International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 6 (2013) pp. 206 - 216

Automated Testing of Web Services System Based on OWL-S

Xi Xu[1] and Shawkat Hasan Shafin[2]

[1] School of Computer and Communication Engineering, University of Science and Technology Beijing, Xueyuan Road, Haidian District, Beijing 100083, China
xuxihasan@126.com (corresponding author)

[2] School of Computer Science and Engineering, Beijing University of Aeronautics and Astronautics, Xueyuan Road, Haidian District, Beijing 100191, China
shafin878@yahoo.com

***Abstract*: As Web Services are more and more mature and popular, large numbers of practical Web Services are published on Internet and they are increasingly integrated together, forming Web Services systems to carry out coherent tasks. However, the distributed application of Web Services always involves plenty of standard protocols and various runtime behaviors. Therefore automated testing of Web Services becomes more difficult than testing previous paradigms for software application development. In this paper we propose a series of applicable automated testing algorithms and implement an automatic testing prototype system for Web Services system based on OWL-S (Web Language for Services). First, deduce abstract test cases from interaction requirement properties of Web Services system. The properties are included in OWL-S Requirement Model extended by our research group. Second, specify test cases according to SWRL (Semantic Web Rule Language) properties and abstract test cases. In consideration of the attributes of Fit (Framework for Integrated Test), specific test case is formatted in tables and then html document. Finally, generate mutants under AOP (Aspect-Oriented Programming) technology support, drive them by specific test cases using improved Fit, and then kill mutants based on business logic implied in Requirement Model. We employ two sufficient measurement criteria to evaluate testing process. Experiments have shown that our algorithms are feasible and efficient, and the prototype system not only meets the applied demands but also performs well as an automated testing tool for Web Services system.**

***Keywords*:** automated testing, Web Services system, OWL-S Requirement Model, specific test case generation, mutant, sufficient measurement criteria

I. Introduction

Web Services form a new distributed computing paradigm that has made its way into the standard mechanism and open platform of the integration of distributed service components. As this emerging technology is increasingly mature and popular, more and more practical Web Services are published on the Internet and used by many consumers. Web Services from different vendors are always integrated together, forming Web Services system to carry out a coherent task, but the Services may contain faults in their implementation and so they will not behave as consumers' expected. The distributed application of Web Services involves plenty of standard protocols and various runtime behaviors. Therefore different defects may emerge in all aspects, such as hardware, software, communication and object management. In other words, there is a trustworthiness problem between consumers and providers and how to guarantee the quality of Web Services system has been a tough problem.

Testing plays a great role in the reliability of target Web Services system. The growing complexity of system architecture and application, continual changing of techniques and rules, etc. have imposed numerous great challenges of our traditional testing techniques. Thus, testing Web Services system is more difficult than testing previous paradigms for software application development. To carry out a coherent task, Web Services' applications may be composed dynamically from different available Services that may be located in different places and have different quality attributes. Not only is the source code of the Service unavailable, but the Service might be hosted on servers at remote, even competing organizations. In addition, individual Web Service may contain unknown faults, and it may experience intruding attempts since any consumer can bind to a Web Service deployed. Therefore lots of research on novel testing techniques for Web Services has being done recent years. Automated testing has become a significant topic in Web Services testing. The degree of automation will profoundly influence the efficiency, quality and cost of Web Services testing. If any fault is found during the testing phase, Web Services in Web Services system will be re-composed. The testing is to validate whether the Web Services system exhibits the desired properties as guaranteed on the Requirement Model and does not exhibits the undesired properties.

Lots of research on novel Web Services testing techniques

has being done recent years [1]–[4]. Existing works [5] and [6] research on how to assist Web Services testing using SOAP message. Although, the SOAP message contains the communication information, it is inadequate for Web Services system testing. Thus, as shown in Literature [7] and [8], test technique research based on formal or semi-formal specification has drawn greater attention. Model checking using formal methods provides comprehensive and detailed testing that can validate whether the composite Web Service software model meets property requirements such as logic, timing sequence. But the process is more complex because OWL-S must be used as an intermediate transformation and the testing is only valid for the programs before implementation [9]. Runtime verification combined with model checking is proposed to solve the previous problem of software testing [10], [11]. Runtime verification does not need the state space beforehand, but simply tracks state changes in a running program. It is easily implemented. Literature [12] presents a mutation-based inter procedural criterion, named Interface Mutation, suitable for use during integration testing. A case study to evaluate the proposed criterion is reported. The results suggest that Interface Mutation offers a viable test adequacy criterion for use at the integration level. Traditional mutation testing has strong ability to debug, and proved to be effective and highly automated [13]. However, the testing is a white-box testing as it mutates in source code levels. Thus, it is only adopted when source code is visible as a unit testing method. Moreover, enormous mutants cause high testing cost.

This paper presents a series of applicable automated testing algorithms for ontology-based, event-oriented, real-time embedded Web Services system, an integration of published Web services. Instance of CompositeProcess is used to describe the application flow of the target Web Services system and SWRL (Semantic Web Rule Language) in OWL-S is applied to characterize the constraints. FTLTL (Future Time Linear Temporal Logic) is combined with Requirement Model to generate test cases for the requirement properties planned to be verified. As characteristics of Web Services systems are different from traditional software systems, mutation testing is adopted in our research after comprehensive comparison of several methods. We propose a new mutation testing [14] to solve problems of traditional mutation testing. In order to execute the mutants automatically, Fit (Framework for Integrated Test) is also absorbed into our research.

The remaining section of the paper is organized as follow. Section II describes the method to generate abstract test cases. Section III details the output of specific test cases. Section IV introduces OWL-S based mutation testing. Section V shows the automatic testing prototype system and some experiment results based on two sufficient measurement criteria. Finally, Section VI contains a conclusion and discussion of our work.

II. Abstract Test Case Generation

Testing for Web Services system is driven automatically by test cases. As each operation of Web Services system driven by the generated test case is associated with certain requirement properties, we generate test cases based on requirement properties to increase automation and efficiency of the testing process. Requirements properties focus on the interaction properties of Web Services system. They are included in Requirement Model of extensional OWL-S. Requirement Model is a ProcessModel instance in OWL-S and described by SWRL and FTLTL formulae. In Requirement Model, CompositeProcess instance describes an application flow of Web Services system in detail and each AtomicProcess is corresponding to an operation in WSDL document of Web Service. Test case generation process is shown in Fig. 1.

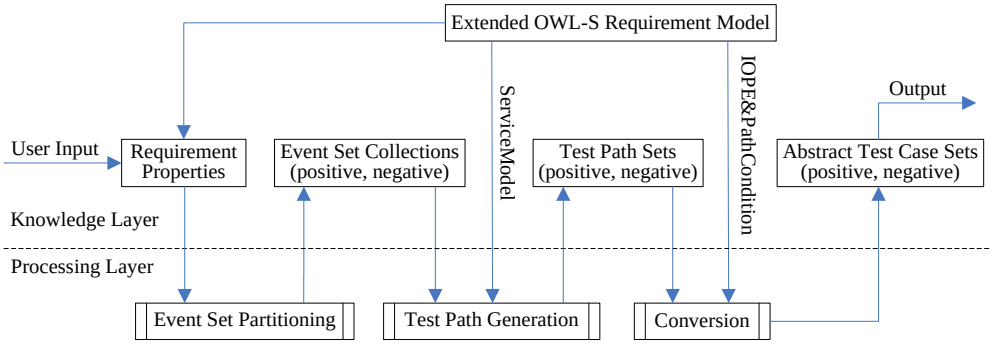

Figure 1. Process of abstract test case generation.

Event Set Partitioning module processes logical formulae expressing requirement properties and educes two types of event set collections, positive collection and negative collection, used to respectively generate positive test paths and negative test paths. In the collections, every event set validates at least one formula and all event sets complete verification of all formulae. In order to obtain test cases associated with requirement properties, Test Path Generation module searches test paths, including at least one positive event set, in CompositeProcess instances treated as a directed graph. These paths are called positive test paths. Negative test paths are derived utilizing the positive test paths combined with related negative event sets. Conversion module transforms the obtained positive test paths and negative test paths into positive test cases and negative test cases respectively. The process is mainly based on the information

that each node of the test paths has, such as Conditions of control structure [15], IOPE (Inputs, Outputs, Preconditions and Effects) properties.

A. *Event set partitioning*

Validation of interactions is our testing target, so we mainly focus on the requirement properties related to interactions among Web Services, such as temporal properties, application data properties, response time properties and amount of invocation time properties. Event Set Partitioning module derives concrete event set collections from information provided by requirement property formulae to select test paths and make sure that test cases are associated with requirement properties.

Requirement properties are described by OWL-S, which is defined in OWL ontology language and has good semantic foundation for describing interaction requirements. ServiceModel of OWL-S, viewed as a Process, describes the business process of Web Services system, which coordinates individual services [16]. It is a specification of the ways a client may interact with a service. There are three types of Processes: Atomic, Composite, and Simple. OWL-S only characterizes Composite Process of individual services, rather than some elements of non-temporal property, such as starting and finishing of calls, so we extend Requirement Model of Web Services system in OWL-S as follows:

$$WSS = (CP, AP, CC, RP). \quad (1)$$

where WSS stands for Web Services system; CP, a Composite Process; AP = {Atomic Process}; CC = {Sequence, Split, Split-Join, Choice, Unordered, If-Then-Else, Repeat-While, Repeat-Until}; RP = {Interaction Requirement Property}. Atomic Process is modeled on interaction between Web Services system and invoked individual service. Each element in CC stands for a control construct in business process of Web Services system. Control construct unites some Atomic Processes to model on business process which only implicitly expresses some temporal relations among individual services, rather than describes interaction requirements explicitly. There is no RP in standard OWL-S Model. We extend the Model by adding a series of classes to expression interaction requirement properties and its availability already has been verified by our research group [17].

To facilitate its processing, requirement properties are divided into two categories, temporal properties and non-temporal properties. FTLTL, based on a rewriting-based algorithm for generating a minimal special observer FSM (Finite State Machine), or an automaton, from an LTL (linear temporal logic) formula, is suitable to describe temporal constraints of interaction requirement properties [18]. It provides temporal operators that refer to the future/remaining part of an execution trace relative to a current reference point. We define the operators in OWL-S Model to constitute temporal logic formula. Temporal property can be any legal LTL formula. It is stored in the form of binary tree in self-defined class: PropertyFormula and mainly constraints the calling sequence of Atomic Processes. Most properties involve several calls of Atomic Processes. Non-temporal property can be treated as attribute of certain class. We introduce SWRL into OWL-S to characterize non-temporal property of Requirement Model, including constraints of invoking times, application data constraints and constrains of response time. SWRL allows users to write rules that can be expressed in terms of OWL concepts to provide more powerful deductive reasoning capabilities than OWL alone. Semantically, SWRL is built on the same description logic foundation as OWL and provides similar strong formal guarantees when performing inference.

Events in this paper refer to the invoking of related Atomic Processes. An event is associated with either a non-temporal property or a predicate of temporal property formula. If the latter, it is expressed by property hasAtomicProcess of self-defined class SoapAction which is the leaf node of formula binary tree. Positive event set is derived from non-temporal property and antecedent of temporal property. It contains series of events that may make the property formula true. Currently non-temporal property is simply associated with calls of single Atomic Process, so there is only one positive event set in their event set collections and the event set contains just one event, namely the invoking of Atomic Process that the property is associated with. Negative event set is educed from converse consequent of temporal property (refer to our previous paper [17] for details). If these event sets have duplicate ones in the collections, remove the duplicates. The corresponding relationship between event sets and formulae are stored during the processing.

B. *Test path generation*

Requirement properties are described by logical Formulae. They are abstract properties of demand level, not suitable for selecting test paths directly. We utilize concrete event set collections, which already have been obtained by Event Set Partitioning, to select test paths.

Test path generation algorithm is a recursive algorithm. It searches test paths in the tree structure storage of CompositeProcess instance straightly. When Web Services system is very complicated, application flow chart of the system, described by CompositeProcess instance, becomes more sophisticated. Control construct of flow chart, branch, concurrency and loop will lead to an explosion of test paths. The search algorithm of control construct Sequence is shown in Fig. 2.

```
Algorithm TestPathGeneration
Input: ControlConstruct Instance rootControlConstruct
Procedure:
1. if ( rootControlConstruct is an instance of Sequence ) {
2.    for each process component of rootControlConstruct {
3.       invoke function TestPathGeneration recursively using
         current process component as parameter
4.       store the return at the end of testPathsList
5.    }
6.    if ( testPathsList.size ( ) > 1 ) {
7.       combine all elements of testPathsList
8.       store the results in testPathsList
9.    }
10.   filter the test paths in testPathsList by event sets of positive
      event set collection
11.   return the first element of testPathList
      //now, there is only one element of type List in testPathsList
12.}
Output: List testPathList
```

Figure 2. Processing of sequence.

There are three ways taken to solve the combinatorial explosion problem.

- Serialize process components of concurrent construct. For concurrent construct, Split and Split + Join, output sequence of its process components is specified, but the meaning of concurrency among the process component does not change.
- Control the cycle index of loop construct. Perhaps there are many sub-test paths in the application flow chart described by CompositeProcess instance. When the algorithm accesses loop construct, Repeat-Until and Repeat-While, we constrain the cycle index, for example only once.
- Filter sub-test paths using obtained event sets and remove paths that have nothing to do with requirement properties. For control constructs whose outputs can be serialized, each of their sub-nodes separates the whole construct into several sub-constructs. As a result, several sub-paths are obtained from each sub-construct. A complete test path consists of these sub-paths combined together in sequence. If there is any event set of collection included in a test path or a sub-path, the path is relevant to the event set collection; otherwise, filter out the test path.

C. Conversion to abstract test case

Test paths are converted into test cases according to IOPE properties of the process ontology and condition information in OWL-S. Conversion from positive test paths to positive test cases is similar to that from negative test paths to negative test cases. In our research, a test case is defined as an ensemble of input data, expected output (specific value or certain constraint of output), operations or calling sequence of Web Services, and set of formulae which it verifies.

The correspondences between test path and test case are shown in Table 1. The information collection of test path is performed automatically. Information for precondition, incondition and conditional effect/output of Atomic Process, is abstracted from properties of the Atomic Process. The rest items' information for test path, listed in first column of Table 1, is obtained directly from test path generation algorithm. Thus, from the last Atomic Process of the test path, we match its precondition and incondition with output and conditionalEffect of Atomic Process before it, until the first Atomic Process of the test path. For Atomic Processes with no constraint, we use the first Result as default value during the matching from back to front. Otherwise, for those have constraints, match all Results with the conditions in condition list one by one until the best matching is find out.

Table 1. Correspondences between test path and test case.

Test Path	Test Case
conditions of path, preconditions, inconditions	inputs
calling sequence of Atomic Processes	operations or calling sequence of Web Services
set of formulae which it verifies	set of formulae which it verifies
conditional effects\outputs of Atomic Processes in the test path	expected outputs

According to this section, abstract test case, including abstract input data (data types and constraints), expected output, operations or calling sequence of Web Services and set of formulae which it verifies, is deduced from interaction requirement properties and stored separately in items.

III. Specific Test Case Output

As mentioned above, only abstract input data, such as data types and constraints, are derived. However, instances of input data are not available. Moreover, we integrate the open-source testing framework Fit to execute mutant automatically according to the property of Fit, which describes the test cases by using HTML table, test cases need to be formatted in tables which are interpreted by a "fixture" written by programmers and saved as HTML files using ordinary business tools such as Microsoft Word. This section mainly details how to output test case documents in tables and html format automatically. The process is shown as follows in Fig. 3.

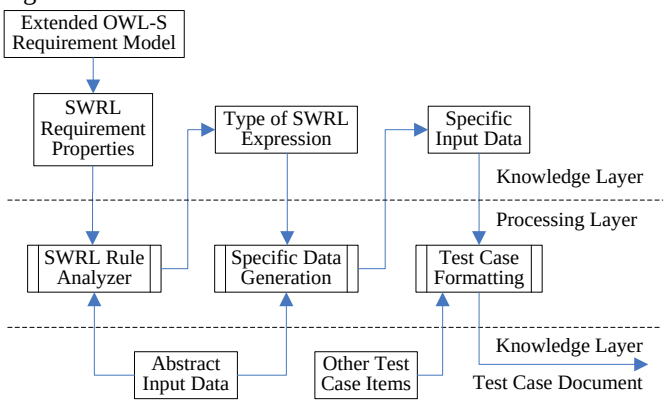

Figure 3. Process of output of test case documents.

A. Specific input data generation

In extended OWL-S Requirement Model, self-defined class ParameterConstraint indicates the running system state that given parameter meets certain constraints. It has two properties. Property hasParameter, ranging over owl: ObjectProperty, represents the constrained object and hasConstraint, ranging over expr: SWRL-Condition, applies SWRL to express specific content of the constraint. Class ParameterConstraint is associated with the antecedent of certain SWRL expression by hasConstraint to express data constraint.

SWRL supports the specification of the dependencies and restrictions of input data. Related SWRL properties of test case are extracted from OWL-S Model. After parsing and analyzing, SWRL expression are classified into several types in accordance with its contents. All these make the generation of parameter instances convenient. We design input data generators, which fill the parameter constraints with real values, for each SWRL expression type. Various generators are extended according to the practical application. The parameter values can be generated based on the constraint analysis of the property, especially the value constraint and cardinality constraint. SWRL expressions are conjunctive normal form with form swrl: ⋯ /\ swrl: ⋯ /\ ⋯. The string after character "?" is the name of the parameter that the SWRL expression describes. The number after "," is a constant of input fields. The intervals of parameters are determined by their types and constraints. Conditions influencing the value of parameters are mapped into input fields. It means that for each parameter, input field is classified into effective equivalence classes and noneffective equivalence classes by equivalence partitioning. Initial input data are generated randomly from different equivalence classes. Then boundary value analysis is applied to supplement data generation. Main algorithm is shown as follows in Fig. 4.

Algorithm SpecificInputDataGeneration
Input: Data Type, type of Parameter P_x,
Exp_j, SWRL expressions relating to P_x
1. DataRegion = InitialRegion(Data Type)
2. for (i = 1 to m) {
// m is the number of SWRL expressions in conjunctive normal form
3. DataRegion = DataRegion ∩ CalRange (Exp_i)
4. $InputData_x$ = ∅
5. for (j = 1 to u) {
6. $DataRange_j$ = GetRange(DataRegion, j)
7. $InputData_x$=$InputData_x$ ∪ GenerateInputData($DataRange_j$, Num)
8. }
9. NoneffectiveDataRegion = CalNoneffectiveRegion(DataRegion)
10. for (k = 1 to v) {
11. $NoneffectiveDataRange_k$=GetRange(NoneffectiveDataRegion, k)
12. $InputData_x$ = $InputData_x$ ∪ GenerateInputData
(NoneffectivDataRange$_k$, InvalidNum)
13. }
14.}
Output: InputData, input data set of P_x.

Figure 4. Main algorithm of specific input data generation

Functions in algorithm are defined as follows:

- InitialRegion (Data Type) returns the region in which computers can handle data of Data Type.
- DataRegion is effective equivalence class. It consists of a group of input intervals that have no intersection.

$$DataRegion = \bigcup_{1 \le j \le u} DataRange_j, \quad (2)$$

$$for\ \forall j_1 \neq j_2, DataRange_{j_1} \cap DataRange_{j_2} = \varnothing. \quad (3)$$

- CalRange (Exp_i) computes the range of parameter P_x under SWRL constraint Exp_i.
- GetRange(DataRegion, j) returns the j_{st} input interval of Data Region.
- GenerateInputData (DataRangej, Num) generates certain number, Num, of input data randomly. Num is given by user. It returns input data set following uniform distribution, including boundary values.
- CalNoneffectiveRegion (DataRegion) returns non-effective equivalence class: NoneffectiveDataRegion.

B. Test case formatting

Java2word is a java program called Microsoft Office Word document components (Class). It creates Microsoft Word Documents from java code without using any component or library. In order to realize batch generation of word documents containing tables, the template file of Microsoft Word Document must be replaced with new test case template. New template has two tables, Setups and fit.ActionFixture (demonstrated in Fig. 9) and they can be interpreted by a "fixture" written by programmer. Bookmarks are added to each cell so that test case data can be inserted into correct position. The tables are modified according to concrete applications.

When Word documents of test cases are obtained, we use Jacob transform them to html documents. Jacob is a java-com bridge that allows programmer to call COM automation components from Java. It uses JNI to make native calls into the COM and Win32 libraries.

IV. OWL-S Based Mutation Testing

From microcosmic perspective, there are two classes of errors in Web Services system. One is hidden in the Web Services' combination process. It is due to programmer's misunderstanding of system requirements. To avoid this type of error, program should be written in accordance with Requirements Model of Web Services system described by extended OWL-S. The other is potential within sub Web Services. Compared with the former, it is more difficult to be detected as source codes of sub Web Services are always invisible. The theoretical foundation of mutation testing is the competent programmer hypothesis and the coupling effect. The former means competent programmers tend to write programs close to be correct. The latter states that a test data set that detects all simple faults in a program is so sensitive that it will also detect more complex faults. It is proved that if a software system has an error, then there must be a group of corresponding mutants, and the mutants can be killed by a certain test case set [19].

In general, for a given program P and its corresponding

test case set T, the procedure of traditional mutation testing is as follows[20]:

- Use mutation operators to generate a group of mutants from P.
- Drive all mutants and P respectively by T; record their outputs. If the output of a mutant is different from that of P driven by the same test case, the mutant is killed by the test case. Otherwise, the mutant is alive.
- Ensure that none of the living mutants is equivalent to P.
- Add new test case and test nonequivalent mutants further until satisfied mutation score MS is achieved. MS is computed as follows:

$$MS = \frac{D}{M - E}. \quad (4)$$

where M is the totality of mutants; D, the number of dead mutants; E, the number of equivalent mutants. Thus, MS represents the percentage of nonequivalent dead mutants. First, tester predefines MS based on software requirements and testing strategies, and then kill individual living mutant by adding new test cases, verifying correctness, analyzing equivalence. The mutant killing operation is repeated until MS is satisfying. Software errors in P are finally found and corrected by this process. Fig. 5 shows the process of our new mutation testing.

A. *Mutation operator*

In our testing, mutants are generated using mutation operators from original Web Services system. The operators have several features resulting from differences between traditional software testing and Web Services system testing like the operators are more focused on misunderstanding of requirement model, especially errors caused by combination of sub Web Services, and they only work on interface of sub Web Services.

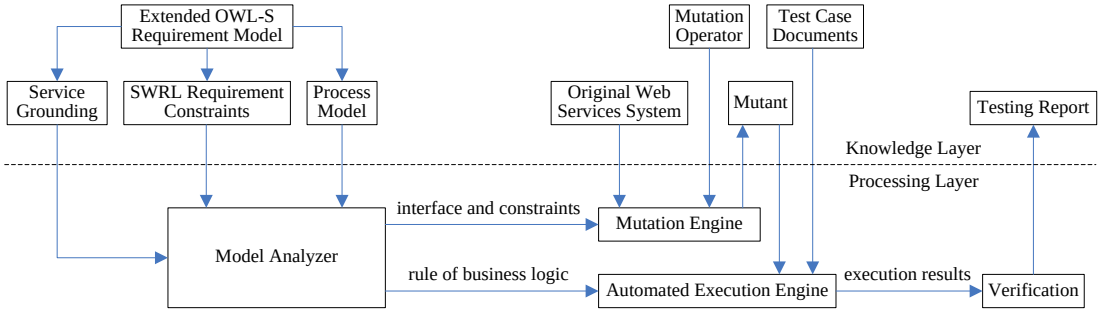

Figure 5. Process of mutation testing based on OWL-S.

Abundant requirement constraints in extended OWL-S Model are very beneficial to the testing. Assume all syntax errors injected in mutants are ignored. Mutation operators based on OWL-S Requirement Model are defined by our research group as follows [17]:

- EMP (Empty Parameter). For example, a given sub Web Service interface, boolean login(String name, String passwd), is mutated to login(null, passwd) or login("", passwd).
- EXC (Exchange Parameter). E.g., boolean login(String name, String passwd) to login(passwd, name).
- IUO (Insert Unary Operator). E.g., Boolean save(String cardNum, int amount) to save(cardNum, -amount) or save(cardNum, ~amount) or save(cardNum, ++amount) etc..
- PS (Parameter Shift). E.g., boolean save(String cardNum, int amount) would be mutated to save(cardNum, amount>>1) or save(cardNum, amount<<1).
- PLT (Parameter Length Transform). E.g., if there is a constraint swrl: stringLength(?passwd, 6), boolean logon(String name, String passwd) is mutated by appending one character to passwd filed and the length of passwd filed becomes seven.
- PA (Parameter assignment). E.g., if there is a constraint swrlb: lessThan (?amount, 2000), boolean save (String cardNum, int amount) to save(cardNum, 2000) by set the boundary value.
- SSRTT (Sub Service Response Time Transform). When timeout constraint occurs on a given sub Web Service, delay the sub Web Service's invocation according to the value of property hasTimeout.
- SSIS (Sub Service Interface Swap). E.g., given two sub service interfaces boolean f(int) and boolean g(int), replace one with the other when invoking.

B. *Mutant generation*

Mutant generation means injecting errors into the original Web Services system. There are two way of injecting. One is in source code level, called compile-time error injection. For example, $i = i - 1$ is mutated to $i = i + 1$. The other is in binary code level, called runtime error injection. Compile-time error injection needs complex syntax and grammar analysis of source code, while runtime error injection requires special platform (software or hardware). Considering Requirement Model and Service Grounding which map the abstract Atomic Process to concrete operation in WSDL documentation, we have brought forward mutation using a new technology, AOP (Aspect-Oriented Programming).

AOP is a new technology and its compiler could construct new system by recompiling original software system with AOP files. Our researches are based on java platform and Web Service instances are constructed in WebLogic platform. Besides, AOP prototype has the best java implementation, AspejctJ. Therefore AspectJ is introduced to give technical

support. Using AOP to generate mutants demands neither source code of the original system nor support of special (hardware or software) platform. It just needs binary code of original system. Moreover, mutation based on AOP takes good advantage of our self-defined mutation operators.

To mutation testing, the most important concepts of AOP are "pointcut" and "advice". Low level information on system implementation is required to form "pointcut". The formal definition of "pointcut" is show in Fig. 6. We parse Service Grounding by using OWL-S API of Mindswap [20] to get the information. We use "advice", especially "around advice", to mutate, as errors are generated into "around advice" body by mutation operators. There are two other advices, "before advice" and "after advice". They together with "pointcut" aid tester with tracing the execution of program. The formal definition of "advice" is show in Fig. 7.

```
<pointcut> ::= <access_type> pointcut <pointcut_name> ({<parameters>})
        : {[!] designator [&& | ||]};
<access_type> ::= public | private [abstract]
<pointcut_name> ::= {<identifier>}
<paramters> ::= {<type> <identifier> }
<designator> ::= designator_identifier(<signature> | <typePattern> | <pointctu>)
<designator_identifier> := call | execution | target | args | cflow | within | ..
<identifier> ::= letter {letter | digit}
<type> ::= defined valid Java type
<typePattern> ::= Java class type
```

Figure 6. Formal definition of "pointcut".

```
<advice> ::= <declaration> "{" [ <body>] "}"
<declaration> ::= <advice_type> ({<parameters>}) <after_qualifier>
        : [args(<parameters>) && ] <pointcut>
<advice_type> ::= before | after | around
<paramters> ::= {<paramter>}
<paramter> ::= <type> <identifier>
<after_qualifier> ::= returning (<paramter>) | throwing (<paramter>)
<body> ::= valid Java code
```

Figure 7. Formal definition of "advice".

When a mutant executes, the execution trace will be recorded. We use the record to kill the mutants according to business logic implied in OWL-S Requirement Model.

An example of AspejctJ file, injected errors of sub service response time, is demonstrated in Fig. 8.

```
pointcut LogSoap_logOutWithID_bSS (String id, String passwd, LogSoap thisObj):
                Call (boolean logOutWithID (String,String));
                && args (id, passwd);
                &&target (thisObj);
boolean around (String id, String passwd, LogSoap thisObj):
                LogSoap_logOutWithID_bSS(id, passwd, thisObj)
                {
                  Thread.sleep(5000);
                  boolean retValue;
                  retValue = proceed (id, passwd, thisObj);
                  return retValue;
                }
```

Figure 8. AspectJ file of sub service response time mutation.

C. Mutant execution

Fit, Framework for Integrated Test, is an open-source tool for automated customer testing. It enhances communication and collaboration, and integrates the work of customers, testers and programmers [21]. Customers provide examples of how their software should work. Those examples are then connected to the software with programmer-written test fixtures and automatically checked for correctness. The customers' examples are formatted in tables which are interpreted by a "fixture" written by programmers, and saved as HTML files using ordinary business tools such as Microsoft Excel, Microsoft Word. We integrate Fit to execute mutants automatically. Fit parses HTML, finds tables, and then pass the information in the tables to fixtures. The fixtures take the information from the tables, turn them into method calls in the actual application, and check the examples in the table by running the actual program. Then, Fit creates a copy and colors the tables green, red, and yellow according to whether the software behaved as expected.

There are two steps to verify a mutant. Firstly, the mutant is filtered by verification rules of sub service combining, which are extracted from business logic described by set CC of Requirement Model. This step obviously reduces the testing cost. CC, containing eight control structures, is an abstract of Process Model. The control structures organize control flow of sub Web Services. Construct a directed graph G = (V, E) from CC. Element in set V represents Atomic Process of WSS. Element in set E, such as <a, b>, means that in software requirements two Atomic Process have timing relations; namely a must happen before b. As discussed in Mutant Generation, the trace of mutant execution has already been achieved. In the implementation, for example, there are two sub Web Services, named a, b and both are combined to a complex web service. If the system accesses service a before service b, then the tracing execution information would be like as follows:

```
…
start sub service a and timestamp is 1209885668203
end sub service a and timestamp is 1209885669207
start sub service b and timestamp is 1209885670901
end sub service b and timestamp is 1209885671402
```

Now we use the trace to verify the mutant. If the trace does not conform to G, stop the mutant execution and mark the mutant dead. Otherwise, the mutant goes to the next step, verifying outputs of the mutant execution using Fit. Automatic execution engine of mutants is built on the basis of framework Fit which supports multiple programming languages and describes the test cases by using HTML table in human readable, writable and understandable way.

V. Experiments

To verify the effectiveness and feasibility of extended OWL-S Model and methodology proposed in previous section, an automatic testing prototype system for Web Service system based on OWL-S is implemented in this paper according to the algorithms we have proposed.

A. Instance of Web Services system

We construct an instance of Web Services system, Online Bank [22]. It is combined by five sub Web Services implemented in WebLogic platform, including CCBank, ICBank, Encryption, Login and Verify. The names imply their functions. The business logic of Online Bank is shown in Fig. 9.

Users log in the Web Services system by inputting user id or card number and password must be supplied as well. The system can encrypt the input data, verify their validation, log in a selected bank and do some business process such as saving or withdrawing. OWL-S Requirement Model of Online Bank web service is constructed by protégé. The Model has a CompositeProcess including AtomicProcess instances: cp_Logon, cp_ICBank and cp_CCBank (shown in Fig. 10).

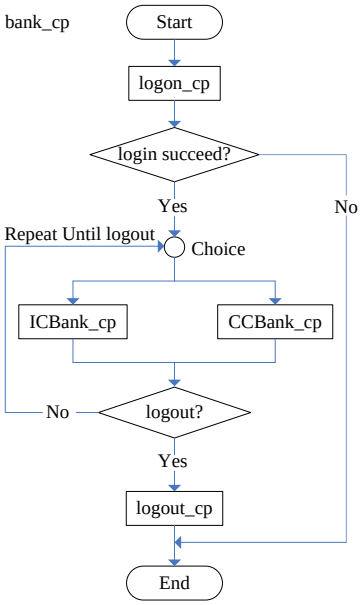

Figure 9. Business logic of Online Bank. cp means composite Web Service.

When Online Bank integrates sub Web Services and implements corresponding business logic, we supply two Sub Web Service sets, Bank Service1 and Bank Service2. Both are realization of the five sub Web Services and can be used as sub Web Service set of Online Bank. Software errors are injected into Bank Service1 intentionally, while Bank Service2 strictly obeys OWL-S Requirement Model. The interaction requirement properties which contain SWRL expressions are illustrated in Table 2.

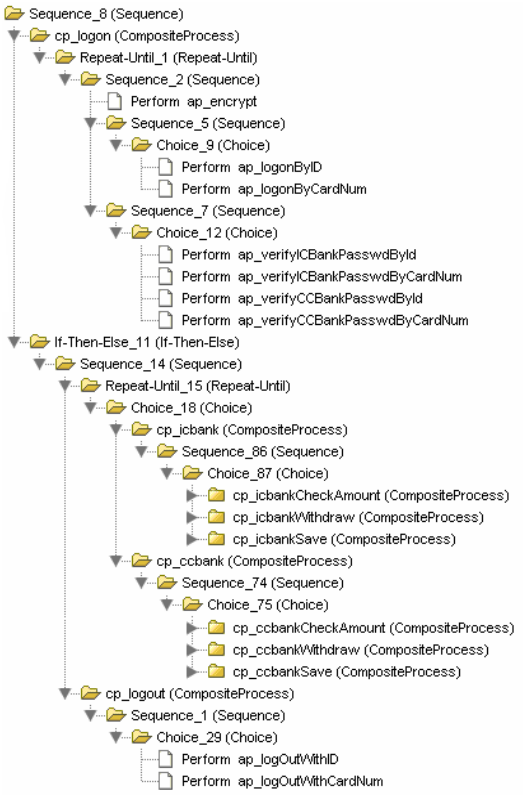

Figure 10. Tree structure of Online Bank Requirement Model

Table 2. SWRL requirement constraints of Online Bank.

Requirement constraints	Comments
hasTimeout(logon, 20)	Login is limited in 20 sec.
hasTimeout(save, 20)	Save is limited in 20 sec.
hasTimeout(withdraw, 20)	Withdraw is limited in 20 sec.
swrl:stringLength(?id, 18)	User id length is 18
swrl:stringLength(?cardNum, 19)	Card number length is 19.
swrl:stringLength(?passwd, 6)	Password length is 6.
swrl:greaterThan(?save_amount, 0)	Save amount is greater than 0.
swrl:greaterThan(?withdraw_amount,0)/\ swrl:lessThanOrEqual(?withdraw_amount, 2500)	Withdrawn amount is in (0, 2500].
Eight control structures	Describe business logic .

B. Automated testing

After constructing instance of Web Services system, including Online Bank, Bank Service1 and Bank Service2, the automatic testing is executed twice. The objects under test are as follows:

- Web Services system Online Bank + sub Web Service set Bank Service1, there are software errors in sub Web Services of Bank Service1.
- Web Services system Online Bank + sub Web Service set Bank Service2, Bank Service2 has no software errors and strictly obeys OWL-S Requirement Model.

According to the properties of automated testing of Web Services system based on OWL-S Requirement Model, we applied two types of measurement criteria. One is previously described MS. Higher MS represents better testing. The other is SSIC, short of "sub Web Service interface coverage". SSIC evaluates testing adequacy of certain sub Web Service that is

combined to composite Web Service. It is defined: given certain sub Web Service SS, the number of its interfaces called by the composite Web Service N, and the number of its interfaces which have been mutated at least one time N_1,

$$SSIC = \frac{N_1}{N}. \quad (5)$$

Compared with MS, SSIC requires neither statements nor paths covered. This feature accords with invisibility of source code of sub Web Services. MS is more rigorous than SSIC. If MS approximates to 100%, SSIC must be bound to 100%. However, the full marks of MS could hardly be achieved in practical application due to high testing cost. SSIC is supplementary of MS to evaluate the testing adequacy of sub Web Services better.

As two objects have the same OWL-S Requirement Model and they are both implemented by Online Bank Web Services system, all results of both objects before mutant execution are the same in the user interfaces (shown in Fig. 11, 12, and 13).

Figure 11. Generate mutants.

Figure 12. Compile mutants.

The documents of test case are in htm format and the specific contents are shown in Fig. 14.

When come to the interface of mutant execution, we select a group of the same test cases for the two objects to drive the mutant respectively (Fig. 15).

However, the statistical data of mutant execution, driven by the same test case set are quite different, as sub Web Service sets of the two objects are distinct—one is from Bank Service1 which has software errors and the other is from Web Service2 which obeys Requirement Model strictly.

Figure 13. Generate test cases.

//initialize

Setups					
bank	id	card Number	password	amount	add()
CCBC	568773468103130352	3374928857896548466	924733	5600	true
end	end	end	end	end	true

fit.ActionFixture		
start	TestCase1	
enter	bank name	CCBC
enter	card num	3374928857896548466
enter	password	924733
press	logon by card num	
check	status	logon successful
enter	save amount	1200
press	save button	
check	status	save successful
press	check amount	
check	amount	6800
enter	withdraw amount	300
press	withdraw button	
press	check amount	
check	amount	6500
press	logout	
check	status	logout successful

Figure 14. Specific Contents of HTM Document.

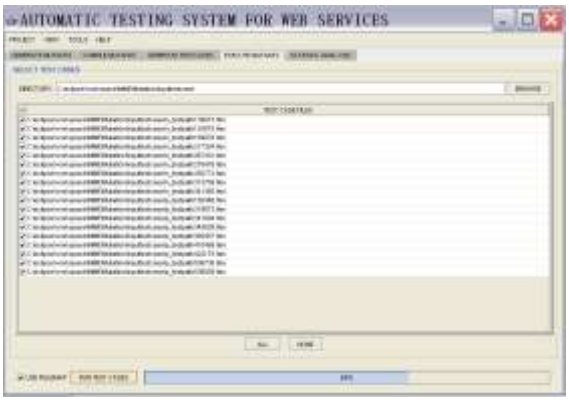

Figure 15. Execute mutants.

Statistical result information for first testing using Web Services system Online Bank + sub Web Service set Bank Service1, is show in Fig. 16. There are software errors like errors of sub service response time in sub Web Services of Bank Service1.

Fig. 17 displays statistical testing data for second testing using Web Services system Online Bank + sub Web Service set Bank Service2, Bank Service2 has no software errors and

strictly obeys OWL-S Requirement Model.

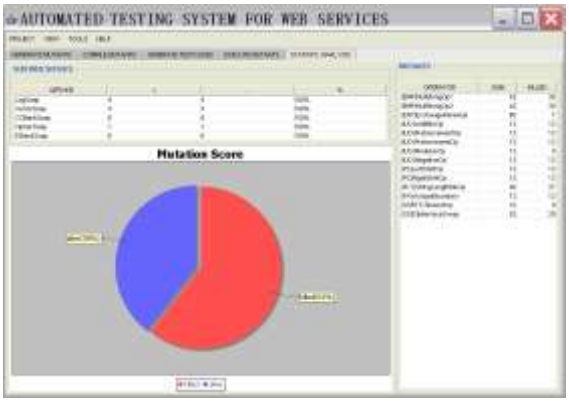

Figure 16. Statistical analysis of first testing.

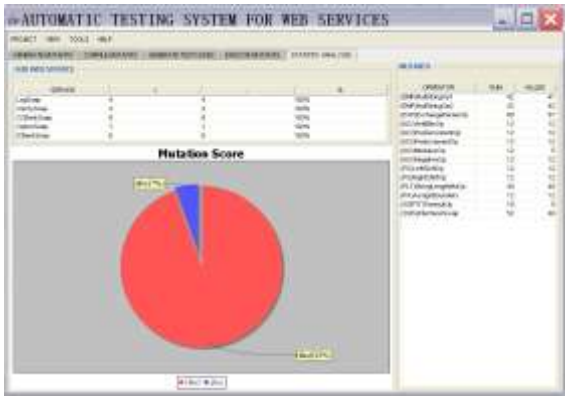

Figure 17. Statistical analysis of second testing.

The statistical data of two object shows that it is more difficult to kill mutants of the system injected errors because it requires more exhaustive test case set. Thus, one of the strong points of our automatic testing system is to find and use the smallest scale of test cases to test the Web Service System completely by adding test cases, which are automatically generated according to the extended OWL-S Requirement Model, to improve the MS. The SSIC reached 100% because the scale of target Web Services system is small. The experiments in [22] have already prove that the satisfied mutation score MS can always be achieved on the premise of the smallest test case set by our testing system.

The experimental results indicate that test methods proposed in this paper can not only evaluate test case sets, but also detect software error in target system under test. It demonstrates the effectiveness of the methods proposed in this paper at the same time.

VI. Conclusion and Discussion

A series of applicable automated testing algorithms for Web Services system is designed and realized based on extended OWL-S Requirement Model in java platform in this paper. Not only does it use requirement constraints effectively to deduce test cases according to application flow, generate mutants under AOP technology support and execute mutants by improving FIT testing framework, but also reduces the testing cost by using business logic implied in extended OWL-S Model to kill mutants and increases the degree of automation for the testing. According to testing characteristics, two sufficient measurement criteria are employed to evaluate the testing process in the system. Experiments have shown that our algorithms meet the applied demands and perform well as an automated testing tool for Web Services system.

Although the algorithms are implemented achieving the research objectives and performs well as an automatic testing tool for Web Service system, there still are a lot of stuff to work on. For example, put forward more effective mutation operators focused on Requirement Model, extend more data types for input data generator.

References

[1] H. Huang, W. –T. Tsai, R. Paul. “Automated Model Checking and Testing for Composite Web Services”. In *Proceedings of the eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 300-307, 2005.

[2] W. Z. Xu, J. Offutt, J. Luo. “Testing Web Services by XML Perturbation”. In *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, pp. 257-266, 2005.

[3] A.T. Endo, A. da Simao, S. Souza, P. Souza. “Web Services Composition Testing: A Strategy Based on Structural Testing of Parallel Programs”. In *Proceedings of Testing: Academic & Industrial Conference on Practice and Research Techniques*, pp. 3-12, 2008.

[4] H. Zhu, Y. F. Zhang. “Collaborative Testing of Web Services”, *IEEE Transactions on Services Computing*, 5(1), pp. 116-130, 2012.

[5] N. Looker, J. Xu. “Assessing the Dependability of SOAP RPC-Based Web Services by Fault Injection”. In *Proceedings of the ninth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pp. 163-170, 2003.

[6] N. Looker, J. Xu. “Assessing the Dependability of OGSA Middleware by Fault Injection”. In *Proceedings of the 22nd International Symposium on Reliable Distributed Systems*, pp. 293-302, 2003.

[7] L. Gallagher, J. Offutt. “Automatically Testing Interacting Software Compontents”. In *Proceedings of the International Workshop on Automation of Software Test*. pp. 57-63, 2006.

[8] A. Ankolekar, M. Paolucci, K. Sycara. “Towards a Formal Verification of OWL-S Process Models”. In *Proceedings of the fourth International Conference on Semantic Web*, pp. 37-51, 2005.

[9] S. H. Lin, Y. F. Lin, J. J. –Y. Chen. “BPEL4WS Verification Environment Using an Enhanced OWL-S and VDM++”. In *Proceedings of the International Conference on Advanced Computing and Communications*, pp. 642-647, 2006.

[10] V. Stolz, F. Huch. “Runtime Verification of Concurrent Haskell Programs”, *Electronic Notes in Theoretical Computer Science*, 113, pp.201-216, 2005.

[11] B. Meyer, J. Woodcock. *Verified Software: Theories, Tools, Experiments*, Springer Berlin Heidelberg, New York, 2008.

[12] M. E. Delamaro, J. C. Maidonado, A. P. Mathur. "Interface Mutation: an Approach for Integration Testing". *IEEE Transactions on Software Engineering*, 27(3), pp. 228-247, 2001.

[13] J. H. Andrews, L. C. Briand, Y. Labiche. "Is Mutation an Appropriate Tool for Testing Experiments?". In *Proceedings of the 27th International Conference on Software Engineering*, pp. 402-411, 2005.

[14] R. Wang, N. Huang. "Requirement Model-Based Mutation Testing for Web Service". In *Proceedings of the fourth International Conference on Next Generation Web Services Practices*, pp. 71-76, 2008.

[15] R. Casado, J. Tuya, C. Codart, M. Younas. "Test Case Design for Transactional Flows Using a Dependency-Based Approach". *International Journal of Computer Information Systems and Industrial Management Applications*, 5(2013), pp.30-40, 2013.

[16] Y. Z. Feng, M. Kirchberg. "Verifying OWL-S Service Process Models". In *Proceedings of the IEEE International Conference on Web Services*, pp. 307-314, 2011.

[17] Y. Yu, N. Huang, Q. Z. Luo. "OWL-S Based Interaction Testing of Web Service-Based System". In *Proceedings of the third International Conference on Next Generation Web Services Practices*, pp. 31-34, 2007.

[18] V. Aravantinos, R. Caferra, N. Peltier. "Linear Temporal Logic and Propositional Schemata, Back and Forth". In *Proceedings of the 18th International Symposium on Temporal Representation and Reasoning*, pp. 80-87, 2011.

[19] R. Geist, A. J. Offutt, F.C. Harris. "Estimation and Enhancement of Real-Time Software Reliability through Mutation Analysis". *IEEE Transactions on Computer*, 41(5), pp. 550-558, 1992.

[20] Y. Jia, M. Harman. "An Analysis and Survey of the Development of Mutation Testing". *IEEE Transactions on Software Engineering*, 37(5), pp. 649-678, 2011.

[21] F. Ricca, M. Di Penta, M. Torchiano. "Guidelines on the Use of Fit Tables in Software Maintenance Task: Lessons Learned from 8 Experiments". In *Proceedings of the IEEE International Conference on Software Maintenance*, pp. 317-326, 2008.

[22] S. H. Shafin, L. Zhang, X. Xu. "Automated Testing of Web Services System Based on OWL-S". In *Proceedings of the World Congress on Information and Communication Technologies*, pp. 1103-1108, 2012.

Author Biographies

Xi Xu received the B. Sc. Degree in computer science in 2007 from University of Science and Technology Beijing, China. She is currently a PHD candidate at University of Science and Technology Beijing. Her research interests include software testing, image processing and pattern recognition.

Shawkat Hasan Shafin received the B.Sc. Degree in computer science in 2007 from University of Science and Technology Beijing, China and the M. Sc. in computer science in 2010 from Beijing University of Aeronautics and Astronautics, China. He is currently a PHD candidate at Beijing University of Aeronautics and Astronautics. His research interests include software testing and Web Services.