Abstract—Although several efforts were made to support the standardization of Web Services, they are not always perfectly compatible to each other due to the distributed nature. In this paper, a novel 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.

II. PROCESS MISMATCHES OF WEB SERVICE

Usually the service requester and the potential service providers have their own communication patterns which determined by each behavior interfaces to express how they 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 [5] [6]. 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 send/receive.

• Mismatches of unexpected conditions. One of the interfaces has some extra conditions imposed on control flow while the corresponding interface expects no conditions.

III. SOLUTION APPROACH

In this section, we present a solution approach to address process mediation. 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 service and an invoking application. However, the WSDL interface does not define sequences of message exchanges within complex interactions. The emerging specifications WS-BPEL [7] and WS-CDL [8] 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.

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 pattern 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 <\text{AN}, MT, MI, CON, MI’, MT’> 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.

Considering a provided interface and a corresponding required interface, there is a set of instances \(\text{MI}_{\text{in}}\) = \(\{I_1, I_2, \ldots, I_p\}\) which indicates all the instances with attribute of “inbound” from both interfaces, and a set of instances \(\text{MI}_{\text{out}}\) = \(\{I_1, I_2, \ldots, I_p\}\) which indicates all the instances with attribute of “outbound”. Likewise, there are \(\text{CON}_{\text{in}}\) and \(\text{CON}_{\text{out}}, \text{MI}’_{\text{in}}\) and \(\text{MI}’_{\text{out}}\).

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

Rule 1: For \(\text{ACT}_a\), if \(\text{MI}_a \neq \text{MI}_{\text{in}} \cap \text{MI}_{\text{out}}\), then there is a mismatch of unexpected messages.

Rule 2: For \(\text{ACT}_a\) and \(\text{ACT}_b\), if \((\text{MI}_{a} \cap \text{MI}_{b} \neq 0) \land (\text{MT}_{a} \neq \text{MT}_{b}) \land (\{\text{MI}_{a} - (\text{MI}_{a} \cap \text{MI}_{b})\} \cap (\text{MI}_{b} \cap \text{MI}_{\text{out}}) \neq 0)\}, then there is a mismatch of message granularity.

Rule 3: For \(\text{ACT}_a\) and \(\text{ACT}_b\), if \((\text{MI}_{a} \cap \text{MI}_{b} \neq 0) \land (\text{MT}_{a} \neq \text{MT}_{b}) \land (\{\text{MI}'_{a}, \text{MI}'_{b}\} \neq \{\text{MI}'_{a}, \text{MI}'_{b}\})\), then there is a mismatch of message order.

Rule 4: For \(\text{ACT}_a\) and \(\text{ACT}_b\), if \((\text{MI}_{a} \cap \text{MI}_{b} \neq 0) \land (\text{MT}_{a} \neq \text{MT}_{b}) \land (\text{CON}_{a} \neq \text{CON}_{b})\), then there is a mismatch of expected conditions.

Rule 5: If \((\text{MI}_{\text{in}} = \text{MI}_{\text{out}}) \land (\text{CON}_{\text{in}} = \text{CON}_{\text{out}}) \land (\text{MI}_{\text{in}} = \text{MI}'_{\text{out}})\), 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 proposed 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

It is a service with the capability of simply storing 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.

B. Simple Generator pattern

It 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 [9]. The Simple Generator pattern can be used to resolve mismatches of extra receiving messages and missing sending messages.

C. Splitter pattern

It is a service with the capability of copying a single message of certain type and splitting it to two or more parts.
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. The Splitter pattern can be used to resolve mismatches of splitting sending messages and combiner receiving messages.

D. Combiner pattern

It 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. The Combiner pattern can be used to resolve mismatches of splitting receiving messages and combiner sending messages.

E. Storing Controller pattern

It is a service with the capability 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.

F. Generating Controller pattern

It 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.

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>.

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. 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.

VIII. A RUNNING EXAMPLE

As shown in Figure 1, a running 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 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.

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 2, 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.

**IX. 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**