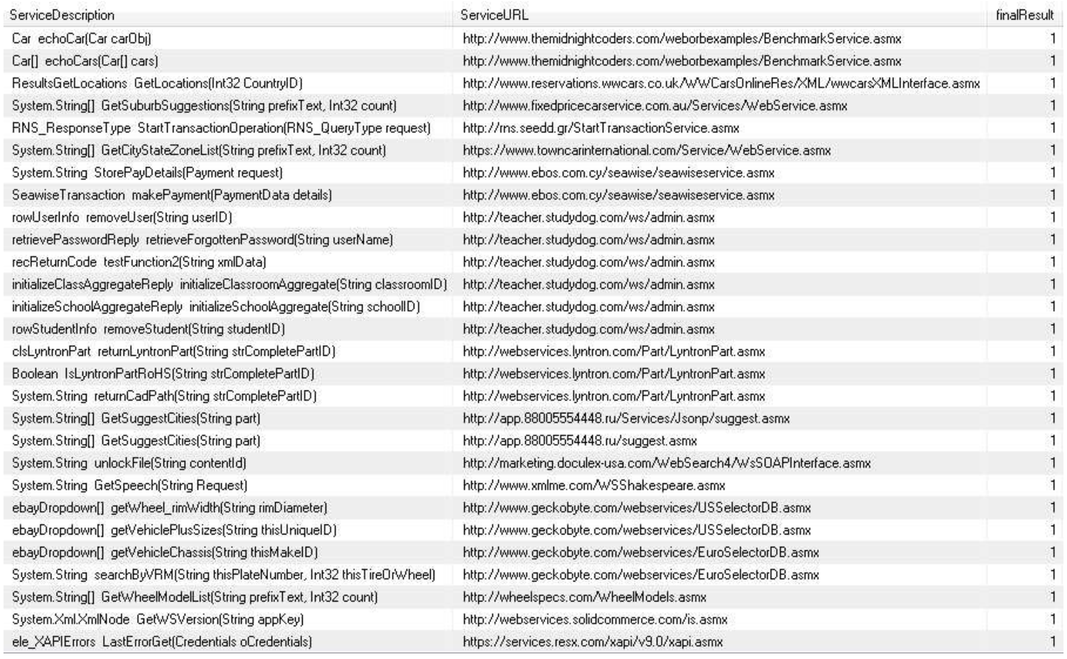
fully.

In ”Car” domain, for 349 web services of all web services, the final test value is 1, it means that they are successfully tested. The final test value of 2545 web services is 0, which means that they are totally failed in testing process. Some of the web services that are totally successfully tested are listed in Figure 5.9. 2501 web services are successfully invoked in all test cases that are generated by using data mutation-based method. 2649 web services are successful in all test cases that are generated by using semantic dependency-based method. 2474 web services are successful in both of two methods. Number of web services that return same values in all successful invocations and are test method is 2453.

In ”Aviation” domain, 28 web services of all web services are successfully tested. Therefore their final test value is 1. These services are listed in Figure 5.10. 10 web services are totally failed. 126 web services are successfully invoked in all test cases that are generated by using data mutation-based method. 80 web services are successful in all test cases that are generated by using semantic dependency-based method. In both of two methods, 125 web services are successful. 48 web services are test method and return same values in all successful invocations.

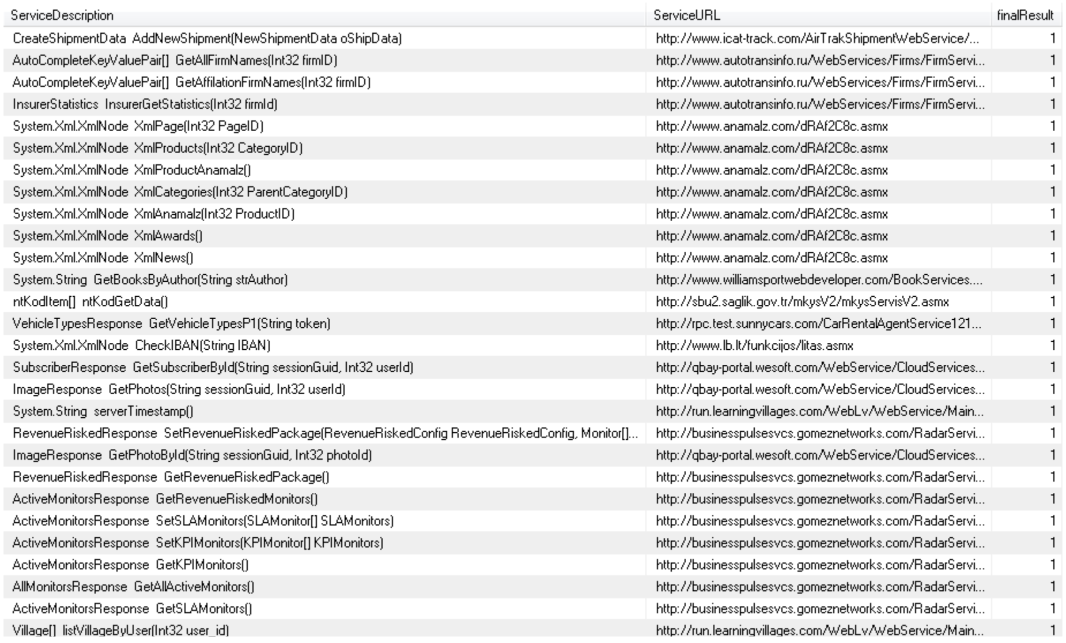
In ”Film” domain, 35 web services of all web services are successfully tested. Therefore their final test value is 1. These services are listed in Figure 5.11. 12 web services are totally failed. 170 web services are successfully invoked in all test cases that are generated by using data mutation-based method. 110 web services are successful in all test cases that are generated by using semantic dependency-based method. In both of two methods, 108 web services are successful. 60 web services are test method and return same values in all successful invocations.

In ”Sports” domain, 36 web services of all web services are successfully tested.Therefore their final test value is 1. These services are listed in Figure 5.12. 12 web services are totally failed. 172 web services are successfully invoked in all test cases that are generated by using data mutation-based method. 108 web services are successful in all test cases that are generated by using semantic dependency-based method. In both of two methods, 108 web services are successful. 58 web services are test method and return same values in all successful invocations.



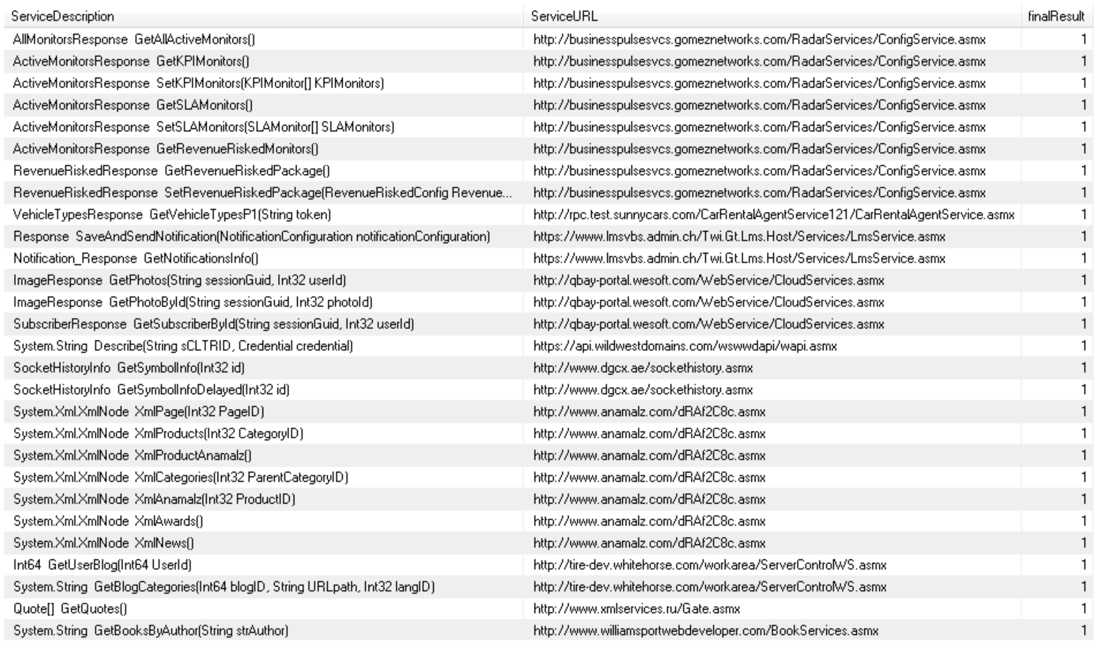
ServiceDescription	ServiceURL	finalResult
Car echoCar(Car carObj)	http://www.themidnightcoders.com/weborbexamples/BenchmarkService.asmx	1
Car[] echoCars(Car[] cars)	http://www.themidnightcoders.com/weborbexamples/BenchmarkService.asmx	1
ResultsGetLocations GetLocations(Int32 CountryID)	http://www.reservations.wwcars.co.uk/WWCarsOnlineRes/XML/wwcarsXMLInterface.asmx	1
System.String[] GetSuburbSuggestions(String prefixText, Int32 count)	http://www.fixedpricecarservice.com.au/Services/WebService.asmx	1
RNS_ResponseType StartTransactionOperation(RNS_QueryType request)	http://rns.seedd.gr/StartTransactionService.asmx	1
System.String[] GetCityStateZoneList(String prefixText, Int32 count)	https://www.towncarinternational.com/Service/WebService.asmx	1
System.String StorePayDetails(Payment request)	http://www.ebos.com.cy/seawise/seawiseservice.asmx	1
SeawiseTransaction makePayment(PaymentData details)	http://www.ebos.com.cy/seawise/seawiseservice.asmx	1
rowUserInfo removeUser(String userID)	http://teacher.studydog.com/ws/admin.asmx	1
retrievePasswordReply retrieveForgottenPassword(String userName)	http://teacher.studydog.com/ws/admin.asmx	1
recReturnCode testFunction2(String xmlData)	http://teacher.studydog.com/ws/admin.asmx	1
initializeClassAggregateReply initializeClassroomAggregate(String classroomID)	http://teacher.studydog.com/ws/admin.asmx	1
initializeSchoolAggregateReply initializeSchoolAggregate(String schoolID)	http://teacher.studydog.com/ws/admin.asmx	1
rowStudentInfo removeStudent(String studentID)	http://teacher.studydog.com/ws/admin.asmx	1
clsLyntronPart returnLyntronPart(String strCompletePartID)	http://webservices.lyntron.com/Part/LyntronPart.asmx	1
Boolean IsLyntronPartRoHS(String strCompletePartID)	http://webservices.lyntron.com/Part/LyntronPart.asmx	1
System.String returnCadPath(String strCompletePartID)	http://webservices.lyntron.com/Part/LyntronPart.asmx	1
System.String[] GetSuggestCities(String part)	http://app.88005554448.ru/Services/Jsonp/suggest.asmx	1
System.String[] GetSuggestCities(String part)	http://app.88005554448.ru/suggest.asmx	1
System.String unlockFile(String contentId)	http://marketing.doculex-usa.com/WebSearch4/WsSOAPInterface.asmx	1
System.String GetSpeech(String Request)	http://www.xmlme.com/WSShakespeare.asmx	1
ebayDropdown[] getWheel_rimWidth(String rimDiameter)	http://www.geckobyte.com/webservices/USSelectorDB.asmx	1
ebayDropdown[] getVehiclePlusSizes(String thisUniqueID)	http://www.geckobyte.com/webservices/USSelectorDB.asmx	1
ebayDropdown[] getVehicleChassis(String thisMakeID)	http://www.geckobyte.com/webservices/EuroSelectorDB.asmx	1
System.String searchByVRM(String thisPlateNumber, Int32 thisTireOrWheel)	http://www.geckobyte.com/webservices/EuroSelectorDB.asmx	1
System.String[] GetWheelModelList(String prefixText, Int32 count)	http://wheelspecs.com/WheelModels.asmx	1
System.Xml.XmlNode GetWSVersion(String appKey)	http://webservices.solidcommerce.com/is.asmx	1
ele_XAPIErrors LastErrorGet(Credentials oCredentials)	https://services.resx.com/xapi/v9.0/xapi.asmx	1

Figure 5.9: Successfully Tested Web Services in "Car" Domain



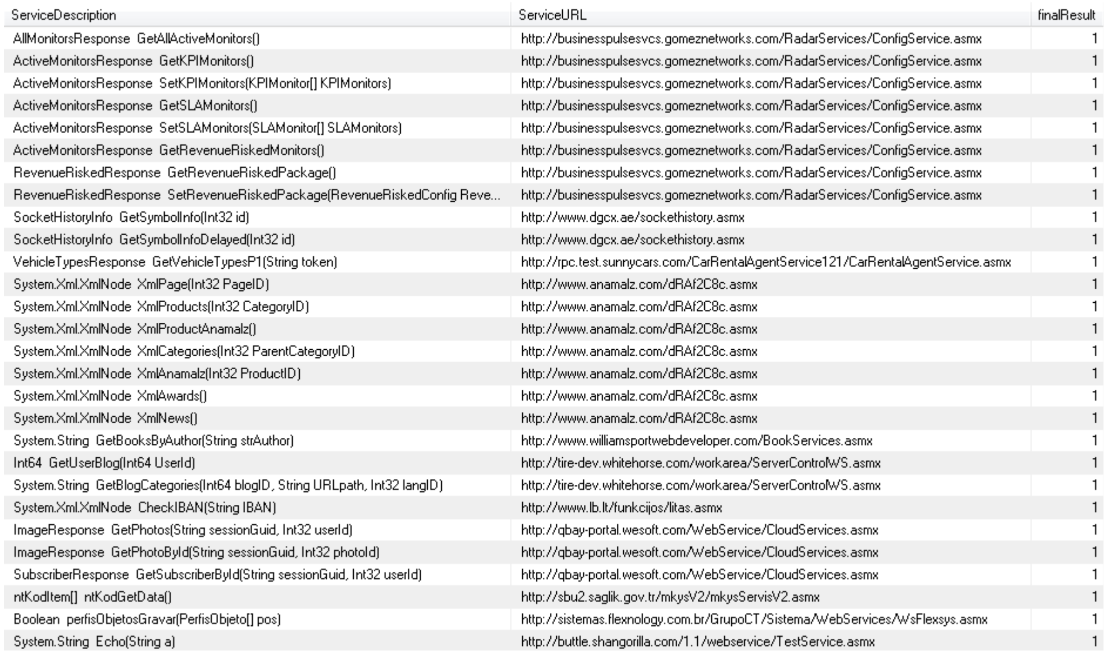
ServiceDescription	ServiceURL	finalResult
CreateShipmentData AddNewShipment(NewShipmentData oShipData)	http://www.icat-track.com/AirTrakShipmentWebService/...	1
AutoCompleteKeyValuePair[] GetAllFirmNames(Int32 firmID)	http://www.autotransinfo.ru/WebServices/Firms/FirmServi...	1
AutoCompleteKeyValuePair[] GetAffilationFirmNames(Int32 firmID)	http://www.autotransinfo.ru/WebServices/Firms/FirmServi...	1
InsurerStatistics InsurerGetStatistics(Int32 firmId)	http://www.autotransinfo.ru/WebServices/Firms/FirmServi...	1
System.Xml.XmlNode XmlPage(Int32 PageID)	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlProducts(Int32 CategoryID)	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlProductAnamalz()	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlCategories(Int32 ParentCategoryID)	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlAnamalz(Int32 ProductID)	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlAwards()	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlNews()	http://www.anamalz.com/dRAf2C8c.asmx	1
System.String GetBooksByAuthor(String strAuthor)	http://www.williamsportwebdeveloper.com/BookServices....	1
ntKodItem[] ntKodGetData()	http://sbu2.saglik.gov.tr/mkysV2/mkysServisV2.asmx	1
VehicleTypesResponse GetVehicleTypesP1(String token)	http://rpc.test.sunnycars.com/CarRentalAgentService121...	1
System.Xml.XmlNode CheckIBAN(String IBAN)	http://www.lb.lt/funkcijos/litas.asmx	1
SubscriberResponse GetSubscriberById(String sessionGuid, Int32 userId)	http://qbay-portal.wesoft.com/WebService/CloudServices...	1
ImageResponse GetPhotos(String sessionGuid, Int32 userId)	http://qbay-portal.wesoft.com/WebService/CloudServices...	1
System.String serverTimestamp()	http://run.learningvillages.com/WebLv/WebService/Main...	1
RevenueRiskedResponse SetRevenueRiskedPackage(RevenueRiskedConfig RevenueRiskedConfig, Monitor[]...	http://businesspulsesvcs.gomeznetworks.com/RadarServi...	1
ImageResponse GetPhotoById(String sessionGuid, Int32 photoId)	http://qbay-portal.wesoft.com/WebService/CloudServices...	1
RevenueRiskedResponse GetRevenueRiskedPackage()	http://businesspulsesvcs.gomeznetworks.com/RadarServi...	1
ActiveMonitorsResponse GetRevenueRiskedMonitors()	http://businesspulsesvcs.gomeznetworks.com/RadarServi...	1
ActiveMonitorsResponse SetSLAMonitors(SLAMonitor[] SLAMonitors)	http://businesspulsesvcs.gomeznetworks.com/RadarServi...	1
ActiveMonitorsResponse SetKPIMonitors(KPIMonitor[] KPIMonitors)	http://businesspulsesvcs.gomeznetworks.com/RadarServi...	1
ActiveMonitorsResponse GetKPIMonitors()	http://businesspulsesvcs.gomeznetworks.com/RadarServi...	1
AllMonitorsResponse GetAllActiveMonitors()	http://businesspulsesvcs.gomeznetworks.com/RadarServi...	1
ActiveMonitorsResponse GetSLAMonitors()	http://businesspulsesvcs.gomeznetworks.com/RadarServi...	1
Village[] listVillageByUser(Int32 user_id)	http://run.learningvillages.com/WebLv/WebService/Main...	1

Figure 5.10: Successfully Tested Web Services in "Aviation" Domain



ServiceDescription	ServiceURL	finalResult
AllMonitorsResponse GetAllActiveMonitors()	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
ActiveMonitorsResponse GetKPIMonitors()	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
ActiveMonitorsResponse SetKPIMonitors(KPIMonitor[] KPIMonitors)	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
ActiveMonitorsResponse GetSLAMonitors()	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
ActiveMonitorsResponse SetSLAMonitors(SLAMonitor[] SLAMonitors)	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
ActiveMonitorsResponse GetRevenueRiskedMonitors()	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
RevenueRiskedResponse GetRevenueRiskedPackage()	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
RevenueRiskedResponse SetRevenueRiskedPackage(RevenueRiskedConfig Revenue...	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
VehicleTypesResponse GetVehicleTypesP1(String token)	http://rpc.test.sunnycars.com/CarRentalAgentService121/CarRentalAgentService.asmx	1
Response SaveAndSendNotification(NotificationConfiguration notificationConfiguration)	https://www.lmsvbs.admin.ch/Twi.Gt.Lms.Host/Services/LmsService.asmx	1
Notification_Response GetNotificationsInfo()	https://www.lmsvbs.admin.ch/Twi.Gt.Lms.Host/Services/LmsService.asmx	1
ImageResponse GetPhotos(String sessionGuid, Int32 userId)	http://qbay-portal.wesoft.com/WebService/CloudServices.asmx	1
ImageResponse GetPhotoById(String sessionGuid, Int32 photoId)	http://qbay-portal.wesoft.com/WebService/CloudServices.asmx	1
SubscriberResponse GetSubscriberById(String sessionGuid, Int32 userId)	http://qbay-portal.wesoft.com/WebService/CloudServices.asmx	1
System.String Describe(String sCLTRID, Credential credential)	https://api.wildwestdomains.com/wswwdapi/wapi.asmx	1
SocketHistoryInfo GetSymbolInfo(Int32 id)	http://www.dgcx.ae/sockethistory.asmx	1
SocketHistoryInfo GetSymbolInfoDelayed(Int32 id)	http://www.dgcx.ae/sockethistory.asmx	1
System.Xml.XmlNode XmlPage(Int32 PageID)	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlProducts(Int32 CategoryID)	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlProductAnamalz()	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlCategories(Int32 ParentCategoryID)	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlAnamalz(Int32 ProductID)	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlAwards()	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlNews()	http://www.anamalz.com/dRAf2C8c.asmx	1
Int64 GetUserBlog(Int64 UserId)	http://tire-dev.whitehorse.com/workarea/ServerControlWS.asmx	1
System.String GetBlogCategories(Int64 blogID, String URLpath, Int32 langID)	http://tire-dev.whitehorse.com/workarea/ServerControlWS.asmx	1
Quote[] GetQuotes()	http://www.xmlservices.ru/Gate.asmx	1
System.String GetBooksByAuthor(String strAuthor)	http://www.williamsportwebdeveloper.com/BookServices.asmx	1

Figure 5.11: Successfully Tested Web Services in ”Film” Domain



ServiceDescription	ServiceURL	finalResult
AllMonitorsResponse GetAllActiveMonitors()	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
ActiveMonitorsResponse GetKPIMonitors()	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
ActiveMonitorsResponse SetKPIMonitors(KPIMonitor[] KPIMonitors)	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
ActiveMonitorsResponse GetSLAMonitors()	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
ActiveMonitorsResponse SetSLAMonitors(SLAMonitor[] SLAMonitors)	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
ActiveMonitorsResponse GetRevenueRiskedMonitors()	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
RevenueRiskedResponse GetRevenueRiskedPackage()	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
RevenueRiskedResponse SetRevenueRiskedPackage(RevenueRiskedConfig Reve...	http://businesspulsesvcs.gomeznetworks.com/RadarServices/ConfigService.asmx	1
SocketHistoryInfo GetSymbolInfo(Int32 id)	http://www.dgcx.ae/sockethistory.asmx	1
SocketHistoryInfo GetSymbolInfoDelayed(Int32 id)	http://www.dgcx.ae/sockethistory.asmx	1
VehicleTypesResponse GetVehicleTypesP1(String token)	http://rpc.test.sunnycars.com/CarRentalAgentService121/CarRentalAgentService.asmx	1
System.Xml.XmlNode XmlPage(Int32 PageID)	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlProducts(Int32 CategoryID)	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlProductAnamalz()	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlCategories(Int32 ParentCategoryID)	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlAnamalz(Int32 ProductID)	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlAwards()	http://www.anamalz.com/dRAf2C8c.asmx	1
System.Xml.XmlNode XmlNews()	http://www.anamalz.com/dRAf2C8c.asmx	1
System.String GetBooksByAuthor(String strAuthor)	http://www.williamsportwebdeveloper.com/BookServices.asmx	1
Int64 GetUserBlog(Int64 UserId)	http://tire-dev.whitehorse.com/workarea/ServerControlWS.asmx	1
System.String GetBlogCategories(Int64 blogID, String URLpath, Int32 langID)	http://tire-dev.whitehorse.com/workarea/ServerControlWS.asmx	1
System.Xml.XmlNode CheckIBAN(String IBAN)	http://www.lb.lt/funkcijos/litas.asmx	1
ImageResponse GetPhotos(String sessionGuid, Int32 userId)	http://qbay-portal.wesoft.com/WebService/CloudServices.asmx	1
ImageResponse GetPhotoById(String sessionGuid, Int32 photoId)	http://qbay-portal.wesoft.com/WebService/CloudServices.asmx	1
SubscriberResponse GetSubscriberById(String sessionGuid, Int32 userId)	http://qbay-portal.wesoft.com/WebService/CloudServices.asmx	1
ntKodItem[] ntKodGetData()	http://sbu2.saglik.gov.tr/mkysV2/mkysServisV2.asmx	1
Boolean perfisObjetosGravar(PerfisObjeto[] pos)	http://sistemas.flexnology.com.br/GrupoCT/Sistema/WebServices/WsFlexsys.asmx	1
System.String Echo(String a)	http://buttle.shangorilla.com/1.1/webservice/TestService.asmx	1

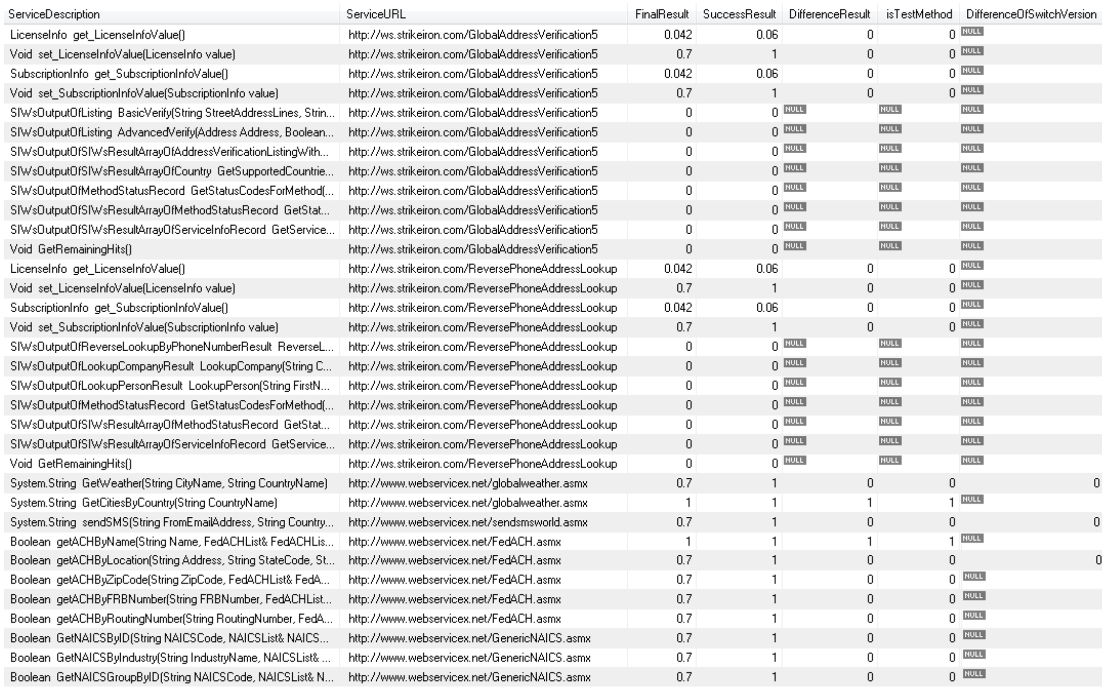
Figure 5.12: Successfully Tested Web Services in ”Sports” Domain

5.3 Experiments with External Web Services

In addition to the collected web services in DSWSD nodes, the testing process is performed for a set of web services obtained from several external services resources. These service resources are strikeiron.com [17] and webserviceX.net [18].

StrikeIron offers the commercial data services such as Email Verification, Global Address Verification, SMS Alerts and Notifications, Online Sales Tax etc. and provides an engine to reduce the complexity for developers and business users who want to integrate live data from any number of sources [17]. WebserviceX.net provides programmable business logic components and standing data that serve as ”black boxes ” to provide access to functionality and data via web services. It offers web services such as Stock Quote, Currency Convertor, Global Weather, London Gold and Silver Fixing, SendSMSWorld [18].

There are many studies that use these rich service providers for different purposes. One of them is the study that is presented by AbuJarour et al. [4] to generate annotations for web services. In our study, a total of 34 web services provided by strikeiron.com and webserviceX.net are tested and the obtained test results are listed in Figure 5.13.



ServiceDescription	ServiceURL	FinalResult	SuccessResult	DifferenceResult	isTestMethod	DifferenceOfSwitchVersion
LicenseInfo get_LicenseInfoValue()	http://ws.strikeiron.com/GlobalAddressVerification5	0.042	0.06	0	0	NULL
Void set_LicenseInfoValue(LicenseInfo value)	http://ws.strikeiron.com/GlobalAddressVerification5	0.7	1	0	0	NULL
SubscriptionInfo get_SubscriptionInfoValue()	http://ws.strikeiron.com/GlobalAddressVerification5	0.042	0.06	0	0	NULL
Void set_SubscriptionInfoValue(SubscriptionInfo value)	http://ws.strikeiron.com/GlobalAddressVerification5	0.7	1	0	0	NULL
SIWsOutputOfListing BasicVerify(String StreetAddressLines, Strin...	http://ws.strikeiron.com/GlobalAddressVerification5	0	0	NULL	NULL	NULL
SIWsOutputOfListing AdvancedVerify(Address Address, Boolean...	http://ws.strikeiron.com/GlobalAddressVerification5	0	0	NULL	NULL	NULL
SIWsOutputOfSIWsResultArrayOfAddressVerificationListingWith...	http://ws.strikeiron.com/GlobalAddressVerification5	0	0	NULL	NULL	NULL
SIWsOutputOfSIWsResultArrayOfCountry GetSupportedCountrie...	http://ws.strikeiron.com/GlobalAddressVerification5	0	0	NULL	NULL	NULL
SIWsOutputOfMethodStatusRecord GetStatusCodesForMethod(...	http://ws.strikeiron.com/GlobalAddressVerification5	0	0	NULL	NULL	NULL
SIWsOutputOfSIWsResultArrayOfMethodStatusRecord GetStat...	http://ws.strikeiron.com/GlobalAddressVerification5	0	0	NULL	NULL	NULL
SIWsOutputOfSIWsResultArrayOfServiceInfoRecord GetService...	http://ws.strikeiron.com/GlobalAddressVerification5	0	0	NULL	NULL	NULL
Void GetRemainingHits()	http://ws.strikeiron.com/GlobalAddressVerification5	0	0	NULL	NULL	NULL
LicenseInfo get_LicenseInfoValue()	http://ws.strikeiron.com/ReversePhoneAddressLookup	0.042	0.06	0	0	NULL
Void set_LicenseInfoValue(LicenseInfo value)	http://ws.strikeiron.com/ReversePhoneAddressLookup	0.7	1	0	0	NULL
SubscriptionInfo get_SubscriptionInfoValue()	http://ws.strikeiron.com/ReversePhoneAddressLookup	0.042	0.06	0	0	NULL
Void set_SubscriptionInfoValue(SubscriptionInfo value)	http://ws.strikeiron.com/ReversePhoneAddressLookup	0.7	1	0	0	NULL
SIWsOutputOfReverseLookupByPhoneNumberResult ReverseL...	http://ws.strikeiron.com/ReversePhoneAddressLookup	0	0	NULL	NULL	NULL
SIWsOutputOfLookupCompanyResult LookupCompany(String C...	http://ws.strikeiron.com/ReversePhoneAddressLookup	0	0	NULL	NULL	NULL
SIWsOutputOfLookupPersonResult LookupPerson(String FirstN...	http://ws.strikeiron.com/ReversePhoneAddressLookup	0	0	NULL	NULL	NULL
SIWsOutputOfMethodStatusRecord GetStatusCodesForMethod(...	http://ws.strikeiron.com/ReversePhoneAddressLookup	0	0	NULL	NULL	NULL
SIWsOutputOfSIWsResultArrayOfMethodStatusRecord GetStat...	http://ws.strikeiron.com/ReversePhoneAddressLookup	0	0	NULL	NULL	NULL
SIWsOutputOfSIWsResultArrayOfServiceInfoRecord GetService...	http://ws.strikeiron.com/ReversePhoneAddressLookup	0	0	NULL	NULL	NULL
Void GetRemainingHits()	http://ws.strikeiron.com/ReversePhoneAddressLookup	0	0	NULL	NULL	NULL
System.String GetWeather(String CityName, String CountryName)	http://www.webservicex.net/globalweather.asmx	0.7	1	0	0	0
System.String GetCitiesByCountry(String CountryName)	http://www.webservicex.net/globalweather.asmx	1	1	1	1	NULL
System.String sendSMS(String FromEmailAddress, String Country...	http://www.webservicex.net/sendsmsworld.asmx	0.7	1	0	0	0
Boolean getACHByName(String Name, FedACHList& FedACHLis...	http://www.webservicex.net/FedACH.asmx	1	1	1	1	NULL
Boolean getACHByLocation(String Address, String StateCode, St...	http://www.webservicex.net/FedACH.asmx	0.7	1	0	0	0
Boolean getACHByZipCode(String ZipCode, FedACHList& FedA...	http://www.webservicex.net/FedACH.asmx	0.7	1	0	0	NULL
Boolean getACHByFRBNumber(String FRBNumber, FedACHList...	http://www.webservicex.net/FedACH.asmx	0.7	1	0	0	NULL
Boolean getACHByRoutingNumber(String RoutingNumber, FedA...	http://www.webservicex.net/FedACH.asmx	0.7	1	0	0	NULL
Boolean GetNAICSByID(String NAICSCode, NAICSList& NAICS...	http://www.webservicex.net/GenericNAICS.asmx	0.7	1	0	0	NULL
Boolean GetNAICSByIndustry(String IndustryName, NAICSList& ...	http://www.webservicex.net/GenericNAICS.asmx	0.7	1	0	0	NULL
Boolean GetNAICSGroupByID(String NAICSCode, NAICSList& N...	http://www.webservicex.net/GenericNAICS.asmx	0.7	1	0	0	NULL

Figure 5.13: The Test Results of External Web Services

5.4 Experiments with Syntactically Generated Web Services

In order to evaluate the accuracy of the proposed method, we created web service providers that include various services, whose behaviors are already known. These web services are pertained to ”Aviation” and ”Car ”domains.

For ”Aviation” domain, the provided web services are used to get the information about flights in Turkey and the definitions of these web services are given in Table 5.7.

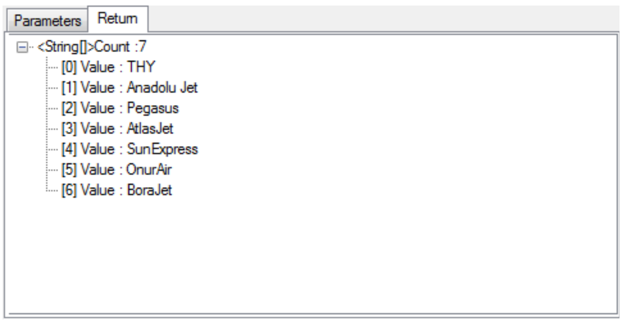
Table 5.7: The Definitions of Web Services of ”Flight Management System”

Web Service	**Definitions**
System.String sayHello()	It always return the same value, ”helloWorld” string.
System.String[] getAirways()	It returns the list of the airline in system
AirportClass[] getAirports()	It returns the list of the airports in system
FlightLineClass[] getFlightLines()	It returns the list of the flightlines in system
System.String[] getAirportsInCity (String city)	It returns the available airports in city given as input parameter.
FlightLineClass[] getFlightLinesTo (String coming)	It returns the flightlines whose arrival airport is given as input parameter.
FlightLineClass[] getFlightLines-From (String takeoff)	It returns the flightlines whose departure airport is given as input parameter.
System.String[] getAirwaysIn-FlightLine (String takeoff, String arrival)	It returns the list of airways, which have flightline, names of the arrival and departure airports are given as parameter
System.String[] getAirwaysIn-FlightLineByID (Int32 takeof-fAerodromeID, Int32 arrivalAero-dromeID)	It returns the list of airways, which have flightline, IDs of the arrival and departure airports are given as parameter.
FlightLineClass[] getFlightLineO-fAirline (String airline)	It returns the list of flightlines of the airline, which is given as parameter.
Boolean isRouteBidirectional (String cityName1, String city-Name2, String airway)	If the flights of the between the cities given as parameter are bi-directional, it returns true. Otherwise, it returns false.

For each web service, the proposed approach is applied and the test results are obtained after the testing process. As we mentioned in previous sections, different input values are generated for each parameter and with these generated input parameters, web services are invoked. After the invocation, the return values of web services are analyzed. Some examples of the generated values of input parameters and the obtained return values are given bellow.

System.String sayHello(): This web service is a test web service and always returns the same value, ”helloWorld” string. Although it is invoked successfully and returns a value, the test result is low since it is a test web service.

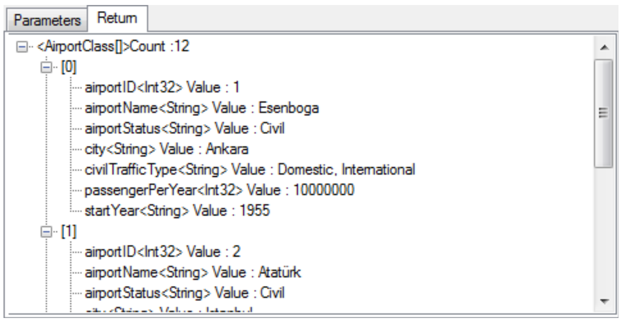
System.String[] getAirways(): When this service is invoked, the obtained list is shown in Figure 5.14.



```
<String[]>Count :7
    [0] Value : THY
    [1] Value : Anadolu Jet
    [2] Value : Pegasus
    [3] Value : AtlasJet
    [4] Value : SunExpress
    [5] Value : OnurAir
    [6] Value : BoraJet
```

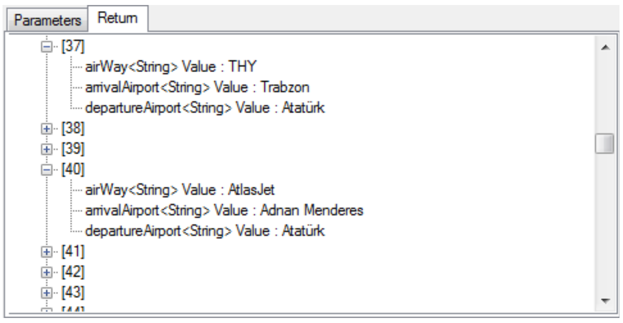
Figure 5.14: The Return Value of Web Service, ”getAirways”

AirportClass[] getAirports(): When this service is invoked, the obtained list is shown in Figure 5.15.



```
<AirportClass[]>Count :12
    [0]
        airportID<Int32> Value : 1
        airportName<String> Value : Esenboga
        airportStatus<String> Value : Civil
        city<String> Value : Ankara
        civilTrafficType<String> Value : Domestic, International
        passengerPerYear<Int32> Value : 10000000
        startYear<String> Value : 1955
    [1]
        airportID<Int32> Value : 2
        airportName<String> Value : Atatürk
        airportStatus<String> Value : Civil
```

Figure 5.15: The Return Value of Web Service, ”getAirports”



```
    [37]
        airWay<String> Value : THY
        arrivalAirport<String> Value : Trabzon
        departureAirport<String> Value : Atatürk
    [38]
    [39]
    [40]
        airWay<String> Value : AtlasJet
        arrivalAirport<String> Value : Adnan Menderes
        departureAirport<String> Value : Atatürk
    [41]
    [42]
    [43]
```

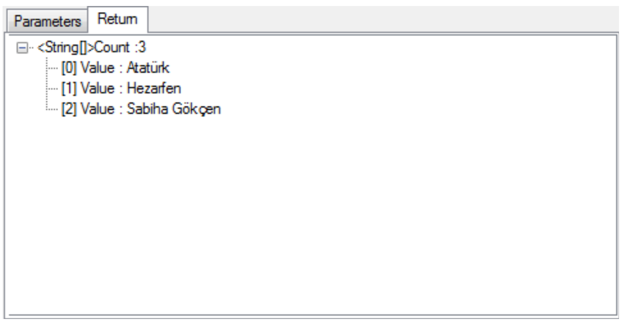
Figure 5.16: The Return Value of Web Service, ”getFlightLines”

FlightLineClass[] getFlightLines(): When this service is invoked, the obtained list is shown in Figure 5.16.

System.String[] getAirportsInCity(String city): This service has one input parameter whose type is *String* and the name is *city*. With semantic dependency analysis-based method, the return types of other web services are retrieved in order to find a dependent output field. After analysis, it is found out that *getAirports* web service whose output parameter type has a field with type *String* and name *city* provides the most similar input/output dependency. A randomly selected value from the *AirportClass* list items is used as the input parameter of *getAirportsInCity*.

When the following value is generated for input parameter, namely city, Figure 5.17 shows the obtained return value.

- city< $String$ > value: ”Istanbul”



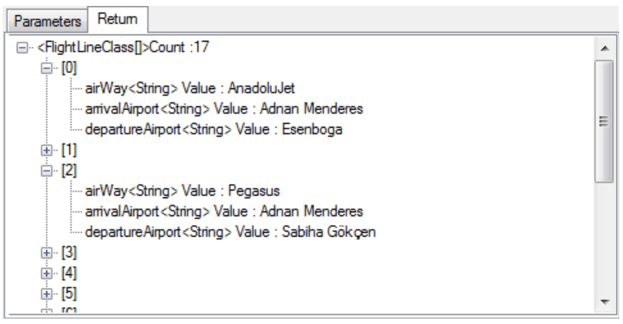
```
⊟- <String[]>Count :3
    [0] Value : Atatürk
    [1] Value : Hezarfen
    [2] Value : Sabiha Gökçen
```

Figure 5.17: The Return Value of Web Service, ”getAirportsInCity”

FlightLineClass[] getFlightLinesTo(String coming): This service has one input parameter whose type of is *String* and the name is *coming*. When all fields of output parameters having the same type are searched, it is found that *arrivalAirport* field of output parameter of *getFlightLines* web service is most semantically similar to the input parameter *coming*. The similarity degree is calculated as 0.967. By using the return value of *getFlightLines*, different values for input parameter *coming* are generated.

When the following value is generated for input parameter *coming*, the output value is obtained as shown in Figure 5.18.

- coming< $String$ > value: ”Adnan Menderes”



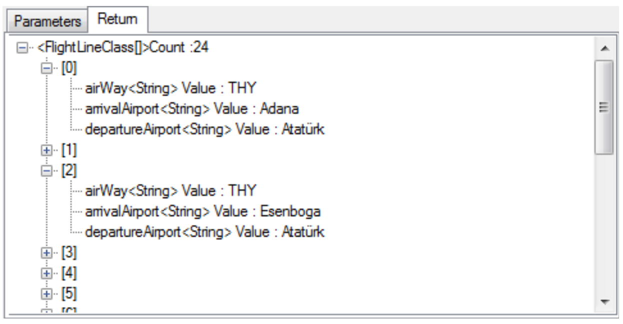
```
Parameters | Return
⊟ <FlightLineClass[]>Count :17
  ⊟ [0]
      airWay<String> Value : AnadoluJet
      arrivalAirport<String> Value : Adnan Menderes
      departureAirport<String> Value : Esenboga
  ⊞ [1]
  ⊟ [2]
      airWay<String> Value : Pegasus
      arrivalAirport<String> Value : Adnan Menderes
      departureAirport<String> Value : Sabiha Gökçen
  ⊞ [3]
  ⊞ [4]
  ⊞ [5]
```

Figure 5.18: The Return Value of Web Service, ”getFlightLinesTo”

FlightLineClass[] getFlightLinesFrom(String takeoff): This service has one input parameter whose type is *String* and the name is *takeoff*. When all fields of output parameters having the same type are searched, it is found that *departureAirport* field of output parameter of *getFlightLines* web service is semantically most similar to the input parameter *takeoff*. The similarity degree is calculated as 0.973. By using the return value of *getFlightLines*, different values for input parameter *takeoff* are generated.

When the following value is generated for departure airport, Figure 5.19 shows the obtained return value.

- takeoff$< String >$ value: ”Atatürk”



```
Parameters | Return
⊟ <FlightLineClass[]>Count :24
  ⊟ [0]
      airWay<String> Value : THY
      arrivalAirport<String> Value : Adana
      departureAirport<String> Value : Atatürk
  ⊞ [1]
  ⊟ [2]
      airWay<String> Value : THY
      arrivalAirport<String> Value : Esenboga
      departureAirport<String> Value : Atatürk
  ⊞ [3]
  ⊞ [4]
  ⊞ [5]
```

Figure 5.19: The Return Value of Web Service, ”getFlightLinesFrom”

System.String[] getAirwaysInFlightLine(String takeoff, String arrival): For two input parameters of *getAirwaysInFlightLine*, all fields of output parameters having the same type are searched. It is found that *departureAirport* field of output parameter of *getFlightLines* web service is most semantically similar to the input parameter *takeoff* and *arrivalAirport* field of output parameter of *getFlightLines* web service is semantically most similar to the input parameter *arrival*. The similarity degree between *takeoff* and *departureAirport* is 0.973 and the similarity degree between *arrival* and *arrivalAirport* is 1. By using the return values of *getFlightLines*, different values for input parameters *takeoff* and *arrival* are generated.

When the following values are generated for input parameters, the output value is obtained as shown in Figure 5.20.

- takeoff$< String >$ value: "Atatürk"
- arrival$< String >$ value: "Adnan Menderes"

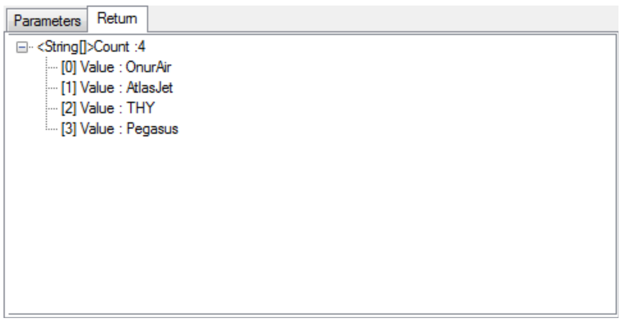
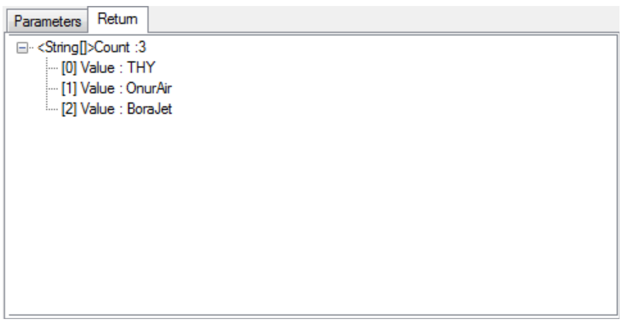

```
Parameters  Return
<String[]>Count :4
    [0] Value : OnurAir
    [1] Value : AtlasJet
    [2] Value : THY
    [3] Value : Pegasus
```

Figure 5.20: The Return Value of Web Service, "getAirwaysInFlightLine"

System.String[] getAirwaysInFlightLineByID(Int32 takeoffAerodromeID, Int32 arrivalAerodromeID): For two input parameters of *getAirwaysInFlightLineByID*, all fields of output parameters having the same type are searched. It is found that *airportID* field of output parameter of *getAirports* web service is most semantically similar to the input parameter *takeoffAerodromeID* and *airportID* field of output parameter of *getAirports* web service is semantically most similar to the input parameter *arrivalAerodromeID*. The similarity degree between *takeoffAerodromeID* and *airportID* is 0.696 and the similarity degree between *arrivalAerodromeID* and *airportID* is 0.712. By using the return values of *getAirports*, different values for input parameters *takeoffAerodromeID* and *arrivalAerodromeID* are generated.

When the following values are generated for input parameters, the output value is obtained as shown in Figure 5.21.

- takeoffAerodromeID< $Int32$ > value: 2
- arrivalAerodromeID< $Int32$ > value: 13

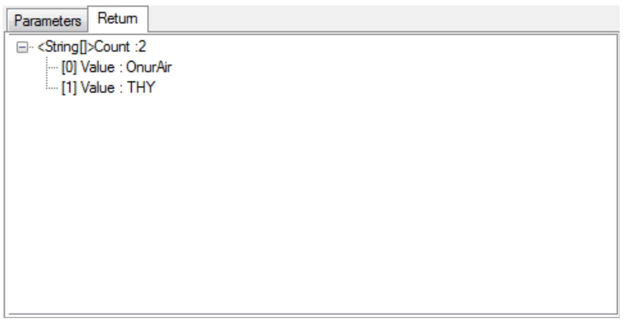


```
Parameters  Return
<String[]>Count :3
    [0] Value : THY
    [1] Value : OnurAir
    [2] Value : BoraJet
```

Figure 5.21: The Return Value of Web Service, ”getAirwaysInFlightLineByID”

The switch parameters version is applied on the service, *System.String[] getAirwaysInFlight-LineByID(Int32 takeoffAerodromeID, Int32 arrivalAerodromeID)* and different results as shown in Figure 5.22 are obtained.

- takeoffAerodromeID< $Int32$ > value: 13
- arrivalAerodromeID< $Int32$ > value: 2



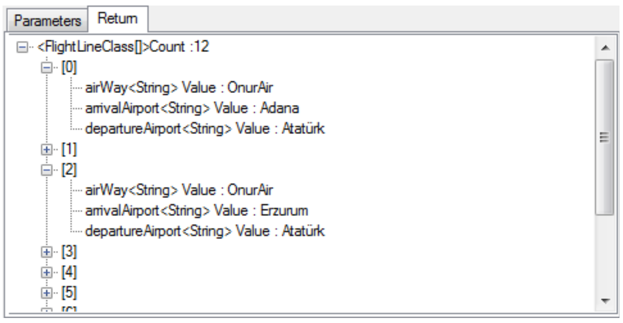
```
Parameters  Return
<String[]>Count :2
    [0] Value : OnurAir
    [1] Value : THY
```

Figure 5.22: The Return Value of Web Service, ”getAirwaysInFlightLineByID” with Switch Version

FlightLineClass[] getFlightLineOfAirline(String airline): This service has one input parameter whose type is *String* and the name is *airline*. When all fields of output parameters having the same type are searched, it is found that *airWay* field of output parameter of *getFlightLines* web service is semantically most similar to the input parameter *airline*. The similarity degree is calculated as 0.920. By using the return value of *getFlightLines*, different values for input parameter *airline* are generated.

When the following value is generated for input parameter, the return value that can be shown in Figure 5.23 is obtained.

- airline< $String$ > value: ”OnurAir”



```
⊟ <FlightLineClass[]>Count :12
   ⊟ [0]
      airWay<String> Value : OnurAir
      arrivalAirport<String> Value : Adana
      departureAirport<String> Value : Atatürk
   ⊞ [1]
   ⊟ [2]
      airWay<String> Value : OnurAir
      arrivalAirport<String> Value : Erzurum
      departureAirport<String> Value : Atatürk
   ⊞ [3]
   ⊞ [4]
   ⊞ [5]
```

Figure 5.23: The Return Value of Web Service, ”getFlightLineOfAirline”

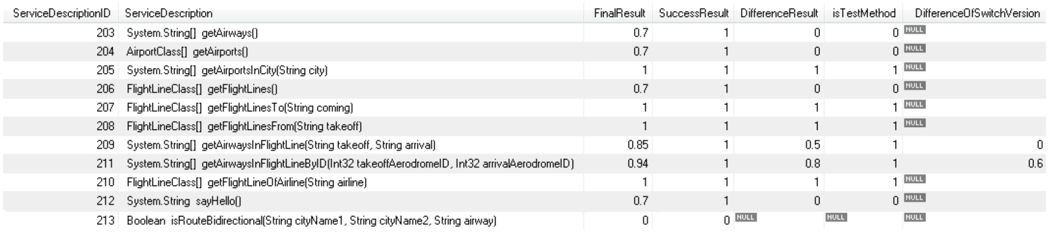
Boolean isRouteBidirectional (String cityName1, String cityName2, String airway): For three input parameters of *isRouteBidirectional*, all fields of output parameters having the same type are searched. It is found that *city* field of output parameter of *getAirports* web service is semantically most similar to the input parameter *cityName1* and *cityName2*. The similarity degree between *cityName1* and *city* is 0.75 and the similarity degree between *cityName2* and *city* is 0.75. For the input parameter *airway*, *airWay* field of output parameter of *getFlightLines* web service has the highest semantic similarity. The similarity degree between *airway* and *airWay* is 0.865. By using the return values of *getAirports* and *getFlightLines*, different values for input parameters *cityName1*, *cityName2* and *airway* are generated.

Whenever this web service invoked with the generated values like the following values for input parameters, it always throws exception. Therefore, this web service fails in testing process.

- cityName1< $String$ > value: ”Trabzon”
- cityName2< $String$ > value: ”Istanbul”

- airway< $String$ > value: ”Pegasus”

For the web services of ”Flight Management System”, with analysis of invocation results and the obtained return values of these web services, difference checking result, successful invocation result and final test result are obtained. These test results of web services are listed in Figure 5.24.



ServiceDescriptionID	ServiceDescription	FinalResult	SuccessResult	DifferenceResult	isTestMethod	DifferenceOfSwitchVersion
203	System.String[] getAirways()	0.7	1	0	0	NULL
204	AirportClass[] getAirports()	0.7	1	0	0	NULL
205	System.String[] getAirportsInCity(String city)	1	1	1	1	NULL
206	FlightLineClass[] getFlightLines()	0.7	1	0	0	NULL
207	FlightLineClass[] getFlightLinesTo(String coming)	1	1	1	1	NULL
208	FlightLineClass[] getFlightLinesFrom(String takeoff)	1	1	1	1	NULL
209	System.String[] getAirwaysInFlightLine(String takeoff, String arrival)	0.85	1	0.5	1	0
211	System.String[] getAirwaysInFlightLineByID(Int32 takeoffAerodromeID, Int32 arrivalAerodromeID)	0.94	1	0.8	1	0.6
210	FlightLineClass[] getFlightLineOfAirline(String airline)	1	1	1	1	NULL
212	System.String sayHello()	0.7	1	0	0	NULL
213	Boolean isRouteBidirectional(String cityName1, String cityName2, String airway)	0	0	NULL	NULL	NULL

Figure 5.24: The Test Results of Web Services of ”Flight Management System”

For ”Car” domain, the generated web services are on car reservation management. Definitions of these web services are given in Table 5.8.

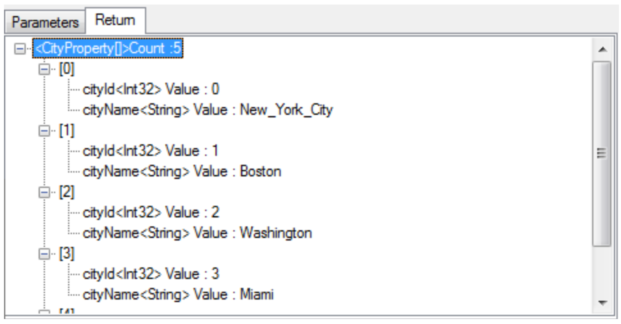
Table 5.8: The Definitions of Web Services of ”Car Reservation System”

Web Service	**Definitions**
System.String getValue()	It always return the same value, ”helloWorld” string.
CityProperty[] getAllCities()	This service returns the list of the cities in which car reservation can be made.
ModelProperty[] getAllCarModels()	It returns the list of the car models, which can be reserved.
PersonInformation[] getCostumers()	It returns the list of the costumers in reservation system.
CarProperties[] getAvailableCarsInTown (Int32 townId, Int32 modelId)	It returns the list of cars, which has the model given as parameter and is available in the town whose id is given as parameter.
CarBookingClass[] getCarBookingsByTownId(Int32 townId)	It queries the car reservations in town whose id is given as parameter and returns the list of found reservation.
CarBookingClass[] getCarBookingsByTownName(String townName)	It queries the car reservations in town whose name is given as parameter and returns the list of found reservation.
CarBookingClass[] getCarBookingsByDate (String startDate)	It queries the car reservations whose start date is given as parameter and returns the list of found reservation.
CarBookingClass[] getCarBookingsOfPerson(String personName)	It is used for getting the list of the car reservations made by the person whose name is given as parameter.
Int32 insertNewCarBookingItem (Int32 personId, String personName, Int32 rentalCarId, String startDate, String stopDate)	It inserts a new car reservation to the system with the given parameters.

The testing process are performed for each web service and the test results are obtained. Some examples of the generated values of input parameters and the obtained return values are given bellow.

System.String getValue(): This web service is a test web service and always returns the same value, ”helloWorld” string. Although it is invoked successfully and return a result, the test result is low since it is a test web service.

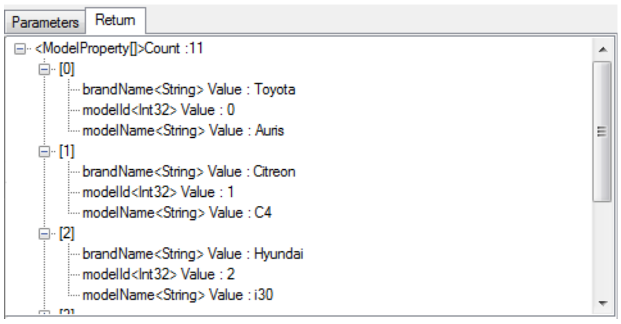
CityProperty[] getAllCities(): When this web service is invoked, the obtained return value is as shown in Figure 5.25.



```
<CityProperty[]>Count :5
  [0]
    cityId<Int32> Value : 0
    cityName<String> Value : New_York_City
  [1]
    cityId<Int32> Value : 1
    cityName<String> Value : Boston
  [2]
    cityId<Int32> Value : 2
    cityName<String> Value : Washington
  [3]
    cityId<Int32> Value : 3
    cityName<String> Value : Miami
```

Figure 5.25: The Return Value of Web Service, ”getAllCities”

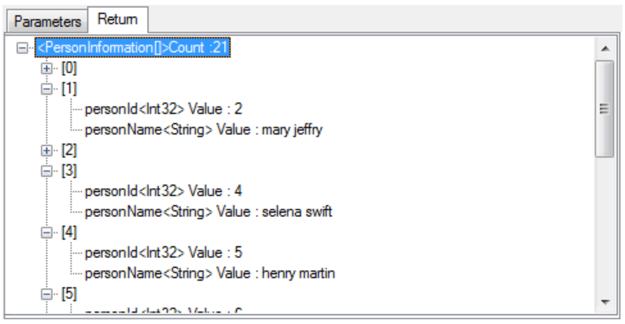
ModelProperty[] getAllCarModels(): When this web service is invoked, the obtained return value is as shown in Figure 5.26.



```
<ModelProperty[]>Count :11
  [0]
    brandName<String> Value : Toyota
    modelId<Int32> Value : 0
    modelName<String> Value : Auris
  [1]
    brandName<String> Value : Citreon
    modelId<Int32> Value : 1
    modelName<String> Value : C4
  [2]
    brandName<String> Value : Hyundai
    modelId<Int32> Value : 2
    modelName<String> Value : i30
```

Figure 5.26: The Return Value of Web Service, ”getAllModels”

PersonInformation[] getCostumers(): When this web service is invoked, the obtained return value is as shown in Figure 5.27.

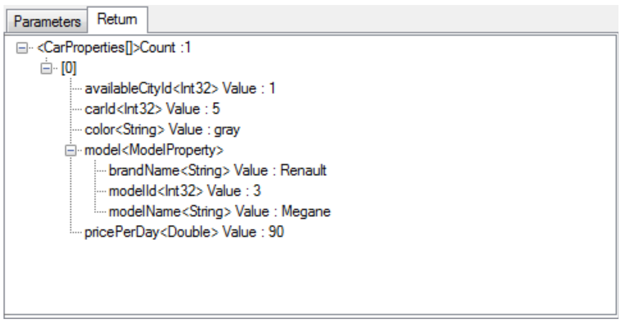


```
Parameters  Return
⊟ <PersonInformation[]>Count :21
   ⊞ [0]
   ⊟ [1]
        personId<Int32> Value : 2
        personName<String> Value : mary jeffry
   ⊞ [2]
   ⊟ [3]
        personId<Int32> Value : 4
        personName<String> Value : selena swift
   ⊟ [4]
        personId<Int32> Value : 5
        personName<String> Value : henry martin
   ⊟ [5]
```

Figure 5.27: The Return Value of Web Service, ”getCostumers”

CarProperties[] getAvailableCarsInTown(Int32 townId, Int32 modelId): For the parameters of this web service, the following values are generated for input parameters and when it is invoked with the generated input values, the return value shown in Figure 5.28 is obtained.

- townId< $Int32$ > value: 1
- modelId< $Int32$ > value: 3



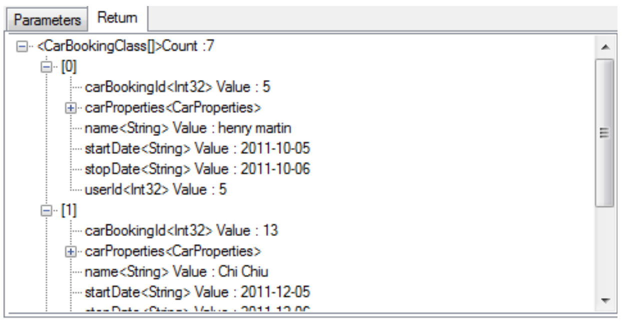
```
Parameters  Return
⊟ <CarProperties[]>Count :1
   ⊟ [0]
        availableCityId<Int32> Value : 1
        carId<Int32> Value : 5
        color<String> Value : gray
      ⊟ model<ModelProperty>
           brandName<String> Value : Renault
           modelId<Int32> Value : 3
           modelName<String> Value : Megane
        pricePerDay<Double> Value : 90
```

Figure 5.28: The Return Value of Web Service, ”getAvailableCarsInTown”

CarBookingClass[] getCarBookingsByTownId(Int32 townId): For the parameter of this web service, the following value is generated for input parameters and when it is invoked with the generated input values, the return value that can be shown in Figure 5.29 is obtained.

- townId< $Int32$ > value: 3

CarBookingClass[] getCarBookingsByTownName(String townName): When it is invoked with the generated input values, it sometimes returns a value successfully. However, in some



```
Parameters | Return
<CarBookingClass[]>Count :7
  [0]
    carBookingId<Int32> Value : 5
    carProperties<CarProperties>
    name<String> Value : henry martin
    startDate<String> Value : 2011-10-05
    stopDate<String> Value : 2011-10-06
    userId<Int32> Value : 5
  [1]
    carBookingId<Int32> Value : 13
    carProperties<CarProperties>
    name<String> Value : Chi Chiu
    startDate<String> Value : 2011-12-05
```

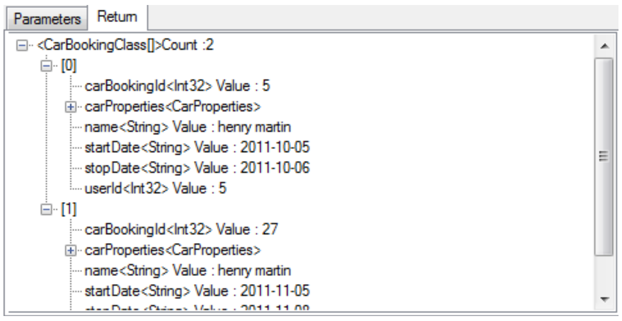
Figure 5.29: The Return Value of Web Service, ”getCarBookingsByTownId”

of test cases, it throws exception. Therefore, the final test result is low. For instance, the test case that uses the following input parameter results with an exception.

- townName$< String >$ value: Boston

CarBookingClass[] getCarBookingsOfPerson(String personName): For the *personName* input parameter of this web service, different values can be generated as follows. The output generated with the first input instance for personName is shown in Figure 5.30.

- personName$< String >$ value: ”henry martin”

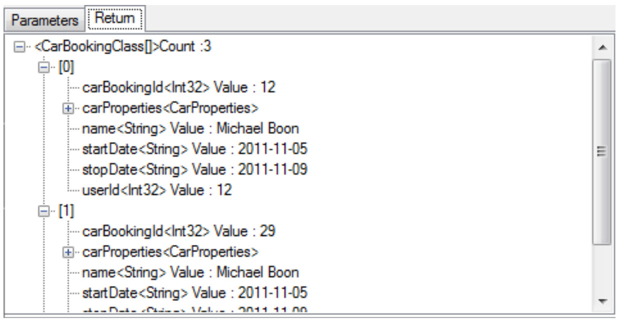


```
Parameters | Return
<CarBookingClass[]>Count :2
  [0]
    carBookingId<Int32> Value : 5
    carProperties<CarProperties>
    name<String> Value : henry martin
    startDate<String> Value : 2011-10-05
    stopDate<String> Value : 2011-10-06
    userId<Int32> Value : 5
  [1]
    carBookingId<Int32> Value : 27
    carProperties<CarProperties>
    name<String> Value : henry martin
    startDate<String> Value : 2011-11-05
```

Figure 5.30: The Return Value of Web Service, ”getCarBookingsOfPerson” - I

The output generated for the second instance of ”getCarBookingsOfPerson” web service is shown in Figure 5.31.

- personName$< String >$ value: ”Michael Boon”



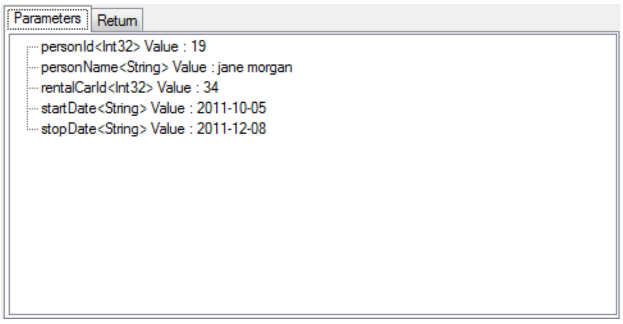
Parameters | Return

```
<CarBookingClass[]>Count :3
  [0]
    carBookingId<Int32> Value : 12
    carProperties<CarProperties>
    name<String> Value : Michael Boon
    startDate<String> Value : 2011-11-05
    stopDate<String> Value : 2011-11-09
    userId<Int32> Value : 12
  [1]
    carBookingId<Int32> Value : 29
    carProperties<CarProperties>
    name<String> Value : Michael Boon
    startDate<String> Value : 2011-11-05
```

Figure 5.31: The Return Value of Web Service, "getCarBookingsOfPerson" - II

Int32 insertNewCarBookingItem(Int32 personId, String personName, Int32 rentalCarId, String startDate, String stopDate): For the input parameters of this web service, the generated values are given in Figure 5.32. With the generated input paremeter values, it returns 1 and it means that the reservation is made successfully.

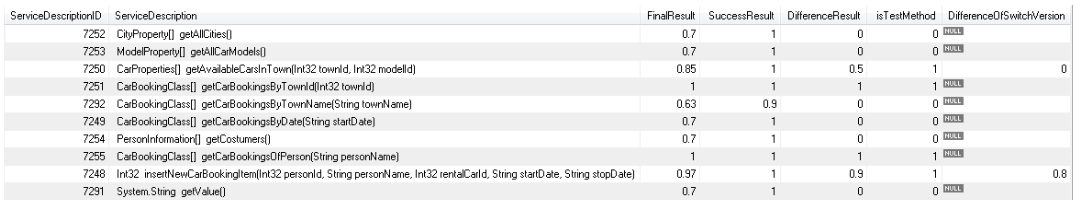
- return $< Int32 >$: 1



Parameters | Return

```
personId<Int32> Value : 19
personName<String> Value : jane morgan
rentalCarId<Int32> Value : 34
startDate<String> Value : 2011-10-05
stopDate<String> Value : 2011-12-08
```

Figure 5.32: The Input Parameter Values of Web Service, "insertNewCarBookingItem"



ServiceDescriptionID	ServiceDescription	FinalResult	SuccessResult	DifferenceResult	isTestMethod	DifferenceOfSwitchVersion
7252	CityProperty[] getAllCities()	0.7	1	0	0	NULL
7253	ModelProperty[] getAllCarModels()	0.7	1	0	0	NULL
7250	CarProperties[] getAvailableCarsInTown(Int32 townId, Int32 modelId)	0.85	1	0.5	1	0
7251	CarBookingClass[] getCarBookingsByTownId(Int32 townId)	1	1	1	1	NULL
7292	CarBookingClass[] getCarBookingsByTownName(String townName)	0.63	0.9	0	0	NULL
7249	CarBookingClass[] getCarBookingsByDate(String startDate)	0.7	1	0	0	NULL
7254	PersonInformation[] getCostumers()	0.7	1	0	0	NULL
7255	CarBookingClass[] getCarBookingsOfPerson(String personName)	1	1	1	1	NULL
7248	Int32 insertNewCarBookingItem(Int32 personId, String personName, Int32 rentalCarId, String startDate, String stopDate)	0.97	1	0.9	1	0.8
7291	System.String getValue()	0.7	1	0	0	NULL

Figure 5.33: The Test Results of Web Services of "Car Reservation System"

For the web services of "Car Reservation System", with analysis of invocation results and the obtained return values of these web services, difference checking result, successful invocation result and final test result are obtained. These test results of web services are listed in Figure 5.33.

After testing of syntactically generated web services, in order to obtain the accuracy of our proposed algorithm, we calculate the root mean square error (RMSE) by using the obtained final test result. The root mean squared error of test value is evaluated by Equation 5.8.

$$\sqrt{\frac{1}{n}\sum_{k=1}^{N}(E_k - T_k)^2} \tag{5.8}$$

where E_k is the estimated test value by our algorithm, T_k is the target expected test value of test web service and N is the length of the test web service set.

Accuracy results for the syntactically generated web services and average accuracy are presented in Table 5.9. The sum of square of the errors is 0.7595 and the average of square of the errors is 0.055. As the final result, the root mean square error is 0.2346, which means that the proposed algorithm within thesis predicts the test values of web services with 0.2346 accuracy error.

Table 5.9: Mean Square Error Values of Test Results

Web Service Name	**Expected (T)**	**Estimated (E)**	**(E - T)^2**
sayHello	0.5	0.7	0.04
getAirways	1	0.7	0.09
getAirports	1	0.7	0.09
getFlightLines	1	0.7	0.09
getAirportsInCity	1	1	0
getFlightLinesTo	1	1	0
getFlightLinesFrom	1	1	0
getAirwaysInFlightLine	1	0.85	0.0225
getAirwaysInFlightLineByID	1	0.94	0.0036
getFlightLineOfAirline	1	1	0
isRouteBidirectional	0	0	0
getValue	0.5	0.7	0.04
getAllCities	1	0.7	0.09
getAllCarModels	1	0.7	0.09
getCostumers	1	0.7	0.09
getAvailableCarsInTown	1	0.85	0.0225
getCarBookingsByTownId	1	1	0
getCarBookingsByTownName	0	0.63	0.3969
getCarBookingsByDate	1	0.7	0.09
getCarBookingsOfPerson	1	1	0
insertNewCarBookingItem	1	0.97	0.0009
The root mean square error	0.2346		

CHAPTER 6

CONCLUSION

In this thesis, we proposed a method to automatically test discovered web services. This method is realized within a web service discovery framework, DSWSD-S that aims to facilitate web service discovery process by finding the most appropriate and successfully tested web services. The module of the DSWSD-S System, which is responsible for testing is developed. To realize the proposed method, an application is designed and a graphical user interface is also presented to test the web services automatically and run the algorithm step by step for the web services of each service provider.

In our approach, mutation-based and semantics-based input data generation methods are used and with these methods, the most suitable input values are aimed to generate. The first method is data mutation. By using this method, the web services are tested by using random and specific values in different ranges. Values in these ranges are generated according to the input type. Different data groups are constructed to generate values for input parameters. The proposed data mutation analysis is inspired from the studies of Martin et al. [2], which emphasizes robustness testing of web services and Siblini et al. [1] which proposes a mutation testing method. However techniques proposed in these studies are not directly applied, but rather modified and extended with new properties. Unlike [2], our work includes various error-prone input value generation for different types in order to check the robustness of web services. Siblini et al. [1] defines mutant operators to the WSDL document of web services. Unlike it, data mutation is applied for input parameters in this study. Switching of generated values of input parameters is one of the applied operations. In this method, the ontology-based data generation is performed by utilizing the instances of DSWSD-S ontology and WordNet ontology.

In the second method, the web services are analyzed semantically to generate different test cases by using the obtained results of analysis. AbuJarour et al. [4] and Bai et al. [3] propose a method using relations between the web services. In the study of Bai et al., the dependencies are syntactically analyzed and the dependency results are used for operation flow generation. However, in our study, both syntactically and semantically dependency analysis are performed for input parameter value generation. We follow a method, which is similar to AbuJarour and colleagues' method to generate values for input parameters. However, while they use the obtained values to generate annotations for web services, we used them in testing of web services.

The web services are invoked by using the input parameters that are generated with these two methods and the invocation results are used for calculating the test results. The test results are obtained by the analysis of invocation results in two different ways. In the first direction, it is checked whether an exception occurs during the execution of the web service request or not. The web services, which throw no exceptions are accepted as successful. In the second direction, it is checked whether the web service returns different results for different test cases.

We can also develop a structure, which enables the user to specify how to evaluate the functions of the web services in testing process. For example, we currently expect that a web service returns different values for different input values and the web services which always return the same value have has low test values. However, the user can prefer that he/she does not care about the difference of return values of web services in testing process.

In this study, we test web services using WSDL and to generate values for the input parameters of web services, we semantically analyze the relations between the input parameters and ontology terms of DSWSD-S System ontologies and WordNet ontology. However, if we test the web services using OWL-S, it is not needed. The OWL-S enables the ontology-based semantic specification of the service inputs, outputs, preconditions and effects. The relations between the input parameters and terms in its own ontology are given and the related ontology instances can be directly used for input parameter values without no semantically analysis.

As a future work, test case generation methods can be extended and the generated test cases of successfully tested web services can be used in testing of syntactically and semantically similar web services.

REFERENCES

[1] R. Siblini and N. Mansour, “Testing web services,” *Proc. 3rd ACS/IEEE Int Computer Systems and Applications Conf*, 2005.

[2] E. Martin, S. Basu, and T. Xie, “Automated robustness testing of web services,” *In Proceedings of the 4th SOAWS*, October 2006.

[3] X. Bai, W. Dong, W. T. Tsai, and Y. Chen, “Wsdl-based automatic test case generation for web services testing,” *Proc. of SOSE*, pp. 207–212, 2005.

[4] M. AbuJarour and S. Oergel, “Automatic sampling of web services,” *Proc. IEEE Int Web Services (ICWS) Conf*, pp. 291–298, 2011.

[5] Y. Wang, X. Bai, J. Li, and R. Huang, “Ontology-based test case generation for testing web services,” *Proc. of ISADS*, pp. 43–50, 2007.

[6] D. Dranidis, D. Kourtesis, and E. Ramollari, “Formal verification of web service behavioural conformance through testing,” *Annals of Mathematics, Computing & Teleinformatics*, vol. 1, pp. 36–43, 2007.

[7] “Certpedia-articles. available: http://www.certpedia.com/?p=2533. last accessed 06th jan 2012..”

[8] Y. Zhang and H. Zhu, “Ontology for service oriented testing of web services,” *in Proc. SOSE*, pp. 129–134, 2008.

[9] D. Canturk and P. Senkul, “Using semantic information for distributed web service discovery,” *International journal of Web Science, in press*, 2011.

[10] D. Canturk and P. Senkul, “Service acquisition and validation in a distributed service discovery system consisting of domain-specific sub-systems,” *Proc. of ICEIS*, pp. 93–99, 2010.

[11] X. Bai, S. Lee, W. T. Tsai, and Y. Chen, “Ontology-based test modeling and partition testing of web services,” *Proc. of ICWS*, pp. 465–472, 2008.

[12] “Apache, Axis. available: http://ws.apache.org/axis/. last accessed 28th jan 2012..”

[13] C. Csallner and Y. Smaragdakis, “JCrasher: An automatic robustness tester for Java,” *Software: Practice & Experience*, vol. 34, pp. 1025–1050, 2004.

[14] D. Canturk and P. Senkul, “Semantic annotation of web services with lexicon-based alignment,” *IEEE 7th World Congress on Services (Services 2011)*, pp. 355–362, 2011.

[15] “Wikipedia, ”mutation testing”. available: http://en.wikipedia.org/wiki/mutation_testing. last accessed 06th jan 2012..”

[16] "Wordnet, "an electronic lexical database". available: http://wordnet.princeton.edu/wordnet/. last accessed 06th jan 2012."

[17] "Strikeiron. available: http://www.strikeiron.com/. last accessed 19th nov 2011.."

[18] "Webservicex.net. available: http://www.webservicex.net/ws/. last accessed 19th nov 2011.."

[19] X. Bai, Y. Wang, G. Dai, W. T. Tsai, and Y. Chen, "A framework for contract-based collaborative verification and validation of web services," *CBSE'07 Proceedings of the 10th international conference on Component-based software engineering*, pp. 258–273, 2007.

[20] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini, "Ws-taxi: A wsdl-based testing tool for web services," in *Proc. Int. Conf. Software Testing Verification and Validation ICST '09*, pp. 326–335, 2009.

[21] T. Cao, P. Felix., R. Castanet, and I. Berrada, "Online testing framework for web services," in *Proc. Third Int Software Testing, Verification and Validation (ICST) Conf*, pp. 363–372, 2010.

[22] T. Cao, T. Phan-Quang, P. Felix, and R. Castanet, "Automated runtime verification for web services," *Proc. of ICWS*, pp. 76–82, 2010.

[23] W. Dong and H. Yu, "Web service testing method based on fault-coverage," in *Proc. 10th IEEE Int. Enterprise Distributed Object Computing Conf. Workshops EDOCW '06*, 2006.

[24] Y. Jiang, Y. Li, S. Hou, and L. Zhang, "Test-data generation for web services based on contract mutation," in *Proc. Third IEEE Int. Conf. Secure Software Integration and Reliability Improvement SSIRI 2009*, pp. 281–286, 2009.

[25] M. S. Jokhio, "Goal-based testing of semantic web services," *Proc. of ASE*, pp. 707–711, 2009.

[26] S. Lee, X. Bai, and Y. Chen, "Automatic mutation testing and simulation on owl-s specified web services," in *Proc. 41st Annual Simulation Symp. ANSS 2008*, pp. 149–156, 2008.

[27] H. Mei and L. Zhang, "A framework for testing web services and its supporting tool," *Proc. of SOSE*, pp. 199–206, 2005.

[28] L. Shan and H. Zhu, "Generating structurally complex test cases by data mutation: A case study of testing an automated modelling tool.," *Computer Journal*, pp. 571–588, 2007.

[29] W. T. Tsai, Y. Chen, and R. Paul, "Specification-based verification and validation of web services and service-oriented operating systems," in *Proc. 10th IEEE Int. Workshop Object-Oriented Real-Time Dependable Systems WORDS 2005*, pp. 139–147, 2005.

[30] W. T. Tsai, R. Paul, Y. Wang, C. Fan, and D. Wang, "Extending wsdl to facilitate web services testing," *Proc. of HASE*, pp. 171–172, 2002.

[31] W. T. Tsai, X. Wei, Y. Chen, and R. Paul, "A robust testing framework for verifying web services by completeness and consistency analysis," in *Proc. IEEE Int. Workshop Service-Oriented System Engineering SOSE 2005*, pp. 151–158, 2005.