WSTester: Testing Web Service for Behavior Conformance

Bixin Li[1,2], Lili Yang[1], Shunhui Ji[1], Dong Qiu[1], and Xufang Gong[1]
[1]School of Computer Science and Engineering, Southeast University
Nanjing 210096, Jiangsu Province, P.R.China. Email: bx.li@seu.edu.cn
[2]Department of Computer Science and Engineering, University of California
Riverside, CA92521, USA. Email: lbxin@cs.ucr.edu

Abstract

In this paper, WSTester is introduced simply to show how to test behavior conformance between Web services based on interaction behavior specification and extended Labeled Transition System, corresponding experiment results analysis show the significance of the tool.

1 Introduction

In Web service times, it is necessary to realize the precise interaction and rapid integration of heterogeneous applications. Old independent point-to-point solution is unable to satisfy such new application requirement, it must be replaced with service-guided distributed computing architecture. SOA (Service-Oriented Architecture) is a new paradigm that can be used to satisfy user's current requirement. As a typical application case of SOA, Web service has won the wide support from academia and industries. Web service is a network component over open software platform, it supports interactive operation between different machines connected by the internet, it inherits the merit of XML language, it adapts and supports international open technology standards and specifications, where WSDL (Web Service Definition Language) is used to describe Web service, UDDI (Universal Description, Discovery, and Integration) is responsible for publishing and registering Web service in a *Register Center* so that it is easy for service requestor to find his wanted service, SOAP (Simple Object Access Protocol) protocol is used to bind and call it after a service is found [1].

Since some ideas of Web service are originated from object-orientation and component technology, they have some common features. However, the obvious difference between them is that a COTS component or an object is physically integrated into application system developed by the users, while only functions of Web services can be used in their application by remote calling, the real running of a service body is performed in the server located on the end of service provider. Those services interacting each other in an application are in fact distributed in different organizations or departments. This reason makes it more complicated and difficult to test the interactive behavior of Web service.

Next, WSTester is discussed about how to test behavior conformance from the user's viewpoint based on an extended Labeled Transition System called xLTS discussed in [2], Labeled Transition System in [3], and *interaction behavior specification* introduced in [2], which was enlightened by the idea in both [4] and [5].

2 WSTester

WSTester(Web Services Tester) is a conformance testing experimental tool, which integrates xLTS model and UML sequence diagram with OCL constraints. WSTester also provides a set of tools for supporting specification analysis, model transformation and model-based testing, it also provides an exchangeable format for integrating with other UML tool since its interface is based on XMI. WSTester has functional components for generating test case and executing test process automatically, where xLTS model which is transferred from UML model is the solid base for generating test case.

The flowchart of WSTester is described in Figure 1, which includes four parts:

- *Generate formal behavioral model*: based on UML 2.0 sequence diagram and OCL constraint, LTS is extended to be xLTS with semantic information, both data flow and control flow information can be captured in xLTS and more rich information can be provided by xLTS to generate test case.

- *Generate test sequences and test cases*: based on xLTS, enough and wanted test sequences and test cases are obtained.

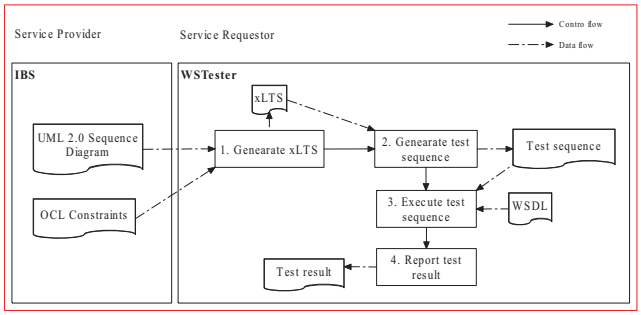

Figure 1. Flow chart of WSTester

SN...	Implemetation
1	correct implementation
2	when inputed quantity of goods is 0, quote of goods can be requested
3	quote information of a non-existence is requested
4	there is inconsistency between requested goods and returned goods
5	amount of requested goods is less than amount of returned goods
6	order form number generated automatically is inconsistent with ordering process
7	amount of requested goods is beyond biggest repertory, but it is ordered
8	order form numbers generated in different times are not identified
9	order is recored in system even it ihas been cancelled
10	payment amount is computed incorrectly

Figure 2. Different implementation of CSW

- *Execute test sequence*: use test cases to test service by remote calling and executing related services.
- *Output test result*: test report is output based on test process record and final test result.

WSTester has four functional components: *xLTS model transformer*, *Test case generator*, *Main tester* and *wsCaller*, their main functions are introduced as follows:

(1) xLTS model transformer will be used to transfer interaction behavior specification into xLTS so as to generate wanted test case to satisfy test requirement.

(2) Test case generator is used to generate test case based on xLTS.

(3) Dominant tester is the center modular for controlling test execution process that determines which test actions will be taken.

(4) wsCaller is used to call service to be tested.

3 CSW: an Illustration Example

Let's see a sample service CSW (Customer-Supplier-Warehouse), which is borrowed from [6]. In the CSW service, the expected behaviors will be described as follows:

(1) Customer sends Supplier a request message *requestQuote* for inquiring the goods about quote price information;

(2) Supplier returns a response message *requestQuote_r* to reply the request

(3) Customer accepts the quote price and sends a message *orderGoods* to Supplier for ordering the goods;

(4) Supplier will send a message *checkShipment* to Warehouse after he accepts the order form;

(5) Warehouse checks the repertory to check whether it is all right to consignment or not;

(6) Supplier will make different decisions according to the checking result to Warehouse:

- (6a) If it is no problem to shipment now, Supplier will send an acknowledgement message to Customer for telling him that the order form has been accepted, and Customer should do a `makePayment` operation to make a payment, then Supplier asks Warehouse to shipment, Warehouse sends Customer the message *getShipmentDetail* to ask Customer the shipment detail, Customer sends Warehouse the message *confirmShipment* to confirm this shipment, finally Warehouse sends Supplier a message to confirm the message *confirmShipment*.
- (6b) If it is impossible to shipment now, Supplier sends Customer a message *cancelOrder* to cancel order form.

4 Experiment Result Analysis

For validating xLTS based method, one kind of correct implementation and nine kinds of error implementations of sample example CSW are designed in Figure 2 to check the ability of our method and WSTester. For each implementation, we generate test case from LTS and xLTS model respectively and observe what differences will happen when each of them is used independently.

4.1 Evaluation factors

Two important factors needed to be checked to determine the ability of a test method: *error-checking capability* and *test expensive*. Error-checking capability is usually measured using *test coverage rate*(or TCR), while test expensive (or TE) is measured by the *length of test sequence*(or LOT). Our test goal is to perform a test with smallest test expensive and strongest test capability. However the two aspects are usually contradictory each other, so we pursuit a strongest test capability when the contradiction can not be solved. In our method, test coverage rate is computed using the number of checked out errors (NCE) and the number of total errors (NTE):

$$TCR = \frac{NCE}{NTE}$$

$$TE = LOT$$

SN	Test case	LOT
1	$\underline{m}$; pass[] otherwise; fail	0
2	?requestQuote;(!requestQuote; ($\underline{m}$; pass[] otherwise;fail)[]otherwise;fail))	2
3	?requestQuote;!requestQuote;?orderGoods;(!confirmOrder;($\underline{m}$; pass[]otherwise; fail) []otherwise;fail)	4
4	?requestQuote;!requestQuote;?orderGoods;(!cancelOrder;($\underline{m}$; pass[]otherwise; fail) []otherwise;fail)	4
5	?requestQuote;!requestQuote;?orderGoods!confirmOrder;?makePayment;($\underline{m}$; pass[]otherwise;fail)	5

Figure 3. LTS-based test case

SN	Test results	
	TCR (%)	Test conclusion
1	0.0	Pass
2	0.0	Pass
3	60.0	Fail
4	80.0	Fail
5	0.0	Pass
6	60.0	Fail
7	60.0	Fail
8	0.0	Pass
9	0.0	Pass
10	20.0	Fail

Figure 4. Test conclusion

4.2 LTS-based test

At first, let's see what will happen when we use LTS-based test method proposed by Jiang [3], supposing the biggest loop times is 1, then five test cases are generated from xLTS in Figure 3, where $\underline{m}$ represents empty message θ. The five test cases are used to test 10 kinds of different implementations of CSW service, and the test result is given in Figure 4.

We can see from Figure 4, only five kinds of error implementations of CSW service have been discovered, which is caused by the construction of test execution trace with sequence dependence relation. Because control-flow information is considered in LTS-based method, it is easy to check out the error produced in sequence operation. However, for LTS-based method, it is needed to add data into test case manually, which limits the test capability of LTS based method, added data will affect directly whether more errors can be checked out or not. The reason for both the 2nd and 5th error haven't been checked out is that LTS is short of related data-flow information. It is needed to declare that it is our limitation to loop times, which causes both 8th and 9th error haven't been checked out.

SN	Test case	LOT
1	?requestQuote<prodA,500>!requestQuote<1,prodA,490,10.0,4900> ?OrderGoods<1,400>!confirmOrder<1>?makePayment<1,4000>	5
2	?requestQuote<prodA,0>!quiescence	2
3	?requestQuote<sun,400>!quiescence	2
4	?requestQuote<prodA,1>!requestQuote<1,prodA,400, 10.0,4000>	2
5	?requestQuote<prodA,500>!requestQuote<1,prodA,490,10.0,4900> ?OrderGoods<1,400>!confirmOrder<1>?makePatment<2,4000>	5
6	?requestQuote<prodA,500>!requestQuote<1,prodA,490,10.0,4900> ?OrderGoods<1,500>!quiescence	4
7	?requestQuote<prodA,500>!requestQuote<1,prodA,490,10.0, 4900 >?OrderGoods<1,400>!confirmOrder<1>?makePayment<1,4000>?requestQuote<prodB,500>!requestQuote<2,prodB,300,20.0,6000>? OrderGoods<1,300>!confirmOrder<1>?makePayment<2,6000>	10
8	?requestQuote<prodA,500>!requestQuote<1,prodA,490,10.0,4900> ?OrderGoods<1,400>?cancelOrder<1>?requestQuote<prodA,500>!requestQuote<2,prodA,490,10.0,4900>	6
9	?requestQuote<prodA,500>!requestQuote<1,prodA,490,10.0,4900> ?OrderGoods<1,400>!confirmOrder<1>?makePayment<1,4000>	5

Figure 5. xLTS-based test case

SN	Real results	
	TCR (%)	Test conclusion
1	0.0	Pass
2	11.1	Fail
3	11.1	Fail
4	77.8	Fail
5	77.8	Fail
6	44.4	Fail
7	11.1	Fail
8	11.1	Fail
9	11.1	Fail
10	44.4	Fail

Figure 6. Test conclusion

4.3 xLTS-based test

Now we will observe what will happen when we use our xLTS-based test method. Test cases are listed in Figure 5, where test sequences with different lengths are adopted, and requests with different amounts for requesting quote are input.

As we can see from Figure 6, all the errors included in the nine kinds of incorrect implementation of CSW service have been checked out. But the discovering of most of them is based on the construction of sequence dependence test execution trace, such as 6th, 8th and 9th errors. It is very hard to check out them to use current methods based on single service operation. In error checking, the check capability of a tool is affected by length of test trace obviously. In above example, some errors are easy to find by checking quote information of goods. For example, the error of inconsistency between requested goods and returned goods is checked out easily by checking a trace with length equaling to 2: observing the response after the quote information is requested.

However, there are a lot of errors that can not be found so

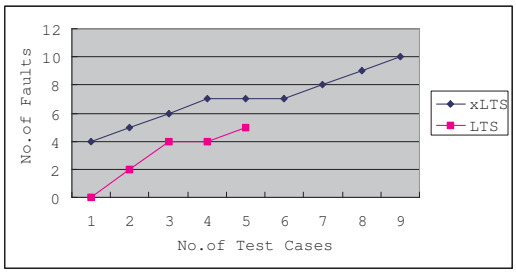

Figure 7. Test comparision

easily, they can only be found in sequence execution process of multiple operations. For example, to check out the error that the numbers of two order forms of goods are the same for twice successful advance booking in different time, it is needed to run two complete advance booking process where the length of trace is 10. This shows, for the error existing in sequence operation process, the longer the test trace is, the stronger test capability is, but the test expensive for generating test case is increasing accordingly. To overcome this shortcomings, we borrow on-the-fly test strategy, i.e., test action is accompanying with test case generation, stopping the test process as soon as the error is found, instead of starting test after all of test cases have been generated, which ignores the space explosion problem caused by model-driven approach and reduces test expensive.

We can see from Figure 7 that the error checking coverage rate of xLTS-based method is higher than that of LTS-based method, the main reason is that data-flow information has been included in former method. But it is needed to point that coverage rate 100% doesn't mean that our method has 100% coverage for any test. It is no meaning for coverage rate itself, but it can be used as a reference data when we do a compare between two methods.

5 Conclusion

There are some people who are doing research on behavior conformance testing, such as [5], [7] and [8]. Enlightened by the main idea in these related work, both *interaction behavior specification* and xLTS are introduced to generate test case for testing behavior conformance in this paper and its earlier version [2]. However, there are still some shortcomings with our method, which encourages us to do further work in our future time. Two representative issues are summarized as follows: (1) the inconsistency caused by synthesizing xLTS from sequence diagrams must be checked and solved, because the xLTS model generated automatically from specification is just an approximation of system, the prototype tool must support user's manual modification; (2) the way for combining xLTS model from many different sequence diagrams must be considered in next work. Because different sequence diagrams may have some same or similar behaviors, how to combine and confirm these behaviors is a challenge problem.

Acknowledgements

The authors thank Prof. Rajiv Gupta in University of California Riverside for providing a very comfortable Lab. This work is partially supported by the National Nature Science Foundation of China under No.60773105, partially by the Natural Science Foundation of Jiangsu Province of China under Grant No.BK2007513, and partially by National High Technology Research and Development Program under Grant No. 2008AA01Z113.

References

[1] W3C. *Web Services Activity*. http: // www. w3. org / 2002 / ws /.

[2] B. Li, X. Fan, and L. Yang. *Extending Labeled Transition Systems for Conformance Testing of Web Services*. Technical Report, Southeast University, 2009.

[3] F. Jiang, Z. Ning. *Automatic Test Case Generation Based on Labeled Transition System*. Chinese Journal of Computer Research and Development, 2001, vol 38, no. 12.

[4] S. Pickin, C. Jard, T. Jeron, J. Jezequel, and Y. Traon. *Test Synthesis from UML Models of Distributed Software*. IEEE Transaction on software engineering, April 2007, vol 33, no.4.

[5] E. Cartaxo, F. Neto, and P. Machado. *Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems*. In: Proceedings of IEEE International Conference on Systems, Man and Cybernetics,7-10 Oct. 2007.

[6] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer Verlag, 2004. ISBN 3-540-44008.

[7] J. Tretmans. *Conformance testing with labeled transition systems: Implementation relations and test generation*. Computer Networks and ISDN Systems, 1996, 29:49-79.

[8] R. Heckel, L. Mariani. *Automatic conformance testing of web services*. In: Proceedings of FASE, Edinburgh, Scotland, Apr., 2005, 2-10.