The Role of Semantics in Testing Semantic Web Services

Hazlfah Mohd Rusli¹, Suhaimi Ibrahim², Mazidah Puteh¹ and Naz’ri Mahrin²
¹Faculty of Computer Science & Mathematics, Universiti Teknologi MARA, Terengganu, Malaysia
²Advanced Informatics School, Universiti Teknologi Malaysia, Kuala Lumpur, Malaysia

Abstract – Semantic Web Services are Web Services that are semantically annotated in order to make the services machine understandable, thus allowing service discovery, selection, composition, and invocation to be done automatically or with minimum human intervention. The Semantic Web services research community has been focusing on how these semantics can facilitate service discovery, selection, composition, and invocation. As of late, there have been some growing research interests in the area of Semantic Web services testing. However, it is not stated how the semantic annotations in the Web services description can help improve testing and how different it is from testing normal Web services. This paper discusses current ongoing research on testing Semantic Web services and classifies how testing uses the semantics of the Semantic Web services description.

Keywords: A Semantic Web Service; Software Testing;

1 Introduction

More and more web based applications are being developed according to the service oriented architecture (SOA) framework. It is estimated that by the year 2015, 80% of web based applications will be developed using this architectural strategy [1]. One way of implementing SOA is by using Web services. Web services are service providing web applications whose service descriptions are advertised in a repository such as Universal Description, Discovery and Integration Protocol (UDDI) registry. Web service enables interoperability between heterogeneous web applications by leveraging on standards such as Extensible Markup Language (XML), Web Service Definition Language (WSDL), Simple Object Access Protocol (SOAP) and, UDDI. While these standards allow interoperability between applications developed using different languages and on different platforms, they do not facilitate the automation of Web services tasks such as service discovery, selection, composition, and invocation. Semantic Web services (SWS) were introduced to provide solutions to the automation of these tasks. Adding semantics to the Web service descriptions enables the descriptions to be understood by machines, thus allowing automation of Web services tasks. Although there are many research conducted in the area of SWS discovery, selection, composition and, mediation, research in the areas of Web services testing, particularly, SWS testing is still new and starting to garner interest.

This paper will focus on Semantic Web services testing and offers an overview and classification of state-of-the-art approaches. The question is how semantics in the service description is used in testing Semantic Web services? Therefore the objective of this article is to find out the role of semantics in testing Semantic Web services and to provide a classification of the testing approaches used. It is hoped that the result of this paper will be able to provide an essential perspective on how testing of Semantic Web services differ from that of normal syntax based Web services.

The remainder of this paper is organized as follows. A brief overview of current syntax based Web services testing is discussed in Section 2. Section 3 briefly describes current prominent Semantic Web services approaches as well as the rule language used. A classification of the SWS testing approaches is presented in Section 4. Section 5 summarizes and discusses the testing approaches. Section 6 concludes the review.

2 Web Services Testing

At the very basic, testing a Web service consists of generating SOAP message request to be sent to the service provider from the service client. The SOAP message response is then analyzed to determine whether it is the same as what is expected of it. Testing a Web service involves having a test data generator, SOAP message generator, message executor, test oracle generator and, a test analyzer at the very least. Generating test cases based solely on the WSDL files have been researched by Sneed [2] and Bartolini [3]. For composite services, several approaches [4, 5] have been proposed for testing using Business Process Execution Language (BPEL) [6].

Both WSDL and BPEL are syntax based Web service description. Syntax based description lacks the necessary information to facilitate the automation of tasks such as selection, discovery, composition and, mediation. Semantic Web services were introduced to provide solution to the problem. While numerous researches catered to finding solutions to service selection, discovery, composition, and mediation, not many were focused on Web services testing. Only recently, there has been a growing research interest in testing Semantic Web services. This paper will discuss several

¹ Suhaimi Ibrahim and Naz’ri Mahrin are supported by Research University Grant Vot 00H68
3 Semantic Web Services Approach

Semantic Web services are Web services that are semantically annotated using ontology and rule languages. The word semantics itself means meaning and the semantic annotations provide meaning to the Web service description. Several initiatives have been proposed to implement Semantic Web services such as OWL-S [7], WSMO [8] and, WSDL-S [9]. This section briefly discusses the initiatives, focusing on the elements of the initiatives that are used in testing.

3.1 OWL-S

OWL-S is an ontology for describing Web services, hence the name Web Ontology Language for services. OWL-S ontology includes three primary sub-ontologies which are service profile, process model and grounding. The profile ontology describes what the service does, the process model ontology describes how the service is used, and finally the grounding ontology describes how to interact with it. While the service profile provides a way to describe the services offered by the providers and the services needed by the requesters, the process model describes the interaction protocol between a Web service and its client. It is organized as a workflow of processes where each processes is described by three components which are inputs, preconditions and results. Results specify the output and effect based on the process condition. The inputs and outputs use Web Ontology Language (OWL) [10] as representation whilst the precondition and postcondition use rule languages such as Semantic Web Rule Language (SWRL) [11].

3.2 WSMO

WSMO stands for Web Service Modeling Ontology. WSMO identifies four main top level elements which are ontologies, Web services, goals and mediators. Ontologies provide the terminology used by the other elements of the WSMO. Web services are computational entities that provide some value in a specific domain. Goals describe a user’s desire or objectives when consulting Web services. Mediators describe elements that handle interoperability problems between WSMO elements. Web services and goals have two main sub-elements which are capability and interface. The capability of the service contains a set of axioms that describes the precondition, postcondition, assumption and effect. The interface of a service describes how to interact with the web service (choreography) and how the service provides its functionality by making use of other services (orchestration). Both choreography and orchestration are defined in terms of their state signature and transition rules. A state signature defines the state ontology and the transition rules change the state according to the given condition.

3.3 WSDL-S

WSDL-S or Web Service Semantics is a semantic web services approach that builds on existing Web services standards by annotating WSDL with ontological information for the domain model along with pre-conditions and effects for each operation in the service. Unlike OWL-S and WSMO, it does not duplicate descriptions in the existing WSDL, but rather enhances it. WSDL-S also allows developers to use any semantic language of their choice and not fixed to just OWL, WSML or UML. In WSDL-S the WSDL operations, input and output are annotated with semantics. The Web service operations are associated with pre and post conditions using modelReference annotation on WSDL portType. Similar to OWL-S, the preconditions and postconditions can be described using rule languages such as SWRL [11] and Object Constraint Language (OCL) [12].

4 Classification of SWS Testing

Literature search on Semantic Web services testing resulted in the discovery of sixteen research papers with the earliest publication in 2005. The research papers were searched from IEEE Xplore, Springer Link, Science Direct as well as Google scholar using search strings “Semantic Web services testing”, “OWL-S and testing”, “WSMO and testing” and “WSDL-S and testing”. Once a research paper has been identified as a SWS testing paper, other relevant papers were also discovered by looking at the references of the identified paper as well searching for other papers that have cited the identified paper. Based on the issues discussed in the selected papers, we have classified the papers into three categories which are test case generation (12 papers), mutation testing (3 papers) and, test selection (1 paper). The research papers worked on SWS initiatives or specification languages such as OWL-S (8 papers), WSMO (3 papers) and WSDL-S (3 papers). The remaining two papers did not specify the exact SWS used but rather, they used the general notion of Input, Output, Precondition and Effect (IOPE) that is inherent in OWL-S, WSMO and WSDL-S. Table 1 describes the test issues addressed by the papers as well as the description language used. The following section will further discuss the issues highlighted in Table 1.

<table>
<thead>
<tr>
<th>Test Issues</th>
<th>OWL-S</th>
<th>WSMO</th>
<th>WSDL-S</th>
<th>General IOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case Generation</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Mutation Testing</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Test Selection</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: SWS Testing Research Paper Breakdown
4.1 Test Case Generation

Test cases are a set of test inputs and expected results created to exercise a particular program path or to verify compliance to a specification. Manually creating test cases can be tedious as well as time consuming. Furthermore, creating test cases manually does not support the Semantic Web services objective which is to facilitate the automation of Web services usage. Generating test input data from web services description is done by selecting data from a database that corresponds to the input parameter type. The intelligence of the test generator can be improved by selecting data that not only corresponds to the parameter type, but also satisfy the precondition of the Web service [13].

<table>
<thead>
<tr>
<th>Test Case Generation Technique</th>
<th>SWS Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OWL-S</td>
</tr>
<tr>
<td>EFSM based</td>
<td>-</td>
</tr>
<tr>
<td>Petri-Net based</td>
<td>-</td>
</tr>
<tr>
<td>Model Checking</td>
<td>1</td>
</tr>
<tr>
<td>Planner based</td>
<td>-</td>
</tr>
<tr>
<td>Decision Table</td>
<td>-</td>
</tr>
<tr>
<td>Equivalence Partition &amp; Boundary Condition testing</td>
<td>1</td>
</tr>
<tr>
<td>Pair Wise Testing</td>
<td>-</td>
</tr>
</tbody>
</table>

Several testing techniques have been employed in generating test cases such as Extended Finite State Machine (EFSM) based, Petri-net based testing, model checking, planner based, decision table, equivalence partitioning and boundary condition testing and, pair wise testing. Table 2 shows the number of research done for each test generation techniques as well as the semantic web services approaches used. The following section will discuss the different techniques in generating test cases using different semantic web services approaches.

4.1.1 Extended Finite State Machine Based (EFSM)

Finite state machines (FSM) are behavioural model used in designing computer programs and is often used for defining the temporal order of Web service interaction. However it is not sufficient to describe all aspect of a Web service as Web service have input and output messages with data parameter. In order to better describe Web service behavior, EFSM extends FSM with variables, statements and conditions [14]. Semantic Web services testing based on EFSM involves creating an EFSM representation of the Web service behaviour and applying existing EFSM test case generation to the created EFSM model [15, 16].

4.1.2 Petri-Net Based

In testing composite Web services, it is not sufficient to just evaluate the preconditions and postconditions of the service operations, as the internal execution process needs to be tested as well. Dai [13] and Wang [17] proposed the use of Petri-Net model to represent the structure and operational semantics of composite Web services due to its ability to analyze and verify properties such as reachability, liveness and deadlocks. The Perform construct in OWL-S process model is represented by Petri-Net transition, the input and preconditions mapped to Petri-Net input places and output and effects mapped to Petri-Net output places. The Petri-Net model is then traversed and test cases are generated to cover all branches of the Petri-Net. A Petri-Net ontology that catered for the IOPE semantics was created to enhance the modeling capability of the derived Petri-Net model. The derived Petri-Net ontology can be used to generate test data based on ontology reasoning.

4.1.3 Model Checking

Model checking technique has been used in generating test cases based on semantic web services description by Huang [18] and Jokhio [19, 20]. The issue in using model checking approach for generating test cases for semantic web services is how to convert or translate the existing specification language into the input language of the model checker, taking into account the semantic description. Another issue is the derivation of trap properties. In all approaches the trap properties are embedded into the input specification language as assertions. In [19], mapping rules were created to map the goal capability and interface into B elements in order to create a B specification. Like WSMO, B language is based on abstract state machine. The state signatures are mapped to B variable types, transition rules are mapped to B operations and state ontology is mapped to sets in B machine. Huang [18] converted the OWL-S process model control construct and IOPE logical formulas expressed in Planning Domain Definition Language (PDDL) into BLAST’s C-like specification input language.
4.1.4 Planner Based

Planning, or more specifically AI (artificial intelligence) planning is a process that chooses and organizes actions by anticipating the expected outcomes in order to change the state of the system. The final result of planning is a plan. Given the initial state of the world and a goal state, the planner is able to come up with a plan that consists of a sequence of actions that corresponds to a sequence of state transitions. Paradkar [21] created a planner that is able to generate a sequence of web service invocations as a test case. The test goals are generated by refining the preconditions from each Web service using a set of fault models. These goals are then fed into the planner, together with the initial state of the world and web service definitions in order to generate test cases. Similar to Ramollari’s approach [16], Paradkar does not state the actual SWS used in the approach.

4.1.5 Decision Table

Decision table is a black box testing technique used to represent complex logical relationship, where a number of combinations of actions are taken based on varying sets of conditions. A decision table has four portions, which are condition stub, action stub, condition entries and action entries. A column in the entry portion is a rule. Each rule indicate actions to be taken for the conditional circumstances indicated in the condition part of the rule. Generating test cases from decision tables involves indentifying the input conditions (causes) and action (effect), generating a cause-effect graph and converting it into a decision table and finally convert the decision table rules into test cases. Noikajana [22] used this technique to generate test cases from WSDL-S where the preconditions and effects of the operations in WSDL-S were described using SWRL. The idea is to automatically populate the condition and action entries using SWRL’s antecedent and consequents. Conditions are obtained from antecedents of precondition and post-condition whilst actions are obtained from consequents of precondition and post-condition.

4.1.6 Equivalence Partitioning & Boundary Condition Testing

In equivalence partition testing, the program’s input domain is divided into sub-domains where the sub-domains have properties that cause that program under test to either produce incorrect answer for every input element in the sub-domain or correct answers for every input element in the sub-domain. One of the issues of equivalence partitioning testing is the generation of equivalence class. It is mostly performed manually based on heuristics. Bai [23] tries to provide a solution to this problem by using the ontology information of the input parameter in OWL-S to generate input partitions. The input partitions are derived by analyzing the relationship, property and restriction of the input parameter class. Similarly, Jokhio [24] proposed to identify equivalence class using the transition rules of WSMO goals.

The guarded condition of the transition rules are obtained from the goal’s choreography interface. Jokhio [24] also proposed the use of precondition’s logical predicate to identify boundary condition for test data generation. Boundary value condition testing is an extension of equivalence partition testing in which values at the equivalence class boundaries are selected as test input.

4.1.7 Pair-Wise Testing

Exercising a set of all possible input combination of software under test is not practical as the test case would be too large to be executed exhaustively. Pair wise testing is an economical alternative to testing all possible input combinations, based on the observation that most faults are caused by interactions of at most two factors. In pair-wise testing a tester needs to first select data values for the system input variables, after which test cases are generated by covering all combinations of the selected test data values for each pair of input variables. As is it is very tedious to generate these reduced test cases combinations by hand, several pair wise testing algorithms exists to automate the process.

Noikajana [25] uses OCL to specify pre and post conditions of Web services operations described in WSDL-S. Based on the service rules and operations definitions derived from both WSDL-S and OCL file analysis, an input parameter model (IPM) is generated. The IPM consist of a set of input parameters and a set of values for each parameter. A pair-wise testing technique is then applied to the IPM. However, it is not stated which pair-wise testing technique is used.

4.2 Mutation Testing

Mutation testing is a technique where two or more mutant programs are executed together with the same test cases in order to determine the ability of the test cases to detect the mutants. Lee [26] proposed mutation testing based on OWL-S specification where the typical errors that might occur are incorrect use of ontology class in the input/output parameter, mistake in the rules defining the precondition and postconditions, control flow and data flow error in the OWL-S composite process. Based on these possible errors, Lee have identified four categories of mutants which are input/output data, condition, control flow and data flow mutation. The input/output data mutant operators were generated from two perspectives which are input type and ontology class definition, and they were the only mutants discussed in the paper.

Similar to Lee’ work [26], Wang [27] and Wang [28] also conducted research on mutation testing based on OWL-S. However, instead of using existing specification in OWL-S, the authors proposed extensions to OWL-S to accommodate interaction requirements in terms of the temporal, invocation time, application data and response time properties. The mutation operators were then generated from these extensions which use Future Time Linear Temporal Logic (FTLTL) and SWRL to describe the extended properties.
4.3 Test Selection

Although comprehensive testing is necessary to ensure the quality of a software system, executing all possible test cases can be expensive as it takes up valuable time, machine and tester resources. Bai [29] proposed an ontology based risk assessment approach to test selection where test cases for service features with high risks are given priority. Software risk assessment identifies crucial parts of the system which have a high failure probability rate or which cause serious consequences due to its failure. Thus the service features’ risks are assessed based on their failure probability and importance. Bai [29] estimates the failure probability of ontology class and service’s interface which is obtained from domain experts or historical data. The failure probability for ontology class is adjusted according to the failure probability of its dependencies (parent and property class) whilst the failure probability of service’s interface is adjusted according to its input and output parameters. Failure probability of a composite service can be estimated based on the control flow analysis obtained from the process model of OWL-S.

Table 3: Summary of SWS Testing Approaches

<table>
<thead>
<tr>
<th>Reference</th>
<th>Year</th>
<th>Semantic Description Utilization</th>
<th>Approach</th>
<th>SWS Description</th>
<th>Rule Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ramollari [16]</td>
<td>2009</td>
<td>Describe service behaviour</td>
<td>EFSM</td>
<td>General IOPE</td>
<td>RIF-PRD</td>
</tr>
<tr>
<td>Huang [18]</td>
<td>2005</td>
<td>Describe service orchestration</td>
<td>Model Checking</td>
<td>OWL-S</td>
<td>Not mentioned</td>
</tr>
<tr>
<td>Jokhio [19, 20]</td>
<td>2009</td>
<td>Describe service behaviour</td>
<td>Model Checking</td>
<td>WSMO</td>
<td>Axioms</td>
</tr>
<tr>
<td>Paradkar [21]</td>
<td>2007</td>
<td>Generate test goals for test planner input</td>
<td>Planner Based</td>
<td>General IOPE</td>
<td>Not mentioned</td>
</tr>
<tr>
<td>Noikajana [22]</td>
<td>2008</td>
<td>Determine test partitions for test data selection</td>
<td>Decision Table</td>
<td>WSDL-S</td>
<td>SWRL</td>
</tr>
<tr>
<td>Bai [23]</td>
<td>2008</td>
<td>Determine test partitions for test data selection</td>
<td>Equivalence Partition &amp; Boundary Condition</td>
<td>OWL-S</td>
<td>Not mentioned</td>
</tr>
<tr>
<td>Lee [26]</td>
<td>2008</td>
<td>Generate test mutants</td>
<td>Existing semantic desc.</td>
<td>OWL-S</td>
<td>Not mentioned</td>
</tr>
<tr>
<td>Wang [27, 28]</td>
<td>2008, 2009</td>
<td>Generate test mutants</td>
<td>Extension to OWL-S</td>
<td>OWL-S</td>
<td>Not mentioned</td>
</tr>
<tr>
<td>Bai [29]</td>
<td>2009</td>
<td>Calculate service feature risk</td>
<td>Test selection</td>
<td>OWL-S</td>
<td>Not mentioned</td>
</tr>
</tbody>
</table>

5 Discussion

A summary of the test issues discussed in the previous section has been provided in Table 3. As mentioned previously, a majority of the test issues involves test case generation, which involves test data generation and for composite services, test process generation is included as well. In the approaches, the semantic elements of the services description were utilized for 1) describing the service behavior in terms of a test model 2) describing the service behavior of a composite service in terms of a test model 3) determining test partitions for test data selection 4) calculating service feature risks 5) generating test mutants.

Sinha [15], Ramollari [16] and Jokhio [19, 20] developed test models to represent behavior of single services and used the developed test models to generate test cases using existing testing tools associated to the test models. The test cases developed consist of a sequence of input and outputs that goes through all possible state transitions. All four approaches focused on how to create formal representation of the service behavior using the IOPE information, a task which normally requires a developer’s or tester’s intuitive, experience as well as understanding of the system under test.

Similarly, Dai [13], Wang [17] and Huang [18] also focused on the creation of formal test models using semantic descriptions. However, the generated test model represents the orchestration of a composite service instead of a single service behavior. Apart from using the IOPE information, these approaches also utilizes the OWL-S process model information in order to generate test cases. The generated test cases consisted of a test process that contains a sequence of service invocations as well as the input and output from one service to another in the composite service. Paradkar [21] also generated a sequence of service invocations as test cases. However, unlike Wang [17] and Huang [18], no formal test model were generated. Instead, test goals were generated by refining the service preconditions using a set of fault models which was eventually fed into a planner based test generator.

Preconditions and postconditions allow test generators to generate test input data and expected output that not only conforms to input and output parameter type, but also satisfies rules associated with them. Approaches such as those of Bai [23], Jokhio [24] and Noikajana [25] are also able to systematically select test data from a data pool according to some partition criteria such as equivalence partitioning and boundary condition testing. Again, the task of determining these partitions are usually based on tester’s heuristic and understanding of the system under test. Fortunately this task can be automated to some extent via the use of input parameter ontology information, rules in IOPE, and guarded
condition of transition rules. Semantic descriptions of Web services have also been used in calculating risks associated with a service failure [29]. The result is used to select and prioritize test cases such that test cases will be tested according to their priority whenever there is limitation in testing time.

All the approaches we have discussed so far have shown how semantic descriptions of Web services can be used to improve test generator’s intelligence for generating test models, determining test data partitions and calculating service feature risks. However, approaches by Lee [26], Wang [27] and Wang [28] tries to verify the proper use of the semantics elements by creating mutants out of the ontology class, rules defining preconditions and postconditions, as well as control and data flow error in composite process model.

Based on the discussions, semantic description of Web services is able to assist in increasing automation of Web service testing by enhancing the intelligence of the test generator such as deriving test model, determining test data partition, generating ontology based mutants and, calculating ontology risks. Normally these testing tasks require the experience and knowledge of a human tester in order to perform them. We believe the use of semantics can minimize human efforts in performing testing tasks. However, most of the approaches are still in their early stages with many of them only reporting results of early findings. More work needs to be done to enhance research in this promising area of Semantic Web services testing.

6 Conclusion

In conclusion, this paper has provided a review on Semantic Web services testing with the aim of understanding how testing of Semantic Web services differ from testing normal Web services and what are the advantages of using semantic descriptions of Web services in testing SWS. An introduction to syntax based WS testing is presented. This is followed by a brief description of the Semantic Web services approaches focusing on the elements of each Semantic Web services approach that are used in testing. The most prominent approaches of testing Semantic Web services are then presented and classified into several categories which are test case generation, mutation testing and test selection, yet it is not possible to claim that the list is exhaustive. A summary of the approaches are presented and the results are discussed. Result of the literature study indicates that semantic annotations of Semantic Web services can improve test intelligence by assisting with tasks that normally require human tester’s experience and knowledge. We are currently in the process of studying how these semantic annotations can be used to describe the interaction behavior of Semantic Web services in order to support interaction testing of composite Web services.

7 References


