# Testing Discovered Web Services Automatically

Pinar Karagoz[1] and Selma Utku[2]
[1]*Department of Computer Eng, Middle East Technical University, Ankara, Turkey*
[2]*Aselsan Inc, Ankara, Turkey*

Keywords: Web Service, Web Service Testing, Semantic Dependency Analysis, Mutation Analysis.

Abstract: The reliability of web services is important for both users and software developers. In order to guarantee the reliability of the web services that are invoked and integrated at runtime, mechanisms for automatic testing of web services are needed. A basic issue in web service testing is to be able to generate appropriate input values for web services and to estimate whether the output obtained is proper for the functionality. In this work, we propose a method for automatic web service testing that uses semantics dependency-based and data mutation-based techniques to generate different test cases and to analyze web services. We check whether the services function properly under the input values generated and enriched from various data sources and we check robustness of web services by generating random and erronous data inputs. Experimental evaluation with real web services show that proposed mechanisms provide promising results for automatic testing of web services.

## 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 web users. They are published in high numbers, they become outdated very rapidly, and there is no standard control mechanism for their reliability. Traditional offline and manual testing processes are not always applicable to testing of web services. Therefore, it is necessary to have a mechanism for online and automated testing of dynamically discovered and selected web services to be included in a software application (Wang et al., 2007), (Dranidis et al., 2007).

In this work, we present a method for automated testing of web services. An important issue that we target in this work is to generate appropriate input values for web services automatically. To this aim, we adopt two previous techniques from different areas: mutation analysis, and semantic analysis. In mutation analysis, a web service is tested by using random and specified values in different ranges that are set according to the parameter type. Whereas in semantic analysis, dependencies among web services that are published by the same service provider are semantically analysed. Different test cases are generated and the test score of a web service is obtained by examining the outputs in two dimensions. Since we do not know the exact behaviour of the services, we aim to employ simple yet effective methods here. In the first dimension, if an exception occurs during the execution of the web service request, the web service is considered unsuccessful. The second dimension is based on checking whether the web service returns different results for different test cases.

In this study, we work on web services specified in Web Services Description Language (WSDL)[1], which is the prominent specification method for web services. The methods proposed in this work do not need semantic description of services. The reason behind this choice is based on the observation that almost all of the published web services do not have semantic definitions. One basic assumption is that web services have providers and they publish a set of services, and these service are generally related such that they belong to the same domain and output of one service may be input for another one. We use such dependencies in semantic analysis of web services. The proposed method is implemented in a domain specific web service discovery system. The web service testing module in this system is in charge of checking whether the discovered services function correctly. The performance of the proposed method is evaluated on synthetic and real world web services.

[1]http://www.w3.org/TR/wsdl

The experimental results indicate the usability of the approach.

The main contribution of this work in comparison to previous work can be summarized as follows: The proposed method does not rely on an external mechanism or framework, it can be easily applied. It does not assume availability of semantic service description or description of internal mechanism of web service. For input generation for web services, it combines data mutation based technique with semantic dependency analysis among web services. Test score generation process is performed automatically.

The rest of this paper is organized as follows. In Section 2, related work is summarized. In Section 3, general architecture of the proposed web service testing method is described. In Section 4, mutation-based analysis is presented. In Section 5, semantic based analysis is given and this is followed by the score generation described in Section 6. Evaluation results are presented in Section 7. The paper is concluded in Section 8 with an overview and future work.

## 2 RELATED WORK

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. (Martin et al., 2006) is one of studies that emphasize the robustness testing of web services by using WSDL. It presents a framework to generate and execute robustness test cases automatically.

In (Bai et al., 2005), Bai et al. propose an approach about WSDL-based test generation and test case documentation to provide reusability of generated test cases. Test cases are generated in four levels: test data generation, individual test operation generation, operation flow generation, and test specification generation. Test data is generated by analyzing WSDL message definitions. In individual test operation generation level, input parameters of web services are analyzed and test operations are generated. In operation flow generation level, the sequence of the web services are determined by the analysis of dependencies between the web services. In this approach, three dependencies are used; input dependency, input/output dependency and output dependency.

Siblini et al. (Siblini and Mansour, 2005) propose a mutation testing method for web services. The aim of this approach is to find errors relevant to both the WSDL interface and the logic of web service programming. In their 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 a mutant. Mutant operators are applied to input parameters and output parameter of web services and the data types that are defined in the WSDL document. .

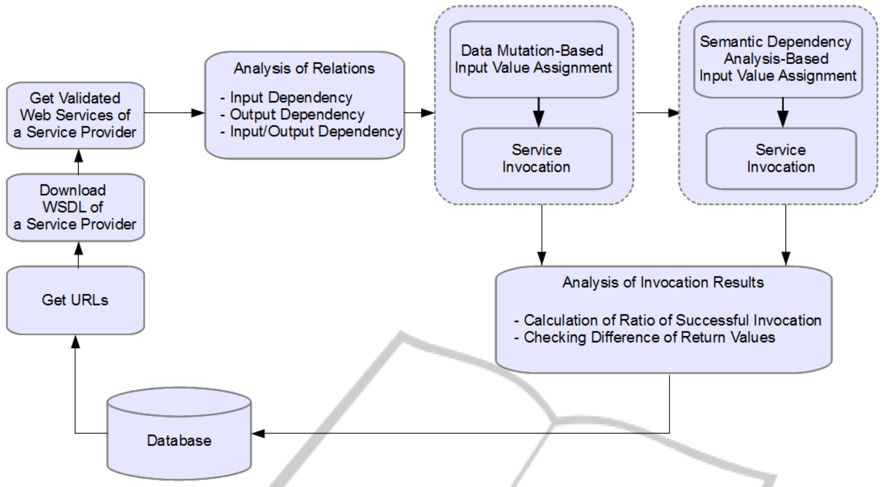
Obtaining valid inputs and outputs is a tedious work and often such information is not readily available. AbuJarour et al. (AbuJarour and Oergel, 2011) propose an approach to generate annotations for web services, i.e.., valid input parameters, examples of expected outputs and tags, by sampling invocations 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 such as WordNet, DBpedia, Freebase, are used.

Although the proposed method have similarities with (Bai et al., 2005), (Siblini and Mansour, 2005) and (AbuJarour and Oergel, 2011), as the basic difference, in this study, the emphasis is on generating appropriate inputs for testing and generate an overall test score automatically.

## 3 GENERAL ARCHITECTURE

The proposed work aims to automatically test 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. This document provides valuable information for service invocation. However, this information is not sufficient to test the web services. Since the behavioral information of web service is not available, only black-box testing can be performed for validation and testing of web service.

Within the scope of this work, web service validation is defined as 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. On the other hand, 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. To this aim, firstly, the types of the input and output parameters of a given web service should be identified. As the atomic parameter types, boolean, character, integer and floating point number are the most common ones. By using



Get URLs
Download WSDL of a Service Provider
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
Analysis of Invocation Results
- Calculation of Ratio of Successful Invocation
- Checking Difference of Return Values
Database

Figure 1: General Architecture of Web Testing Approach.

the atomic types, user-defined complex types can be constructed. To analyze and generate a value for basic types is more straightforward than that of complex types. In order to generate test data of complex data types, the structure is recursively analyzed until it encounters with basic data types. Since 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, setting simple default values to input parameters is sufficient for validation process.

In order to test a web service, one criteria is that no exception is taken when the web service is invoked by using the generated input values. Another important point is to check whether the service generates 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 fail the test.

The general architecture of the proposed approach is shown in Figure 1. Web service testing process starts with getting the URLs of service providers. At this point, we assume that the service is already discovered, and hence service provider's address is available. By downloading and analyzing 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. On the validated services set, input dependency, output dependency and input/output dependency relations are analyzed.

The next step is test case generation with data mutation-based method. By this method, the values of input parameters are generated according to the parameter type declarations in the service description. With the generated input parameters, the web service is invoked and the invocation result is saved for score calculation. Following this, test case generation with semantic dependency analysis is performed. For each web service, the values for input parameters of web service are determined according to the semantic dependency analysis 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 on the basis of the results of all invocations.

## 4 MUTATION ANALYSIS

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. 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 cases. In traditional mutation testing, mutation operators are used to transform the program under test. In contrast, this method is applied for generating random and erroneous 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 behaviours.

We mention that types of input parameters can be fundamental or complex type. In case of complex type, the generator recursively analyzes the structure of the input parameter type until it reaches the fundamental type. 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. We apply several versions of data nutation. 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, erroneous 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 behaviour and the web service can be crashed. 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 the parameter type.

For String typed parameters, it is very hard to generate valid values. Although the web services are not annotated, if we know the domain of the web services, it is possible that the instances can be directly taken from the ontology or the ontology can be populated with instances by using public resources. 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. 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 used, if available.

## 5 SEMANTIC DEPENDENCY ANALYSIS

A service provider generally publish multiple services and some of them have interactions with each other. In such a situation, for testing atomic web services, test cases can be generated by using these interactions. Semantic dependency analysis considers the following three types of dependencies:

- Input dependency: 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 dependency: A web service WS1 is output dependent on WS2 if and only if output parameters of WS1 and WS2 have the same type and the same name.
- Input/output dependency: 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 dependency* is 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. On this basis of this analysis, test cases are generated. 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 are analyzed.

For semantic input/output dependency analysis, similarity between the names of the parameters is found through matching degree calculation. For this calculation phase, the functions of word matching library presented in (Canturk and Senkul, 2011) is used. This matching method is extended from WordNet and it performs both syntactic and semantic matching. It is possible to use any other word matching tool that returns a score for similarity. However, we preferred to use this new matching technique since it has shown to provide promising improvement over similar techniques, especially for semantic matching.

While finding matches, each term in the first parameter is compared with the all of the terms in second parameter. As the result of this similarity calculation, 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 1.

$$sim(term_1, term_2) = \frac{1}{n}\sum_{j=1}^{n} sim(term_1, term_{2_{word_j}}) \tag{1}$$

We devise also another method to calculate the similarity degree between input and output terms where domain ontology is available. Note this ontology is not for service description, but for describing the domain of the services, such as Car or Movie domain. In this method, the distance between the input term and each ontology term and the distance between each output term and each ontology term are calculated. These distances are used to determine the ontological positions 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 2.

$$ontsim(term_1, term_2) = 1 - \frac{1}{n}\sum_{k=1}^{n} |sim(term_1, ontTerm_k) - sim(term_2, ontTerm_k)|$$

where $n$ is the number of ontology terms.

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 2.

## 6 OVERALL TEST SCORE GENERATION

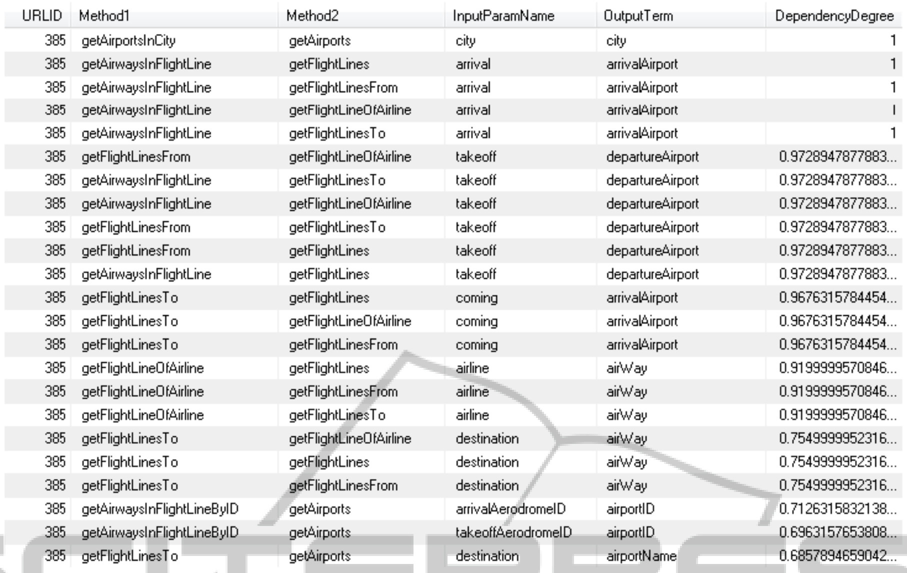
Overall test score of a web service is calculated through its performance in mutation-based and semantic dependency analysis based test generation phases. For all of validated web services, different test cases based on both semantic analysis operation and data mutations are generated.

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, five 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 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



| 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 | I |
| 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 2: Input/Output Dependency Relations of Web Services.

generation and the analysis is continued with the next web service in the dependency list.

## 7 EXPERIMENTAL EVALUATION

### 7.1 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 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. On the other hand, if it works properly and causes no exception, it is accepted as successful.

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.

The other 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 fails the test.

By using the obtained results of the web services for two dimensions 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 having different outputs for different input values. Therefore, a weighted average is applied, where the weight of first dimension is set to 0.7, under empirical evaluation.

### 7.2 Experiments with Synthesized Web Services

In this set of experiments, we created web service providers that include various services, whose behaviours are already known. These web services belong to *Aviation* and *Car* domains. For Aviation domain, the provided web services are used to get the information about flights in Turkey.

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.

Table 1: MSE on Synthesized Data Set.

| **Web Service Name** | **Expected**$(\mathbf{T})$ | **Estimated**$(\mathbf{E})$ | $(\mathbf{E}-\mathbf{T})^2$ |
|---|---|---|---|
| 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 |
| **MSE** | 0.2346 | | |

Table 2: Statistical Information on Web services in 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 |

In order to obtain the accuracy of our proposed algorithm, we calculate the root mean square error (RMSE). Accuracy results for the syntactically generated web services and average accuracy are presented in Table 1. The proposed testing method predicts the reliability of this set of web services with 0.2346 accuracy error. Since the similar studies in the literature do not produce a comparable overall test score, it is not possible to make a direct comparison with the literature. However, this error value is promising for the usability and effectiveness of the proposed method.

## 7.3 Experiments with Real Web Services

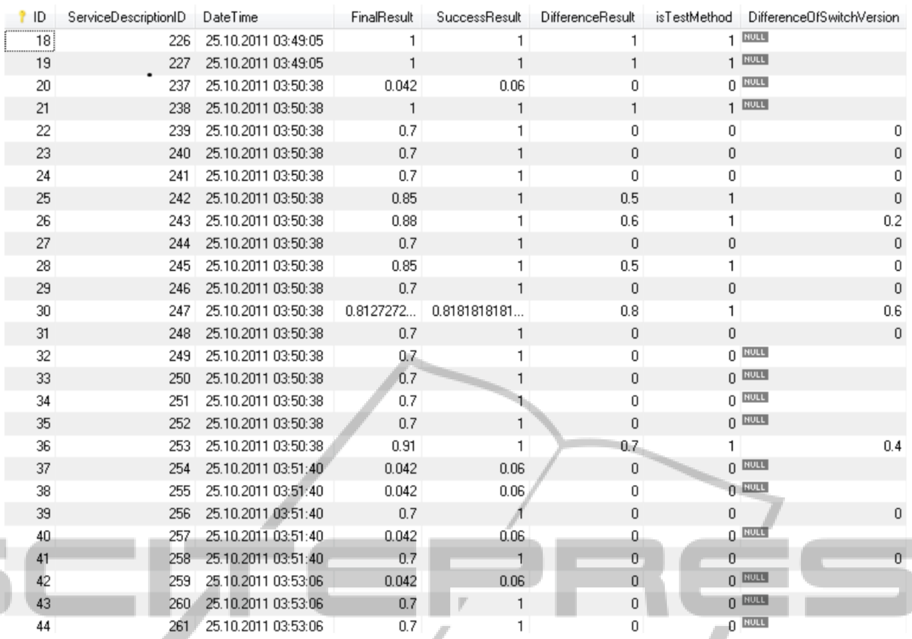
In addition to experiments with synthetic data set, the proposed method is tested on a set of real web services collected in *Car, Aviation, Film* and *Sports* domains. In this set of experiments, it is not possible to obtain a success rate, as the services are not annotated. The results rather provides an approximate picture of the current situation for the reliability of the published services. The web services are collected through web search given the domain name as the keyword. The number of the web services that are provided by the same provider, and the number and type of the parameters have a high variation in this collection. Due to space limitation, we present the results for web services in *Car* domain.

Some of the statistical information for the web services collected in Car domain is given in Table 2. In this collection, out of all web services, 349 services have the overall test score of 1 hence they successfully passed the test cases. The overall test score of 2545 web services is 0, which means that they are totally failed in the test process. 2501 web services are successfully invoked in all test cases of 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 value under switching of input values is 2453. Sample results for this set of web services is given in Figure 3.

## 8 CONCLUSION

In this work, we proposed a method to test discovered web services and to generate a test scores automatically. To realize the proposed method, an application is designed and a graphical user interface is also presented to test the web services automatically through running the method step by step for a given web service.

In our approach, data mutation based and seman-



| 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 3: Sample test results for Web Services in Car Domain.

tic dependency based input data generation methods are used. In data mutation, web services are tested by using random and specific values in different ranges. Values in the specified ranges are generated according to the input type. Different data mutation groups are constructed to generate values for input parameters. Since the internal working mechanism of web services are not available in practice, and for most of the time, the exact respond is not known as well, we prefer to use simple, yet as much as possible techniques for testing.

To be able to evaluate the success of the proposed method, we generated a synthetic data set whose behaviour is known. In this data set, error value is 0.23 under RMSE. It is not possible to make a direct comparison with similar studies since they do not generate such an overall test score. However, this error value is promising for the applicability of the approach. In addition to synthetic data set, we tested the real web services. In this evaluation, it is not possible to provide a success rate as these services are not annotated. This evaluation show that the number of web services that can pass all test cases is about 20%.

## REFERENCES

AbuJarour, M. and Oergel, S. (2011). Automatic sampling of web services. In *Proc. IEEE Int. Web Services Conf. (ICWS)*, pages 291–298.

Bai, X., Dong, W., Tsai, W. T., and Chen, Y. (2005). WSDL-based automatic test case generation for web services testing. In *Proc. of SOSE*, pages 207–212.

Canturk, D. and Senkul, P. (2011). Semantic annotation of web services with lexicon-based alignment. In *Proc. of IEEE 7th World Congress on Services (SERVICES)*, page 355362.

Dranidis, D., Kourtesis, D., and Ramollari, E. (2007). Formal verification of web service behavioral conformance through testing. *Annals of Mathematics, Computing and Teleinformatics*, 1:36–43.

Martin, E., Basu, S., and Xie, T. (2006). Automated robustness testing of web services. In *Proc. of the 4th SOAWS*.

Siblini, R. and Mansour, N. (2005). Testing web services. In *Proc. 3rd ACS/IEEE Int. Computer Systems and Applications Conf.*

Wang, Y., Bai, X., Li, J., and Huang, R. (2007). Ontology-based test case generation for testing web services. In *JProc. of ISADS*, page 4350.