Observability and Controllability Issues in Conformance Testing of Web Service Compositions

Jose Pablo Escobedo[1], Christophe Gaston[2], Pascale Le Gall[3] and Ana Cavalli[1]

[1] TELECOM & Management SudParis (ex INT) - CNRS SAMOVAR
9 rue Charles Fourier, 91011 Évry, France
email: {jose.escobedo, ana.cavalli}@it-sudparis.eu
[2] CEA LIST, Point Courrier 94, 91191, Gif-sur-Yvette, France
email: christophe.gaston@cea.fr
[3] Programme d'Épigénomique Université d'Évry-Val d'Essonne, 91000 Évry, France
Laboratoire MAS, Grande Voie des Vignes, 92195 Châtenay-Malabry, France
email: pascale.legall@ecp.fr

Abstract. We propose a model-based black-box testing approach to test conformance of Web Service Compositions (WSC). When a WSC under test makes use of implementations of Web Services, two situations may occur: either communications between the WSC and the Web Services are observable or hidden internal actions. We show by means of an example how to generate test cases whose verdicts are provided with explanations taking into account the status of the Web Services.
Keywords: Conformance testing, Web Service composition, observability and controllability, verdict testing report.

1 Introduction

Web Services (WS) are available through the network to be accessed by any computer that wants to communicate with them. *Web Services Compositions* (WSC) are mechanisms to produce new services by making use of already existing WSs. In this paper we address the problem of testing such kind of WSC following a model-based approach like in [4], using Input/Output Symbolic Transition System (IOSTS) to represent WSC specifications. Contrarily to systems considered in [4], the system under test may include not only the WSC system, but also some Web Services interacting with it. As the approach described in [1], the most natural solution to overcome that problem is to reduce it to the case of unit testing case by isolating WSCs from their Web Services so that the tester may directly interact with WSCs. Communication channels with Web Services are then qualified as **controllable**. However, it may occur that channels used to communicate with Web Services may technically be uncontrollable by the tester. In such cases, two situations may occur: either channels are **observable**, which means that the tester may observe exchanges of values through them, or they are **hidden**, and in that case, exchanges of messages through them can

only be considered as internal actions. The paper presents a model-based testing approach for testing WSC implementations in the context of a system also containing some Web Services.

The paper is structured as follows. The test architecture and a running example are presented in Section 2. In Section 3, we present the key ideas of our approach by illustrating it on the example.

2 The context: Test architecture

WS-BPEL stands for Web Services Business Process Execution Language [3], and is a standard for specifying Web Services Compositions in the form of orchestrations (or choreographies), where a central entity guides the composition. Basically, it defines how the interaction between Web Services is coordinated in order to achieve a business goal.

Let us introduce the Loan Request example depicted in Fig. 1. Three partners interact in the process: the user (bank's client), the WS-BPEL process (LR-WSC) and the Loan Service (LS, the Web Service). The behavior of the Loan Request WS-BPEL is as follows. The client initiates the process by sending his ID, the amount of money he wants to borrow and the maximal reimbursement duration. The LR-WSC sends this information to the LS, which first decides whether the loan is approved or not. If approved, it returns the minimum duration and the monthly fee and then, the LR-WSC sends the monthly fee to the user. Otherwise, the process ends and the user gets a `Good bye` reply message. If the duration computed by the bank and the one proposed by the client are the same, the loan is approved by sending a `Good bye` message to the user. Otherwise, if the duration proposed by the client is greater than the duration computed by the LS, the client can try again by proposing a new value for the duration until the loan is approved or until he decides to finish the process.

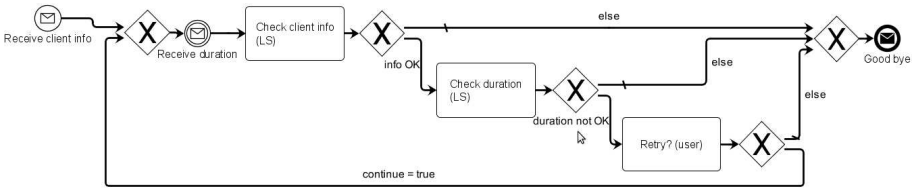

Fig. 1. Diagram of the Loan Request (LR) example.

Following the classification given in the Introduction, a communication channel between the WSC and a WS is said to be (see Figure 2 for an illustration):

Controllable, if communications over the channel are controllable by the tester through control points **CP**. The tester has a complete control over the channel, both for stimulating the System Under Test (SUT) implementing the WSC with inputs and for observing its outputs. If all communication channels are controlled, the WSC becomes an independent component which can be tested according to a classical unit testing framework.

Hidden, if the tester cannot access the communication channel. This is the usual case when using the implementations of Web Services. The SUT is not just the WSC in isolation but a system composed of the WSC providing an interface to the user and of some encapsulated Web Services.

Observable, if the tester can only observe the communication channel through observation points **OP**.

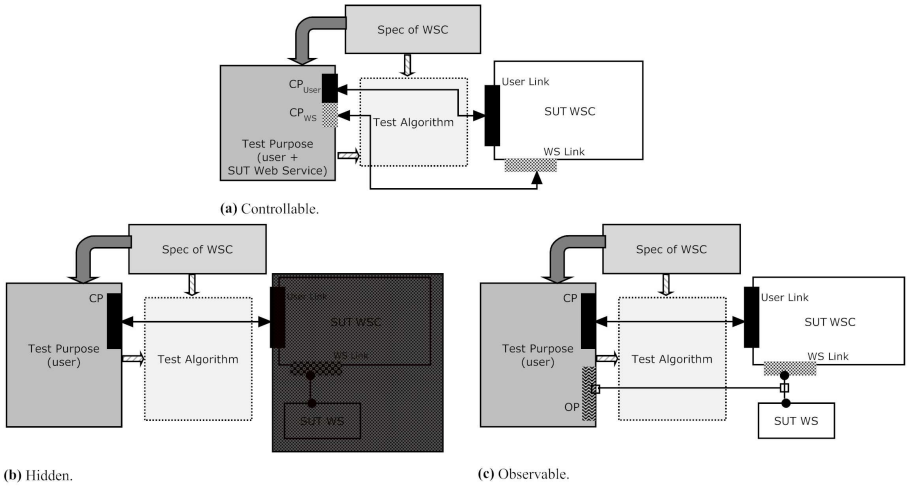

Fig. 2. Testing scenarios according to the accessibility to the communication channels.

Figure 3 shows the Loan Request example written as an IOSTS [4]. The user is represented by the letter u. IOSTS contain attribute variables useful to store data values. In the case of the LR-WSC, attribute variables are: d, for the (maximal) duration proposed by the client to pay the loan; a, for the amount of money the user wants to borrow; id, for the client's ID; b, for the boolean variable indicating if the user wants to try (if possible) to get another loan proposition with a smaller duration; min, for the minimal duration computed by the LS for the reimbursement period allowed; $approve$, for the boolean variable used by LS to indicate if the duration proposed by the client is satisfactory or not; and mf, for the monthly fee computed by the LS. Channels are represented, for readability reasons, as *EntityName_ChannelName_OperationName*. The initial state is q_0 and the final state is q_9.

Transitions of IOSTS are defined by a guard, an action (reception, emission or internal action) and assignments of attribute variables. Transition $q_0 \rightarrow q_1$ represents the initialization of the WS-BPEL process (LR-WSC). Transitions $q_1 \rightarrow q_2$, $q_2 \rightarrow q_3$ represent the first communication between the LR-WSC and the LS to check if the information given by the client is correct. Transition $q_4 \rightarrow q_5$ represents a positive answer from LS, and transition $q_5 \rightarrow q_6$ represents the corresponding following information sent by the LR-WSC to the user informing him about the monthly fees. Transition $q_4 \rightarrow q_9$ represents a negative answer from LS and the corresponding `Good bye` message sent to the user. Transition

$q_6 \rightarrow q_9$ represents the approval of the loan request and the corresponding `Good bye` message notifying the user. Transitions $q_6 \rightarrow q_7$, $q_7 \rightarrow q_8$ $q_8 \rightarrow q_2 \cdots$, $q_8 \rightarrow q_9$ represent respectively, the case where, if $min \neq m$, the user is asked to propose a new value for the duration of the reimbursement and start the process again or to finish the loan request process.

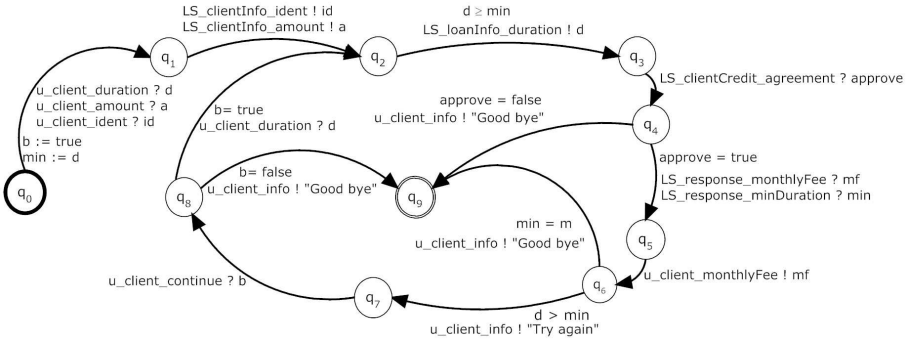

Fig. 3. The Loan Request example expressed as an IOSTS.

3 Our approach of Conformance testing for WSC

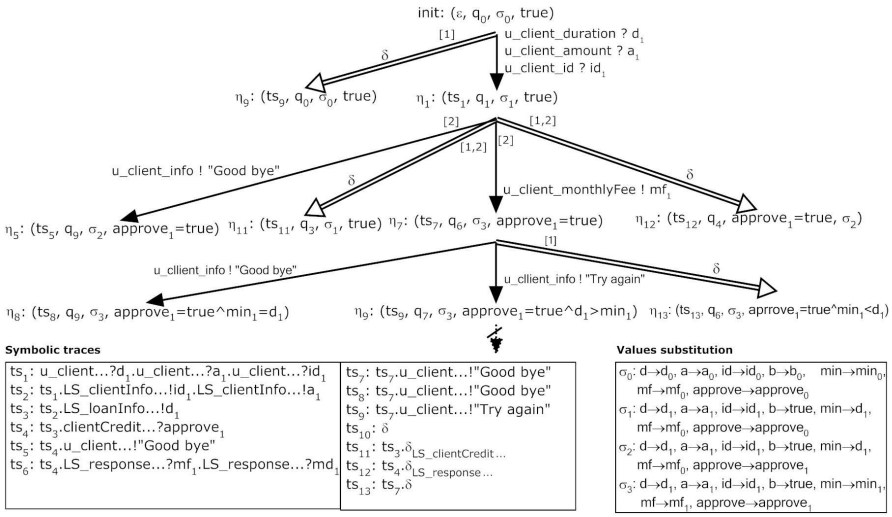

Fig. 4. Observable behaviors of LR-WSC, assuming that LS is hidden. Transitions labeled by [1] (resp. [2]) are introduced by applying *enrichment by quiescence* (resp. by *hiding transitions on hidden channels* and *internal actions reduction*).

The first step consists in symbolically executing the IOSTS representing the WSC under test. The technique uses symbols instead of concrete data and allows us to represent all behaviors in a symbolic execution tree. The tree is then modified to take into account observable and hidden communication channels. The symbolic tree is first enriched by transitions carrying on the quiescence action δ and indicating that the WSC is waiting for a request either from the user or from a WS on an hidden or observable channel. Then, WS requests on observable channels are transformed into special observations (denoted as outputs). Finally, all transitions on hidden channels are translated into internal actions and the resulting symbolic tree is reduced by removing all internal actions which are useless from a testing point of view. Fig. 4 shows the dedicated symbolic execution tree of the LR example with the assumption that the LS's communication channels are all *hidden*.

With the dedicated symbolic execution tree, we adapt the rule-based algorithm of test case generation presented in [4] and defined for the classical ioco conformance relation. In few words, the algorithm consists in a simultaneous traversal of the modified execution tree and a test purpose defined as selected finite paths of the tree, in order to stimulate and observe the SUT in such a way that, as long as possible, the sequence of stimulations and observations corresponds to a trace of the test purpose. In practice, each path of the tree beginning in the initial state and ending in the final one can be used as a test purpose. Each algorithm step involves the computation of constraints issued from the values of messages exchanged in the beginning of the test sequence and the constraints stored in the symbolic states of the tree and of the test purpose. This process is repeated until a verdict is emitted together with a testing report illustrating the trace (exchange of values) and explaining which conditions lead to that verdict.

(a)

Action	Values
stim	u_client_duration→12 u_client_amount→1000 u_client_id→42
obs	δ
Verdict	INCONC, {(u_client…?d_1.u_client…?a_1.u_client…?id_1.LS_clientInfo…!id_1.LS_clientInfo…!a_1. LS_loanInfo…!d_1.LS_clientCredit…?$approve_1$.δ, d=12^a=1000^id=12^b=true^min=12^mf=mf_0^approve=true), (u_client…?d_1.u_client…?a_1.u_client…?id_1.LS_clientInfo…!id_1.LS_clientInfo…!a_1. LS_loanInfo…!d_1.δ,d=12^a=1000^id=12^b=true^min=12 ^mf=0^approve=$approve_0$)}

(b)

Action	Values
stim	u_client_duration→12 u_client_amount→1000 u_client_id→42
obs	u_client_monthlyFee→100
obs	u_client_info→"Good bye"
Verdict	PASS, {(u_client…?d_1.u_client…?a_1.u_client…?id_1.LS_clientInfo…!id_1.LS_clientInfo…!a_1. LS_loanInfo…!d_1.LS_clientCredit…?$approve_1$. LS_response…?mf_1.LS_response…?min_1. u_client…!mf_1.u_client…!"Good bye".,d=12^a=1000^id=12^b=true^min=12^mf=100 ^approve=true)}

Fig. 5. Verdicts for hidden channels.

Fig. 5 shows results when the algorihm is performed with the test purpose characterised as the restriction of the tree in Fig. 4 with η_8 as the (accept)

state. In case *(a)*, the tester sends the stimulation $u\_client\_duration \rightarrow 12$, $u\_client\_amount \rightarrow 1000$, $u\_client\_id \rightarrow 42$ and observes δ. The observation belongs to a trace of the execution tree but not of the test purpose. So, we obtain the *inconclusive* verdict. The report produces the two scenarios as shown in Fig. 6: (1) the LS does not send the *approve* answer or, (2) the LS sends the approve answer, but does not send the monthly fee and minimum duration. In case *(b)*, the tester stimulates the SUT with the sequence $u\_client\_duration \rightarrow 12$, $u\_client\_amount \rightarrow 1000$, $u\_client\_id \rightarrow 42$ and observes the monthly fee proposed by the LR-WSC, followed by the message *Good bye*. The algorithm produces the verdict *pass* with its corresponding scenario report (Fig. 5). In [2], more examples are available.

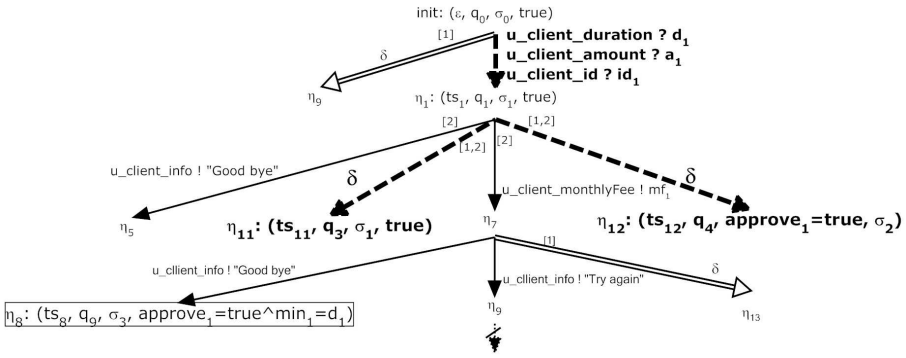

Fig. 6. Two possible scenarios when observing δ after sending the client information.

In this article, we have shown an example of how to test a WSC (orchestration) from its specification taking into account that WSs are connected to it during the testing phase. This corresponds to a situation of *test in context* or *embedded testing* for which no knowledge is available of the rest of the system (made of WSs invoked by the WSC). The complete formal description can be found in [2].

References

1. L. Bentakouk, P. Poizat, and F. Zaïdi. A Formal Framework for Service Orchestration Testing based on Symbolic Transition Systems. In *International Conference TESTCOM/FATES 2009*, Eindhoven, the Netherlands, 2009. Long version as technical report of LRI, University of Paris-Sud.
2. J. Escobedo, P. Le Gall, C. Gaston, and A. Cavalli. Examples of testing scenarios for web service composition. Technical Report 09003_LOR, TELECOM & Management SudParis, http://www.it-sudparis.eu/, 22 pages, 2009.
3. A. Alves et al. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, April 2007.
4. C. Gaston, P. Le Gall, N. Rapin, and A. Touil. Symbolic Execution Techniques for Test Purpose Definition. In *Proc. of the 18th Int. Conference TestCom 2006*, volume 3964 of *LNCS*, pages 1–18. Springer-Verlag, 2006.