Web Service Testing Method Based on Fault-coverage*

Wen-Li Dong[1], Hang YU[1]
[1]*Department of Computer Science and Engineering, Tsinghua University Beijing, China, 100084.*
E-mail: wldong@mail.tsinghua.edu.cn

Abstract

The aim of web service verification is how well the web service conforms to the WSDL specification and it is the key of web service popular adoption at present. In web service testing, the adequacy of test is the essence to verify if the software satisfies WSDL specification with the increasing complexity and applications of web service. This paper presents a systematic approach for web service testing based on fault-coverage which is intended to be used for service testing automation. In this method, HPNs representing operations are produced from WSDL specification after parsing process. A graph transforming from the HPN representing a web service is used to generate the adapted UIO sequence, then test sequence based on the adapted UIO sequence is given to acquire high fault coverage. Constraints-based test data generation for service testing getting sufficient test data to kill mutant program is presented in this paper. Constrains are departed to two kinds: user-defined and policy based on the syntactic and semantic analysis for WSDL specification. At last, we applied a test script language based on XML to effectively describe the test sequence. The test sequence and constrains for test data are expressed by this language. The test scenario is built on this formal language. The prototype system based on above method automating the web service test is developed in our lab.

1. Introduction

The aim of web service verification is how well the web service conforms to the WSDL (Web Services Description Language) [1] specification and it is the key of web service popular adoption at present. Testing is the most important way to verify web services. In web service testing, the adequacy of test (generally, test case and test data adequacy) is the essence to verify if the software satisfies WSDL specification with the increasing complexity and applications of web service.

On one hand, web services are a set of emerging standards that enable interoperable integration between heterogeneous information technology processes and systems. It can be regarded as a new breed of web application that is self-contained and self-describing, and which can provide functionality and interoperation ranging from the very basic to the most complicated business and scientific processes. Based on component techniques, web service can be composed, re-composed, and re-configured dynamic according to requirement. The integration number is large and the integration way is complex. In thus large software organization, manually generating test case and test data is a time-consuming effort and the fault frequency of test case and test data is high. In order to produce a reasonable small set of test case and test data that satisfy the WSDL specification, automatic tool is required.

On the other hand, web services hold the promise for providing a common standard mechanism for interoperable integration among disparate systems. And the key to their utility is their standardization, WSDL specification etc. With abundant data type and structure, WSDL allows the exchange of information in a decentralized, distributed environment, and the interface of web service at the same time. Thus, the automation of test case and test data generation is possible.

Recently, focusing on the cooperation point of view, some researchers have the attempt to verify for reliable web service by using model-checking techniques [2][3][4]. However, in these researches, the test case and test data generation is either semi-automatic or simple designed to just satisfy the whole framework work [5][6][7][8]. The test case and test data are not sufficient to check whether all specified parts specified

* This work is supported by National Post-doctor Science Foundation of China under Grant 2005038089.

by WSDL specification are implemented correctly and completely.

In general, a web service can produce unexpected behavior caused by errors common to sequential programs. Initialing parameters incorrectly will deviate its use by other operation. In this paper, by analyzing the fault type and produce adapted UIO (Unique Input Output) sequence of HPN (High-level Petri Net) referring to the UIO sequence of FSM that has been proven the fault coverage can exceed 99% [9], the actual error causing such an incorrect behavior can be detected. On the other hand, generating a wrong event might be not found because coincidental correctness, an incorrect program appears to be coincidentally correct if a intermediate state executed by an incorrect program is identical with a intermediate state executed by a correct program, a extent technique, constraints-based test data generation is adopted in this paper to meet the testing requirements.

Our test case generation strategy for web service testing is based on fault-coverage. By analyzing WSDL specification, web service is represented by a HPN, the algorithm computing adapted UIO for each state before the transition in HPN is designed to obtain high fault coverage. With user-defined constrains and policy constrains, the test data by analyzing the syntactic and semantic change will be generated to satisfy the sufficiency conditions. Finally, a Test Script Language (TSL) is proposed, test environment establishment, assignment of value to variables base on constrains, perform of necessary set and cleanup operation, test case of adapted UIO sequence deployment, and the result to check the correctness of execution are described in TSL.

The test method framework consists of above three techniques mainly. All the process can be executed automatically. A prototype system has been developed as a part of web service test system based on a HPN tool, poses++, in our lab.

The rest of this paper is organized as follows. Section 2 is the overview of the testing method. HPN generation from a WSDL specification is described in section 3. In section 4, we discussed the test case generation method based on adapted UIO sequence. Section 5 presents test data generation techniques, two constrains for test data generation is described. Test script language is given in section 6, followed by conclusion and future work in section 7.

2. Overview of The Testing Method

At first, two basic terms are imported here for late reference [10].

First, test case. A test case consists of

a) A combination of defined input conditions and,
b) A description of the expected output condition achieved by the execution of the tested operation under the input conditions mentioned in a).

Second, test data. By assigned value to the input parameter in tested operation, the input condition can be represented. Test input data of a tested operation are initial value corresponding to the defined input conditions of a test case. Also, a value can be interpreted as an output condition; Test output data of tested operation are final returning value (e.g. of a simulation run) corresponding to an output condition of a test case. Test data is a collection of test input data and test output data.

In short, test case describes the structure for testing an operation, and can be regarded as the control part definition of testing. Test data is the data information set. The loose combination of the control part and data part by the separation of control part and data part can improve the repeatable ability of test data and the flexibility of test case [11].

The relationship of the test case generation and test data generation can be shown in our testing method structure in figure 1.

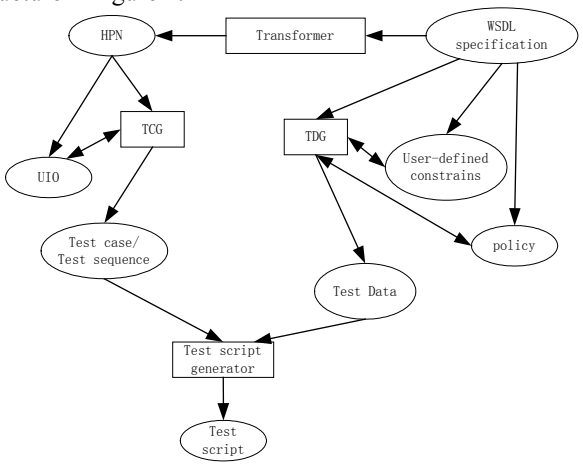

Figure1. Structure of testing method

The testing scenario is described as follows:

1) Translating an operation consists of input message and output message into a HPN. The input, output message should be de-compose based on their data type;

2) Using Test Case Generator (TCG) to generate test cases from the HPN. Here, the test case is represented by adapted UIO sequence derived from the HPN transition. These sequence contain the executions of send and receive events;

3) For each test cases, by analyzing WSDL specification, the constrains will be attached to the input and output messages, Test Data Generator (TDG) will generate test data satisfying the constrains;

4) Feeding the test case and test data into test script, the variables, values, constrains, input, output, and their relationship will be written in test script for test execution.

3. HPN Construction

A web service takes a set of input data, performs some operations, and outputs the excepted results. The operations are the unit execution of web service and will be our study object. In test case design, the first step to extract the necessary information is decomposing the specification. As have been mentioned, the function unit is operation. The specification of the service is first parsed into a DOM tree representation based on the WSDL schema. The DOM-tree will be represented by HPN for adapted UIO sequence generation to satisfy the fault-coverage.

High-level Petri nets are a widely used modeling and specification language for information system behavior since it has the ability to model concurrency of the systems, analyze concurrent behavior, and express the dynamically changed software [12][13]. The related research and tool for Petri net is popular and rather complete. They combine the advantages of a simple graphical notation with mathematical foundation. Moreover, they allow to model the behavior of an information system and its data structures in one integrated scheme [14]. Petri nets are directly executable by a net interpreter. Thus, Petri net simulation may be used for verification of web service. This can reduce effort in later stages of the web service system testing and development process.

Based on WSDL, two levels in operation will be analyzed in mapping procedure: message and part. The input message specifies the signature of an operation, and the output message specifies the expected testing results. Message is consists of parts. A part is mapped to a parameter of an operation. The mapping is a de-composite procedure from input and output to part, where operation is represented by transition, and input and output are represented by incoming arc and outgoing arc respectively.

Figure 2 illustrates the HPN for an operation. All parts of a input message will be presented by the place with token whose type specified by the part type used as the interface of test case generation, so do all parts of a output message. With multiple input and output, there will be a composition for all parts in the input and output messages of an operation. All parts of all input messages concatenated by "+" will be labeled in the incoming arc of corresponding transition. All parts of all output messages concatenated by "+" will be labeled in the outgoing arc of corresponding transition. An arc is used to link the transition with place linking to the input and output message consisting of those parts. The physical preconditions described in WSDL are embodied in the HPN by places. And the cause-effect analysis [10] is adopted in this mapping.

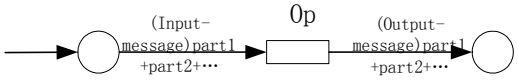

Figure 2. HPN for operation in web service

4. Test Case Generation Method

Fault-based adequacy criteria focus on the faults that could possibly be contained in the software. The adequacy of a test sequence is measured according to its ability to detect such faults. In [15][16], the use of unique input and output (UIO) test sequence is proposed for the purpose of testing a protocol. Improvements to the approach of [15] are proposed in [16][18] for optimizing the test procedure. Protocol testing has also been evaluated with respect to fault-coverage. A detailed study of formal methods with regard to protocol testing can be found in [17][19]: a comparative evaluation of various method [17] is carried out using extensive simulation. A further evaluation in estimation of fault coverage using the UIO method can be found in [19]. In [17][19], it has been shown that the UIO method can achieve a high fault coverage. WSDL specification is a kind of protocol, the UIO method for fault-coverage can be applied to WSDL-based test case generation. Because UIO sequence is designed for FSM (Finite State Machine), for applying UIO sequence to our test case generation based on HPN, UIO sequence is adapted in this paper. The objective of this section is to provide a comprehensive analysis of UIO sequence for WSDL-based testing.

A WSDL can be specified as a HPN, where $P=\{p_1, p_2, \ldots, p_n\}$ is a finite set of predicates before operation, and $T=\{t_1, t_2, \ldots, t_n\}$ is a finite set of transition representing the operation specified by WSDL specification. $I=\{i_1, i_2, \ldots, i_m\}$ is a finite set of inputs, and $O=\{o_1, o_2, \ldots, o_s\}$ is a finite set of outputs. The transition In HPN translating from WSDL specification based on the following mapping: $P \times I \rightarrow P \times O$. A HPN can be represented by a 3-tuple H={P, T, A}, where the set P represents the set of

specified predicates of the HPN, T, the set of specified transition, operation in WSDL, and A, arcs with label indicating input and output.

Now, the algorithm of test case sequence based on adapted UIO will be introduced [8].

For each transition (t_i , t_j ; i, o), construct the corresponding shortest adapted UIO sequence, where t_i is the initial transition t_j , t_j is the transition after the i/o transition. Assuming the tested system can be represented by a intermediate structure, a graph G={T, A} based on the HPN translated from WSDL specification depending on a HPN tool, where node T is state sets before a transition, we used the transition name to represent the state, and A is the arc set. Based on G, a symmetrical graph G^* ={ T, A^* }, at last, the test sequence will be generated by Euler Traverse. The steps are listed as follows:

1) computing the shortest adapted UIO sequence of each state in G;
2) if existing an Euler Route starting at t_i and ending at t_j , then the testing sequence is UIO(t_j)
3) transforming G to a symmetrical graph G^* ={ T, A^* }, where $A^* \subset$ A, that is, adding as fewer as possible arcs from A in G, transforming G to a symmetrical graph G^* and forming an Euler Route;
4) constructing Euler Traverse ET, where the initial node is the start of ET, ET is the testing sequence of final generation.

An example is presented to illustrate the construct process. A HPN corresponding to an ATM web service is shown in figure 3. The operation name is added in the input if two operations with the same incoming node have the same input parameters, e.g., the *deposit* and *withdraw* in the ATM web service, "deposit" and "withdraw" are integrated in input to distinguish the two states, as shown in figure 3.

A graph G transforming from above HPN can be shown as figure 4, where the parts of input and output are integrated in a single arc, and the predicates are omitted for adapted UIO sequence generation, as shown in figure 4.

For convenience, this paper defines:

State: t_1 :connect; t_2 : logon; t_3 : deposit; t_4 :withdraw; t_5 :logoff; t_6 :status; t_7 :disconnect.

Arc: a_1 :connectReauest/sestionID; a_2 :logonRequest/sestionID+customName; a_3 :Deposit+amount+customeName/banlance; a_4 :withdraw+amount+customeName/balance; a_5 :customName (from node *deposit*); a_6 :customName (from node *withdraw*); a_7 : sestionID (from node *logoff*); a_8 : sessionMessage/status; a_9 : sestionID (from node *status*).

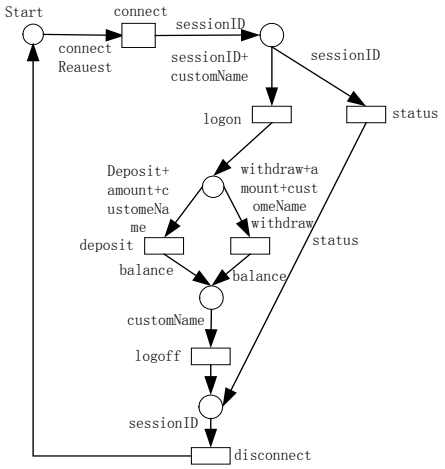

Figure 3. A HPN for an ATM web service

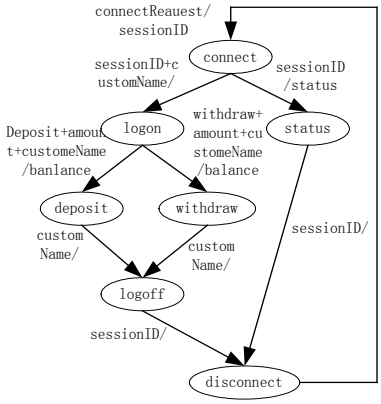

Figure 4. The graph G generated from a HPN of ATM

Based on the graph shown in figure 4, first, the shortest adapted UIO sequence and the end state t_j after UIO execution for each shown as figure 4, as shown in table 1.

t_i	UIO sequence	t_j
t_1	a_8	t_6
t_2	a_3	t_3
t_3	a_5	t_5
t_4	a_6	t_5

t_5	a_7	t_7
t_6	a_9	t_7
t_7	a_1	t_1

Table 1. the UIO sequence for graph G

Adding the necessary arc, forming a symmetrical graph G^* (figure 5)

The corresponding test sequence is:

$\{a_2, a_3, a_5, a_7, a_1, a_2, a_4, a_6, a_7, a_1, a_8, a_9\}$

Actually, test case can be generated automatically from our tool on a HPN tool, poses++. Because poses++ is based on C, the symmetrical graph can be generated automatically by a C program. The generated symmetrical graph G^* can be stored in memory and transforming to its equivalent HPN representation in the form of production rules. Afterwards, all the test case generation and test sequence based on that representation can be generated automatically. The transformation of WSDL to HPN should be done at the beginning of testing, so that the WSDL specification can be verified before test case generation. Therefore, during testing phase, we already have the suitable representation for test case generation.

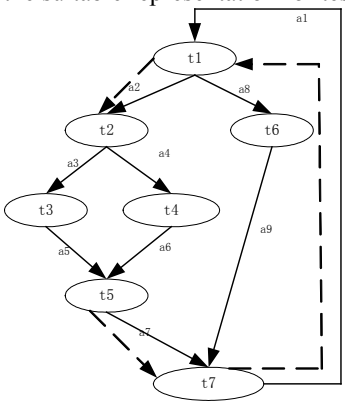

Figure 5. Symmetrical graph G^*

5. Constrains-based Test Data Generation

Our current constrains for test data generation consists of two parts: user-defined constrains and policy constrains. Based on white-box approach in structure testing, the user-defined constrains can meet the syntactic change effected by the mutation result in the message definition specified by WSDL specification [21]. The policy constrains is used to satisfy the sufficiency condition to verify the fault causing by data computation in semantic analysis for program predicates.

5.1. *User-defined constrains*

Based on Schema, WSDL provides the rich data type associating with common programming language. Besides build-in data types, e.g., string, integer, boolean, data, and time, The specification also provides the capability for defining new type. The build-in as well as user-defined data types to effective define and constrain XML document attributes and element values is supported. To constrain the value of defined component in application, the value/lexical spaces is adopted.

Generally, traditional mutation system takes account into the expressions, statements, and variables. But, they don't define mutant type declaration [22]. However, as a key feature of Schema, the user-defined type is an important part of Schema application in WSDL. It distinguishes those aspects of one datatype which differ from other datatypes. It is very likely that user-defined type but build-in types contain many defects such as missing content model, beyond value, occurrence change, etc. Nineteen mutation operators is designed to test related faults and classified them into 5 groups based on the relationship defined in Schema [23]: element, complexType, attribute, and identity constraint. The simpleType mutation is embodied in element and attribute mutation because simpleType is tightly related with the both.

Because of the space limit, we only give the example for Element Content Model Change (ECMC) here.

Element Content Model Change (ECMC)

The content model of element can be "empty", "any", "mixed", or "children". If element is empty, that is, the element has no content at all, the content model is "empty"; if there is no constraint on its content in any way, its content model is "any"; if character data may appear alongside sub-elements and is not confined to the deepest sub-elements, its content model is "mixed"; otherwise, its content model is "children". In WSDL testing, the element whose content model is "any" can not be checked because of its unconstrained. The original code:

```
<element name="mail">
<complexType>
<sequence>
<!—element definition→
</sequence>
<complexType>
</element>
ECMC mutant:
<xsd:element name="mail">
 <xsd:complexType>
  <!--attribute definition-->
```

```
    </xsd:complexType>
  </xsd:element>
```

5.2. Policy-based constrains

The second way to satisfy the sufficiency condition is designing policy to cause difference in implement predicates. [21] shows over 60% of the constrains involved predicates in a program. The test data generating from not involving predicates only killed 90% mutants while those involving predicates killed 65%. An example web service is an ATM service. It contains a operation *withdraw*, the control policy we need to specify is the following:

when a user withdraw a amount of money, the server will check the balance of the account, assuming the cash balance must greater than 10 $, if the balance-amount>10, then the transaction can go on.

Although the test case generated causes an effect based on above constrains and test case generation method for above operation. The result of the mutated predicate may be the same as that of the original program. As shown in figure 6, where a constant replace the variable balance with "1000", although the test case in figure 6 satisfy the user-defined constrains, balance is decimal and balance>amount, the result of the predicate test remains unchanged. balance-amount=100, 1000- amount=500, both of which are greater than 10.

Original program segment	balance-amount>10
mutated program segment	1000-amount>10
User-defined constrains	balance>amount &balance is decimal
Test case	Balance=600, amount=500

Figure 6. Constrains Problem

To resolve this problem, we constructs following inequality:

(100-amount>10) ≠ (balance-amount>10)

The above policy can be presented as follows.

```
<policy><rule><not-
equal><greater><substract><element1>1000</element
1><element2>amount</element2><substract/></greate
r><greater><substract><element1>balance</element1
><element2>amount</element2></substract></greater
></not-equal></rule></policy>
```

The test case satisfies above policy can be balance=2000, amount=999.

Other XML-based policy [24] can be added to web service testing to ensure its security, coordinate the partner relationship etc. Here, we just introduce a simple application of XML-based policy, and further research can improve the fault-coverage.

6. Test Script Language (TSL)

A simple test script language based on XML is provided in this paper to describe the test sequence, pre-condition, variable, constrains, and policy. Figure 7 shows the general form of the language [25][26].

```
<TESTSEQUENCE name="{service-name}">
[<!--{test sequence description}-->]
(<TESTCASE name="{operation-name}">
<!--{test case description}-->
<PRECONDITION>
[<!--{pre-condition description}-->]
<VARIABLES>
(<"{operator-name}">
(<{varialble-name}>{variable-value}
</{varialble-name}>)+
</"{operator-name}")*
</VARIABLES>
</PRECONDITION>
<INPUT>
[<!--{input description}-->]
<INPUT-MESSAGE name="{input-message-name}">
<CONSTRAINS>
[<!--{constraint description}-->]
{constrain}*
</CONSTRAINS>
<POLICYS>
[<!--{policy description}-->]
<VARIABLES>
(<"{operator-name}">
(<{varialble-name}>{variable-value}
</{varialble-name}>)+
</"{operator-name}")*
</VARIABLES>
</POLICYS>
(<PART name="{part-name}">
{part-value}
</PART>)*
</INPUT-MESSAGE>
</INPUT>
<OUTPUT>
[<!--{OUTPUT description}-->]
<OUTPUT-MESSAGE name="{output-message-name}">
<CONSTRAINS>
[<!--{constraint description}-->]
{constrain}*
</CONSTRAINS>
<POLICYS>
[<!--{policy description}-->]
<VARIABLES>
(<"{operator-name}">
(<{varialble-name}>{variable-value}
</{varialble-name}>)+
</"{operator-name}")*
</VARIABLES>
</POLICYS>
(<PART name="{part-name}">
{part-value}
</PART>)*
</OUTPUT-MESSAGE>
</OUTPUT>
<RESULT>
[<!--{result description}-->]
<CONSTRAINS>
[<!--{constraint description}-->]
{constrain}*
</CONSTRAINS>
<VARIABLES>
(<{varialble-name}>{variable-value}
</{varialble-name}>)+
</VARIABLES>
</RESULT>
</TESTCASE>)*
</TESTSEQUENCE>
```

Figure 7. General form of TSL

It contains a TESTSEQUENCE section consisting of a series of TESTCASE section. And each TESTCASE section consists of PRECONDITION, INPUT, OUTPUT, and RESULT. Upper case strings

are keywords. Square brackets ([]) represent optional item. Curly brace ({}) denote a conceptual construct that will be replaced by an actual instance value. The character "+", "*", and "?" has the same semantic as that described in []. The parentheses are used to group a set of sections.

The TESTSEQUENCE section provides the place for test sequence arrangement. The test sequence generation method is provided in section 4 in this paper. The test case will be described in this section in turn. The test sequence will be executed to meet fault-coverage requirement. Here, the test-sequence-name will be specified as the service name to denote the test object.

The TESTCASE section is used to describe an operation, with PRECONDITION denoting the pre-condition of the test case execution, the VARIABLES section is used to denote the variables and their value related the predicates, where OPERATOR section is the operators exerting on the variables, the operators and variables forms an operation tree corresponding combining condition, the same explanation for the VARIABLES section in POLICY section. In each INPUT section, the input message of the operation will be de-composed to identify the component parts whose information is stated in PART section. The OUTPUT section is designed for the output message of an operation that is similar with that of input message. Frequently, the values of each part interact with each other in specified way that will affect the test result. The user-defined constrains and policy are specified to decide this relationship, which is represented in CONSTRAIN section and POLICY section.

The RESULT section is designed to provide identifying the expected results that are produced by the operation execution. It also includes a CONSTRIANS section to denote the user-defined constrains on result variable.

In order to illustrate the method proposed in this paper, applying test case generation and test data generation theory to the ATM service described in figure 3, the test script segment for this web service can be shown in figure 8.

7. Conclusion and Future Work

This paper presents a systematic approach for web service testing based on fault-coverage that is intended to be used for service testing automation. In the method, HPNs representing operations are produced from WSDL specification after parsing process, and the HPNs will be integrated as a HPN to complete a service. A graph transforming from the HPN is used to generate the adapted UIO sequence, then test sequence based on the adapted UIO sequence is given to meet the fault-coverage requirements. This is mainly control flow testing aspect.

On the other hand, data flow testing aspect. Though the mutation analysis is an effective way to detect fault, its shortcoming is that it doesn't generate test data by itself. To resolve this problem, constraints-based test data generation for service testing getting sufficient test data to kill mutant program. Constrains are departed to two kinds: user-defined and policy based on the syntactic and semantic analysis for WSDL specification.

```
<TESTSEQUENCE name="ATM">
<!--automatic transaction machine-->
<TESTCASE name="connect">
<INPUT><INPUT-MESSAGE name="connectRequest"/>
<INPUT>
<OUTPUT><OUTPUT-MESSAGE name="sessionMessage">
<CONSTRAINS>
<!--the type of sessionID is changed to
integer from string-->
<element name="sessionID" type="integer" />
<PART name="sessionID">123</PART>
</OUTPUT-MESSAGE><OUTPUT>
<RESULT>
<CONSTRAINS>
<!--the type of connect is changed to
 integer from boolean-->
<element name="connect" type=integer"/>
</CONSTRAINS>
<VARIABLES>
<connected>true</connected>
</VARIABLES>
</RESULT>
</TESTCASE><TESTCASE name="logon">
<PRECONDITION>
<!--logon can be executed only after
 connect is successful-->
<VARIABLES>
<equal>
<element1>connect<element1>
<element2>true<element2>
</equal></VARIABLES></PRECONDITION>
...
</TESTCASE><TESTCASE name="deposit">
...</TESTCASE><TESTCASE name="logoff">
...</TESTCASE><TESTCASE name="disconnect">
...</TESTCASE><TESTCASE name="connect">...
</TESTCASE>
<TESTCASE name="withdarw">...
<policy>
<rule><not-equal><greater><substract><
element1>1000</element1><element2>
amount</element2><substract/></greater>
<greater><substract><element1>balance
</element1><element2>amount </element2>
</substract></greater></not-equal></rule>
</policy>
...</TESTCASE>
<TESTCASE name="disconnect">...</TESTCASE>
<TESTCASE name="connect">...</TESTCASE>
<TESTCASE name="status">...</TESTCASE>
<TESTCASE name="disconnect">...</TESTCASE>
</TESTSEQUENCE>
```

Figure 8. The test script for an ATM service

To effectively describe the test process, we applied a test script language based on XML. The test sequence and constrains for test data are expressed by this language. The test scenario is built on this formal language. The test script language is flexible in manipulating and user-friend. It is very useful for test process automation.

In fact, the whole testing process is automatic in our lab. After inputting the WSDL specification, the HPN construct, the UIO sequence analysis, the test sequence

generation, simple constrains design, and test data generation, at last the test script production can be automated by our prototype system based on a HPN tool supported by C language, poses++.

However, with the extraordinary growth of distributed yet coordinated service-oriented frame work that can be directly or dynamically composed of web services through workflow, the service composition becomes complex, their quality and reliability become crucial. As for future work, we plan to fully study the test of web service composition based on BPEL (Business Process Execution Language for web services) [27] and OWL-S (Semantic Markup for web services) [28] that provide the mechanism to integrate web service. Future research also includes the collaboration testing based on CPP/CPA (Collaboration-Protocol Profile and Agreement) [29] proposed by OASIS removes barriers to entry and eliminating the high cost and complexity of trading partner on-boarding.

References

[1] Web Services Description Language (WSDL 1.1) [s]. W3C Note, http://www.w3.org/TR/WSDL/, 15 March, 2001.

[2] Howard Foster, Sebastian Uchitel, Jeff Magee, Jeff Kramer. Model-based Verification of Web Service Compositions. Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE'03) (2003).

[3] Xiang fu, Tevik Bultan, Jianwen SU. Analysis of Interaction Web Services. WWW2004, pp. 621-630 May 17-22,2004. ACM.

[4] Srini Narayanan, Sheila A. Mcllraith. Simulation, Verification and Automated Composition of Web Services. WWW2002, pp. 77-88 May 17-22,2002. ACM.

[5] eTest, available at http://www.empirix.com.

[6] eValid, available at http://www.soft.com.

[7] SOAPtest, available at http://www.parasoft.com.

[8] WS-I, available at http://www.ws-i.org.

[9] C Ramanmoorthy, S Ho, W Chen. On the automated generation of program test data [J]. IEEE Transaction on Software Engineering, 1776, 2(4):293-300.

[10] Jorg Desel, Andres oberweis, Torsten Zimmer. Validation of Information system Models: Petri Nets and Test Case Generation. IEEE, 1997 : 3401-3406.

[11] Carl K. Chang, Cheng-Chung song and Rong-Fa wang. Distributed Software Testing with Specification. IEEE 1990: 112-117.

[12] J. L. Peterson, "Petri Net Theory and the Modeling of Systems", Englewood Cliffs, New Jersey, Prentice Hall Inc., 1981,

[13] J. Desel, T. Freytag, A. Oberweis and T. Zimmer, A partial-order-based simulation and validation approach for high-level Petri nets, to appear in: Proc. of the I f h IMACS World Congress on Scientific Computation, Modeling and Applied Mathematics, Berlin, Aug. 1997.

[14] A. Oberweis, An integrated approach for the specification of processes and related complex structured objects in business applications, Decision Support Systems, Vol. 17, 1996, pp. 31-53.

[15] K.K. Sabnani and A.T. Dahbura, "A protocol test generation proce- dure," Computer Networks, vol. 15, no. 4, pp. 285-297, 1988.

[16] A. V. Aho, A.T. Dahbura, D. Lee, and M. U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese Postman Tours," in Protocol Specification,Testing, and Verification, VII/, S. Aggarwal and K.K. Sabnani, Eds. New York: Elsevier North-Holland, 1988, pp. 75-86.

[17] D. Sidhu and T.-K. Leung, "Formal methods for protocol testing: A detailed study," IEEE Trans. Sop. Eng., vol. SE15, pp. 413-426, 1989.

[18] Y. -N. Shen, F. Lombardi, and A. T. Dahbura, "Protocol conformance testing by multiple UIO sequences,'' Proc. 9th Int. Symp. on Protocol Spec., Test. and Ver., Twente, June 1989, also in Protocol Spec., Testing and Verification /X, E. Brinksma, G. Scollo, and C.A. Vissers, Eds., Amsterdam: Elsevier Sci., 1990, pp. 131-143.

[19] A.T. Dahbura and K. Sabnani, "An experience in estimating fault coverage of a protocol test," Proc. IEEE INFOCOM, New Orleans, LA, 1988, pp. 71-79.

[20] Ma XiuFei, Gao Xiang, Mei Shaochun. Tow kinds of Generating Algorithms of Test Sequences Based On UIO Sequences and Their Comparison.Computer Engineering and Applications,2005(22),pp:76-80.

[21] Richard A. DeMillo, A. Jefferson Offutt. Constraint-Based Automatic Test Data Generation. IEEE Transactions on Software Engineering, 17(9):900--910, September 1991.

[22] Jian Bing Li, James Miller. Testing the Semantics of W3C XML Schema. IEEE 2005, COMPSAC'05.

[23] Wenli Dong, Luoming Meng. tML Schema Based ICS Proforma and Generation Method. The Chinese Journal of Electronics, 2005(4):681-685.

[24] Michiharu Kudo. XACML Policy Test Case: - Access Control Policy for XML Resource-.December 16, 2001

[25] Marc J. Baker, William M. Hasling, and Thomas J. Ostrand. Automatic Generation of Test Scripts from Formal Test Specifications. ACM 1998: 210-218.

[26] Aki Watanabe, Ken Sakamura. A Specification-Based Adaptive Test Case Generation Strategy for Open Operating System Standards. IEEE 1996, Proceedings of ICSE-18: 81-89.

[27] Business Process Execution Language for web services Version 1.1, 5 May 2003

[28] OWL-S: Semantic Markup for web services. W3C Member Submission 22 November 2004. http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

[29] Collaboration-Protocol Profile and Agreement Specification Version 2.0. OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee, September 23, 2002.