BESERIAL: Behavioural Service Interface Analyser

Ali Aït-Bachir[1]*, Marlon Dumas[2] and Marie-Christine Fauvet[1]

[1] University of Grenoble, LIG (MRIM)
385 rue de la bibliotheque – B.P. 53
38041 Grenoble Cedex 9, France
[2] University of Tartu Estonia

Abstract. In a service-oriented architecture, software services interact by means of message exchanges that follow certain patterns documented in the form of behavioural interfaces. As any software artifact, a service interface evolves over time. When this happens, incompatibility problems may arise. We demonstrate a tool, namely *BESERIAL*, that can pinpoint incompatibilities between behavioural interfaces.

1 Motivation

The interface of a software service establishes a contract between the service and its clients or peers. In its basic form, a service interface defines the operations provided by the service and the schema of the messages that the service can receive and send. This *structural interface* can be captured for example using WSDL. In the case of *conversational services* that provide several inter-related operations, a service interface may also capture the inter-dependencies between these operations. Such *behavioural interfaces* can be captured for example using BPEL business protocols, or more simply using state machines as we consider in this paper [2].

As a service evolves, its interface is likely to undergo changes. These changes may lead to the situation where the interface provided by a service no longer matches the interfaces that its peers expect from it. This may result in incompatibilities between the service and the client applications and other services that interact with it.

This paper presents a tool, namely BESERIAL, which is able to automatically detect incompatibilities between behavioural service interfaces and to report them graphically. This feature enables designers to pinpoint the exact locations of these incompatibilities and to fix them. BESERIAL is able to detect elementary changes in the flow of service operations that lead to incompatibilities (e.g. adding an operation, deleting an operation and modifying an

* This author is partially funded by the Web Intelligence project granted by the French Rhône-Alpes Region and a scholarship from Estonian Ministry of Education and Research.

operation). Existing tools, such as WS-Engineer[3], are able to detect if two behavioural interfaces are compatible. But, they will only detect one incompatibility at a time, whereas BESERIAL identifies several incompatibilities at once. Also BESERIAL allows one to compare a given interface with multiple other interfaces in order to identify which one most closely matches the given interface. This can be used for service selection. A screencast of the demo is available online at `http://www-clips.imag.fr/mrim/User/ali.ait-bachir/webServices/webServices.html`.

2 BESERIAL tool

In BESERIAL behavioural service interfaces are modeled by means of Finite State Machines (FSM)[4]. These FSMs are serialized in SCXML[5]. Figure 1 (*Process1*) depicts an FSM representing the behaviour of a given interface (the underlying service supports orders and deliveries of goods). In this work, we consider only the external behaviour of an interface (sent messages and received messages) and we abstract away from internal steps of the underlying business process. Accordingly, transitions are labelled by the types of messages to be sent (with prefix '!') or received (with prefix '?').

2.1 Incompatibility Detection

One of the main features of BESERIAL is to detect changes between a new version of an interface and a previous one. BESERIAL specifically detects changes that may cause the new version to be incompatible vis-a-vis of clients or peers that use the previous version. To that end, BESERIAL relies on a simulation algorithm incorporating an incompatibility diagnosis and recovery mechanism. The originality of BESERIAL is that the simulation algorithm does not stop at the first incompatibility encountered [2,4] but tries to search further to identify a series of incompatibilities leading up to one of the final states of the old version of the interface.

A screenshot of the tool is shown in Figure 1. The screenshot displays the result of comparing two versions of an interface FSM (*Process2* versus *Process1*). The operation that allows customers to cancel an order has been deleted as well as the operation that allows the supplier to send updated information about an order to the customer (*!OrderResponse*). Accordingly, we can see two state pairs ($updateOrderResonse$, $\_invoice$) and ($transfer$, $\_transfer$) linked by a dashed edge labelled *deletion*. The deleted operations are *!OrderResponse* and *?CancelOrder* shown by dotted arrows.

For validation purposes, we built a test collection consisting of 14 process scenarios from the xCBL[6] textual description of order management choreographies.

[3] `http://www.doc.ic.ac.uk/ltsa/eclipse/wsengineer/`

[4] Transformations exist between other languages for describing behavioural service interfaces (e.g. BPEL) and FSMs – see for example the WS-Engineer and Tools4BPEL toolsets referenced above.

[5] `http://www.w3.org/TR/scxml/`

[6] XML Common Business Library (`http://www.xcbl.org/`).

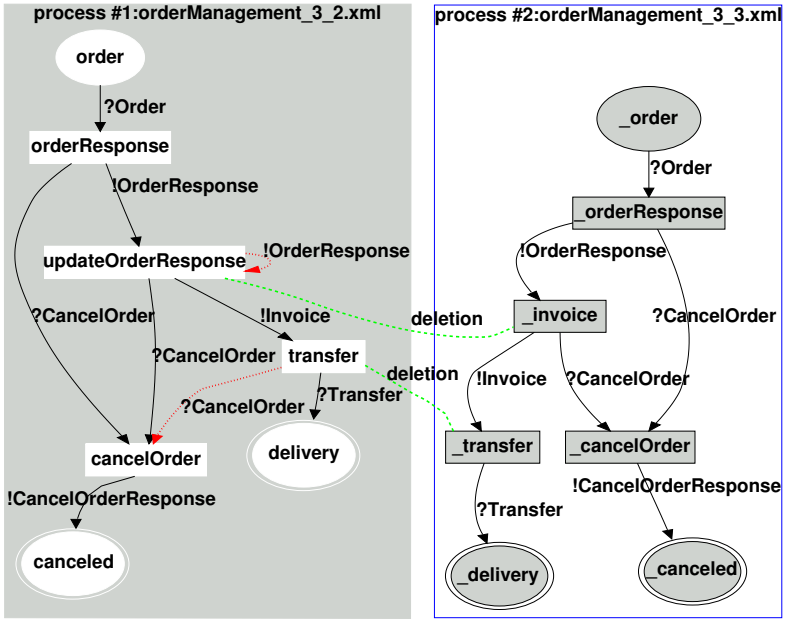

Fig. 1. Detected incompatibilities in two FSMs.

These two-party choreographies describe possible document exchanges between trading partners in an Order Management business process.

2.2 BESERIAL in action

One of the two features of BESERIAL is to detect changes between two behavioural interfaces that cause that one interface does not simulate the behaviour of another interface. A typical usage scenario is one where the compared interfaces correspond to consecutive versions of a service. The algorithm simulates the two FSMs by visiting state pairs (one state from each of the two interfaces). Given a state pair, the algorithm determines if an incompatibility exists and classifies it as addition, deletion or modification (i.e. replacement of one operation with another). If an *addition* is detected the algorithm moves along the transition of the added operation in the new version only. Conversely, if the change is a *deletion*, the algorithm will move along the transition of the deleted operation in the old version only. However, if a *modification* is detected, the algorithm progresses along both FSMs simultaneously. Detection results are written down in a text area showing the test and its outcomes (states, changes). Results can be viewed graphically, as in Figure 1, to better pinpoint the incompatibilities.

BESERIAL can also compare one interface to a collection of interfaces. The comparison results are sorted increasingly according to the number of detected incompatibilities. This functionality may be used to select which service interface is most closely compatible with the required interface.

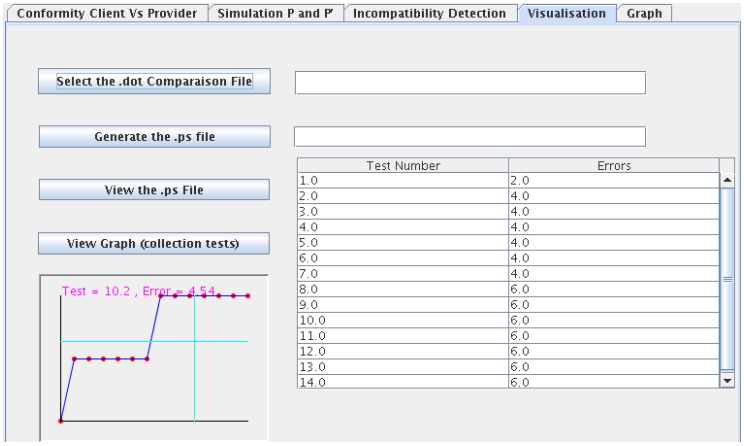

Fig. 2. Test results of incompatibility detections in BESERIAL tool.

In Figure 2, one given interface is compared to other interfaces and the detection result is rendered and sorted in the table area. Results can be sorted increasingly and a graph can be viewed. The graph shows which interface yields less incompatibilities with respect to the interface given as reference. In this example, the closest interface to the given one yields two incompatibilities and the worst result is six incompatibilities.

3 Future work

In this paper we focused on elementary incompatibility detection. Ongoing work aims at extending BESERIAL towards two directions:

- Detecting complex incompatibilities combining elementary ones,
- Fixing detected incompatibilities.

As BESERIAL covers synchronous communications only, it also needs to be extended to address the asynchronous case along the lines of [3].

References

1. L. Bordeaux, G. Salan, D. Berardi, and M. Mecella. When are two web services compatible? In *Proc. 5th Int. on Technologies for E-Services*, pages 15–28, Canada, 2004. Springer Verlag.
2. R. Hamid, N. Motahari, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-automated adaptation of service interactions. In *Proc. of the 16th WWW Int. Conf.*, pages 993–1002, Canada, 2007. ACM.
3. J. Wu and Z. Wu. Similarity-based web service matchmaking. In *Proc. of the Int. Conference on Services Computing*, pages 287 – 294, Florida, 2005.