Towards Specification Based Testing for Semantic Web Services

M. Shaban Jokhio, Gillian Dobbie and Jing Sun
ADepartment of Computer Science, The University of Auckland,
Private Bag 92019, 38 Princes Street, Auckland, New Zealand
mjok002@aucklanduni.ac.nz, {gill, j.sun}@cs.auckland.ac.nz

Abstract

Web services have become popular in the modern infrastructure of the World Wide Web. They aim to provide automatic discovery, selection, and invocation of required applications (services) across the internet. However, the quality assurance aspects of web services remain a challenge. Recently, the semantic web has been introduced as an emerging technology which emphasizes presenting the meaning of the web content to achieve a machine processable automation. In this paper, we explore the synergy of applying specification based software testing techniques to semantic web services. Our approach investigates the possibility of deriving concrete test cases from the goal specification of a semantic web service in order to determine the correctness of a service implementation. Furthermore, we also propose coverage criteria to evaluate the generated test cases at both the goal and the service description levels. We demonstrate the generation and evaluation of the test cases from a goal specification with the help of a simplified discount example.

1. Introduction

Web services have revolutionized the ways enterprises conduct business on the internet. Based on the standard technologies, such as WSDL (Web Service Description Language) for description and publication, SOAP (Simple Object Access Protocol) for communication, and UDDI (Universal Description, Discovery and Integration) for registry and lookup, web services enable easy integration and interoperability between heterogeneous applications. However, a major limitation of web services is the lack of automation. The semantic web has been introduced to overcome this barrier. Semantic Web Services (SWS) add metadata and a semantic layer to the current web service technologies to enable automated processing of web tasks in order to achieve the full potential of Service-Oriented Architecture (SOA). Recently, there has been a lot of research into SWS, but most of it has been focused on web service design, composition and automation. There has been little research on how this extra layer of semantic information could be used in the testing of web service behaviors.

Most of the current research on web service testing has been limited to syntax based service descriptions. For example, Tsai et. al. [1] proposed an extension to the basic WSDL specification to facilitate web service testing, including input-output dependency testing, functional hierarchy testing and the testing of the invocation sequence of operations. Tsai [2] proposed Coyote, a tool for web service testing, which supports both unit and regression testing. Siblini [3] proposed a mutation-based testing technique by applying mutation operators to WSDL documents in order to generate mutated interfaces that can be used for testing WSDL specification. Bai et. al. [4] proposed a technique to automatically generate test cases based on WSDL by parsing and transforming it into structured DOM trees using a test case generator tool. The test cases were generated for two purposes, i.e., generating test data and generating test operations. Sneed [5] proposed WSDLTest- a web service testing tool based on static analysis of the WSDL schema. It parses the WSDL schema recursively and extracts the basic XML element sets and assigns them representative test values. Xu et. al. [6] proposed a technique for testing web services using schema perturbation operators to modify the XML schema. Perturbation operators were defined on the WSDL model, and are used to generate the test inputs for web services that use the XML message. All this research helps make testing easier with WSDL specifications and provides efficient and automated testing tools such as Coyote and WSDLTest.

The above research is mainly based on the WSDL aspects of web service testing, which provides only a syntactic description of the services. The testing examines the correctness of different parts of the WSDL specification, e.g., signatures, input and output messages of the interface, operations and message parts, and the type and sequence of the operation messages [3], [6]. However, the semantic aspects of the web service seems to be neglected, e.g., logical correctness and functional behaviors of the web services. Because these approaches rely solely on the WSDL specifications, where the semantic information about the services are not available, the contribution of those approaches is limited. Recently, Dai et. al. [7] and Wang et. al. [8] proposed a method based on OWLS specifications, a semantic description of web services. Dai proposed a contract-based technique for testing web services, testing whether the web services fulfill
Whereas, Wang proposed a methodology for ontology-based web service testing. The given OWLS specification is divided into input-output information and operational behavior as a state-machine. Finally the test-cases are generated by parsing different paths of the state-machine, and analyzing the inputs and outputs of every atomic process. Semantic based testing of web services ensures that a web service correctly fulfills its specification, as described in a semantic profile of the web service, e.g., the OWLS specification in the above case. The research presented in this paper is an initial step in this direction.

Recently, the World Wide Web Consortium (W3C) proposed a new recommendation for SWS, i.e., the Web Service Modelling Ontology (WSMO) framework [9]. It provides a semantically enriched platform for documenting, discovering and executing SWS. Its goal specification can be used as a means for deriving test cases to verify the behaviors of a corresponding web service. Before we present our approach, let us consider a scenario of semantic based testing. A goal specification documents the precise description of the intended fulfillment for a particular web service, e.g., a flight booking service. It is constructed by domain experts using the goal meta model specified in the WSMO framework [9]. This goal specification is stored in the goal repositories of the WSMO execution environment. After a web service has been implemented, the web service tester retrieves the goal specification from the WSMO repository and generates the test cases from it. The generated set of the test cases are applied to the web service that supposedly fulfils that particular goal. Based on the evaluation of the result of the test, the web service tester can guarantee the quality of service with respect to its goal specification.

Figure 1 provides an overview of our proposed goal based web service testing approach. We use the Web Service Modelling Ontology (WSMO) framework to realize the goal based testing of web services. WSMO defines user objectives in the form of WSMO goals as the top level elements. The reason WSMO was chosen is that the WSMO goal specification has a rich semantic description for specifying the user objectives that can be used to identify test cases for web service testing. WSMO components consists of four elements, i.e., the goals, web services, mediators and the ontology base. The Ontology Base has two components, i.e., the domain Ontology, which consists of static domain information used by the goal specifications and the Ontology Reasoner, which provides useful inferencing for the purpose of reasoning based on the domain ontology. The Goal Analyser component parses the goal specification and extracts the useful information from which the test cases can be generated, such as goal capability, precondition, postcondition and transition rule, discussed in detail in Section 2. It passes this information to the Test Case Generator component to produce the concrete Test cases based on standard software testing techniques. The test cases are evaluated by the Test case Evaluation process, which assures the completeness and effectiveness of the generated test cases, using the proposed coverage criteria.
discussed in Section 4. Evaluation Results are fed back to the Ontology Base to improve the test case generation process with the help of the ontology reasoner. Test case Execution is achieved by running the test cases on corresponding services which are connected to that particular goal by mediators. The Execution Results are also fed back to the Ontology Base to improve the test case generation process. A WSMO goal defines the detailed objectives through the use of the web service. The novelty of our approach is exploring the full use of this semantic description as a formal specification in the testing of web services.

The rest of the paper is organized as follows. Section 2 introduces the necessary background information about the WSMO framework. Section 3 describes the goal specification based generation of test cases. Section 4 discusses the evaluation criteria for the test cases generated. Section 5 presents the implementation issues and discusses the lessons learned. Section 6 concludes the paper and describes future work.

2. Background

WSMO is an emerging framework for SWS. It is based on the design principles of scalable mediation and strict decoupling between its elements. It defines four top level elements, i.e., ontology to formally specify the terms to be used by other elements, webservice to model the description of services, goal to formally specify the user objectives and mediator to link any of these four components and resolve mismatches. The Web Services Modelling Language (WSML) provides the ontology language and the Web Services Execution Environment (WSMX). The overview of the relationship between the components is shown in Figure 2 [9].

2.1. The example of a discount calculation goal

To clarify WSMO goal specifications and help demonstrate our approach, we present a small example case study, which we call Discount Calculation Goal example. Different types of discounts are offered based on the customer’s age and amount of purchase, according to the following business rules.

- If the age of the customer is less than 40 years and the amount of the purchase is also less than 1000 dollars then a junior normal discount is offered.
- If the age of the customer is less than 40 years and the amount of the purchase is at least 1000 dollars then a junior special discount is offered.
- If the age of the customer is greater than or equal to 40 years and the amount of the purchase is less than 1000 dollars then a senior normal discount is offered.
- If the age of the customer is greater than or equal to 40 years and the amount of the purchase is at least 1000 dollars then a senior special discount is offered.

A specification for this generic goal can be modelled according to WSML v0.2 specification [10] as follows.

```xml
goal discountCalculationServiceGoal
importsOntology _"http://www.example.org/
discountCal#discountOntology"
capability
sharedVariables
precondition
definedBy
?customer[customerAge hasValue ?cAge]
memberOf customer and
```

Figure 2. Overview of the WSMO framework

WSMO goals provide a formal specification of the user’s expectations from the execution of the web service. A goal has two important parts, i.e., requested capability - specifying what functionality the user wants and requested interface - specifying how the user wants to interact with the service. In addition to this, a goal has some descriptive annotations known as “non-functional properties” and can import an external ontology and use mediators to resolve any mismatch at the data, function and behaviour levels. A requested capability is defined in terms of a precondition, postcondition, assumption and effect as WSML predicates. Precondition defines the constrained information space for goal fulfillment where as the postcondition defines the constrained output space after the goal is fulfilled. The assumption and effect describes conditions on the world state that must hold before and after fulfillment of the goal. The requested interface defines the requested view of interaction for goal fulfilment defined in terms of a requested choreography (a desired format of interaction with the service that fulfills particular goals) and a requested orchestration (a desired service flow composition). Both contain two sub-elements, i.e., the state signature, defining the states of concepts, which are used to interact with the service and the transition rules which dictate how the states of these concepts are changed based on different conditions.
?purchase[purchaseAmount hasValue ?pAmount]  
memberOf purchasing and  
wsml#numericLessGreaterThan(?cAge,0) and  
wsml#numericLessGreaterThan(?pAmount,500)  
postcondition  
definedBy ?disc memberOf discount  
interface discountCalc_Interface  
choreography  
stateSignature  
in  
concept customer,  
concept purchasing  
out  
concept SeniorSpecialDiscount,  
concept juniorNormalDiscount,  
concept juniorSpecialDiscount,  
concept SeniorNormalDiscount  
transitionRules discountCalcGoalRules  
if (?purchase[customer hasValue ?customer,  
purchaseAmount hasValue ?pAmount]  
memberOf purchasing) then  
choose (?cAge, ?pAmount) with  
{?customer[customerAge hasValue ?cAge]  
memberOf customer and  
?purchase [purchaseAmount hasValue  
?pAmount] memberOf purchasing) do  
// Nested Rule 1.  
if (greaterEqual(?cAge,40) and  
greaterEqual(?pAmount,1000)) then  
add(?x1 memberOf SeniorSpecialDiscount)  
endif  
// Nested Rule 2.  
if (greaterEqual(?cAge,40) and  
LessThan(?pAmount,1000)) then  
add(?x2 memberOf SeniorNormalDiscount)  
endif  
// Nested Rule 3.  
if (LessThan(?cAge,40) and  
greaterEqual(?pAmount,1000)) then  
add(?y1 memberOf JuniorSpecialDiscount)  
endif  
// Nested Rule 4.  
if (LessThan(?cAge,40) and  
LessThan(?pAmount,1000)) then  
add(?y2 memberOf JuniorNormalDiscount)  
endif  
endChoose  
endif

The importsOntology section of the goal specification imports a discount ontology, which contains the definition of concepts such as customer, purchasing, discount and various types of discounts offered as mentioned in the business rules listed above. These concept definitions are used in modelling of the goal specification. The sharedVariable part of the capability section in the goal defines the variables that are shared across the different parts of the capability. The precondition part of the capability section specifies that in order to successfully achieve this goal, the instances of each customer and purchasing should be received with the non-zero value of the customer's age, customerAge, and value greater than 500 for the purchase amount purchaseAmount. Whereas the postcondition part of the capability states a discount should be offered as a consequence. The choreography section of the interface DiscountCalc_Interface consists of the stateSignature and transitionRules that describe the interaction among different states. The stateSignature consists of the list of concepts used for input and output. For example, the customer and purchasing are defined as input states specified by in mode, whereas the four discount types are defined as output states specified by (out) mode.

Finally, the transitionRules part of the goal interface specifies how the required behavior is achieved, i.e., how the discount is actually offered. Complicated business rules can be modeled through the nesting the transition rules. The outer most rule in the goal specification states that the transitions are enabled as soon as some instance of purchasing is received with some values of customer and purchaseAmount attributes. The next nested rule states that the particular value of the discount is chosen based on the value of the variables ?cAge and ?pAmount, specified using the choose construct. Finally, the inner most transitions define how exactly the discount is offered based on different values of the customerAge and purchaseAmount attributes, i.e., 'customerAge < 40 and purchaseAmount < 1000', 'customerAge < 40 and purchaseAmount ≥ 1000', 'customerAge ≥ 40 and purchaseAmount < 1000', and 'customerAge ≥ 40 and purchaseAmount ≥ 1000'.

3. Generating test cases from a goal specification

A closer look at a WSMO goal specification shows that it contains sufficient information for test case generation. This information can be obtained from two main parts of the goal specification, i.e., the goal capability and goal interface specification. In WSMO, a goal provides a high-level description of the objectives of a service, which can be used for black-box testing. In this section, we investigate the application of two widely used black-box testing techniques on a WSMO goal specification, i.e., boundary value analysis and equivalence class testing. To generate concrete test cases from a goal specification, a test case generation process was designed, which contains the three steps shown in Figure 3.

- **IdentifyTestInputClasses** - Identifies input classes for generating test cases, such as boundary conditions and equivalence classes. Each boundary condition or equivalence partition serves as the class of input which can be used for the generation of test data.
- **GenerateTestInputData** - Generates the test input data from the input classes identified by the IdentifyTestInputClasses module. The test data is created according to standard software testing techniques. For an input
class of a boundary condition or equivalence class, it generates test input data according to the robust boundary analysis testing technique.

- **LinkInputWithOutput** - Captures the expected output from the postconditions of the goal specification and links the generated input data produced by the **IdentityTestInputClasses** module with the expected outputs through the transition rules. In this way, it produces the concrete test cases, i.e., the pairs of the generated inputs and their expected outputs.

### 3.1. Identifying test input classes

In this section, we will demonstrate how test input values can be derived from the various parts of the WSMO goal specification. In order to show this, we first present an overall structure of the goal specification in Figure 4 and use it as a reference in the later sections.

#### 3.1.1. Identifying boundary conditions

Boundary value condition testing techniques test the boundary sensitivity of an input domain for a particular program. The test case generation algorithm follows the approach as described below, for identifying the boundary conditions from the goal specification. The goal specification is parsed to obtain the goal capability, which is further parsed to get the goal precondition. See Figure 4 for the relationship between the different parts of a goal specification. The precondition logical predicate is then analyzed. If it contains boundary conditions on the input space, then it is identified. At the same time, the concepts used in the preconditions are also examined. This is done by looking at the WSML **memberOf** keyword in the precondition logical predicate. The definition of the concepts identified from the precondition part is obtained by referring to the ontology domain with the help of the **importsOntology** part of the goal specification. The concept definition can be parsed to identify any implicit boundary condition on the input space, e.g., the concept **month** has, by definition, values of its attribute **monthValue** between 1 to 12. Therefore, in order to identify the complete boundary conditions, we look at the definition of both the precondition and the concept definitions. The boundary condition in both cases can be identified by examining the WSML relational operators [10]. The boundary conditions obtained in this way are considered as the input classes, which are used for the purpose of test input data generation.

#### 3.1.2. Identifying equivalence classes

The equivalence class technique tests a program by deriving test cases from the partitions of the input domain associated with unique outputs, called equivalence classes. In our case, equivalence classes can be identified from the transition rule part of a goal specification in Figure 4. First of all, a goal specification is parsed to obtain its interface and choreography. The goal **choreography** contains the transition rules that describe the exact behaviors of the service in terms of the transitions on state signatures. The transition rules represent an abstract state machine view of the desired behaviors. The guarded conditions in the transition rules can be used to classify equivalence classes on the input spaces with respect to an expect output. The expected outputs are realized by the update facts as well as the out variables of the goal.

Once the choreography, transition rules and state signature are extracted from the goal specification, all the transition rules are processed for identifying the equivalence classes. For every transition rule, the guarded condition of the rule is analyzed. A condition defining a specific range in a transition rule that has an in type concept associated with out type concepts is identified as a possible equivalence class. We can also identify the equivalence classes based on multiple variables, e.g., if customerAge\(\geq 40\) and purchaseAmount\(\geq 1000\) then a special discount is offered. In this case, an instance of the out type concept would be added to the state signature based on the condition to the controlled type states. To automatically identify such equivalence classes, we analyze the instances of controlled states and associate the transition rule guarded condition with these instance variables in the state signature space. The guarded condition can also be retrieved from the temporary variables in other transition rule(s), which might list more complicated transitions with the help of conditions on the internal states. Similarly, the boundary symbols are identified with the help of the WSML relational operators [10]. The type of the instance of concept added in the update fact is determined from the state signature, which lists the types of all the concepts used in the goal specification, as shown in Figure 4.

The boundary conditions and equivalence classes identified on the **Discount Calculation Goal** example are listed in
Table 1, where \( cAge \) represent the value of the attribute ‘customerAge’ and \( pAmount \) represents the value of the attribute ‘purchaseAmount’.

<table>
<thead>
<tr>
<th>Input Class</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cAge ( \geq 0 ) Bound Cond</td>
</tr>
<tr>
<td>2</td>
<td>pAmount ( \geq 500 ) Bound Cond</td>
</tr>
<tr>
<td>3</td>
<td>cAge ( \leq 40 ) and pAmount ( &lt; 1000 ) Eq Class 1</td>
</tr>
<tr>
<td>4</td>
<td>cAge ( \leq 40 ) and pAmount ( \geq 1000 ) Eq Class 2</td>
</tr>
<tr>
<td>5</td>
<td>cAge ( \geq 40 ) and pAmount ( &lt; 1000 ) Eq Class 3</td>
</tr>
<tr>
<td>6</td>
<td>cAge ( \geq 40 ) and pAmount ( \geq 1000 ) Eq Class 4</td>
</tr>
</tbody>
</table>

Table 1. Identified boundary conditions and equivalence classes

### 3.2. Generating test input data

Once the boundary value condition and equivalence class information is derived from goal specifications as described above, the test input data can be generated by the procedure `GenerateTestInputData`. This procedure generates test input data according to the boundary testing technique. According to this technique, a given boundary condition is tested with three values, i.e., one value exactly on the boundary, one value near the boundary in valid range and one value near the boundary in invalid range. In addition to these three values it also suggests picking one value in the middle of the valid range. For example, for the first boundary condition from Table 1, we pick values -1, 0, 1 and a normal non-zero value 25. Similarly, the second boundary condition values are 499, 500, 501 and the normal value is 600. The complete set of the test inputs are generated using boundary values of one variable and keeping other variables fixed at the normal value and repeating this process for each variable in the program. In this way, we generate seven test inputs for the identified boundary conditions as shown for test cases from 1 to 7 in Table 2.

A similar technique is used for generating test input data for the identified equivalence classes from 3 to 6 as listed in Table 1. For example, customerAge 39, 40, 41 for the equivalence class condition customerAge \( \geq 40 \), and purchaseAmount 999, 1000, 1001 for the equivalence class condition purchaseAmount \( < 1000 \). We also chose two normal values for each equivalence class condition, i.e., 20, 60 for age and 750, 1250 for the purchaseAmount. The test sets are generated using boundary values of one variable with normal values of the other variables as shown in Table 2 for test cases from 8 to 23. Note that we generated the test input data for the boundary value conditions and the equivalence classes in two separate steps, where we took different normal values for the customerAge and purchaseAmount. It is possible to use the same normal values of
Table 2. Robust boundary and equivalence class test input data

<table>
<thead>
<tr>
<th>ID</th>
<th>Test Input Data</th>
<th>ID</th>
<th>Test Input Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cAge=600</td>
<td>13</td>
<td>cAge=20</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>14</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>15</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>16</td>
<td>39</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>17</td>
<td>1250</td>
</tr>
<tr>
<td>6</td>
<td>501</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>600</td>
<td>19</td>
<td>999</td>
</tr>
<tr>
<td>8</td>
<td>750</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>9</td>
<td>40</td>
<td>21</td>
<td>60</td>
</tr>
<tr>
<td>10</td>
<td>750</td>
<td>22</td>
<td>60</td>
</tr>
<tr>
<td>11</td>
<td>20</td>
<td>23</td>
<td>60</td>
</tr>
<tr>
<td>12</td>
<td>999</td>
<td>24</td>
<td>1250</td>
</tr>
</tbody>
</table>

Table 3. Robust boundary and equivalence class test cases

<table>
<thead>
<tr>
<th>Test IDs of Input Data</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2, 4</td>
<td>Error</td>
</tr>
<tr>
<td>3, 5, 6, 7, 8, 11, 12</td>
<td>Discount1</td>
</tr>
<tr>
<td>13, 14, 15, 18</td>
<td>Discount2</td>
</tr>
<tr>
<td>9, 10, 19, 22</td>
<td>Discount3</td>
</tr>
<tr>
<td>16, 17, 20, 21, 23</td>
<td>Discount4</td>
</tr>
</tbody>
</table>

4. Evaluation criteria for test cases

It must be noted that test case generation and test case evaluation are two different and independent steps in the software testing process. It is well known that no matter which test case selection or generation method is used, some programming errors could still escape detection. Therefore, it is essential to be able to measure the effectiveness of the set of generated test cases. We propose different criteria for measuring the effectiveness of the test cases generated from WSMO goal specifications. These criteria can be measured at different levels of service usage, such as the goal level and the service description levels.

4.1. Boundary coverage criterion

Boundary conditions are derived from the goal capability specification as discussed in Section 3.1. The boundary condition coverage criteria specifies the number of the boundaries covered or tested by a given set of test cases out of the total boundary conditions present in goal precondition specifications. According to this criterion the set of test cases is said to be effective or complete if it tests all the boundary conditions identified from the goal specification completely.

Boundary Coverage Criterion: A set of test cases is said to be adequate according to the boundary coverage criterion if it covers or tests all the boundary conditions found in a goal specification, completely, according to a particular boundary testing technique.

Completeness of boundary condition testing depends on the type of the boundary analysis technique used. There are four standard boundary analysis techniques namely simple boundary testing, robust boundary testing, worst case boundary testing and robust worst case boundary testing. For simple boundary testing a set of five values are used for each boundary range, i.e., min, min+, norm, max-, max; whereas for the robust boundary testing, a boundary range is said to be tested completely if a set of seven values is used with the addition of one value below the min and another value above the max. Worst case and robust worst-case techniques increase the number of the tests which is usually impractical to achieve if the number of variables are high.

The test cases generated by the proposed algorithm as shown in Table 3 are complete according to the boundary coverage criterion. All the identified boundary condition and equivalence classes are covered according to the robust boundary testing technique. The test cases from 1 to 7 in Table 2 test the boundary conditions completely, where as the test cases from 8 to 23 test all the boundary conditions of all the identified equivalence classes. We use the robust boundary testing technique because it tests boundary condition in a reasonable way.

4.2. Transition rules path coverage criterion

Similar to an Abstract State Machine (ASM), transition rules in the WSMO goal specification are defined in terms of state transitions using conditional constructs, such as if, forAll and choose. Each transition rule produces different execution path(s). This criterion specifies the number of transition rule paths covered by the set of test cases.
Transition Rules Path Coverage Criterion: A set of test cases is said to be adequate according to this criterion, if it covers every path in the set of paths of transition rules in the goal specification at least once.

The test cases generated by the proposed algorithm for the Discount Calculation Goal example as shown in Table 3 are complete test cases according to the transition rules path coverage criterion. There are four independent paths of execution of transition rules for the Discount Calculation Goal example as shown in Figure 5. It can be easily verified that every transition path is covered more than once with respect to the test cases in Table 3.

4.3. Service level coverage criteria

The test cases that are generated can be evaluated against the goal specification or the service level specification. WSMO also has a strong semantic description at the web services level [9]. A WSMO webservice description has a similar structure to the goal specification, such as, the capability and interface, which defines the provider’s view of the available functionalities and interfaces. Capability is defined in terms of the precondition, postcondition, assumption and effect. The service interface defines choreography and orchestration. Choreography describes the interaction pattern of the service with its clients in terms of state signature and transition rules. Orchestration defines the service composition in terms of state signature and clients. A service specification can potentially strengthen or loosen its preconditions or postconditions with respect to a goal definition. For example, the web services corresponding to a particular goal could impose more boundary conditions on its input space or remove some of them. Similarly, a service can have different interface specification than that of its corresponding goal specification, e.g., different choreography description with more or less transition rules. Therefore, using the service level specification could provide additional means for evaluating the effectiveness of the test cases. After all, these test cases will be executed on the actual selected web service. For example, a test set is said to be adequate at the service level if it satisfies the boundary condition coverage and transition rule path coverage criteria derived from a service specification. Furthermore, these coverage criteria could potentially be used to help the selection of a service that is tested effectively by the set of test cases generated from the goal specification. That is, the completeness of the test cases at the service level depends on the particular service that is used for the goal fulfillment purpose. For a given set of goal specifications different services may provide different coverage as discussed above.

5. Discussion and future directions

5.1. Towards implementation

To implement the goal based testing approach, we used the WSMO4jv2.01 [11] - a Java based open source API for building SWS applications in WSMO. Since goal based test case generation needs extensive parsing and analysis of the goal specifications, we have implemented the required functionalities as a set of reusable external libraries. The test case generation algorithm as proposed in Section 3 is currently being implemented. We have also identified many challenges and opportunities during the implementation. Some limitations are the lack of mature tools for pursuing the research. The WSMO4j API and execution environment are still evolving and do not provide the complete implementation of WSMO definitions. For example, at the conceptual level the service can be discovered at four different levels, i.e., exact match, subsumption match, plug-in match and intersection match, whereas at the implementation level (in the WSMO execution environment) only the exact match between the goal and the service is implemented [12].

5.2. Formal specification based testing

So far the proposed generation of test cases is based only on two standard black box testing techniques, i.e., boundary value condition and equivalence class testing. These techniques are still primitive and may not be sufficient for producing a complete set of test cases for web service testing purposes. We are therefore investigating the application and integration of other techniques to our proposed approach. Since the WSMO goal specification is a formal representation of user objectives and formal specification based software testing has its mature testing techniques, it is natural to apply formal specification based testing techniques to the goal based web service testing, especially in the test case generation process. One possible application of such
Formal specification based testing techniques is shown in Figure 6.

- **Model checking** - Model checking tools are good at identifying counter examples with respect to a formal specification. They can be used to generate test sequences for state based specifications, which later can be mapped into the concrete test cases. Specific counter examples are generated by supplying the model checker with certain derived test predicates. The test predicates are checked against the specification and a violation trace is returned as the counter examples, starting from the initial state to the state which violates the predicate or property [13]. Test predicates from WSMO goal specifications can also be generated in such a way and utilized by the model checker for the creation of specific test sequences.

- **Theorem proving** - Theorem proving tools are good at providing formal proofs of properties with respect to a formal specification. They can be used to prove the disjointness and completeness of the partitions on an input domain for appropriate test selection purposes [14]. This is helpful to ensure the correctness of the selected test input space.

- **B related testing tools** - B specification has powerful tool support especially for testing, such as ProB, ProTest and BZTT tools [15], [16], [17]. Since B specification is a description of an abstract state machine, WSMO goal specifications can be easily translated into their corresponding B specifications and use the B testing tools for our goal based web service testing. Currently, we are investigating automated test case generation from WSMO goals using the ProB toolset.

6. Conclusion

As a recent development in the semantic web area, WSMO provides a semantically enriched framework for documenting, matching and discovery of web services. This paper investigates the possibility of deriving test cases from a WSMO goal specification for debugging the correct fulfillment of a actual service. The novelty of our approach is that the test case generation is semantics based, as compared to the syntax-based approaches mentioned in section 1. We demonstrated that the WSMO goal description can be used as a kind of formal specification for deriving more accurate test cases for testing the behaviors of web services. The contributions of the paper can be summarized as follows. First, we described an approach to derive test cases from WSMO goals using the standard boundary value condition and the equivalence class testing methods. Second, we identified the parts of a WSMO goal specification that can be used to generate test cases automatically. Third, we also proposed metrics for evaluating the effectiveness of the generated test cases from both WSMO goal and service descriptions.

In the future, we plan to complete the implementation of our goal based test case generation tool, which will enable fully automated test case generation from WSMO...
goals. This involves applying formal specification based testing techniques, such as model checking, to automatically generate test cases from goal specifications. As a matter of fact, we are currently using the ProB [15] model checker to automate the process at the moment. We also plan to investigate more measurement criteria for the evaluation of test cases, such as the predicate coverage. Furthermore, the evaluation results need to be analyzed and fed back to the test case generation process to improve the quality of the test cases. Such analysis can be conducted through the ontology knowledge base and its reasoners. Finally, we plan to apply our goal based test case generation and evaluation strategies to web service discovery and selection, where it can be determined whether the goals are fulfilled more accurately and intelligently in a semantic web environment.

7. Acknowledgement

This research work is being funded by the University of Auckland, New Zealand and Higher Education Commission (HEC), Government of Pakistan.

References


