A Mediator-Based Approach for Process Mediation of Web Services

Liyi Zhang
Center for Studies of Information Resources, Wuhan University, Wuhan, China
Email: lyzhang@whu.edu.cn

Si Zhou[1] and Zhefeng Sun[2]
1. School of Information Management, Wuhan University, Wuhan, China
2. Department of Computer Science, Huazhong Normal University Wuhan, China
Email: si_chow@msn.com, Sunzhefengsandy@sina.com

***Abstract*—Although several efforts were made to support the standardization of Web Services (such as WSDL, UDDI, SOAP, BPEL, etc.), they are not always perfectly compatible to each other due to the distributed nature. Both data and process heterogeneity required and offered by services' requesters and providers hamper the usability of Web services, thus service mediation becomes one key working area in SOA. While the former has received considerable attention, process mediation is still open and current approaches provide only partial solutions. In this paper, a mediator-based approach for process mediation of web services is proposed to adjust the partially compatible messages interchange behaviors to suit the requested/expected interface of each party. Based on the identification of message exchanging sequences in service interactions, several basic process mismatch patterns are presented to develop basic mediator patterns, which can be used to modularly construct advanced mediators that can resolve all possible process mismatches.**

***Index Terms*—web service, mediator, mediation, process, compatibility**

I. INTRODUCTION

In recent years, web services have become an active research area in both academia and industry. Web services which decouple application interfaces from implementations and use XML-based languages (usually WSDL) to describe the interfaces, were born as a solution to (or at least as a simplification of) the integration problem [1]. The main benefit they bring is that of standardization, in terms of data format (XML), interface definition language (WSDL), transport mechanism (SOAP) and many other interoperability aspects. Standardization reduces heterogeneity and makes it therefore easier to develop business logic that integrates different (Web service-based) applications. Web services also represent the most promising technologies for the realization of service-oriented architectures (SOAs), not only within but also outside companies' boundaries, as they are designed to enable loosely-coupled, distributed interaction [2].

However, web services are not always perfectly compatible due to its principle of decentralization and autonomy. In fact, although the lower levels of the interaction stacks are standardized, different Web services may still be represented using different languages and different terminologies of the same domain, similarly their functionalities are described in different ways and expect the clients to align with various interaction patterns in order to consume them.

An effective solution to this challenge is service mediation which is recognized as the act of reconciling existing services by intercepting, storing, transforming, and (re-) routing messages going into and out of these services [3]. Generally, service mediation can be classified into data mediation and process mediation. Data mediation, where the focus is on message types, has received considerable attention [4]. In comparison, process mediation, where the focus is on resolving mismatches occurring at the communication behaviors between services, is still open.

In the remainder of this paper, we present in section 2 the motivation of our work. In section 3, we describe the general solution approach. A services compatibility checking mechanism is offered in section 4. Six basic mediator patterns are proposed in section 5. Then, we explain their configurability and compositionability in section 6. Finally, Section 7 summarizes our conclusions and future work.

II. MOTIVATION

A. Related Work

Process mediation is still a poorly explored research field, in the context of web services. The most existing work represents only visions of mediator systems able to resolve in a (semi-) automatic manner the processes heterogeneity problems, without presenting sufficient details about their architectural elements. Still, these visions represent the starting points and valuable references for the future concrete implementations.

Benatallah et al. [5] identify a set of "mismatch patterns" between behavioural interfaces and provide templates of BPEL code that developers may reuse to build adaptors that resolve these mismatches. However, the compositionality of these BPEL templates is not

doi:10.4304/jsw.5.8.915-922

considered and thus the approach is not systematic. Similar mismatch patterns are identified in [6] where high-level architectures for addressing such mismatches are proposed. Altenhofen et al. [7] propose a formal model for process mediation based on Abstract State Machine (ASM) specifications. They show how these ASMs can be refined to deal with mismatch patterns such as those identified in [6]. In [8], the authors present the purpose of a process mediator within WSMX, which is a message broker among the partners. Process mediator needs to decide which data belongs to which partner(s) based on choreography and ontology of the partner(s). This work extends process mediation to multi-lateral interactions, and focuses on message forwarding among the partners. However, this data distribution among the partners is actually, only a part of task that should be addressed by process mediation. Fuchs [9] proposes another approach to interface adaptation. However, this contribution focuses on reconciling operational differences such as security policies, service level agreement, etc.

B. Problem Definition

Usually the service requester and the potential service providers have their own communication patterns which determined by each behavior interfaces to express how them want to communicate with each other. Unfortunately, those interfaces are defined separately, so the two parties will not be able to directly communicate, even if they can understand the same data formats. The existing researches have identified this kind of mismatches [6] [10]. However, few paper claims its identification is complete in any sense. To achieve a complete identification, we have proposed four basic mismatch patterns. Particularly, we have pointed out that all possible process mismatches can be composed by these basic patterns.

- Mismatches of unexpected messages. One of the interfaces has some extra messages the corresponding interface does not expect to send/receive. Or one of the interfaces does not have some messages the other interface expects to send/receive.
- Mismatches of message granularity. One of the interfaces has some messages the corresponding interface expects to split to send/receive. Or one of the interfaces has some messages the corresponding interface expects to merge to send/receive.
- Mismatches of message order. One of the interfaces sends the messages in a different order than the corresponding interface expects to receive them.
- Mismatches of unexpected conditions. One of the interfaces has some extra conditions imposed on control flow while the corresponding interface expects no conditions.

In order to communicate they must be able either to redefine their communication patterns (at least one of them has to) or to use an external mediation system as part of the process. The first solution is generally a very expensive one implying changes in the entities' business logic, and it is not suitable in a dynamic environment since every participant would have to readjust its pattern (through re-programming) each time it gets involved in a new partnership. As a consequence, the role of the mediator system will be to compensate the communication patterns in order to obtain equivalent processes.

A set of assumptions are made regarding the two parties to mediate between:

- Each of the parties has to make public the expected/requested way of interoperating with its partner.
- As many exiting solutions of data mediation, which is the prerequisite of process mediation, aim to the semantic coordination in manner of ontologies, the messages exchanged between the two parties have to contain data represented in terms of the used ontologies, that is, ontology instances.
- The heterogeneity problems at data level are resolved by a Data Mediator. This implies that a failure of the Data Mediator in solving the data heterogeneity problems has as a direct effect the failure of the Process Mediator.

The scope of the process mediator is to make this conversation possible by the use of different techniques as message blocking, message splitting or aggregation, acknowledgements generation and so on.

C. Motivating Example

As shown in Figure 1, a motivating example comes from a composition scenario of a company M and a company N. It is presumed that the two services from the companies, CM and CN respectively, provide complementary functionality. However, they do not fit each other exactly, due to process mismatches identified

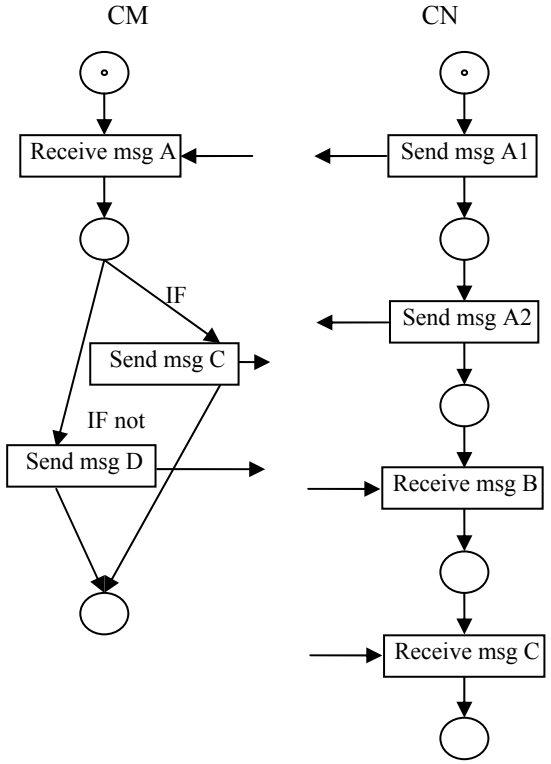


Figure 1. A motivating example of service composition with process mismatches

as follows:

• CM expects a whole message A which includes A1 and A2, while CN sends them separately.

• CN expects a message B which CM don't send.

• CM sends message C only when condition x is satisfied, or sends message D. But CN always wants to receive message C.

To make the two services compatibly interact with each other, process mismatches between them needs to be identified so that appropriate mediator patterns are selected to reconcile the mismatches.

III. Solution Approach

In this section, we present a solution approach to address process mediation, as shown in Figure 2. There are three steps, described as mismatches identification, mediator generation, and mediator implementation.

A. Mismatches identification

The interface of a web service is currently described by WSDL. The WSDL interface defines the messages exchanged between the described web service and an invoking application. However, the WSDL interface does not define sequences of message exchanges within complex interactions. The emerging specifications WS-BPEL [11] and WS-CDL [12] have made a step forward to cope with this requirement: they both can define a complex message exchange sequence on top of WSDL message definitions. For the purpose of mismatch identification, communication behaviors of service are abstracted and described using formal models (which illustrated in section 4). Then, in terms of basic mismatch patterns, developers analyze the actions between two interacting services and identify all possible process mismatches.

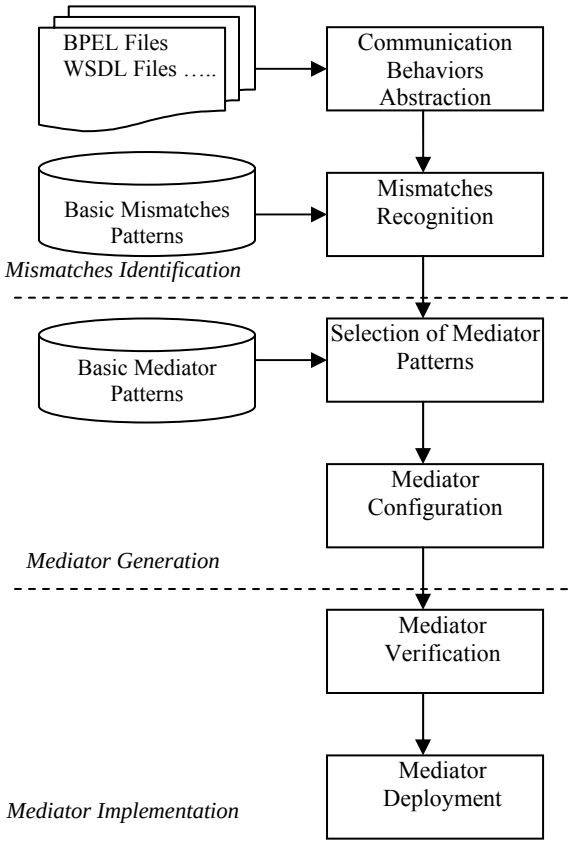


Figure 2. Solution approach to process mediation

B. Mediator generation

Firstly, with basic process mismatches, developers select corresponding mediator patterns which are proposed in section 5. Then, the structures and control logics of the mediator patterns need to be configured as parameters by developers, according to the identified mismatches. Finally, the configured mediator patterns are composed to construct a composite mediator that can resolve all identified process mismatches.

C. Mediator implementation

The mediator generated in the above procedure is only conceptual and should be placed between the two interacting services. The composition model of the two services and the mediator need to be formally verified. If any deadlock exists, we consider that the mediation has failed. Otherwise, the mediation is successful. After the Mediator verification, the conceptual mediator will be transformed to deployable/ executable service mediators, like BPEL-based mediators, which are pattern-specific codes and need developers' refinement.

IV. Compatibility Checking

Compatibility checking is the operation of both assessing the compatibility and identifying basic incompatibility factors of service interfaces, which is necessary for the mediator generation.

We defined a process as a set of communication actions which can be described in terms of communication action schemas. A communication action schema is a statement that a service may send or receive a message of a given type. We represent a communication action (ACT) as a tuple <AN, MT, MI, CON, MI^-, MI^+> where AN is the name of the action, MT indicates whether the action is inbound (receive) or outbound (send) with respect to the service being described, MI is the set of instances been received or send, CON indicates the conditional branch which cause this action, MI^-/MI^+ indicates the possible instances of preceding/next communication action in the same interface. $MI/CON/MI^-/MI^+$ can be NULL if there is no instance/conditional branch/preceding instance/next instance.

With the above notation, all the communication actions in the motivating example are listed in table 1.

TABLE I.
Communication Actions In Motivating Example

<CM.ReceivemsgA, inbound, (A1, A2), NULL, NULL, [C, D]>
<CM.SendmsgC, outbound, C, condition(x), (A1, A2), NULL>
<CM.SendmsgD, outbound, D, condition(~~x~~), (A1, A2), NULL>
<CN.SendmsgA1, outbound, A1, NULL, NULL, A2>
<CN.SendmsgA2, outbound, A2, NULL, A1, B>
<CN.ReceivemsgB, inbound, B, NULL, A2, C >
<CN.ReceivemsgC, inbound, C, NULL, B, NULL>

In the CM.SendmsgC, "condition(x)" indicates that the msgC is sent under the condition x. In the CM.SendmsgD, "condition(⁎)" indicates that the msgD is sent when the condition x doesn't hold. In the CM.ReceivemsgA, "[C, D]" indicates that one of C and D will be sent.

Considering a provided interface and a corresponding required interface, there is a set of instances MI_{in}= <I_1, I_2, …, I_n> which indicates all the instances with attribute of "inbound" from both interfaces, and a set of instances MI_{out}= <I_1, I_2, …, I_n> which indicates all the instances with attribute of "outbound". Likewise, there are CON_{in} and CON_{out}, MI^-_{in} and MI^-_{out}, MI^+_{in} and MI^+_{out}.

Then we introduce some rules for identifying process mismatches between the two interfaces.

Rule 1: For ACT_a, if $MI_a \neq MI_{in} \cap MI_{out}$, then there is a mismatch of unexpected messages.

Rule 2: For ACT_a and ACTb, if $(MI_a \cap MI_b \neq 0) \wedge (MT_a \neq MT_b) \wedge \{[MI_a - (MI_a \cap MI_b)] \cap (MI_{in} \cap MI_{out}) \neq 0\}$, then there is a mismatch of message granularity.

Rule 3: For ACT_a and ACT_b, if $(MI_a \cap MI_b \neq 0) \wedge (MT_a \neq MT_b) \wedge ([MI^-_a, MI^+_a] \neq [MI^-_b, MI^+_b])$, then there is a mismatch of message order.

Rule 4: For ACT_a and ACT_b, if $(MI_a \cap MI_b \neq 0) \wedge (MT_a \neq MT_b) \wedge (CON_a \neq CON_b)$, then there is a mismatch of unexpected conditions.

Rule 5: If $(MI_{in} = MI_{out}) \wedge (CON_{in} = CON_{out}) \wedge (MI^-_{in} = MI^-_{out}) \wedge (MI^+_{in} = MI^+_{out})$, then there is no mismatch.

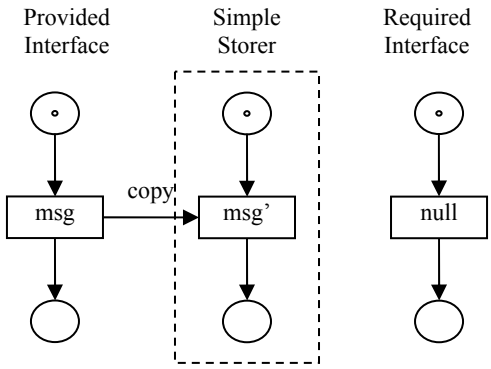
V. Basic Mediator Patterns

Six basic mediator patterns are proposed in this section. It should be pointed out that the six basic mediators can be treated as basic patterns to modularly construct service mediators which can be used to resolve all possible process mismatches. In addition, the basic mediators presented in this paper are conceptual patterns which can provide pseudo-code to develop executable codes for mediation, like BPEL code. The intended benefit of this work is to help developers produce service mediators through an engineering methodology and semi-automatically generate mediation codes by using these patterns.

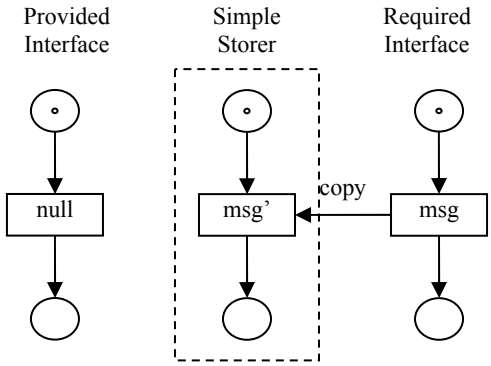
A. Simple Storer pattern

The Simple Storer is a service with the capability of simply copying, storing, and transmitting messages of certain specific type.

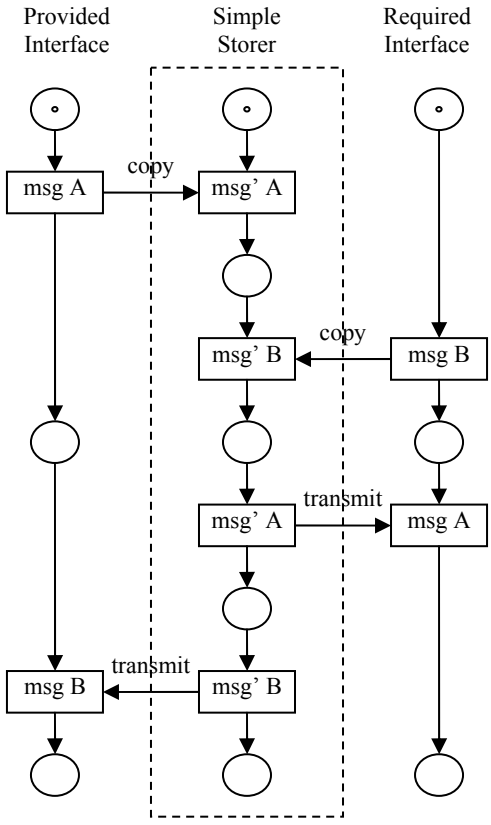
The Simple Storer pattern can be used to resolve mismatches of extra sending messages, missing receiving messages, and message order. The three scenarios of using Simple Storer pattern are respectively illustrated in Figure 3(a), Figure 3(b),and Figure 3(c). And the structures of Simple Storer pattern are distinguished with dashed squares. In the figures of this paper, the null depict those actions without sending/receiving any message, the "msg" depict those messages defined by interface, the "msg'" depict those messages generated by basic mediator, and the symbols "copy" and "transmit" stand for certain action type.



(a) Extra sending message scenario



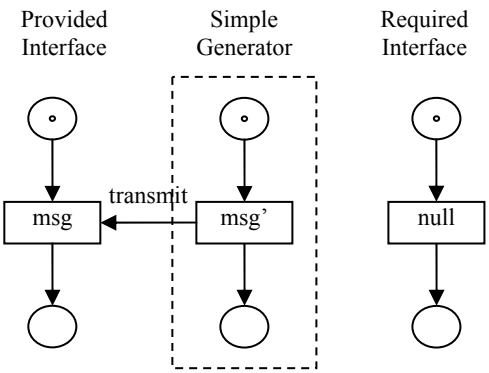
(b) Missing receiving message scenario



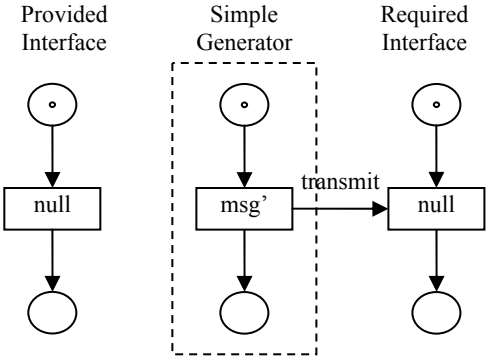
(c) Message order scenario

Figure 3. Scenarios of using Simple Storer

B. Simple Generator pattern



(a) Extra receiving message scenario



(a) Missing sending message scenario

Figure 4. Scenarios of using Simple Generator

The Simple Generator is a service with the capability of simply generating and transmitting messages of certain specific type. It should be pointed out that how to construct a message of certain type from a collection of incoming messages is a non-trivial task and some evidences can be used to address the issue [13].

The Simple Generator pattern can be used to resolve mismatches of extra receiving messages and missing sending messages. The two scenarios of using Simple Constructor pattern are respectively illustrated in Figure 4(a) and Figure 4(b). And the structures of Simple generator pattern are distinguished with dashed squares.

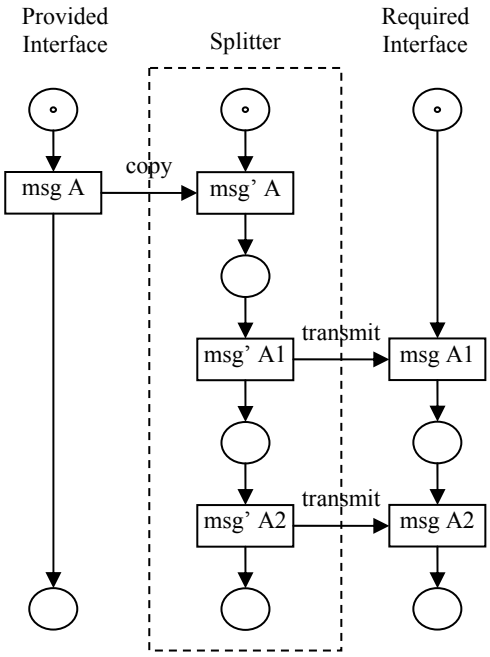


Figure 6. Scenario of using Splitter pattern

C. Splitter pattern

The Splitter is a service with the capability of copying a single message of certain type and splitting it to two or more partial messages. The specific structure of Splitter pattern is variable according to the sequence of partial messages which may be sequential, parallel or mixed structure. If splitting to two partial messages, the structure of Splitter pattern can be two types, as shown in Figure 5(a) and Figure 5(b).

The Splitter pattern can be used to resolve mismatches of splitting sending messages and combiner receiving messages. We only show the former scenario in Figure 6 because they are quite similar. And the structures of

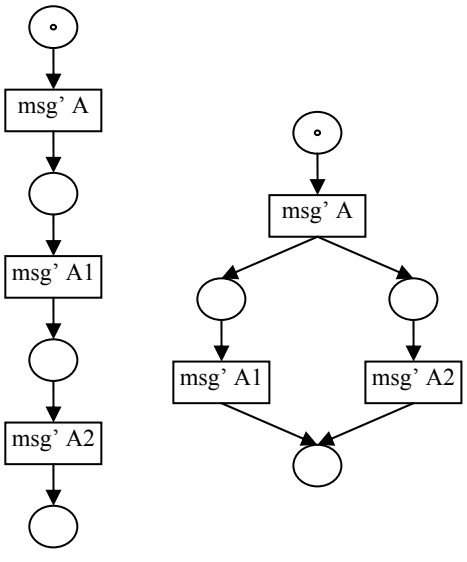


Figure 5. Two types of structures of Splitter pattern with two partial messages

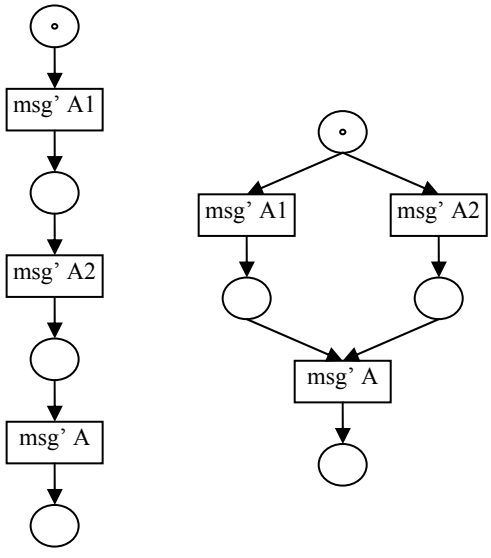


Figure 7. Two types of structures of Combiner pattern with two merged messages

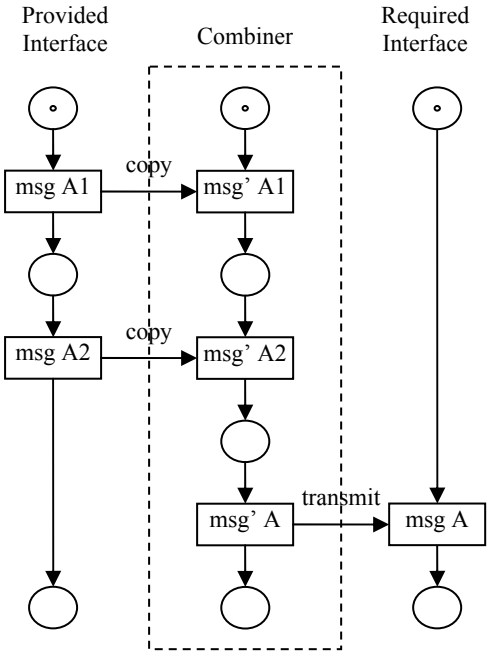


Figure 8. Scenario of using Combiner pattern

Splitter pattern are distinguished with dashed squares.

D. Combiner pattern

The Combiner is a service with the capability of copying two or more partial messages and combining them to a single one. Similar to Splitter pattern, the specific structure of Merger pattern is variable according to the sequence of merged messages which may be sequential, parallel or mixed structure. If combining two messages, the structure of Merger pattern can be two types, as shown in Figure 7(a) and Figure 7(b).

The Combiner pattern can be used to resolve mismatches of splitting receiving messages and combiner sending messages. We only show the former scenario in Figure 8 because they are quite similar. And the structures of Merger pattern are distinguished with dashed squares.

E. Storing Controller pattern

The Storing Controller is a service with the capability

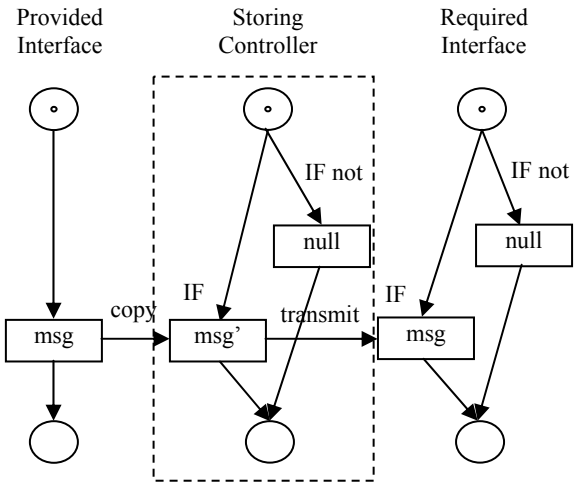


Figure 9. Scenario of using Storing Controller pattern

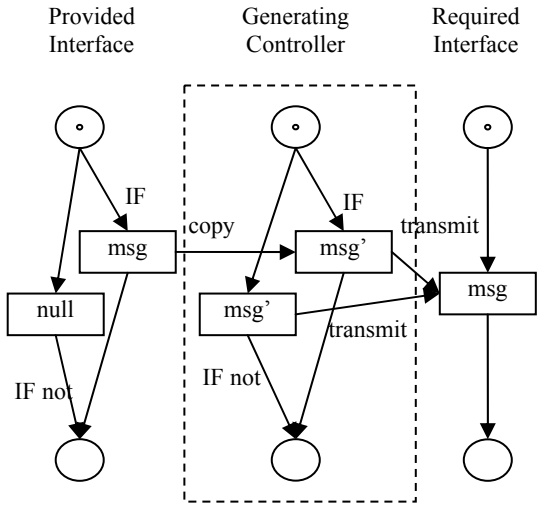


Figure 10. Scenario of using Generating Controller pattern

of storing and conditionally sending some messages of certain type in terms of specific logic.

The Storing Controller pattern can be used to resolve mismatches of extra condition of receiving messages and missing condition of sending messages. We only show the former scenario in Figure 9 because they are quite similar. And the structures of Storing Controller pattern are distinguished with dashed squares.

F. Generating Controller pattern

The Generating Controller is a service with the capability of conditionally generating and sending some messages of certain type in terms of specific logic.

The Generating Controller pattern can be used to resolve mismatches of extra condition of sending messages and missing condition of receiving messages. We only show the former scenario in Figure 10 because they are quite similar. And the structures of Generating Controller pattern are distinguished with dashed squares.

VI. Mediator Configuration and Composition

As mentioned above, some basic mediator patterns are not pre-established, like Splitter, Combiner, Storing Controller and Generating Controller patterns. Thus, specific interfaces should be provided for the basic mediator patterns to configure their structures and control logics.

Before using the Splitter and Combiner patterns, developers should specify the quantities of partial messages which involved as well as the sequence of these messages, that is, sequential, parallel or mixed structure. After configuration, the specific structures of the Splitter and Combiner patterns can be identified and concretized.

When resolving unexpected condition mismatches, developers should specify the condition constraints of the Storing Controller and Generating Controller patterns, according to the condition of the provided or required interfaces of services to be composed. The condition constraints are eventually transformed to such BPEL elements as <switch>, <pick>, <while>, <flow> or <repeatUntil> [14].

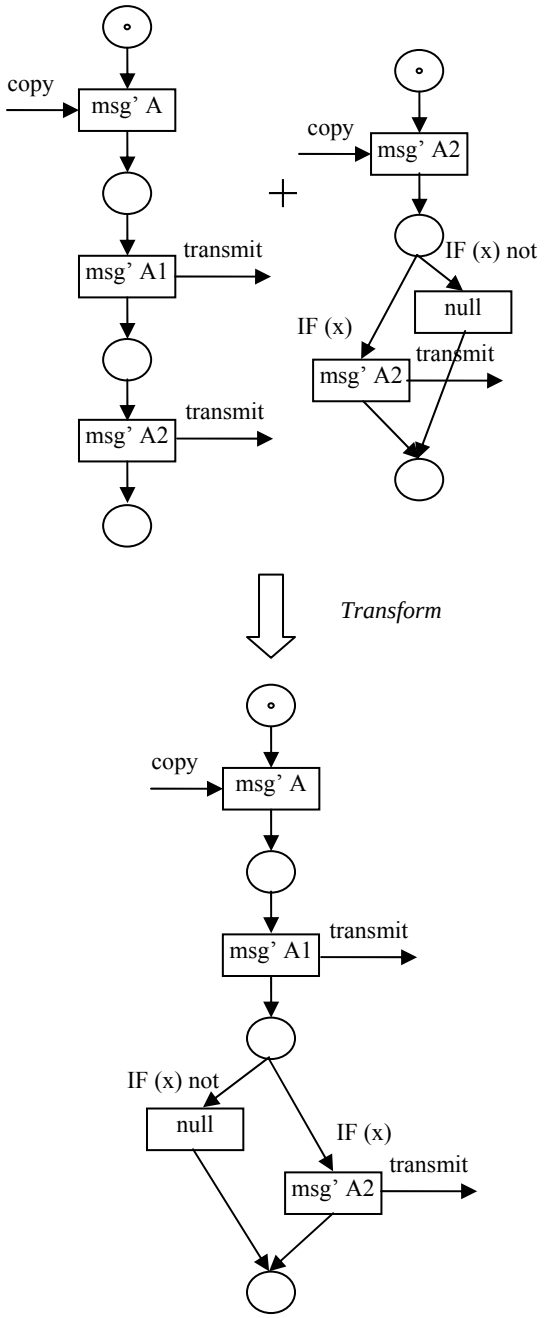


Figure 11. Compositionability of basic mediator patterns

The four basic process mismatches mentioned in section 2 can be resolved by the basic mediator patterns. However, process mismatches are more complicated in practical environments, and should be addressed by advanced mediators with control logics which are composed by these basic mediators. Then a composite mediator can be considered as an integrated one with sophisticated structure, and be used in the future. Each mediator presented in this paper has an initial place and an end place (illustrated by circles in each figure). Informally, the composition of two mediators is performed by merging the end place of one mediator with the initial place of the other as well as the common parts of the two mediators. To illustrate the composition of mediators, herein take a mediator composed by two basic

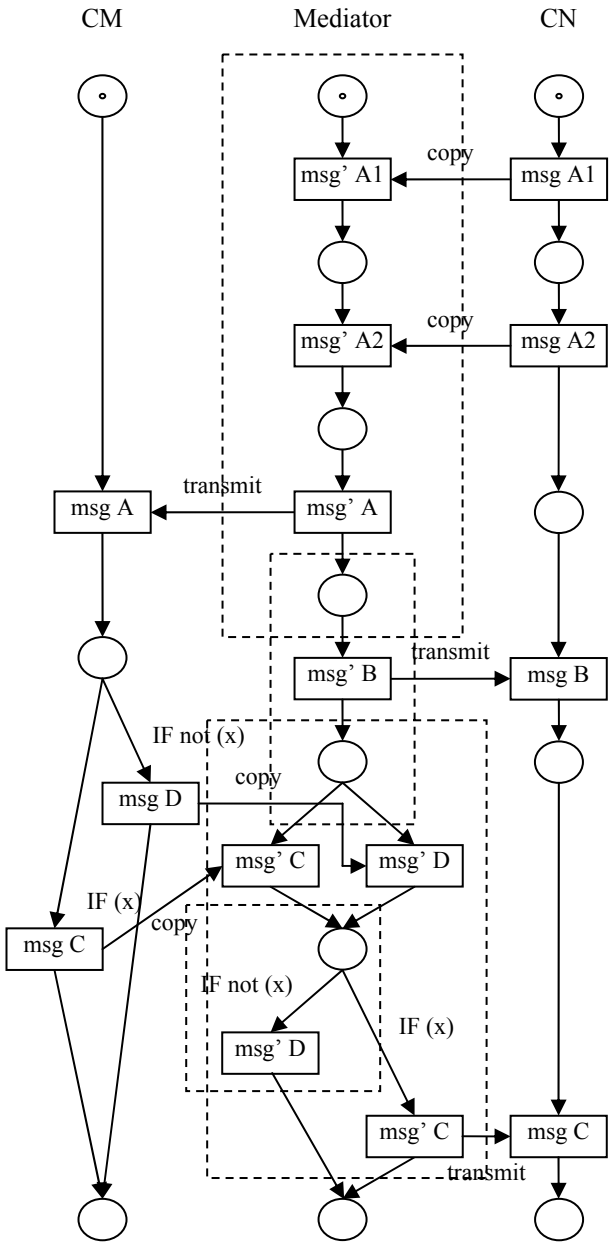


Figure 12. A composite mediator for protocol mediation of CM and CN

mediator patterns (which are Splitter pattern and Storing Controller pattern respectively) as shown in Figure 11. It's easy to see that message A is divided into message A1 and message A2, and A2 will be sent if condition x occurs.

For the motivating example, there are three process mismatches can be found out, and four mediator patterns can be respectively used to address these mismatches as follows:

- A Combiner can be used to receive the messages A1 and A2 from CN, and then it sends message A to CM.
- A Simple generator can be used to construct the message B and send to CN.
- A Storing Controller can be used to send message C until the condition x is satisfied.

• A Simple Storer can be used to handle the message D which CN don't want to receive.

As shown in Figure 12, a composite mediator composed by the above four mediator patterns sits between the two interacting services, CM and CN, and compensates their process mismatches. The four mediator patterns are distinguished with dashed squares.

VIII. Conclusions and Future Work

The main contributions that we have achieved in the paper are:

• We have proposed a mediator-based solution approach to resolve most of possible process mismatches and glue partially compatible services together. Since we abstract the specific definitions of the service interface, the approach is not limited to BPEL-based services and can be used with other definition languages.

• We have presented several basic mediator patterns which are derived from the process mismatch patterns. The well-defined basic mediator patterns can be configured and composed by developers, according to the specific process mismatches.

• Identification and formalization of a set of atomic problems that can be automatically solved by a mediator.

• We have defined a process mismatch identification mechanism.

In the future, we plan to focus on the formal approach to verification of the correctness of service mediation. And a systematic solution is expected to be investigated. In addition, further effort will be made to implement the prototype system.

Acknowledgment

The authors were supported in part by the MOE Project of Key Research Institute of Humanities and Social Science in Chinese Universities (NO: 07JJD870220). The authors would like to express our sincere gratitude to the contributing author and to the referees for reviewing papers for this special issue.

References

[1] G. Alonso, F. Casati, H. Kuno, V. Machiraju, *Web Services: Concepts, Archtectures, and Applications.* Springer Verlag, 2004.

[2] B. Benatallah, F. Casati, F. Toumani, "Web services conversation modeling: A Cornerstone for E-Business Automation," *IEEE Internet Computing,* vol. 8, no.1, pp.46-53, January/February 2004.

[3] M. Dumas, M. Spork, and K. Wang, "Adapt or Perish, Algebra and Visual Notation for Service Interface Adaptation", the 4th Intl. Conf. on Business Process Management, pp. 65-80, 2006.

[4] M. Mrissa, C. Ghedira, D. Benslimane, and Z. Maamar, "Context Model for Semantic Mediation in Web Services Composition", *Lecture Notes in Computer Science*, Vol 4215, pp.12-25, October 2006.

[5] B. Benatallah, F. Casati, D. Grigori, H. R. Motahari Nezhad, and F. Toumani, "Developing Adapters for Web Services Integration", The 17th International Conference on Advanced Information System Engineering, CAiSE 2005, pp.415–429, 2005.

[6] E. Cimpian and A. Mocan, "WSMX Process Mediation Based on Choreographies" BPM 2005 Workshops, LNCS 3812, pp.130–143, 2005.

[7] M. Altenhofen, E. B¨orger, and J. Lemcke, "An abstract model for process mediation", The 7th International Conference on Formal Engineering Methods (ICFEM), pp.81–95, 2005.

[8] T. Haselwanter, P. Kotinurmi, M. Moran, T. Vitvar, and M. Zaremba, "WSMX: A Semantic Service Oriented Middleware for B2B Integration", The 4th International Conference on Service-Oriented Computing (ICSOC 2006), pp.477-483, 2006.

[9] M. Fuchs, "Adapting web services in a heterogeneous environment", the Second IEEE International Conference on Web Services, ICWS 2004, pp.656–664, 2004.

[10] X. Li, Y. Fan, and F.Jiang, "A Classification of Service Composition Mismatches to Support Service Mediation", The Sixth International Conference on Grid and Cooperative Computing, pp.315-321, August 2007.

[11] http://www.oasis-open.org/committees/wsbpel/

[12] http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/

[13] H.R. Motahari Nezhad, A. Martens, and F. Curbera, et al., "Semi-Automated Adaptation of Service Interactions", Proc. of the 16th Intl. World Wide Web Conference, pp. 993-1002, 2007.

[14] X. Li, Y. Fan, J. Wang, L. Wang, and F. Jiang, "A Pattern-Based Approach to Development of Service Mediators for Protocol Mediation", the Seventh Working IEEE/IFIP Conference on Software Architecture, pp.137-146, 2008.

Liyi Zhang received the B.S. and Ph.D. degrees from Wuhan University, Wuhan, China, in 1988 and 1999, respectively.

He is currently a professor and DEAN OF DEPARTMENT of Information & E-commerce in School of Information Management, Wuhan University, Wuhan, China. He has published five books, over 40 Journal papers. In addition, he has organized several conferences in the emerging areas of Electronic Commerce. His research interests include information system, e-commerce and information retrieval.

Mr. Zhang is a member of E-commerce Major Guiding Committee of China, the Secretary-general of Association of Hubei Electronic Commerce, and a member of AIS (Association of Information System).

Si Zhou received the B.S. and Master degrees from Huazhong Normal University, Wuhan, China, in 2004 and 2007, respectively.

He is currently a Ph.D. candidate of electronic commerce, Wuhan University, Wuhan, China. His research interests include SOA, e-commerce and web services.

Zhefeng Sun is currently an undergraduate student of computer science and technology, Huazhong Normal University, Wuhan, China.