Reconciling Protocol Mismatches of Web Services by Using Mediators

Xitong Li, Yushun Fan, Stuart Madnick, Quan Z. Sheng

Working Paper CISL# 2008-17

December 2008

Composite Information Systems Laboratory (CISL)
Sloan School of Management, Room E53-320
Massachusetts Institute of Technology
Cambridge, MA 02142

Reconciling Protocol Mismatches of Web Services by Using Mediators[†]

Xitong Li[1], Yushun Fan[1], Stuart Madnick[2], Quan Z. Sheng[3]
[1] Department of Automation, Tsinghua University, Beijing 100084, P.R. China
[2] MIT Sloan School of Management, 50 Memorial Drive, Cambridge, MA 02142, USA
[3] School of Computer Science, University of Adelaide, Adelaide, SA 5005, Australia

Abstract

In the era of Global Services, Service Oriented Architecture (SOA) has been gaining momentum for building Web-based information systems. Service composition is one of the key objectives for adopting SOA. Unfortunately, Web services are not always exactly compatible and it is a non-trivial task to address the mismatches between them. To this end, an approach based on mediator patterns is proposed to develop mediators for reconciling protocol mismatches of partially compatible services and mediating them together. A heuristic technique is developed for identifying protocol mismatches and selecting appropriate patterns. The main steps of the reconciliation approach are presented.

Abstrait

Les services de Web ne sont pas toujours exactement compatibles et les disparités entre eux devraient être adressées. On propose une approche basée sur des modèles de médiateur pour développer des médiateurs pour réconcilier des disparités de protocole. Une technique heuristique est développée pour identifier des disparités de protocole et choisir les modèles appropriés. Les étapes principales de l'approche de réconciliation sont présentées.

1. Introduction

In the era of Global Services, Service Oriented Architecture (SOA) has been gaining momentum for building Web-based information systems. Service composition, whereby multiple independent services are used in combination to accomplish a more complex task, is one of the key objectives for adopting SOA, as it facilitates the seamless integration of heterogeneous information systems within and/or across enterprise boundaries [1]. Unfortunately, Web services are not always exactly compatible. Reconciling the mismatches of partially compatible services is a non-trivial task and much effort may be needed.

Mismatches of service composition are generally recognized at both the signature level and protocol level. *Signature mismatches*, which occur in the message types, have received considerable attention [2]. In comparison, the problem of *protocol mismatches*, which occur in the message exchanging sequences, is still open. A frequently-used approach to the reconciliation of protocol mismatches is to develop a mediator which is a piece of code that sits between the interacting services [3]. However, current approaches only provide partial solutions. The mediators developed by these approaches have no control logics and fail to compensate complicated mismatches. Few of these approaches can be used to

[†] Acknowledgement: The work is partially supported by the MIT Sloan China Management Education Project.

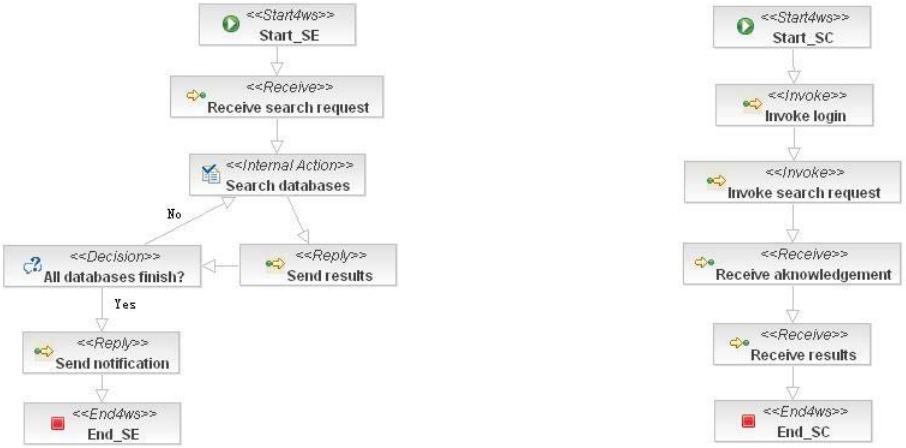
automatically generate deployable codes. Additionally, none of them provides a systematic solution to reconciliation of protocol mismatches. Last but not least, to the best of our knowledge, there exists no software tool which can assist developers to ease their efforts on mediation tasks, such as identifying protocol mismatches and generating mediation codes.

In the previous work, six basic patterns of protocol mismatches and appropriate basic mediator patterns for reconciling these mismatches have been identified [4, 5]. By using the basic mediators as patterns, complicated mediators can be modularly built and contain control logics, which can compensate all possible protocol mismatches. The basic mediator patterns are: 1) *Simple Storer*: the mediator with the capability of simply receiving and storing messages of certain specific type; 2) *Simple Constructor*: the mediator with the capability of simply constructing and sending messages of certain specific type; 3) *Splitter*: the mediator with the capability of receiving a single message of certain type and splitting it into two or more partial messages; 4) *Merger*: the mediator with the capability of receiving two or more partial messages and merging them into a single one; 5) *Storing Controller*: the mediator with the capability of storing and conditionally sending some messages of certain type in terms of specific logic; 6) *Constructing Controller*: the mediator with the capability of conditionally constructing and sending some messages of certain type in terms of specific logic. Details of the mediator patterns and the discussion of the comprehensiveness of these patterns are presented in [4, 5].

To promote the automation of the pattern-based approach, we develop a heuristic technique that assists developers to identify protocol mismatches and select appropriate mediator patterns. The proposed reconciliation approach consists of five main steps which can lead to a systematic solution to the reconciliation of protocol mismatches for Web services composition. This paper is organized as follows. Section 2 introduces a motivating example that will be used to demonstrate our approach throughout the paper. Section 3 presents the heuristic technique for identifying mediator patterns. Section 4 proposes the reconciliation approach. Section 5 presents related work and Section 6 concludes the paper.

2. Motivating Example

The example consists of a search client (S_C) and a search engine (S_E). We take BPEL as the specification language for describing their protocols, as shown in Figure 1.



(a) BPEL of search engine service (b) BPEL of search client service

Figure 1. A motivating example of service composition with protocol mismatches

S_C invokes S_E by sending its login information and the search request respectively. After that, S_C waits for the acknowledgement and the results from S_E. On the other hand, after receiving search request, S_E starts to search several distributed databases one by one (by performing its internal searching action). Once S_E finishes a database and obtains some searched items, it sends these items to its client immediately. If all databases have been searched, S_E sends a completing notification to its client and the search work is finished. To make the two services compatibly interact with each other, protocol mismatches between them needs to be identified so that appropriate mediator patterns are selected to reconcile the mismatches.

3. Identification of Mediator Patterns

We propose a heuristic technique based on message mapping for semi-automatic identification of protocol mismatches and mediator patterns. By semi-automation, we mean that developers should specify the message mappings and adjust the identified patterns.

Message mapping M is a finite set of mapping relations, i.e., $M = \{mr_i\}$. Each mapping relation mr_i is expressed in the form of <*source*, *cnst_s*, *target*, *cnst_t*>, where *source* is a part/element of a sending message and *target* is the corresponding part/element of a receiving message. *source*/*target* is expressed in the form of *Service.Message.Part*. *cnst_s* is the constraint of the operation that sends *source* and *cnst_t* is the constraint of the operation that receives *target*. *cnst_s*/*cnst_t* can be NULL if there is no constraint with the sending/receiving message. In the motivating example, the receiving message *SearchRequest* of S_E has two parts: *login* and *request*. Thus the part *login* is a target and expressed as S_E.sreq.login, where sreq stands for the message *SearchRequest*. For the sake of simplicity, the part name is omitted if the message consists of only one part. For example, the sending message *Login* of S_C has only one part *login*. Thus it is a source and expressed as S_C.login. *source*/*target* can be NULL if the sending/receiving message doesn't exist. The prefix of *source*/*target* is the message name of *source*/*target*, denoted by *prefix*(*source*/*target*), e.g., *prefix*(S_E.sreq.login) = S_E.sreq and *prefix*(S_C.login) = S_C.login. For any two mapping relations, e.g., mr_i and mr_j, it is easy to see the following formulas:

a) $source(mr_i) \neq \text{NULL}$, if $target(mr_i) = \text{NULL}$;
b) $target(mr_i) \neq \text{NULL}$, if $source(mr_i) = \text{NULL}$;
c) $cnst\_s(mr_i) = cnst\_s(mr_j)$, if $prefix(source(mr_i)) = prefix(source(mr_j))$;
d) $cnst\_t(mr_i) = cnst\_t(mr_j)$, if $prefix(target(mr_i)) = prefix(target(mr_j))$.

With the above notation, developers specify the mapping relations, as shown in Table 1.

Table 1. Message Mapping Relations

mapping	source	cnst_s	target	cnst_t
mr_1	S_C.login	NULL	S_E.sreq.login	NULL
mr_2	S_C.sreq	NULL	S_E.sreq.request	NULL
mr_3	NULL	NULL	S_C.ack	NULL
mr_4	S_E.partialResult	<*while*> condition(x)	S_C.totalResult	NULL
mr_5	S_E.ntf	condition($\overline{x}$)	NULL	NULL

The first mapping relation (i.e., mr_1) indicates that S_C sends a login message and S_E receives the message as the part login of its message sreq. There is no constraint with the two operations. We denote that $source(mr_1)$ = S_C.login and $target(mr_1)$ = S_E.sreq.login. In the fourth mapping relation (i.e., mr_4),

"<*while*> condition(x)" indicates that the message "S_E.partialResult" is sent iteratively under the condition x. In the fifth mapping relation (i.e., mr_5), "condition($\overline{x}$)" indicates that the message "S_E.ntf" is sent when the condition x doesn't hold. We also denote that $cnst\_s(mr_5)$ = condition($\overline{x}$) and $cnst\_t(mr_5)$ = NULL.

Herein, we introduce a heuristic rule for identifying which mediator pattern should be selected by using the mapping relations. For two mapping relations, i.e., mr_i and mr_j, the rule is as follows:

Selection Rule of Mediator Patterns

if $(cnst\_s(mr_i) = cnst\_t(mr_i)) \wedge (prefix(source(mr_i)) = prefix(source(mr_j))) \wedge (prefix(target(mr_i)) = prefix(target(mr_j)))$

then there is no need of mediator patterns;

else if $(cnst\_s(mr_i) = cnst\_t(mr_i)) \wedge (target(mr_i) = \text{NULL})$

then a Simple Storer pattern is selected;

else if $(cnst\_s(mr_i) = cnst\_t(mr_i)) \wedge (source(mr_i) = \text{NULL})$

then a Simple Constructor pattern is selected;

else if $(cnst\_s(mr_i) = cnst\_t(mr_i)) \wedge (prefix(source(mr_i)) = prefix(source(mr_j))) \wedge (prefix(target(mr_i)) \neq prefix(target(mr_j)))$

then a Splitter pattern is selected;

else if $(cnst\_s(mr_i) = cnst\_t(mr_i)) \wedge (prefix(source(mr_i)) \neq prefix(source(mr_j))) \wedge (prefix(target(mr_i)) = prefix(target(mr_j)))$

then a Merger pattern is selected;

else if $(cnst\_s(mr_i) \neq cnst\_t(mr_i)) \wedge ((cnst\_s(mr_i) = \text{NULL} \wedge source(mr_i) = target(mr_i)) \vee (cnst\_s(mr_i) \neq \text{NULL} \wedge target(mr_i) = \text{NULL}))$

then a Storing Controller pattern is selected;

else if $(cnst\_s(mr_i) \neq cnst\_t(mr_i)) \wedge ((cnst\_t(mr_i) = \text{NULL} \wedge source(mr_i) = target(mr_i)) \vee (cnst\_t(mr_i) \neq \text{NULL} \wedge source(mr_i) = \text{NULL})$

then a Constructing Controller pattern is selected;

else developers' intervention is needed to build more complicated mediators.

In the motivating example, four mediator patterns can be selected to address the mismatches after performing the selection rule. The selected mediator patterns are as follows:

1) A Merger pattern is used to receive S_C.login and S_C.sreq from S_C, and then it sends S_E.sreq to S_E, where $S_E.sreq = S_E.sreq.(login, request)$. This pattern is selected according to mr_1 and mr_2.
2) A Simple Constructor pattern is used to construct S_C.ack and send it to S_C. This pattern is selected according to mr_3.
3) A Merging Repeater pattern [5] is used to iteratively receive S_E.partialResult from S_E until all partial databases are finished according to mr_4. The Merging Repeater merges all partial results together and sends S_C.totalResult to S_C. Since mr_4 corresponds to a complicated mismatch with iterative structure, the Merging Repeater pattern can be selected by service developers.
4) A Storing Controller pattern is used to conditionally store S_E.ntf which is sent by S_E. This pattern is selected according to mr_5.

As discussed in [5], service developers should configure the structures and control logics of the

selected mediator patterns and compose them together. In the motivating example, the Merging Repeater pattern can successfully compensate the mismatch with iterative structure and there is no need of another Storing Controller pattern. Thus three mediator patterns are eventually selected for reconciliation, i.e., a merger, a simple constructor and a merging repeater.

4. The Reconciliation Approach

We take the BPEL files of two partially compatible services as input and produce deployable mediators as output for reconciling protocol mismatches if the correct mediator exists. The main steps of the reconciliation approach are as follows.

Step 1: *Service model transformation*. In our approach, the protocols of both services and mediators are depicted based on Colored Petri Net (CPN) [6]. The benefit of adopting CPN models lies in that they provide rich analysis capability to support solid verification of protocol mediation. As the first step, BPEL-based services are transformed to CPN-based service models.

Step 2: *Selection of mediator patterns*. In the WSDL/BPEL specifications of Web services, messages exchanged between them are specified as an aggregation of parts and/or elements. For selection of mediator patterns, developers should specify the message mapping relations between the two Web services to be composed. By performing the selection rule (see Section 3), possible protocol mismatches are identified and appropriate mediator patterns are selected automatically based on the message mapping.

Step 3: *Mediator configuration and composition*. The structures and control logics of the mediator patterns need to be configured as parameters by service developers according to the identified mismatches. After configuration, the mediator patterns are composed to construct a composite mediator that reconciles all identified protocol mismatches. Both mediator patterns and composite mediators are depicted as underlying CPN models for the following formal verification.

Step 4: *Mediation verification*. The mediator produced in the above steps is only a conceptual model and should be put between the interacting services. The composition model of the two services and the mediator need to be formally verified. Generally, we consider that the mediation has failed if any deadlock exists. Otherwise, the mediation is successful. Details of mediation verification are beyond the scope of this paper.

Step 5: *Code generation of mediators*. Only successful mediator will be performed in this step. It is the converse of the first step, i.e., transforming CPN models to BPEL-based services. To facilitate code generation of deployable mediators, we have developed the BPEL templates for the corresponding mediator patterns. With these BPEL templates, the pseudo-code for reconciling protocol mismatches can be produced automatically. Due to space constraints, we will not present the details in this paper.

To validate the feasibility of our approach, we have developed a prototype system, namely Service Mediation Toolkit (SMT). The toolkit contains several components that are designed to facilitate the above steps of our approach. The core component of SMT is the Mediation Workspace which provides a GUI workbench for developers to manipulate Web services and mediators. It is implemented as an eclipse plugin that is easy to be integrated with other eclipse-based applications.

5. Related Work

A large number of research works have been developed for reconciling various kinds of composition

mismatches [7]. As a challenging issue, the problem of protocol mismatches is still open and needs further research. It has been recognized that patterns can be used to reconcile protocol mismatches [3, 7]. Five mismatch patterns are identified in [3] and corresponding templates of BPEL codes for building the mediators are presented, but these patterns are not sufficient. The taxonomy of composition mismatches is proposed in [7] and a selection of patterns is presented to reconcile these mismatches. The taxonomy, however, does not sufficiently address protocol mismatches.

Inspired by [3], our approach is significantly different from the existing works in the following aspects: 1) The mediator patterns presented in this paper are derived from our comprehensive identification of protocol mismatches and can be used to sufficiently address those mismatches [4]; 2) A heuristic technique is developed to (semi-) automatically identify protocol mismatches and to select appropriate mediator patterns; 3) A formal modeling method (i.e., CPN models) is adopted as an underlying formalism for depicting the protocols of both services and mediators., which supports solid verification of protocol mediation; 4) The proposed approach is considered to be a systematic and engineering solution to reconciliation of all possible protocol mismatches.

6. Conclusion

As the world economy continually shift towards global collaboration, the need for integrating heterogeneous information systems within and/or across enterprise boundaries has become critically important. To compatibly integrating these systems, we propose a systematic approach that can be used to develop mediators and reconcile protocol mismatches for composing Web services. We have been developing a prototype system to validate the feasibility of our approach. Future work will focus on the further implementation of the prototype system to address more general and real-world cases.

References

[1] S. Staab, W. van der Aalst, V. R. Benjamins *et al.*, "Web services: been there, done that?," *Intelligent Systems, IEEE*, vol. 18, no. 1, pp. 72-85, 2003.

[2] M. Szomszor, T. R. Payne, and L. Moreau, "Automated Syntactic Medation for Web Service Integration," *Proceedings of the International Conference on Web Services (ICWS), Chicago, IL*, 2006.

[3] B. Benatallah, F. Casati, D. Grigori *et al.*, "Developing Adapters for Web Services Integration," *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, 2005.

[4] X. Li, Y. Fan, and F. Jiang, "A Classification of Service Composition Mismatches to Support Service Mediation," *Proceedings of the International Conference on Grid and Cooperative Computing (GCC)*, pp. 315-321, 2007.

[5] X. Li, Y. Fan, J. Wang *et al.*, "A Pattern-Based Approach to Development of Service Mediators for Protocol Mediation," *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pp. 137-146, 2008.

[6] K. Jensen, "Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. vol. 1, Basic Concepts," *Monographs in Theoretical Computer Science. Springer-Verlag*, 1997.

[7] S. Becker, A. Brogi, I. Gorton *et al.*, "Towards an Engineering Approach to Component Adaptation," *Architecting Systems with Trustworthy Components*, vol. 3938, pp. 193–215, 2006.