WSMX Process Mediation Based on Choreographies

Emilia Cimpian, Adrian Mocan

DERI, National University of Ireland, Galway, Ireland
emilia.cimpian@deri.org, adrian.mocan@deri.org

Abstract. One of the most difficult obstacles Web Services have to overcome in the attempt to exploit the true potential of the World Wide Web is heterogeneity. Caused by the nature of the Web itself, heterogeneity problems occur both at data level as well as at behavioral level of business logics, message exchange protocol and Web Service invocation.
Process mediation is one of the crucial points on the road towards establishing new, ad-hoc cooperation on the web between various business partners. If semantic enhanced data enables dynamic solutions for coping with data heterogeneity, semantically enhanced Web Services can do the same for behavioral heterogeneity. Based on Web Service Modeling Ontology (WSMO) specifications that offers support in semantically describing Web Services, we propose a solution that acts on these semantic descriptions and offers the means for defining of what we call a Process Mediator. Such a mediator acts on the public processes (represented as WSMO choreographies) of the parties involved in a communication and adjust the bi-directional flow of messages to suit the requested/expected behavior of each party.

1 Introduction

The advantages offered by the huge amount of information available and by the higher and higher number of services deployed on the Web are overcame by the inherent heterogeneity issues existing between all these resources. The information is represented using different languages and different conceptualizations of the same domain; similarly Web Services describe their functionalities in different ways and expect the clients to align with various interaction patterns in order to consume their functionalities.

Numerous approaches are proposing various solutions to cope with data heterogeneity by adding semantic meaning to data, and make it machine understandable. Even if this area is well explored and many semi-automatic solutions are available for this problem there is one more gap to fill: how the semantic enriched data is interchanged by machines. That is, even if the machine can understand the data they receive, they have to understand the communication process it is part of, as well. As a consequence, a coherent mode of describing the expected interaction patterns is necessary, together with means of mediating between heterogeneous patterns.

The Web Service Modeling Ontology (WSMO) working group started an initiative for standardizing various aspects related to Semantic Web Services. The objective of WSMO and its surrounding efforts is to define a coherent technology for Semantic Web Services by providing means for (semi-)automatic discovery, composition and execution of Web Services which are based on logical inference mechanisms. Web Service Execution Environment (WSMX) [3] is a reference implementation for WSMO, a proof of concept for this specification, aiming to provide reference implementations for the main tasks related to Web Services as envisioned by WSMO.

In this context we propose a solution able to cope with the differences in the way a requester wants to consume the functionality of a Web Service and the way this functionality is made available by the Web Service to the requester. We use WSMO choreography to describe the expected/requested behavior of the two parties, which is in fact a formalization of their public business processes. Using these descriptions and the services of a data mediator (to solve data heterogeneity problems) we introduce the process mediator, a system able to adjust the two parties' behavior and to enable their communication.

In this paper we will present WSMX approach for process mediation, and the WSMX Process Mediator component. Section 2 presents the motivation for developing such a mediator and Section 3 briefly presents WSMO and WSMX. The WSMX Process Mediator is presented in Section 4, followed by an example (Section 5). The final parts of this document present some related efforts in this area (Section 6) and conclusions (Section 6).

2 Motivation

2.1 Overview

The simple scenario of invoking a (Semantic) Web Service may become extremely complicated in a heterogeneous environment such as the existing web.

Usually the client has its own communication pattern (expressing how it wants to communicate with a service) that in general is different from the one used by the corresponding Web Service (which expresses how the service wants to be invoked). As a consequence the two parties will not be able to directly communicate, even if they can understand the same data formats. 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.

2.2 Problem Definition

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

– Each of the parties have to make public the expected/requested way of inter-operating with its partner. Conform to WSMO these represent choreography descriptions and they are included in the goal's interface and in the Web Service's description.
– The involved parties have to refer from their choreographies the ontologies they used to describe their domain. Furthermore these ontologies have to be available and the heterogeneity problems between them 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 messages exchanged between the two parties have to contain data represented in terms of the used ontologies, that is, ontology instances.

The scope of the Process Mediator is to make this conversation possible by the use of different technics as message blocking, message splitting or aggregation, acknowledgements generation and so on. The process mediator is part of WSMX and it can make use of all the functionalities provided by WSMX regarding message receiving and sending, keeping track of the ongoing conversation, access various Data Mediators, resources and so on.

3 WSMO and WSMX

The Web Service Modeling Ontology (WSMO) is a formal ontology for describing various aspects related to Semantic Web Services.

WSMO defines four main modeling elements for describing several aspects of Semantic Web Services: *ontologies*, *Web Services*, *goals* and *mediators*. In what follows, we will describe all these elements, insisting on their importance in reaching a truly Semantic Web Service technology.

As defined in [6], *ontologies* are formal explicit specifications of shared conceptualizations. In WSMO they represent key elements, having a twofold purpose: firstly they define the information's formal semantics and secondly, they allow to link machine and human terminologies. The WSMO ontologies give meaning to the other elements (Web Services, goals and mediators), and provide common semantics, understandable by all the involved entities (both humans and machines).

In WSMO, requestors of a service express their objectives as *goals*, which are high level descriptions of concrete tasks. Every requestor expresses its goal in terms of its own ontology, which, on one hand provides the means for a human user to understand the goal, and on the other hand, allows a machine to interpret it as part of the requestor's ontology. Another advantage of using the goals is that the requestor only has to provide a declarative specification of what it wants, and does not need to have a fixed relation with the Web Service or to browse through an UDDI registry for finding Web Services that provide the appropriate capability.

In order for this goal to be accomplished, the requestor (by means of its information system) has to find an appropriate *Web Service* which may fulfill

the required task. Similar to the way the requestor declares its goal, every Web Service has to declare its capability (that is, what it is able to accomplish) in terms of its own ontology. If the requestor of the service and the Web Service that offers it use the same ontology the matching between the goal and the capability can be directly established. Unfortunately, in most of the cases they use different ontologies, and the equivalence between the goal and the capability can be determined only if a third party is consulted for determining the similarities between the two ontologies. Another problem that may appear is the impossibility of the requester and of the provider of the service to communicate with each other, the reason for this being the heterogeneity of their communication protocols. For these reasons, WSMO introduces the fourth key modeling element: the *mediators*, which have the task of overcoming the heterogeneity problems, both at data level and at communication level.

The Web Service Execution Environment (WSMX) is the reference implementation for WSMO, designed to allow dynamic discovery, invocation and composition of Web Services. WSMX offers complete support for interacting with Semantic Web Services. In addition, WSMX supports the interaction with non-WSMO, but classical Web Services ensuring that a seamless interaction with existing Web Services is possible.

By using WSMX, if two partners want to interoperate with each other, they only have to expose their own functionality and to consume the functionality offered by their partners. Additionally, there can be the case that one entity wants to get involved in ad-hoc business processes without knowing its partner beforehand. In order to accommodate this situation, WSMX offers mechanisms and strategies for dynamic discovery of those entities that expose the desired capability. Furthermore, it extends this search in order to find out if multiple partners are able to fulfill the requester goal, by composing the offered sub-functionalities.

In any of the functionalities offered by WSMX (discovery, invocation and composition) mediation can be needed at both data [8] and process level (behavioral level) [2]. In the following chapter we will describe the WSMX process mediation approach.

4   Process Mediation Based on WSMO Choreography

For addressing the problem of process mediation we have to define first what a process means, and how we represent a WSMX process.

We adopt the standard definition of a process: collection of activities designed to produce a specific output for a particular customer, based on a specific inputs [1], an activity being a function, or a task that occurs over time and has recognizable results.

Depending on the considered level of granularity, each process can be seen as being composed of different, multiple processes. The smallest process possible consists of only one activity. Figure 1 presents a graphical representation of a

process obtain by combining multiple processes. The output of one process (or more processes) is considered the input of one or many other processes.
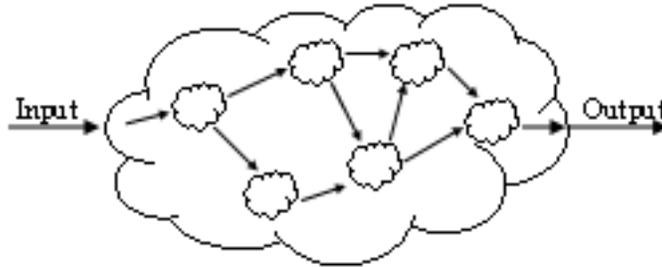
Fig. 1. Process consisting of multiple processes

One can distinguish between two type of processes: *private processes*, which are carried out internally by an organization, and usually are not visible to any other entity, and *public processes*, which are defining the behavior of the organization in collaboration with other entities [WSMF, 2002]. From the process mediation point of view we are interested only in the public processes, the private process not being visible to the exterior, can not be the object of WSMX mediation.

In what follows, we will define the WSMX public process representation, the problems this mediator intends to solve, the Process Mediator's interactions with other WSMX components, and we will describe step by step the process mediation algorithm.

4.1 Process Representation

WSMX process representation is similar with the WSMO choreography [9] definition. This dependency is a straight forward one considering that WSMX Process Mediator is dealing with the communication patterns heterogeneity, and that the WSMO choreography describes the behavior of the service from a user point of view (that is, how the user should interact with the Web Service in order to consume its functionality). In terms of WSMX, every choreography represents a public process, which means that WSMO choreography is a subclass of WSMX process.

In order for this paper to be self-contained we describe in the next paragraphs the main features of WSMX processes, i.e. the main features of WSMO choreography, as described in [9].

The representation of a WSMX business process is based on the Abstract State Machine [7] methodology, and it inherits the core principles of ASMs:

 – it is state-based;

- it represents a state by an algebra;
- it models state changes by guarded transition rules that change the values of functions and relations defined by the signature of the algebra.

A WSMX process consists of *states* and *guarded transitions* [9]. A state is described by a WSMO ontology, and is obtained from the ontology used by the owner of the process by:

- subclassifying all the concepts that the owner needs to make public in order to enable the communication, and
- adding an additional attribute `mode`, which shows who has the right of modifying the instances of the concept. This attribute can take the values:

> **static** - the concept cannot be changed;
> **controlled** - the concept can only be changed by its owner;
> **in** - the concept can only be changed by the environment;
> **shared** - the concept can be changed by its owner and by the environment;
> **out** - the concept can only be changed by its owner.

The guarded transitions (transition rule) are used to express changes of states by means of rules, expressible in the following form:

```
if Cond then Updates
```

`Cond` is an arbitrary Web Service Modeling Language (WSML) [4] axiom, formulated in the given signature of the state.

The `Updates` consist of arbitrary WSMO Ontology instances.

In the Semantic Web Services context the level of granularity for representing a certain process is strictly up to the owner of that process. Each action can be represented as a transition, which is the most detailed level, or more actions can be modelled using only one transition.

4.2 Addressed Problems

Usually a business communication consists of more than one exchange message, and as a consequence, finding the equivalences between the message exchange patterns of the two (or more) parties is not at all a trivial task. Intuitively, the easiest way of doing this is to first determine the mismatches, and than search for a solution of eliminating them. [5] identifies three possible cases that may appear during the message exchange:

Precise match. The two partners have exactly the same pattern in realizing the business process, which means that each of them sends the messages in exactly the order the other one requests them. In this ideal case the communication can take place without using a Process Mediator.

Resolvable message mismatch. This case appears when the two partners use different exchange partners, and several transformations have to be

performed in order to resolve the mismatches. For example when one partner sends multiple instances in a single message, but the other one expects them separately, the mediator can break the initial message, and send the instances one by one.

Unresolvable message mismatch. In this case, one of the partners expects a message that the other one do not intend to send. Unless the mediator can provide this message, the communication reaches a dead-end (one of the partners is waiting indefinitely).

In order to communicate two partners have to either define equivalent processes, or to use an external mediation system as part of the communication process. The mediator's role will be to transform the clients messages and/or Web Services messages, in order to obtain a sequence of equivalent processes.

As illustrated above, not all the communication mismatches can be solved by using a mediator, but only some of them. A list containing the initial set of resolvable mismatches that our mediator intends to address is provided below.

Stopping an unexpected message (Figure 2. a)): in case one of the partners sends a message that the other one does not want to receive, the mediator should just retain and store it. This message can be send later, if needed, or it will just be deleted after the communication ends.

Inversing the order of messages (Figure 2. b)): in case one of the partners sends the messages in a different order than the one the other partner wants to receive them. The messages that are not yet expected will be stored and sent when needed.
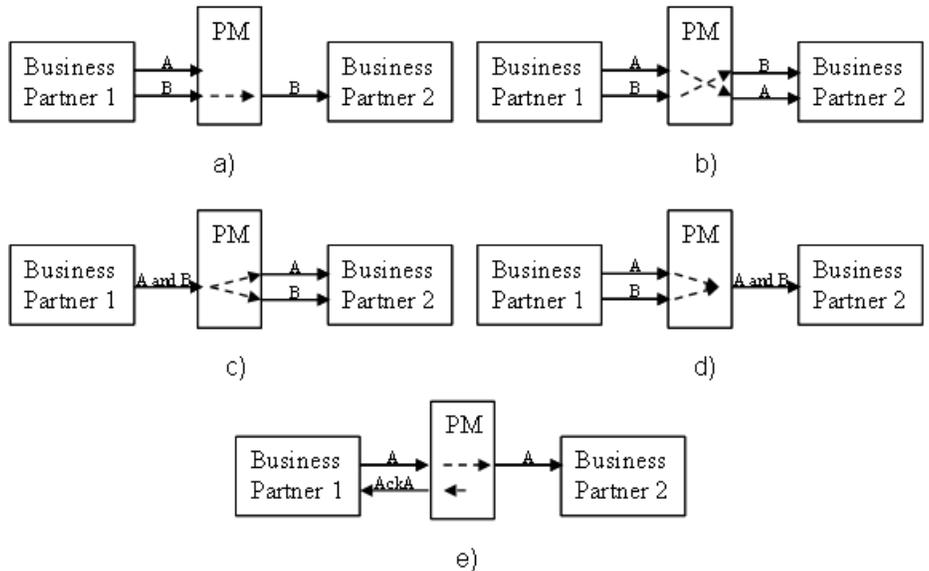
Fig. 2. Addresses Mismatches

Splitting a message (Figure 2. c)): in case one of the partners sends in a single message multiple information that the other one expects to receive in different messages.

Combining messages (Figure 2. d)): in case one of the partners expects a single message, containing information sent by the other one in multiple messages.

Sending a dummy acknowledgement (Figure 2. e)): in case one of the partners expects an acknowledgement for a certain message, and the other partner does not intend to send it, even if it receives the message.
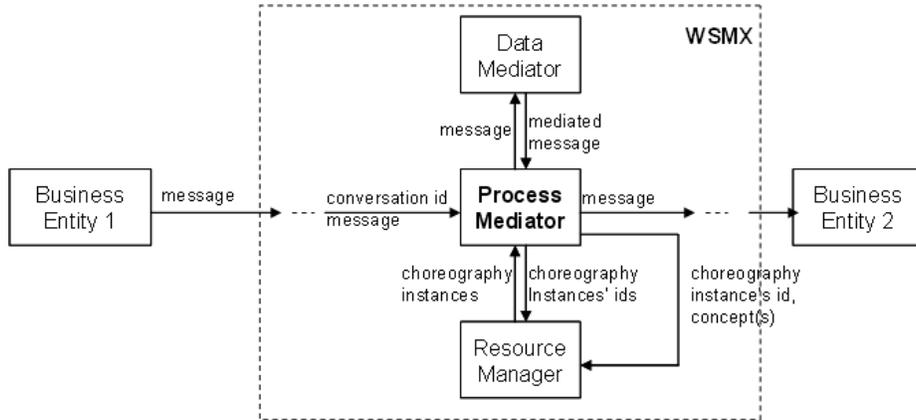
4.3   Process Mediator in WSMX

WSMX process mediation is concerned with determining how two public processes can be combined in order to provide certain functionality. In other words, how two business partners can communicate, considering their public processes.

When WSMX receives a message, either from the requestor of the service or from a Web Service, it has to check if it is the first message in a conversation. If it is the first, WSMX creates copies (instances) of both the sender and the targeted business partner choreographies, and stores these instances in a repository, together with a uniquely identifier of the conversation. If it is not the first message of a conversation, WSMX has to determine the conversation id. These computations performed on the message are done by two WSMX components, Communication Manager and Choreography Engine. Their descriptions are not included in this article, since they are not relevant from the process mediation's point of view; more information about various WSMX components can be found in [10].

After the id of the conversation is obtained, the Process Mediator (PM) receives it, together with the message; the message consists of instances of concepts from the sender's ontology. Based on the id, the PM loads the two choreography instances from the WSMX Repository, by invoking the WSMX Resource Manager. All the transformations performed by the PM will be done on this instances. In case different ontologies have been used for modeling the two choreographies, the PM has to invoke an external Data Mediator for transforming the message in terms of the target ontology.

After various internal computations (described in the next chapter) the PM determines if based on the incoming message it can generate any message expected by either one of the partners. The generation of any message determines a transformation in the chorography instance of the party that receives that message. Simultaneously with sending the message, the process mediator should update the choreography instances and revaluate all the rules, until no further updates are possible.

The interactions between the Process Mediator and other WSMX components is represented in Figure 3.

Fig. 3. Process Mediator interactions

4.4 Execution

The Process Mediator is triggered when it receives a message and a conversation id. The message contains instances of concepts, in terms of the sender's ontology. The conversation id uniquely identifies the instances of the choreographies involved in the communication.

After being invoked, the PM performs the following steps:

1. Loads the two choreography instances from the repository.

2. Adds the instances contained in the message to the corresponding choreography instance (the sender's choreography instance); this step is needed considering that the choreography instances contains the information prior to the transmission of the current message.

3. Mediates the incoming instances in terms of the targeted partner ontology, and checks if the targeted partner is expecting them, at any phase of the communication process. This is done by checking the value of the `mode` attribute, for the mediated instances' owner. If the attribute `mode` for a certain concept is set to `in` or `shared`, than this concept's instances may be needed at some point in time. The instances that are expected by the targeted partner are stored in an internal repository.

4. For all the instances from the repository, the PM has to check if they are expected at this phase of the communication, which is done by evaluating the transition rules. The evaluation of a rule will return the first condition that can not be fulfilled, that is, the next expected instance for that rule. This means that an instance is expected if it can trigger an action (not necessary to change a state, but to eliminate one condition for changing a state).

The possibility that various instances from this repository can be combined in order to obtain a single instance, expected by the targeted business partner is also considered.

5. Each time the PM determines that one instance is expected, it sends it, deletes it it from the repository, updates the targeted partner choreography instance, and restarts the evaluation process (step 4). When a transition rule can be executed, it is marked as such and not re-evaluated at further iterations.

The PM only checks if a transition rule can be executed, and not executes it, since it can not update any of the two choreography instances without receiving input from one of the communication partner. By evaluating a rule, the PM determines that one of the business partner can execute it, without expecting any other inputs.

This process stops when, after performing these checkings for all the instances from the repository, no new message is generated.

6. For each instance forwarded to the targeted partner, the PM has to check if the sender is expecting an acknowledgement. If the sender expects an acknowledgement, but the targeted partner does not intend to send it, the PM generate a dummy acknowledgement and sends it. Simultaneously, it updates the sender's choreography instance.

7. The PM checks all the sender's rules and mark the ones that can be executed.

8. The PM checks the requestor's rule, to see if all of them are marked; when all are marked, the communication is over and PM deletes all the data regarding this conversation, from both its internal repository and WSMX repository.

This algorithm is implemented by the PM in order to solve the communication heterogeneity problem.

5 Examples

We consider a Virtual Travel Agency (VTA) service, able to provide on-line tickets for certain routes, and a client who wants to invoke this service.

For keeping this example as simple as possible, we will present only parts of the requestor's and service's choreographies, but these parts are conclusive enough to illustrate how the Process Mediator works. Additionally, we will consider that some of the concepts have exactly the same semantic for both the service and the client.

5.1 Requestor and Provider's Choreographies

We consider that the two participants have the following concepts in their internal ontologies:

- station - the concept of a station, whose instances can be the starting point or the destination of a trip;
- date - the instance of this concept represents the date the trip should begun;
- time - an instance of this concept represents the departure time;
- price - the price of a certain trip.

Additionally, the requestor's ontology contains the concept `myRoute`, with the following signature[1]:

```
concept myRoute
    nonFunctionalProperties
        dc#description hasValue "concept of myRoute, containing
            the source and the destination locations,
            and the date of the trip"
        mode hasValue out
    endNonFunctionalProperties
    sourceLocation ofType station
    destinationLocation ofType station
    onDate ofType date
```

The choreography of the requestor states that the attribute `mode` has the value `out` for the concept `myRoute`, which means that the client will send the instance of this concept to the environment. `mode` has the value `in` for `time` and `price`, since the client expects to receive information about the departure time and the price of the trip from the service, and `controlled` for `station` and `date` (only the client can decide the starting and the destination point of his trip, as well as the date he wants to travel).

Additionally, it's choreography includes the following rules:

```
?x [sourceLocation hasValue ?sourceLocation_,
    destinationLocation hasValue ?destinationLocation_,
    onDate hasValue ?onDate_]
    memberOf myRoute<-
        ?sourceLocation memberOf station and
        ?endLocation memberOf station and
        ?onDate memberOf date.
```

The above rule creates an instance of `myRoute`, assuming that two instances of `station` and an instance of `date` are already created; since both `station` and `date` have the value of `mode` set to `controlled`, the requestor does not expect any input in order to create the instance of `myRoute`.

```
?x memberOf time <-
        ?myRoute memberOf myRoute.
```

An instance of `time` is expected, after the instance of `myRoute` was sent to the service.

```
?x memberOf price <-
        ?myRoute memberOf myRoute.
```

---

[1] All the concepts described in this section are WSMO concepts and they are modelled using Web Service Modeling Language (www.wsml.org)

An instance of `price` is expected, after the instance of `myRoute` was sent to the service.

As shown by these rules, the client will first send an instance of `myRoute`, and then expects to receive an instance of `time` and `price`. There are no restrictions regarding the order of receiving the `time` and `price` instances.

The service also has some additional concepts in its choreography: `route` and `routeOnDate`, with the following signatures:

```
concept route
    nonFunctionalProperties
        dc#description hasValue "concept of route, having two
            attributes of type station which show the starting
            and the ending point of the route"
        mode hasValue out
    endNonFunctionalProperties
    sourceLocation ofType station
    destinationLocation ofType station

concept routeOnDate
    nonFunctionalProperties
        dc#description hasValue "concept of route on a certain date,
            containing the route, the date, the departure time and
            the price of a ticket"
        mode hasValue out
    endNonFunctionalProperties
    forRoute ofType route
    onDate ofType date
    onTime ofType time
    forPrice ofType price
```

From the service's concepts, the attribute `mode` has the value `in` only for `station` and `date`, and the value `out` for `route` and `routeOndate`. The other concepts have the `mode` set to `static`, which means that their instances' values can not be changed during the communication process.

The service's choreography includes the following rules:

```
?x [startLocation hasValue?startLocation_,
    endLocation hasValue ?endLocation_]
    memberOf route <-
        ?startLocation_ memberOf station and
        ?endLocation_memberOf station.
```

The above rule states that an instance of `route` can be created only after two instances of `station` exists; since the concept `station` has the mode set to `in`, this instances need to be provided by the environment; the instance of `route` will be sent to the requestor of the service.

```
?x [forRoute hasValue ?forRoute_,
   onDate hasValue ?onDate_,
   onTime hasValue ?onTime_
   forPrice hasValue ?forPrice_]
   memberOf routeOnDate<-
       ?forRoute_ memberOf vtasc#route and
       ?onDate_ memberOf vtasc#date and
       ?onTime_ memberOf vtasc#time and
       ?forPricememberOfvtasc#integer.
```

This rule expresses the fact that an instance of `routeOnDate` is created, assuming that instances of `route`, `date`,`time` and `price` already exist; since none of these concepts has the `mode` set to `in`, none of them is expected from the environment.

5.2 Communication Process

In this chapter we illustrate step by step the communication process between the VTA service provider and requestor:

1. The requestor initiates the communication by sending an instance of `myRoute`.

2. PM translates this instance in terms of the service's ontology, obtaining two instances of `station` and one of `date`, which it stores in its internal repository.

3. Conform to the provider's choreography, all these three instances are expected, but the guarded transitions show that only one of them (one instance of `station`) is expected at this phase. Since the targeted choreography does not specify which one of the `station`'s instances is expected, the PM randomly sends one of them, and deletes it from the repository.

The evaluation of the transition rules starts again for the rest of the instances from the repository, and the second instance of `station` is sent and deleted.

4. PM evaluates the requestor's rules, and marks the first of them, which means it will not be reevaluated during further iterations.

5. Internally, the provider creates the instance of `route`, which is sent to WSMX.

6. After translating the `route`'s instance in terms of the requestor's ontology, and analyzing the two choreographies, the PM discards the instance of `route` (nobody is expecting any information contained by that instance) and the mediated instances. By evaluating the transition rules PM determines that the provider expects the previously stored instance of `date`; it sends it and deletes it from its internal repository.

PM marks the first rule from the service's choreography, which means it will not reevaluate it at further iterations.

7. PM marks the first rule from the requestor's choreography.

8. PM checks if all requestor's rules are marked; since there are still unmarked rules, the communication is not over yet.

7. The provider creates an instance of `routeOnDate` and sends it to WSMX.

8. PM translates the `routeOnDate` in terms of the requestor's ontology in two instances of `station`, an instance of `time` and one of `price`. Nobody is expecting instances of `station` anymore, so these two can be deleted. The `price` and `time` instances are sent to the requestor; the order of sending them is not specified in the requestor's choreography, so the PM randomly selects one, sends it to the requestor, and deletes it from the repository; the corresponding rule is marked.

The second instance is sent and deleted at the second evaluation of the instances contained by the repository. The rule triggered by sending this instance is marked.

9. PM evaluates the service's rules and mark the second one.

10. PM checks if all requestor's rules are marked; since they are, the communication is over.

11. PM deletes the two choreography instances (it should also delete any instances from its internal repository, but in this case there are none).

6 Related Work

Processes mediation is still a poorly explored research field, in the context of Semantic Web Services. The 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.

Two integration tools, Contivo[2] and CrossWorlds[3] seemed to be the most advanced ones in this field.

Contivo is an integration framework which uses metadata representing messages organized by semantically defined relationships. One of its functionalities is that it is able to generate transform code based on the semantic of the relationships between data elements, and to use this code for transforming the exchange messages. However, Contivo is limited by the use of a purpose-built vocabulary and of pre-configured data models and formats.

CrossWorlds is an IBM integration tool, meant to facilitate the B2B collaboration through business processes integration. It may be used to implement various e-business models, including enhanced intranets (improving operational efficiency within a business enterprise), extranets (for facilitating electronic trading between a business and its suppliers) and virtual enterprises (allowing enterprises to link to outsourced parts of its organization). The disadvantage of this tool is that different applications need to implement different collaboration and connection modules, in order to interact. As a consequence, the integration of a new application can be done only with additional effort.

Through our approach we aim to provide dynamic mediation between various parties using WSMO for describing goals and Web Services. As described in this paper this is possible without introducing any hard-coded transformations.

---

[2] http://www.contivo.com/
[3] http://www.sars.ws/hl4/ibm-crossworlds.html

7 Conclusions

In this document we proposed an approach and a mechanism for coping with the processes heterogeneity problem. The processes we are addressing in this paper are the public processes of business entities (services or a requestors of services), which express their public processes as WSMO choreographies.

The proposed approach is based on the semantic description of the Web Services and of their behavior, as well as on how a requestor that wants to interact with a Web Service express the expected behaviour of the Web Service.

We showed that an algorithm that considers the concepts' definitions and evaluates the transition rules from the two choreographies can resolve (some of the) heterogeneity problems of the collaborating parties.

References

1. Business Process Trends Glossary. `http://www.bptrends.com/resources_glossary.cfm`, 2003.
2. E. Cimpian and A. Mocan. Process Mediation in WSMX. Technical report, WSMX Working Draft, `http://www.wsmo.org/TR/d13/d13.7/v0.1/`, March 2005.
3. E. Cimpian, M. Moran, E. Oren, T. Vitvar, and M. Zaremba. Overview and Scope of WSMX. Technical report, WSMX Working Draft, `http://www.wsmo.org/TR/d13/d13.0/v0.2/`, February 2005.
4. J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel. The Web Service Modeling Language WSML. Technical report, WSML Working Draft, `http://www.wsmo.org/TR/d16/d16.1/v0.2/`, March 2005.
5. D. Fensel and C. Bussler. The web service modeling framework WSMF. *Electronic Commerce Research and Applications*, 1(2), 2002.
6. T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–229, 1993.
7. Y. Gurevich. Evolving algebras 1993: Lipari Guide. In Egon Börger, editor, *Specification and Validation Methods*, pages 9–37. Oxford University Press, 1994.
8. A. Mocan and E. Cimpian. WSMX Data Mediation. Technical report, WSMX Working Draft, `http://www.wsmo.org/TR/d13/d13.3/v0.2/`, March 2005.
9. D. Roman, J. Scicluna, C. Feier, M. Stollberg, and D. Fensel. Ontology-based Choreography and Orchestration of WSMO Services. Technical report, WSMO Working Draft, `http://www.wsmo.org/TR/d14/v0.1/`, March 2005.
10. M. Zaremba, M. Moran, and T. Haselwanter. WSMX Architecture. Technical report, WSMX Working Draft, `http://www.wsmo.org/TR/d13/d13.4/v0.2/`, March 2005.