Measuring similarity of service interfaces

Ali Aït-Bachir*

Supervised by Pr. Marie-Christine Fauvet
University of Grenoble, LIG (MRIM)
385 rue de la bibliotheque – B.P. 53
38041 Grenoble Cedex 9, France

Abstract. In this paper, we present a similarity measure between behavioural interfaces of Web services. This measure computes the difference value of simulation between two service interfaces. In our previous work we implemented an algorithm to detect the exact location of differences between service interfaces in a tool namely BESERIAL [1]. The similarity measure is based on the results of the detection algorithm. In our case study, this measure is used to select the most suitable service to substitute a previous one, which is no longer available at design time.

1 Introduction

Web service interfaces can be described from two aspects: i) The structural aspect models the provided operations, and the schema of the messages that the service can send and receive. These operations can be described by using WSDL for instance. ii) The behavioural aspect refers to the control flow between the operations and establishes their inter-dependencies. In conversational services, such behavioural interfaces can be described using BPEL for instance. Nevertheless, Finite State Machines (FSM) is the formal model adapted in our work to describe behavioural interfaces [7]. In this paper, we do not consider semantic aspects of operation definitions.

Client applications are meant to consume provided operations in a service interface. Conversations between a client application and a service are loosly coupled. Thus, if the service evolves and provides a new interface, then incompatibilities may arise as a client application does no longer match the new interface. The provided interface of a service evolves from a previous definition to a new one by means of basic differences (*addition, deletion* and *modification* of operations). The exact location of these differences can be detected and resolved instead of programing a new client application whose required interface is compatible with the new interface definition. However, if the service is no more available, there is no choice left to developers than to substitute this service by another one, at design time. If there exists no service whose interface simulates the old service interface, it is interesting to discover another service whose interface has a minimum number of differences with the previous one.

* This author is partially funded by the Web Intelligence project granted by the French Rhône-Alpes Region

This paper is structured as follows. First, Section 2 introduces the running example. Section 3 gives details on the quantitative simulation measure. Section 4 shows some experimental results. Then, Section 5 gives a panel of the related work on the diagnosis of differences in service interfaces. Finally, Section 6 concludes and sketches the future work.

2 Case study

As a running example, we consider a scenario where a car factory interacts with one of its provider of goods and services. The service provider describes its operations in WSDL and the control flow is established using BPEL process protocol. Figure 1 (a) illustrates the activity diagram of the provided interface of the provider service. This provider processes service and goods orders from the car factory. The provider receives a service order which can be updated by its client (the car factory) only if the invoice is not sent yet (see the flow which loops back to the *ReceiveServiceOrder* activity). Once the service invoice is sent, the provider waits for the transfer from the client to finally send him the *ShipmentTrackingNumber* (STN). On the other hand, when a *GoodsOrder* is received, a *GoodsInvoice* is immediately sent to the client. This former can either send his *CreditCardDetails*, to pay the invoice, or update his order by sending a new *GoodsOrder* (see the flow which loops back to the *ReceiveGoodsOrder* activity). The client pays the invoice, and then the provider sends him the *STN*.

If the service provider is no more available, the car factory will send an invitation to tender to substitute the old provider and all candidates will provide their behavioural interfaces. The selection criterion is that the provided interface of the new partner must conform as much as possible to the required interface by the car company. In other words, the new provider is such as there exists a minimum number of changes in the new provided interface in order to simulate the old provided interface.

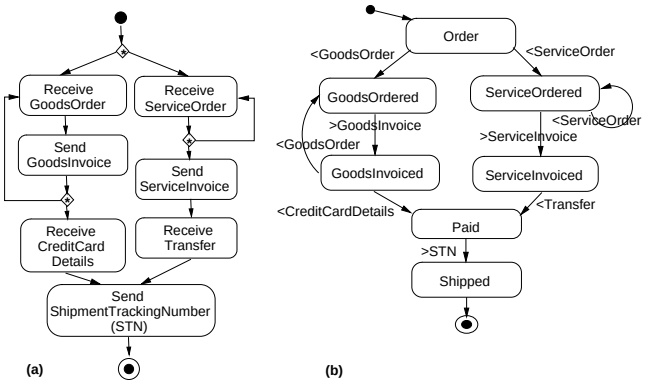

Fig. 1. Activity diagram and FSM of the provided interface.

3 Quantitative simulation

FSM modeling: In our approach we model the behaviour of a Web service interface using *Finite State Machines [6]*. Techniques exist to transform behavioral service interfaces defined in other languages (e.g. BPEL) into FSMs(see for example the WS-Engineer tool [3]). In the FSMs considered in this paper, transitions are labelled with messages to be exchanged. When a message is sent or received, the corresponding transition is fired.

An FSM is a tuple (S, L, T, s_0, F) where: S is a finite set of states, L a set of events (actions), T the transition function ($T : S \times L \longrightarrow S$). s_0 is the initial state such as $s_0 \in S$, and F the set of final states such as $F \subset S$. The transition function T associates a source state $s_1 \in S$ and an event $l_1 \in L$ to a target state $s_2 \in S$. In this model, a transition is defined as a tuple containing a source state, a label and a target state.

Figure 1 (b) illustrates the FSM of the running example which describes the behavioural interface of the service provider. We only consider the observable behaviour of a service, thus internal activities are hidden. Activities meant to send and to receive messages are modeled. The message m is denoted by $>m$ (respectively $<m$) when it is sent (respectively received). Each conversation initiated by a client starts an execution of the corresponding FSM.

We use the following notations (examples refer to the FSM depicted in the right side of the Figure 1(b)):

- $s\bullet$ is the set of outgoing transitions from s.
  (e.g. GoodsInvoiced$\bullet$ = {(GoodsInvoiced,<CreditCardDetails,Paid),
  (GoodsInvoiced, <GoodsOrder,GoodsOrdered) }).
- Label(t) is the label[1] of the transition t.
  (e.g. Label((GoodsInvoiced,<CreditCardDetails,Paid))=<CreditCardDetails)
- The *Label* operator is generalised to a set of transitions. For example, if $T = \bigcup_{i=1}^{n}\{t_i\}$ then $Label(T) = \bigcup_{i=1}^{n}\{Label(t_i)\}$; where $n =\parallel T \parallel$.
- $\parallel X \parallel$ is the cardinality of the set X.

In our previous work, we implemented an algorithm which is meant to detect the exact location of changes while comparing two FSMs P and P' (which respectively models the old provider and the new provider interfaces). A difference is detected if and only if the new interface does not simulate the behaviour of the previous interface. The outcome is a set Res of tuples (si, ti, sj, tj) where si and sj are states of P and P' respectively, while ti and tj are either null values or outgoing transitions of si and sj respectively.

Figure 2 shows three differences between P and P'. The first difference is a deletion of the operation $<$ *ServiceOrder*, which means that the new provider does not allow its client to update its service order. This difference causes an incompatibility with the required interface of the client as he can not use this operation any more. The second difference is an addition of the operation $<$ *Transfer*.

[1] In deterministic FSMs, $\forall t_1 \in s\bullet, t_2 \in s\bullet : Label(t_1) \neq Label(t_2)$.

However, this difference does not cause any incompatibility as the added operation provides a new option to its client. The third difference is the modification of the operation $>$ *STN* by the operation $>$ *ASN* (*Advanced Shipment Notice*). An incompatibility will arise because the client can not recognize this new operation.

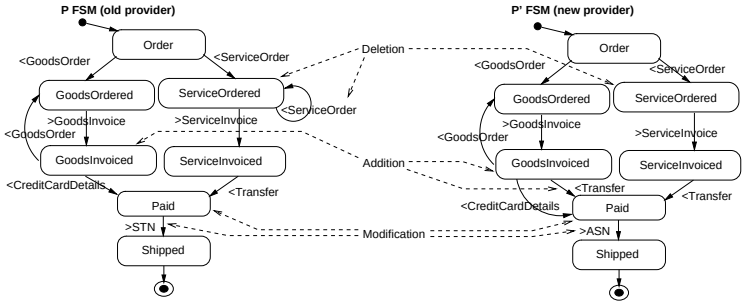

Fig. 2. Differences between the old and the new provider FSMs.

Quantitative simulation: In the detection algorithm, P and P' are traversed in parallel. A set of reached pair of states *Rps* is built such as $Rps \subseteq S \times S'$, where S is a set of P's states and S' is a set of P''s states. For each pair of states $(si, sj) \in Rps$, we compute a quantitative simulation function *Qs*. This function returns a score of differences between si's outgoing transitions and sj's outgoing transitions. $Qs : S \times S' \rightarrow [0..1]$ is defined as follows:

$$Qs((si, sj)) = \begin{cases} 1 & if, si\bullet = \{\} \\ \frac{\sum_{i=1}^{\| Diff((si,sj)) \|} Weight(D_i) + \| Label(si\bullet) \cap Label(sj\bullet) \|}{\| Diff((si,sj)) \| + \| Label(si\bullet) \cap Label(sj\bullet) \|} & otherwise \end{cases} \quad (1)$$

Where: $Diff((si, sj))$ is a set of differences pinpointed at the state pair (si, sj) such as $Diff((si, sj)) \subseteq Res$, and $D_i \in Diff((si, sj))$ for $i = 1.. \parallel Diff((si, sj)) \parallel$. The function *Weight* returns a penalty value[2] for each type of difference, and $0 \leqslant Weight(D_i) < 1$. The sum of all penalties in the state pair is added to the score of the common labels of the outgoing transitions. Common labels of the outgoing transitions of si and sj refer to the case where no difference is detected. Thus, a highest score is attributed (see (1): $\parallel Label(si\bullet) \cap Label(sj\bullet) \parallel$). To compute the quantitative simulation of the pair state, the sum of difference score and similarity score is divided by the number of these differences and similarities between the outgoing transitions of si and sj (see (1): $\parallel Diff((si, sj)) \parallel + \parallel Label(si\bullet) \cap Label(sj\bullet) \parallel$). For example, in Figure 2, if the value of the deletion penalty is set to 0.5, then the quantitative simulation is: $Qs((ServiceOrdered, ServiceOrdered)) = \frac{0.5+1}{1+1} = 0.75$.

[2] How penalty values are set is out of the scope of this paper.

Mean quantitative simulation: Once the quantitative simulation is computed to all state pairs, a mean quantitative simulation value of P and P' can be defined as follows :

$$Mqs(P, P') = \frac{\sum_{i=1}^{\| Qs \|} Qs(PS_i)}{\| Rps \|} \quad (2)$$

Where: PS_i is a pair of states such as $PS_i \in Rps$ for $i = 1.. \| Rps \|$. In the running example, if all the penalty values are set to 0.5 then the mean quantitative simulation is: $Mqs(P, P') = 0.875$.

4 Tests and results in BESERIAL

For validation purposes, we built a test collection consisting of 20 process scenarios from the xCBL[3] textual description of order management choreographies. These two-party choreographies describe possible document exchanges between trading partners in an Order Management business process.

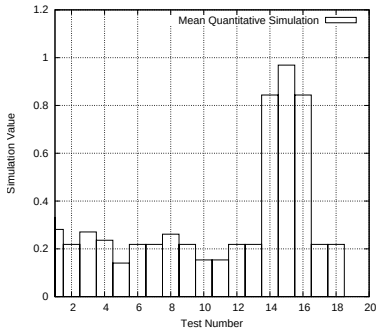

Fig. 3. Graph results of the process order collection test.

In BESERIAL[4], one interface is compared to a collection of interfaces. In Figure 3, the graph shows which interface yields less incompatibilities with respect to the interface given as reference. In this example, the interface which simulates as much as possible the given one yields a mean quantitative simulation value of 0.97. The worst result is 0.14. The interfaces in tests number 14, 15 and 16 are selected as candidates to substitute the old service.

5 Related work

Compatibility test of interfaces has been widely studied in the context of Web service conversations. Most of approaches, which focus on the behavioural dimension of interfaces, rely on similarity calculus to check, *at design time*, whether

[3] XML Common Business Library (`http://www.xcbl.org/`).
[4] http://www-clips.imag.fr/mrim/User/ali.ait-bachir/WebServices/WebServices.html

or not interfaces described for instance by automata are compatible [2]. The behavioural interface describes the structured activities of a business process. Checking interface compatibility is thus based on bi-similarity algorithms [5]. These approaches do not deal with the quantification of interface simulation.

In [6], authors introduced a technique to diagnosis message structure missmatches between service interfaces and to fix them with adapters. An extention of this technique is applied to reslove missmatches between service protocols. The proposed iterative algorithm builds a missmatch tree to help developers to choose the suitable adapter each time and incompatibility is detected. However, this technique can only be applied to protocols which describe a sequence of operations. More complex flow controls, such as loops and options, are not taken into consideration. Recent research has addressed interface similarity measures issues. In [4], authors present a similarity measure for labeled directed graphs inspired by the simulation and bi-simulation relations on labeled transition systems. The presented algorithm returns a value of a simulation measure but does not tell us more about the location of incompatibilities.

6 Future work

In this paper we focused on the calculus of the differences between two behavioural interfaces. Ongoing work aims at extending this work towards two directions: i) detecting complex incompatibilities including structural aspects, ii) guiding analysts in fixing detected incompatibilities. As we compare two different versions of a same service, we identify adequately the delta introduced by the new version. Nevertheless, if we compare two completely different services, the semantics of operations or data types must be considered.

References

1. A. Ait-Bachir, M. Dumas, and M.-C. Fauvet. BESERIAL: Behavioural service analyser. In *Proc. of the BPM Int. Conf.*, pages 374–377. Springer, 2008.
2. L. Bordeaux, G. Salan, D. Berardi, and M. Mecella. When are two web services compatible? In *Proc. of the TES Int. Conf.*, pages 15–28. Springer, 2004.
3. H. Foster, S. Uchitel, J. Magee, and J. Kramer. WS-Engineer: A tool for model-based verification of web service compositions and choreography. In *Proc. of the IEEE Int. Conf. on Software Engineering (ICSE)*, pages 771–774, 2006.
4. N. Lohmann. Correcting deadlocking service choreographies using a simulation-based graph edit distance. In *Proc. of the BPM Int. Conf.*, number 5240 in LNCS, pages 132–147. Springer, 2008.
5. A. Martens, S. Moser, A. Gerhardt, and K. Funk. Analyzing compatibility of bpel processes. In *Proc. of the Advanced Int. Conf. on Telecom. and Int. Conf. on Internet and Web Applications and Services*, pages 147–156. IEEE, 2006.
6. H. Motahari-Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-automated adaptation of service interactions. In *Proceedings of the 16th International Conference on World Wide Web*, pages 993–1002. ACM, 2007.
7. J. Pathak, S. Basu, and V. Honavar. Modeling web service composition using symbolic transition systems. In *Proc. of the 21st Conf. on Artificial Intelligence Workshop on AI-driven Technologies for Service-Oriented Computing*, pages 65–80. AAAI Press, 2006.