An Automaton-based Approach for Web Service Mediation

Marie-Christine FAUVET and Ali AÏT-BACHIR
University of Grenoble, CLIPS-IMAG
385 rue de la bibliothèque – B.P. 53 – 38041 Grenoble Cedex 9, France
Marie-Christine.Fauvet@imag.fr, Ali.Ait-Bachir@imag.fr

Abstract. Interactions between Web services are based on interfaces which describe Web services on both structural and behavioral perspectives. It can happen that the interface provided by a service does no longer match (for instance, because of an evolution) the interface required by its partners. In this situation, and until the required interfaces are fixed, interactions cannot succeed. To address this issue, and focusing on the behavioral part of interfaces, we propose an approach based on a mediator which aims to seamlessly resolve incompatibilities during service interactions. We adopted a formal tool as finite-state automata, particularly Labeled Transition Systems to model the behavioral aspects of Web services.

Keywords. Web services, conversation, mediation, finite state automata, provided interface, required interface

1. Introduction

Message exchanges form the basics of Web service interactions. Therefore, modeling Web services relies on the descriptions of messages they send and receive and on their interdependencies, as well from the structural point of view (types of exchanged messages) as from the behavioral point of view (control flow between interactions). We distinguish[1] the *provided* interface an existing service exposes, from its *required* interface as it is expected by its clients (which could be softwares of any kind, such as services).

Service interfaces are generally seen as a contract between the provider and its clients. Thus, services are expected to respect their interface. However, a provided interface may need to be modified because, for instance, of an evolution of the corresponding service. When it happens, whether *a priori* for a comparison or *a posteriori* for a compliance test, that the provided interface of a service does not correspond any more to the one its partners expect, two solutions apply: (1) modify the service in order to make the interface it requires match the provided interface; (2) introduce an adapter that reconciles the provided interface with those required by the partners. The former solution is not satisfying because the same service may interact with many other partners which consider its original interface. This leads to the situation where the same service has to expose as many provided interfaces as collaborations it is involved in. The latter solution consists

[1] in proceedings of CE'06, Antibes France, September 2006

in providing an adapter which is capable of matchmaking each of the required interfaces with the one provided by the service.

A service is generally described according to its structural or behavioral dimensions, or even according to its non-functional dimension. Thus, interface matchmaking must be studied according each of these dimensions. Dealing with structural matchmaking essentially leads towards reconciliation between different message types. This issue has been widely studied so far (see for example [14,2,12]) and many commercial systems exist (e.g. Microsoft's BizTalk Mapper). Conversely the problem of behavioral matchmaking is still a research topic [13,16,2,4].

As the development of new services by composition of existing ones has gained considerable momentum as a means of integrating heterogeneous applications and realizing business collaborations, reconciliation of service interfaces is likely to become an hot and crucial topic.

The contributions of the study reported in this paper are mainly:

- An automaton-based model of the behavioral dimension of services;
- A mediator which allows clients to keep accessing a service according to the interface they require even after the behavioral interface of the service has been modified.

The rest of the text is structured as follows. In Section 2 we frame the problem addressed and we give an illustrating scenario. Section 3 discusses related work. In section 4 we introduce the automaton-based behavioral interface model and the mediator respectively. Finally, Section 5 concludes and discusses directions for future work.

2. Motivating example

As a working and motivating example we consider a scenario that arises when evolutions of the provided interface of an existing service lead to inconsistent interactions between the service itself and its partners. As stated in the previous section, we focus only on the behavioral dimension of interfaces. The scenario involves two partners: (1) the provider, *Conf* which gives information about scientific conferences (name, important dates, location, organisation and program committees, etc.) and receives from clients papers submitted to a given conference; (2) the client, *Lab* which is meant to be a research group. To keep the example simple we do not model all features expected in a review system (such as for instance paper submission or review process).

The following operations are offered by the service *Conf*:

- *Register(BCN)*: to register by providing a bank card number *BCN*. If the operation succeeded, the returned result is a client identifier, otherwise, an error message is sent to the client.
- *Login(clientId)*: to log in the service *Conf*. In case of success a session identifier is issued and associated to the client identified by *clientId*, otherwise a error message is returned to the client.
- *ConfList(clientId)*: given a valid *clientId* the operation returns details about forthcoming conferences.

The initial behavior of the operation *ConfList()* is described by an UML sequence diagram depicted in Figure 1(a). Figure 1(b)) shows the same operation after its behavior has been changed: now a session has to be open first (by submitting *Login()* before *ConfList()*.

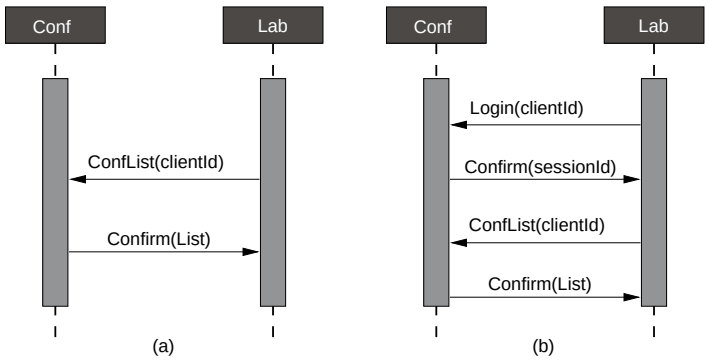

Figure 1. Sequence diagram modeling interactions: (a) initial version, (b) modified version.

The study reported in this article aims at dealing with the issues that arise when clients wish to access a given service, requiring an interface which no longer matches the one provided as operations offered by the service have been changed. Focusing on behavioral point of view of operations, the objective is to seamlessly reconcile interface required by clients with the interface provided by the service, in order to avoid failures.As a first step towards this objective we restrict our focus to interfaces whose behavior is described as a sequence of operations (e.g. operation *Register()* must be executed before operation *Login()*).

3. Related Work

Approaches related to Web service descriptions which focus on behavioral dimension of services are mostly related to standards (see for example WSCI [1,13] and BPEL [16]). In these approaches, the consistency of service interactions rely on collaborations partners have to establish. Specifically, WSCI description does not address the issue related to the matchmaking between service provided interface and those required by its partners. Clients are responsible for using an interface compatible with the one provided by services they wish to access. Other studies propose to translate workflow patterns into a process described in BPEL [16]. In a BPEL process, conversations between partners are orchestrated by the composite service which has the global vision of interactions involved. However, when an evolution of any partner's interface occurs, the compatibility does not hold any more. Then, to fix this incompatibility all partners involved in the composition and concerned by this evolution have to adapt their interfaces.

Web service behavior can be modeled using Petri nets or finite-state automata [9, 7,8,11]. Even thought compatibility checking between two behavioral interfaces can be implemented using bi-similarity algorithms on Petri nets, these approaches do not deal with reconciliation needed when behavioral incompatibilities occur.

In a community-based approach as defined in the SELF-SERV service composition system [6,5], instead of fixing incompatibility of interfaces, the service which provides an incompatible interface is substituted by another taken in the same community. This is satisfying in clients' point of view, but certainly not in providers' point of view: the service whose interface has evolved is no longer accessible via the community, unless an adapter is supplied by the provider.

In dialog theory-based approaches (see for example [3]) paths in a tree model all possible scenario of a conversation. A path in this tree describes a sequence of actions that lead to a consistent dialog between the client and the service. However, this proposal does not consider interpretation of error messages returned to the client so that it can adapt its behavior. Moreover, each time an evolution of the service occurs all client related to it must redefine the tree that models the conversation that has been changed.

Mediation has been already proposed to deal with incompatibilities between structural interfaces in service conversations [15,2]. The basics of the mediation, as it was introduced for data integration, is that all messages exchanged between two partners pass through a service mediator also called virtual supplier. The mediator introduced in this paper is based on the same principle, except it is meant to detect and reconcile behavioral incompatibilities. Moreover, the mediator is hosted by the service, not by the client. It is so for many reasons (e.g. security, knowledge of operation behavior, etc.) we can not detail in this paper because of space limitation.

The mediation approach we propose in this text is described and illustrated in the next section.

4. Mediation Approach

We choose to model the behavior of operations provided by a Web service with *Labelled Transition Systems (LTS) [10]*[2]. As an example, the behavior of the operation *ConfList()* is formalised by the LTS shown in Figure 2. Transitions are labelled by events which could be either a message to receive (with prefix $<$), a message to send (with prefix $>$), or a call to an internal operation (with prefix $*$). Intermediate states model execution of internal operations and each final state models whether the operation succeeded or not.

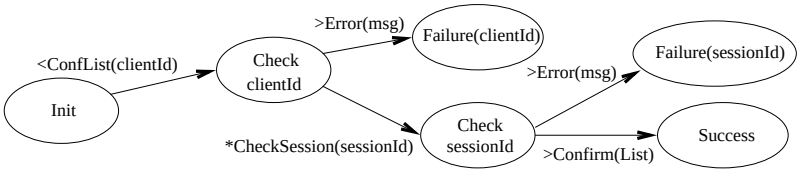

Figure 2. The LTS modeling *ConfList()*.

According to the scenario introduced before, the mediator's role is depicted in the sequence diagram given in Figure 3. Once the message *ConfList()* is received by the mediator, this latter forwards it to the service *Conf* which in turn, checks the validity of the value *clientId* given as parameter and returns an error message because no session

[2]Petri Nets-based modeling was another option. We do not discuss this choice here, as it is out of the scope of this paper.

associated with it has been created before. The mediator then catches the error message and by analysing the LTS associated with *ConfList()*, finds out that the operation *Login()* must be executed first with the value *clientId*, and acting on the behalf of the client, asks the service to perform the login operation *Login(clientId)*. Next, the message requesting the operation *ConfList()* is submitted again, still by the mediator to the service, which this time returns the list of conferences to the client, via the mediator.

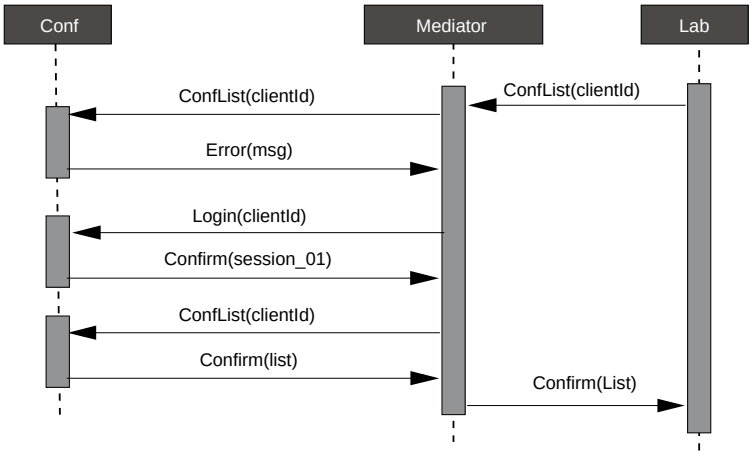

Figure 3. Sequence diagram modeling the reconciliation process

The reconciliation process implemented by the mediator aims to find out, the sequence of operations whose execution may fix an error. This process performs the following tasks:

- *Error diagnosis which returns the faulty information.* Back to the example (see Figure 2), the automaton ended in a failure state (*Failure(sessionId)*). Starting from this state the algorithm backtracks to the preceding (*Check(sessionId)*) associated with the operation that rose the error. According to the signature of this operation, it is returned that the faulty information is *sessionId*.
- *Browsing operation LTS which returns the LTSs of missing operations.* This step aims at collecting LTSs of operations whose outputs contain the information that was seen as faulty. In the example, *sessionId* being the faulty information, the LTS to be returned is the one associated with operation *Login()*.
- *Composition of returned LTSs and execution.* Eventually, LTSs previously returned are composed with the one of the operation that failed and then executed. To express the sequential composition of two LTS, we consider that the successful termination state of the first LTS is the initial state of the LTS which follows it (see Figure 4).

The process above is summarised by the algorithm shown Figure 5: line 5, realises the two first step, while the lines 6, 7 and 8 do the last one.

The algorithm Figure 6 finds out the list of the LTSs components which must be compound in a sequence. The LTS resolving the error is found by trying to match the missing information, in the check state preceding the failure state, with outputs of all the operations of the service (see line 3). The retrieved LTS may have missing information

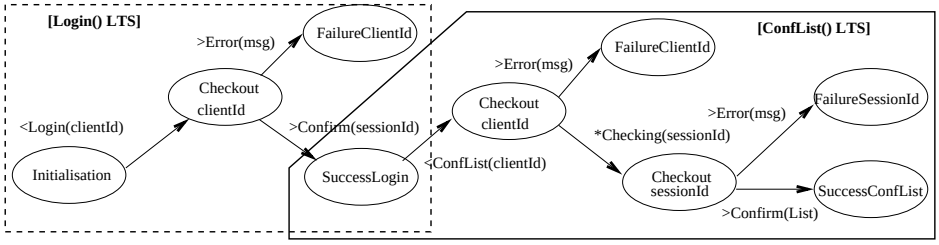

Figure 4. LTSs composition.

```
1  SequenceResolve (Error: Message ; LTSin: LTS )
2      LTSres: LTS                                          { result variable }
3      LTScomponents: List{LTS}, Nb: integer                { LTSComponents size }
4      LTSres ← LTSin                                       { initialisation with the input variable }
5      <LTScomponents, Nb> ← ListLTS(Error, LTSin)
           { ListLTS(E, L) build a LTS list, with the components number. This list has
           the LTSs to compound  }
6      For i← 1 to Nb
7          LTSres ← Compound( LTScomponents[i], LTSres)
8      Process (LTSres)
```

Figure 5. Reconciliation Algorithm

which will generate an error. That's why the algorithm must be applied recursively to the previously found LTS (see line 7).

```
1  ListLTS (Error: Message ; LTSin: LTS ): List{LTS}
2      ListRes: List{LTS}, ListRes←{}
3      ListRes←ListRes+LTSop(AllLTS, Error)
           { + denotes the operator that adds an element to a list }
4      LTSin←LTSop(AllLTS, Error)
5      If there are error messages in LTSin
6          Error ← PreviousError(Error, LTSin)
7          ListRes← ListRes+ ListLTS(Error, LTSin)
8      Return(ListRes)
```

Figure 6. Algorithm of method ListLTS().

The algorithm Figure 7 shows how two LTSs are compound in a sequence. The initial state of the resulting LTS is the initial state of the first LTS of the sequence (see line 4). The final state of the compound LTS is the final state of the second LTS (see line 5). Finally, the success final state of the first LTS is combined with the initial state of the second LTS (see line 6).

5. Conclusion and Future Work

In this paper, we proposed a mediation-based approach to deal with failures that may arise during interactions between web services. More specifically, we focused on con-

```
1  Compound (LTSa: LTS ; LTSb: LTS ): LTS
2      LTSRes: LTS                                          { result variable }
3      InitialState: State, FinalState: State
4      InitialState ←SelectInitialState(LTSb)          { to get the LTSb initial state  }
5      FinalState ←SelectFinalState(LTSa)               { to get the LTSa final state  }
6      LTSRes←Combine(LTSa,LTSb,InitialState,FinalState)
           { Combine the two LTSs according to initial and final states }
7      Return(LTSRes)
```

Figure 7. Algorithm of method Compound().

versations which cannot successfully complete because client and provider interfaces do not match after changes to provider interface.

We use LTS (*Labelled Transition Systems*) to model the behavior of operations exposed by a service. The mediator provides a technique that searches, among all operations exposed by the service, those whose execution must precede the one which produced the failure. It returns a set of operations, which once composed with the LTS of the faulty operation produces a new LST whose execution reconciles the unsuccessful conversation.

The preliminary work reported in this paper opens several directions. Firstly, we focused on the situation that occurs when a service interface, whose behavior is described by a sequence of operations, is modified by adding new operations in the sequence. This has to be extended by considering any kind of control flow constructs (e.g., loops, conditional routing, switches). Another critical extension is that of considering the case when the algorithm returns many LTSs to be composed. Addressing this issue needs further studies such as defining rules or policies to guide the composition of all returned LTS.

References

[1] G. Alonso, F. Cassati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications*. Springer-Verlag, Berlin, 2003.

[2] M. Altenhofen, E. Boerger, and J. Lemcke. A high-level specification for mediators(virtual providers). In *In Proceedings of the BPM'2005 Workshops : Workshop on Choreography and Orchestration for Business Process Managament*, LNCS, pages 116–129, Nancy, France, September 2005. Springer-Verlag.

[3] L. Ardissono, A. Goy, and G. Petron. Enabling conversations with web services. In *Proc AAMAS03: The Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 819–826, Melbourne, Australia, July 2003. ACM.

[4] B. Benatallah, F. Casati, D. Grigori, H.R. Motahari-Nezhad, and F. Toumani. Developing adapters for web services integration. In *Proceedings of the 17th International Conference on Advanced Information System Engineering, CAiSE 2005, Porto, Portugal*, pages 415–429, 2005.

[5] Boualem Benatallah, Marlon Dumas, and Quan Z. Sheng. Facilitating the rapid development and scalable orchestration of compositie web services. *Distributed and Parallel Databases*, 17(1):5–35, 2005.

[6] Boualem Benatallah, Quan Z. Sheng, and Marlon Dumas. The self-serv environnment for web services composition. *Internet Computing*, 7(1):40–48, 2003.

[7] Piotr Chrzastowski-Wachtel, Boualem Benatallah, Rachid Hamadi, Milton O'Dell, and Adi Susanto. A top-down petri net-based approch for dynamic workflow modeling. In *Business Process Management Proceedings of the International Conference on Business Process Management (BPM'03)*, number 2678 in LNCS, pages 336–353, Eindhoven, The Netherlands, June 2003. Springer-Verlag.

[8] Remco Djikman and Marlon Dumas. Service-oriented design: A multi-viewpoint approch. *International Journal of Cooperative Information Systems*, 13(4):337–368, 2004.

[9] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Model-based verification of web service compositions. In *Automated Software Engineering (ASE 2003), 18th IEEE International Conference*, pages 152–161, Montreal Canada, October 2003. IEEE Computer Society Press.

[10] Serge Haddad, Tarak Melliti, Patrice Moreaux, and S. Rampacek. Modelling web services interoperability. In *ICEIS (4)*, pages 287–295, Porto, Portugal, April 2004.

[11] Axel Martens. Consistency between executable and abstract processes. In *e-Technology, e-Commerce and e-Service the 2005 IEEE International Conference*, pages 60–67, Hong Kong, China, March 2005. IEEE Computer Society Press.

[12] P. Oaks and A.H.M. ter Hofstede. Guided interaction: A language and method for incremental revelation of software interfaces for ad hoc interaction. In *In Proceedings of the BPM'2005 Workshops : Workshop on Business Processes and Services*, LNCS, pages 3–17, Nancy, France, September 2005. Springer-Verlag.

[13] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[14] Shankar R. Ponnekanti and Armondo Fox. Interoperability among independently evolving web services. In *Middleware 2004, 5th ACM/IFIP/USENIX International Middleware Conference*, number 3231 in LNCS, pages 331–351, Toronto, Canada, October 2004. Springer-Verlag.

[15] Heinz W. Schmidt and Ralf H. Reussner. Generating adapters for concurrent component protocol synchronisation. In *Formal Methods for Open Object-Based Distributed Systems V, Proceedings of the IFIP TC6/WG6.1 ,The Fifth IFIP International conference on Formal Methods for Open Object-based Distributed Systems*, volume 81, pages 213–229, Enschede, The Netherlands, March 2002. Springer-Verlag.

[16] Petia Wohed, Wil M.P van der Aalst, Marlon Dumas, and Arthur H.M. ter Hofstede. Analysis of web services composition languages: The case of bpel4ws. In *Conceptual Modeling - ER 2003. 22nd International Conference on Conceptual Modeling*, number 2813 in LNCS, pages 200–215. Springer-Verlag, October 2003.