



D13.7 v0.1 Process Mediation in WSMX

WSMX Working Draft 14 April 2005

This version:

<http://www.wsmo.org/TR/d13/d13.7/v0.1/20050414/>

Latest version:

<http://www.wsmo.org/TR/d13/d13.7/v0.1/>

Previous version:

<http://www.wsmo.org/TR/d13/d13.7/v0.1/20050404/>

Authors:

Emilia Cimpian
Adrian Mocan

Editor:

Emilia Cimpian

Reviewer:

This document is also available in non-normative [PDF](#) version.

Table of contents

- [1. Introduction](#)
- [2. Problem Definition](#)
 - [2.1. Process](#)
 - [2.1.1. Public Processes](#)
 - [2.1.2. Private Processes](#)
 - [2.2. Process Equivalence](#)
- [3. State of the Art](#)
 - [3.1. Process Representation](#)
 - [3.2. Process Mediation](#)
- [4. WSMX Process Representation](#)
 - [4.1. State](#)
 - [4.2. Guarded Transitions](#)
- [5. Process Mediation in WSMX](#)
- [6. Example](#)
 - [6.1. Virtual Travel Agency Service](#)
 - [6.2. Virtual Travel Agency Requestor](#)
 - [6.3. Mediation Between the Service and the Requestor](#)
- [7. Conclusions](#)
- [References](#)
- [Acknowledgement](#)

1. Introduction

The Web Service Execution Environment (WSMX) has the ambitious goal of creating an execution environment for the dynamic discovery, selection, invocation and inter-operation of Semantic Web Services. In any of these processes, the mediation may be needed at both data and process level.

This deliverable tackles the process mediation problem, starting with a description of what we understand by a process and process equivalence in the context of Semantic Web Services, the current state of the art in process representation and process mediation, and continuing with our approach for process representation and mediation. In the end of this document, we will present an example and the final conclusions and future steps in the development of a robust and reliable mediation system.

2. Problem Definition

The business process mediation is a complex task, and this deliverable does not intend to address all the problems and all the mediation scenario that may appear in this context, but only a small subset; this subset will be extended as our work will progress. In this section of the document, we will describe what do we understand by a business process, and by mediating between business processes.

2.1. Process

A business process is a collection of activities designed to produce a specific output for a particular customer, based on a specific input, an activity being a function, or a task that occurs over time and has recognizable results [BTP, 2003].

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.

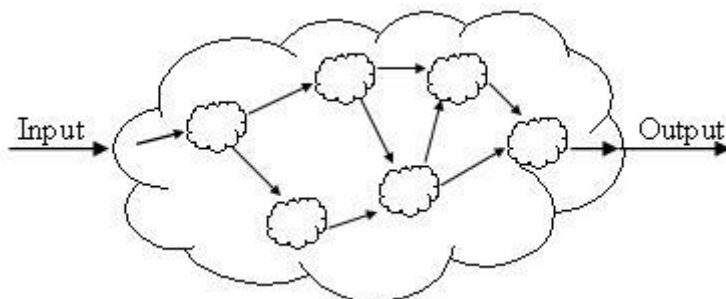


Figure 1. Process consisting of multiple processes

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 behaviour of the organization in

collaboration with other entities [[Fensel and Bussler, 2002](#)].

2.1.1. Public processes

The public processes (called abstract processes or business protocol in Business Process Execution Language for Web Services [[BPEL4WS, 2003](#)]) define the behaviour of an business entity (endpoint) in collaboration with another endpoint, which is expressed by the exchange of messages. For establishing a communication, both of them have to understand the behaviour of the other one, and more important, their behaviors have to match. The matching means that the two endpoints have to have symmetric behaviour, for example when one of them is sending a message, the other one has to know that it is going to receive it.

Consider the following example: a Virtual Travel Agency (VTA) service, offers the possibility of on-line tickets booking for certain routes, and a requestor of such a service attempts to invoke it. In their internal ontology, both of them have the following concepts:

Listing 1. station and route definitions

```

concept station
  nonFunctionalProperties
    dc:description hasValue "concept of station, containing the
      code of the station, and two attributes showing if this
      station is the starting or the ending point of a trip"
  endNonFunctionalProperties
  code ofType xsd:String
  startPoint ofType xsd:Boolean
  endPoint ofType xsd:Boolean

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"
  endNonFunctionalProperties
  sourceLocation ofType station
  destinationLocation ofType station

```

The assumption that both the requestor and the provider of the service understand the same concepts was made only for the sake of simplicity. In case they use different conceptualizations of the same domain, an external Data Mediator [[Mocan, 2005](#)] should be invoked for solving the data heterogeneity problem.

A public process for the requestor could be that after sending two instances of `station`, it expects to receive an instance of `route`. The equivalent public process from the requestor side could be that it expects two instances of `station` in order to generate an instance of `route`. The relations between the two instances of `station` and the instance of `route` are not specified in the public interface of neither one of the two endpoints.

A graphical representation of the requestor's public processes is represented in [Figure 2](#).

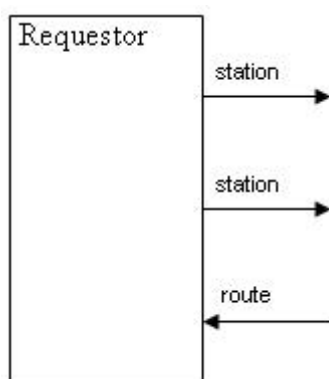


Figure 2. Example of a public process

2.1.2. Private processes

The private processes (called executable business processes in BPEL4WS [BPEL4WS, 2003]) model the actual behaviour of an endpoint involved in an interaction, that is, the internal processing of events.

For example, in the previously described scenario, the computations reformed by the VTA in order to generate the instance of route are not visible for the environment. The private process of generating this instance could check, for example, if the two instances of stations exist in it's own ontology (otherwise it is impossible for it to have a route between them) and if the requestor specified which one of them is the starting and which one is the ending point of the trip.

2.2. Process Equivalence

By process equivalence we understand in this context the full matching of the communication pattern from the source and target of the communication, that is, when one of them is sending a message, the other one is able to receive it (we are not addressing the private process equivalence, these not being visible outside their owner).

Since usually a business communication consists of more than one exchange message, the finding of 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. [Fensel and Bussler, 2002] 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 patterns, and several transformations have to be performed in order to resolve the mismatches (for example when one partner sends more than one concept in a single message, but the other one expects them separately. In this case the mediator can “break” the initial message, and send the concepts 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 (for example, an acknowledgement). 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 endpoints have to define equivalent processes, or to use an external mediation system as part of the communication process. The role of the mediator system will be to transform the client's messages and/or Web service's messages, in order to obtain a sequence of equivalent processes.

In the following subsections we will describe the resolvable message mismatches that our Process Mediator intends to address, and the unresolvable message mismatches that a Process Mediator can not address, the ideal case of precise match not raising any problems from the communication patterns point of view.

2.2.1. Resolvable Message Mismatches

Some of the communication mismatches can be addressed by means of a mediation system. In this section we will provide a list of resolvable mismatches that our mediator intends to address.

- a) Stopping an unexpected message ([Figure 3. 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.
- b) Inversing the order of messages ([Figure 3. 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.
- c) Splitting a message ([Figure 3. c\)](#)) – in case one of the partners sends in a single message multiple information that the other one expects to receive in different messages.
- d) Combining messages ([Figure 3. d\)](#)) – in case one of the partners expects a single message, containing information sent by the other one in multiple messages.
- e) Sending a dummy acknowledgement ([Figure 3. 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.

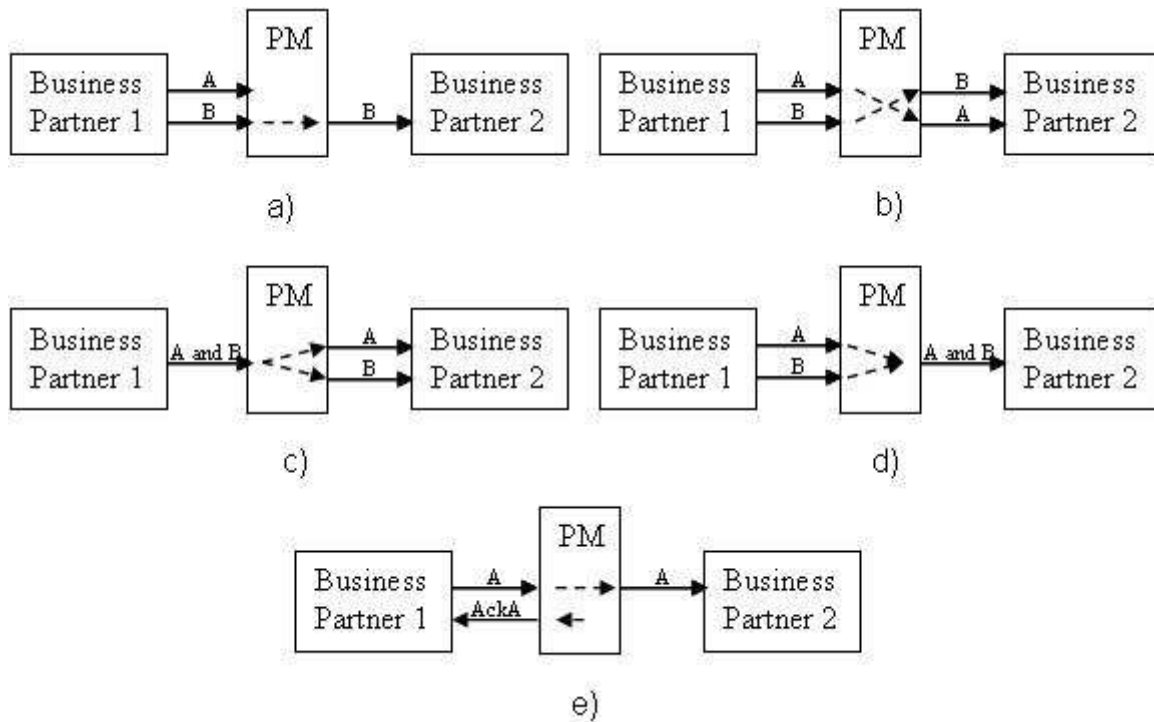


Figure 3. Resolvable message mismatches

This list of resolvable message mismatches contains only the initial set of mismatches that our Process Mediator intends to address. It will be further extended, in order to address more possible resolvable mismatches.

2.2.2. Unresolvable Message Mismatches

There are several communication mismatches that can not be addressed by a Process Mediator.

- a) Generating a message (Figure 4. a)) – in case one of the partners expects a message containing information that the other partner did not previously send. The Process Mediator is not able to generate it, as a consequence of the fact that it does not have the required information
- b) Sending a dummy acknowledgement (Figure 4. b)) – in case one of the partners expects an acknowledgement for a certain message, but the other partner does not want to receive the message.

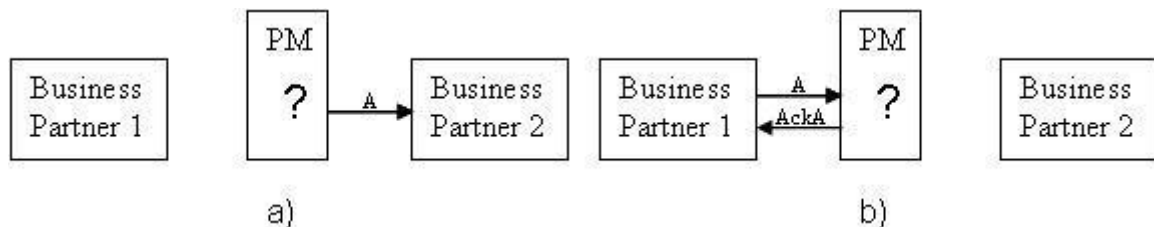


Figure 4. Unresolvable message mismatches

3. State of the Art

This chapter presents the current state of the art in process representation ([Section 3.1](#)) and in process mediation ([Section 3.2](#)). Since there are many approaches for both process representation and process mediation, we do not claim to present all existing approaches in neither one of these fields, but to provide a short introduction of what has already been done.

3.1. Process Representation

In order to address the processes mediation problem, we have to have a clear understanding of what a process means, and what are the existing technologies in representing the processes.

An adequate process representation technology has not only to support the modeling of the processes and to respect the correctness of their execution, but also to allow the automation of business processes within organizations. Since the description of the current existing technologies in representing the processes is out of the scope of this chapter, and there are already a number of surveys in this field (like [\[Solanki and Abela, 2003\]](#) or [\[Peltz, 2003\]](#)), we will briefly present only aspects of BPEL4WS [\[BPEL4WS, 2003\]](#). The reason for focusing on this particular notation, is that BPEL4WS allows the processes to export and import functionality by using Web Services interfaces exclusively.

BPEL4WS distinguishes two ways of describing the business processes: executable business processes and abstract business processes, also known as business protocols. The first ones model the internal behavior of a participant in an interaction, while the second ones describe the message exchange behavior of the involved parties. The key distinction between the two classes of processes is that the executable business processes model data in a private way, that need not be described in the public protocols.

From the mediation point of view, we are only concerned with the second category, which is addressed by the WSMO choreography, the internal behavior of the participants not being relevant for communication. In the rest of this chapter, both terms, business process and business protocol, will be used with the same meaning: the message exchange behavior.

Conform to the BPEL4WS Specifications, the definition of a business protocol involves the specification of all the visible messages exchanged between the involved parties, without any reference to the internal behavior.

The BPEL4WS definitions for both the internal processes and the communication protocols consist of four major components: variables, partnerLinks, faultHandlers, and description of the normal behavior. The variables define the data variables used by the process, in terms of WSDL message types, XML Schema simple types or XML Schema elements; the partnerLinks specify the different parties that are involved in the process; the faultHandlers contain fault handlers that define the responses in case of a failure.

Additionally, the protocol definition must include conditional and time-out constructs, exceptional conditions and recovery sequences and cross partner coordination of the outcome. The conditional and time-out constructs are used for modeling the data dependent behavior; the exceptional conditions and recovery sequences are as important as the ability to define the behavior of a business protocol assuming that

everything is working well; the cross partner coordination of the outcome is needed for different units of work and at different levels of granularity, the reason for this being that long-running interactions may include multiple nested units of work.

3.2. Process Mediation

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 [[Contivo](#)] and CrossWorlds [[CrossWorlds](#)] 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.

4. WSMX Process Representation

As in WSMO Choreography and Orchestration [[Roman et al., 2005](#)] the representation of a WSMX business process is based on the Abstract State Machine [[Gurevich, 1995](#)] methodology. ASMs have been chosen as underlying model of choreography and orchestration for the following three reasons:

1. Minimality: ASMs provide a minimal set of modeling primitives, i.e., enforce minimal ontological commitments. Therefore, they do not introduce any ad hoc elements that would be questionable to be included into a standard proposal.
2. Maximality: ASMs are expressive enough to model any aspect around computation.
3. Formality: ASMs provide a rigid mathematical framework to express dynamics.

For a detailed explanation on ASMs we refer the reader to [Börger, 1998].

In order for this deliverable 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 [Roman et al., 2005].

Taking the ASMs methodology as a starting point, a WSMX process is state-based and consists of two elements: states and guarded transitions.

Listing 6. WSMX process definition

```

Class wsmxProcess
  hasState type ontology
  hasGuardedTransitions type guardedTransition

```

All the `wsmxProcess` elements are defined in WSMO Choreography Orchestration, but for consistency reasons we will provide them here as well.

State

A state is described by an ontology as defined in [Roman et. al., 2004] Section 4.

Guarded transitions

Transition rules that express changes of states by changing the set of instances.

4.1. State

A `state` is described by a set of explicitly defined instances and values of their attributes or through a link to an instance store.

In extension to a standard WSMO ontology, an ontology that is used to describe states in a WSMX process introduces a new non-functional property. When a concept, relation or function in a process is defined, the attribute `mode` can be defined as a new non functional property. It can take one of the following values:

- `static` - meaning that the extension of the concept, relation, or function cannot be changed. If not explicitly defined, the attribute mode takes this value by default.
- `controlled` - meaning that the extension of the concept, relation, or function can only be changed by its owner.
- `in` - meaning that the extension of the concept, relation, or function can only be changed by the environment. A grounding mechanism for this item must be provided that implements *write* access for the environment.
- `shared` - meaning that the extension of the concept, relation, or function can be changed by its owner and by the environment. A grounding mechanism for this item must be provided that implements *read/write* access for the environment.
- `out` - meaning that the extension of the concept, relation, or function can only be changed by its owner. A grounding mechanism for this item must be provided that implements *read* access for the environment.

For more details on WSMO Grounding we refer the reader to [Kopecky and Roman,

2005].

The signature of the states is defined by WSMO identifiers, concepts, relations, functions, and axioms. This signature is the same for all states. The elements that can change and that are used to express different states of a choreography, are the instances (and their attribute values) of concepts, functions, and relations that are not defined as being static. In conclusion, a specific state is described by a set of explicitly defined instances and values of their attributes or through a link to an instance store.

4.2. Guarded transitions

Guarded Transitions are used to express changes of states by means of rules, expressible in the following form:

if *Cond* then *Updates*.

Cond is an arbitrary WSML axiom, formulated in the given signature of the state.

The *Updates* consist of arbitrary WSMO Ontology instance (see [Section 4.7 of WSMO 1.1](#)) statements.

5. Processes Mediation in WSMX

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

When WSMX receives a messages, 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 [Zaremba, 2005].

After the id of the conversation is obtained, the Process Mediator (PM) receives it, together with the message, consisting 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 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 choreography instance of the party that receives that message. Simultaneously with sending the message, the process mediator should update the choreography instances and reevaluate all

the rules, until no further updates are possible.

The interactions between the Process Mediator and other WSMX components is represented in Figure 5.

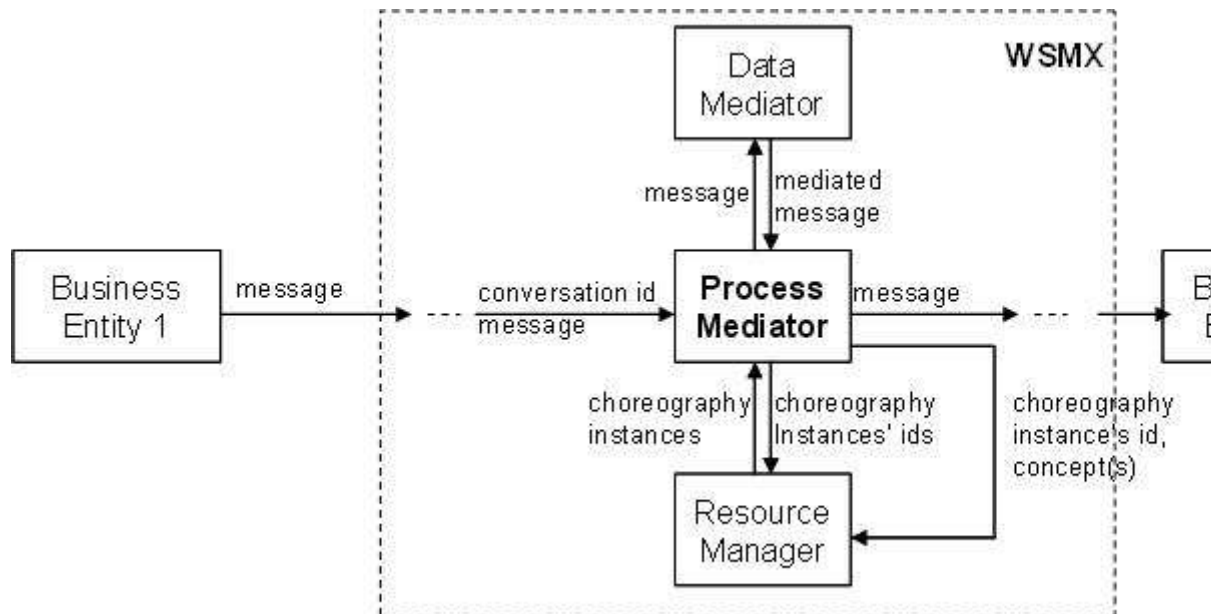


Figure 5. Process Mediator's interactions

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; 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' owners. 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 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 instances created during this conversation (together with the choreography instances), from both its internal repository and WSMX repository.

The only thing that should be kept in an internal storage is the actions the Process Mediator needs to take when receiveing a message. This could be usefull in case the two partners will be involved in a second conversation.

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

6. Example

In this chapter we will present two examples of public processes, the process defined by a Virtual Travel Agency service and the process defined by a requestor of this service.

6.1. Virtual Travel Agency Service

In this subchapter, we will define the Virtual Travel Agency (VTA) service's ontology, its choreography ontology and the corresponding transition rules.

Listing 7 presents the VTA service ontology, defining the concepts needed for performing on-line ticket reservation:

Listing 7. Virtual Travel Agency service ontology

```

namespace
<<http://www.wsmo.org/TR/d13/d13.7/ontologies/VTAServiceOntology/>>
  dc: <<http://purl.org/dc/elements/1.1#>>
  xsd: <<http://www.w3.org/2001/XMLSchema#>>

ontology<<http://www.wsmo.org/TR/d13/d13.7/ontologies/VTAServiceOntology>>

nonFunctionalProperties
  dc:title hasValue "Virtual Travel Agency Service Ontology"
  dc:creator hasValue "Emilia"
  dc:description hasValue "an ontology for describing trip
    reservation service related knowledge"
  dc:publisher hasValue "DERI International"
  dc:contributor hasValues "Adrian, ..."
  dc:date hasValue "03.03.2004"
  dc:type hasValue <<http://www.wsmo.org/2004/d2/#ontology>>
  dc:format hasValue "text/html"
  dc:language hasValue "en-us"
  dc:rights hasValue <<http://deri.at/privacy.html>>
  version hasValue "$Revision 0.1 $"
endNonFunctionalProperties

concept station
  nonFunctionalProperties
    dc:description hasValue "concept of station, containing the
      code of the station, andtwo attributes showing if this
      station is the starting or the ending point of a trip"
  endNonFunctionalProperties
  code ofType xsd:string
  startpoint ofType xsd:boolean
  endpoint ofType xsd:boolean

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"
  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 for a ticket"
  endNonFunctionalProperties
  forRoute ofType route
  onDate ofType date
  onTime ofType time
  forPrice ofType price

concept time subConceptOf xsd:time
  nonFunctionalProperties
    dc:description hasValue "concept of time"
  endNonFunctionalProperties

concept date subConceptOf xsd:date
  nonFunctionalProperties
    dc:description hasValue "concept of date"
  endNonFunctionalProperties

```

```

concept price subConceptOf xsd:integer
  nonFunctionalProperties
    dc:description hasValue "concept of price"
  endNonFunctionalProperties

concept person
  nonFunctionalProperties
    dc:description hasValue "concept of person, containing only
      the name of the person"
  endNonFunctionalProperties
  name ofType xsd:string

concept creditCard
  nonFunctionalProperties
    dc:description hasValue "concept of credit card, containing
      its number, owner and expiration date"
  endNonFunctionalProperties
  number ofType xsd:integer
  owner ofType person
  expirationDate ofType date

concept reservation
  nonFunctionalProperties
    dc:description hasValue "concept of reservation, containing its
      number, owner, and the route for which this reservation
was made"
  endNonFunctionalProperties
  reservationNumber ofType xsd:integer
  reservedRoute ofType routeOnDate
  reservationHolder ofType personn

```

Listing 8 presents the VTA service choreography's ontology (any business partner express its public processes as WSMO choreography).

Listing 8. Virtual Travel Agency service choreography ontology

```

namespace<<http://www.wsmo.org/TR/d13/d13.7/ontologies/VTAServiceChorograph
  vtas: <<http://www.wsmo.org/TR/d13/d13.7/ontologies/VTAServiceOntology:
  dc: <<http://purl.org/dc/elements/1.1#>>

ontology <<http://www.wsmo.org/TR/d13/d13.7/ontologies/VTAServiceChorograph
  nonFunctionalProperties
    dc:title hasValue "Virtual Travel Agency Service Choreography"
    dc:creator hasValue "Emilia"
    dc:description hasValue "an ontology for describing trip
      reservation service chorography related knowledge"
    dc:publisher hasValue "DERI International"
    dc:contributor hasValues "Adrian, ...."
    dc:date hasValue "03.03.2004"
    dc:type hasValue <<http://www.wsmo.org/2004/d2/#ontology>>
    dc:format hasValue "text/html"
    dc:language hasValue "en-us"
    dc:rights hasValue <<http://deri.at/privacy.html>>
    version hasValue "$Revision 0.1 $"
  endNonFunctionalProperties

importedOntologies
<<http://www.wsmo.org/TR/d13/d13.7/ontologies/VTAServiceOntology>>

concept station subConceptOf vtas:station
  nonFunctionalProperties

```

```

        mode hasValue in
    endNonFunctionalProperties

concept route subConceptOf vtas:route
    nonFunctionalProperties
        mode hasValue out
    endNonFunctionalProperties

concept routeOnDate subConceptOf vtas:routeOnDate
    nonFunctionalProperties
        mode hasValue out
    endNonFunctionalProperties

concept time subConceptOf vtas:time

concept date subConceptOf vtas:date
    nonFunctionalProperties
        mode hasValue out
    endNonFunctionalProperties

concept price subConceptOf vtas:price

concept person subConceptOf vtas:person
    nonFunctionalProperties
        mode hasValue in
    endNonFunctionalProperties

concept creditCard subConceptOf vtas:creditCard
    nonFunctionalProperties
        mode hasValue in
    endNonFunctionalProperties

concept reservation subConceptOf vtas:reservation
    nonFunctionalProperties
        mode hasValue out
    endNonFunctionalProperties

```

[Listing 9](#) presents the guarded transitions that lead the execution of the process.

Listing 9. Virtual Travel Agency service transition rules

```

choreographyVTAServiceChoreography

statevtasc
<"http://www.wsmo.org/TR/d13/d13.7/ontologies/VTAServiceChorography">

guardedTransitionsreservationServiceTransitionRules

/*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 providedby the environment; the instance of
route will be send to the requestor of the service*/
?x memberOf vtask#route[
    startLocation hasValue?startLocation_,
    endLocation hasValue ?endLocation_] <-
    ?startLocation_ memberOf vtask#station and
    ?endLocation_ memberOfvtasc#station.

/*an instance of routeOnDate is created, assuming that instances of route, date,time and price
already exist; since only date has the mode set toin, only an instance of date is expected from the

```

```

environment*/
?x memberOf vtask#routeOnDate[
  forRoute hasValue ?forRoute_,
  onDate hasValue ?onDate_,
  onTime hasValue ?onTime_
  forPrice hasValue ?forPrice_] <-
  ?forRoute_ memberOf vtask#route and
  ?onDate_ memberOf vtask#date and
  ?onTime_ memberOf vtask#time and
  ?forPrice_ memberOf vtask#integer.

/*an instance of the reservation concept is created, assuming that instances of routeOnDate,
creditCard and person already exists. The instances of creditCard and person has to be obtained
from the environment*/
?x memberOf vtask#reservation[
  reservationNumber hasValue ?reservationNumber_,
  reservationRoute hasValue ?reservationRoute_,
  reservationHolder hasValue ?reservationHolder_] <-
  ?reservationRoute_ memberOf vtask#routeOnDate and
  ?creditCard_ memberOf vtask#creditCard and
  ?reservationHolder_ memberOf vtask#person.

```

Please notice that this rules does not reflect the actual computations done by the system in order to create certain instances, they just express the message sequencing. For example, for generating an instance of route, the system has to internally check if the two stations are the beginning and the end of the trip, and if it can provide a connection between these two stations. However, all this computations are private, and they are not visible for the environment.

Based on the transition rules, the order of sending/receiving messages (communication pattern) can be establish: Based on the transition rules, the order of sending/receiving messages (communication pattern) can be establish:

1. Receive two instances of `station` (the order of receiving these two instances does not matter);
2. Send an instance of `route`;
3. Receive an instance of `rdate`;
4. Send an instance of `routeOnDate`;
5. Receive an instance of `creditCard` and an instance of `person` (the distinction between the owner of the credit card and the person that makes the reservation is done internally, and doesn't need to be published in the choreography);
6. Send an instance of `reservation`.

6.2. Virtual Travel Agency Requestor

In the previous example we described a service that provides on-line ticket booking facilities. In this chapter, we will describe the requestor of such a service. [Listing 10](#) present its ontology, [Listing 11](#) its choreography's ontology and finally, [Listing 12](#), its transition rules.

Listing 10. Virtual Travel Agency requestor ontology

```

namespace <<http://www.wsmo.org/TR/d13/d13.7/ontologies/VTARrequestorOntolog;
dc <<http://purl.org/dc/elements/1.1#>>,
xsd <<http://www.w3.org/2001/XMLSchema#>>

```

```

ontology <<http://www.wsmo.org/TR/dl3/dl3.7/ontologies/VTARquestorOntolog
nonFunctionalProperties
  dc:title hasValue "Virtual Travel Agency Requestor Ontology"
  dc:creator hasValue "Emilia"
  dc:description hasValue "an ontology for describing trip reservation
    requestor related knowledge"
  dc:publisher hasValue "DERI International"
  dc:contributor hasValues "Adrian, ...."
  dc:date hasValue "03.03.2004"
  dc:type hasValue <<http://www.wsmo.org/2004/d2/#ontology>>
  dc:format hasValue "text/html"
  dc:language hasValue "en-us"
  dc:rights hasValue <<http://deri.at/privacy.html>>
  version hasValue "$Revision 0.1 $"
endNonFunctionalProperties

concept station
  nonFunctionalProperties
    dc:description hasValue "concept of station, containing the
      code of the station, and two attributes showing if this
      station is the starting or the ending point of a trip"
  endNonFunctionalProperties
  code ofType xsd:string
  startpoint ofType xsd:boolean
  endpoint ofType xsd:boolean

concept date subConceptOf xsd:date
  nonFunctionalProperties
    dc:description hasValue "concept of date"
  endNonFunctionalProperties

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

concept person
  nonFunctionalProperties
    dc:description hasValue "concept of person, containing only the
      name of the person"
  endNonFunctionalProperties
  name ofType xsd:string

concept time subConceptOf xsd:time
  nonFunctionalProperties
    dc:description hasValue "concept of time"
  endNonFunctionalProperties

concept price subConceptOf xsd:integer
  nonFunctionalProperties
    dc:description hasValue "concept of price"
  endNonFunctionalProperties

concept creditCard
  nonFunctionalProperties
    dc:description hasValue "concept of credit card, containing its
      number, owner and expiration date"

```

```

endNonFunctionalProperties
number ofType xsd:integer
owner ofType person
expirationDate ofType date

concept creditCardAcknowledgement
  nonFunctionalProperties
    dc:description hasValue "acknowledgement for receiving the
creditCard"
  endNonFunctionalProperties
  confirmation ofType creditCard

concept reservation
  nonFunctionalProperties
    dc:description hasValue "concept of reservation, containing
    its number, owner, and the route for which this reservation
    was made"
  endNonFunctionalProperties
  reservationNumber ofType xsd:integer
  reservedRoute ofType routeOnDate
  reservationHolder ofType person

```

Listing 11. Virtual Travel Agency requestor choreography onto

```

namespace<<http://www.wsmo.org/TR/d13/d13.7/ontologies/VTARrequestorChorogr:
vtar <<http://www.wsmo.org/TR/d13/d13.7/ontologies/VTARrequestorOntology:
dc <<http://purl.org/dc/elements/1.1#>>

ontology <<http://www.wsmo.org/TR/d13/d13.7/ontologies/VTARrequestorChorogr:

nonFunctionalProperties
  dc:title hasValue "Virtual Travel Agency Requestor Choreography"
  dc:creator hasValue "Emilia"
  dc:description hasValue "an ontology for describing trip reservation
  requestor choreography related knowledge"
  dc:publisher hasValue "DERI International"
  dc:contributor hasValues "Adrian, ...."
  dc:date hasValue "03.03.2004"
  dc:type hasValue <<http://www.wsmo.org/2004/d2/#ontology>>
  dc:format hasValue "text/html"
  dc:language hasValue "en-us"
  dc:rights hasValue <<http://deri.at/privacy.html>>
  version hasValue "$Revision 0.1 $"
endNonFunctionalProperties

importedOntologies
<<http://www.wsmo.org/TR/d13/d13.7/ontologies/VTARrequestorOntology>>

concept station subConceptOf vtar:station
  nonFunctionalProperties
    mode hasValue controlled
  endNonFunctionalProperties

concept date subConceptOf vtar:date
  nonFunctionalProperties
    mode hasValue controlled
  endNonFunctionalProperties

concept myRoute subConceptOf vtar:myRoute
  nonFunctionalProperties

```

```

        mode hasValue out
    endNonFunctionalProperties

concept person subConceptOf vtar:person
    nonFunctionalProperties
        mode hasValue out
    endNonFunctionalProperties

concept time subConceptOf vtar:time
    nonFunctionalProperties
        mode hasValue in
    endNonFunctionalProperties

concept price subConceptOf vtar:price
    nonFunctionalProperties
        mode hasValue in
    endNonFunctionalProperties

concept creditCard subConceptOf vtar:creditCard
    nonFunctionalProperties
        mode hasValue out
    endNonFunctionalProperties

concept creditCardAcknowledgement subConceptOf vtar:creditCardAcknowledgement
    nonFunctionalProperties
        mode hasValue in
    endNonFunctionalProperties

concept reservation subConceptOf vtar:reservation
    nonFunctionalProperties
        mode hasValue in
    endNonFunctionalProperties

```

Listing 12. Virtual Travel Agency requestor transition rules ontology

```

choreography VTARquestorChoreography

state vtarc
<"http://www.wsmo.org/TR/d13/d13.7/ontologies/VTARquestorChorography">

guardedTransitions reservationRequestorTransitionRules

/*creates an instance of my route, 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 myRoute[
    sourceLocation hasValue ?sourceLocation_,
    destinationLocation hasValue ?destinationLocation_,
    onDate hasValue ?onDate_] <-
    ?sourceLocation memberOf vtarc:station and
    ?endLocation memberOf vtarc:station and
    ?onDate memberOf vtarc:date.

/*an instance of time is expected (the departure time), after the instance of myRoute was sent to the service*/
?x memberOf time<-
    ?myRoute memberOf vtarc:myRoute.

/*an instance of price is expected (the price of a ticket), after the instance of myRoute was sent to the
service*/

```

```

?x memberOf price<-
    ?myRoute memberOf vtarc:myRoute.

/*after receiveing the time and price instances, the requester is creating an instance of the concept
creditCard*/
?x memberOf creditCard[
    number hasValue ?number_,
    owner hasValue ?owner_,
    expirationDate hasValue ?expirationDate_] <-
    ?time_ memberOf vtarc:time and
    ?price_ memberOf vtarc:price and
    ?number_ memberOf xsd:integer and
    ?owner_ memberOf vtarc:person and
    ?expirationDate_ memberOf vtarc:date.

/*the requestor is expecting the confirmation of the credit card (internally it will check if all the attributes
from the confirmedCreditCard's instance have the same value as the attribute of the creditcard's instance, but
these computations don't have to be public)*/
?x memberOf creditCardAcknowledgement[
    confirmation hasValue ?confirmation_] <-
    ?confirmation_ memberOf vtarc:creditCard.

/*after receiving the confirmation of the credit card, the requestor will send the name of the person who needs
the reservation*/
?x memberOf person[
    name hasValue ?name_] <-
    ?name memberOf xsd:string and
    ?creditCardAcknowledgement_ memberOf
vtarc:creditCardAcknowledgement.

/*an instance of reservation is expected, after instances of myRoute and person has been created*/
?x memberOf reservation[
    reservationNumber hasValue ?reservationNumber_,
    route hasValue ?route_,
    reservationHolder hasValue ?reservationHolder_] <-
    ?reservationNumber_ memberOf xsd:integer and
    ?route_ memberOf vtarc:myRoute and
    ?reservationHolder_ memberOf vtarc:person.

```

In this scenario, the exchange of messages is as follows:

1. Send an instance of myRoute
2. Receive an instance of price and an instance of time (the order of receiving these two messages is not important from the requestor point of view)
3. Send an instance of creditCard
4. Receive an instance of confirmedCreditCard
5. Send the details of the person
6. Receive the reservation instance

6.3. Mediation Between the Service and the Requestor

Considering the previously described choreographies, Figure 6 presents a graphical representation of the communication patterns of the two participants:

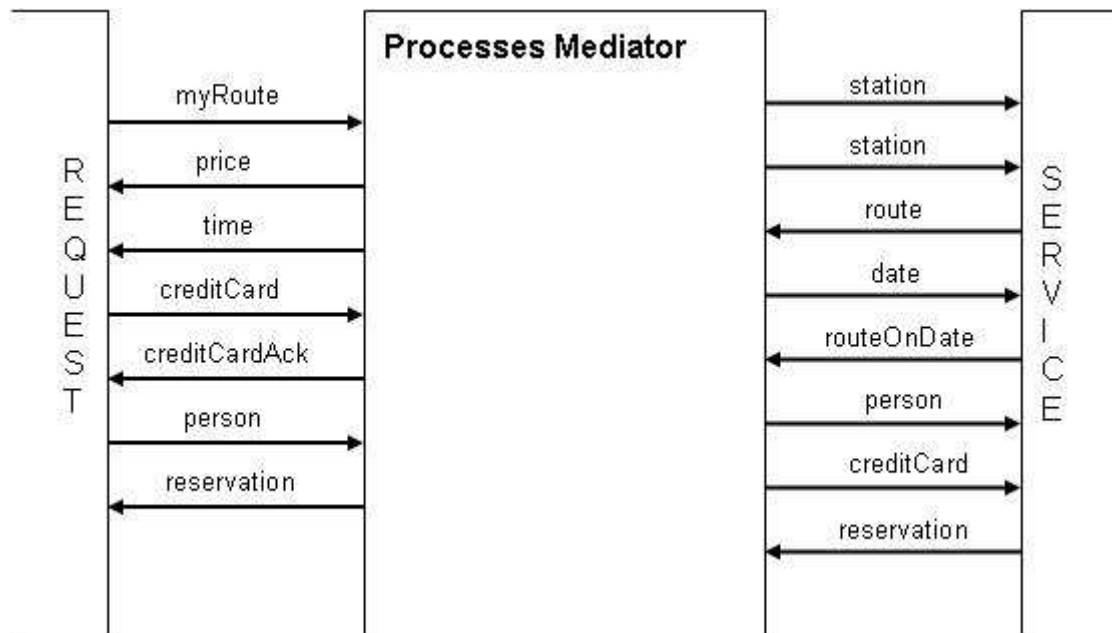


Figure 6. Communication patterns of the service requestor and service provider

The Process Mediator has to translate any incoming message in terms of the targeted partner ontology, and to decide if based on this message, it can generate any message, for either one of the two partners, that could trigger an action (not necessary to change a state, but to eliminate one condition for changing a state).

In what follows, we will analyze step by step the flow of messages and the internal transformations that take place on the requestor and provider side.

1. PM receives an instance of `myRoute` – after translating this instance in terms of the service’s ontology, the PM will obtain two instances of `station` and one of `date`. Conform to the choreography ontology of the requestor, all these three instances are expected, but the guarded transitions show that only two of them (the instances of `station`) are expected at this phase. The instance of `date` is not yet expected since the first condition (`?forRoute_ memberOf vtask:route`) of the rule that checks if an instance of `date` exists is not satisfied. As a consequence, the PM will send the two instances of `station` and store the instance of `date`.
2. Internally, the provider creates the instance of `route`, which will be send to WSMX.
3. 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 send to the provider the previously stored instance of `date`, in the same time deleting it from its internal repository.
4. The provider creates an instance of `routeOnDate` and sends it to WSMX
5. 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 anymore instances of `station`, so these two can be deleted. The `price` and `time` instances are sent to the requestor.

6. The requestor sends the details of a credit card (an instance of the `creditCard` concept) to the WSMX
7. PM send the `creditCard` instance to the provider. Based on the two choreographies, the PM sends the corresponding instance of a `creditCard` for the provider, and creates an acknowledgement (instance of `creditCardAcknowledgement`) to be sent to the requestor
8. The instance of `person` is send by the requestor to WSMX
9. PM sends the corresponding instance of `person` to the provider
10. The service sends to WSMX an instance of the `reservation` concept
11. PM send the `reservation` instance to the requestor. Since in this moment none of the two participants is expecting anything more, the PM considers the communication over.

7. Conclusions

...

References

- [Börger, 1998] E. Börger: High Level System Design and Analysis Using Abstract State Machines, Proceedings of the International Workshop on Current Trends in Applied Formal Method: Applied Formal Methods, p.1-43, October 07-09, 1998.
- [BPT, 2003] Business process Trends, Glossary, available at http://www.bptrends.com/resources_glossary.cfm, 2003.
- [Bussler, 2003] C. Bussler: B2B Integration. Berlin, Heidelberg: Springer, 2003.
- [BPEL4WS, 2003] Specification: Business Process Execution Language for Web Services Version 1.1, available at <http://www-128.ibm.com/developerworks/library/ws-bpel/>, 2003.
- [Contivo] Available at: <http://www.contivo.com/>.
- [CrossWorlds] Available at: <http://www.sars.ws/hl4/ibm-crossworlds.html>
- [Fensel and Bussler, 2002] D. Fensel and C. Bussler: The Web Service Modeling Framework WSMF, In Electronic Commerce Research and Applications, 1(2), 2002.
- [Gurevich, 1995] Y. Gurevich: Evolving Algebras 1993: Lipari Guide, Specification and Validation Methods, Oxford University Press, 1995, 9--36.
- [Hollander, 2003] D. Hollander: Semantic Integration, How You Can Deliver Business Value Today, 2003.
- [Kopecky and Roman] J. Kopecky, and D. Roman: WSMO Grounding, WSMO

deliverable D24.2 version 0.1. available from
<http://www.wsmo.org/2005/d24/d24.2/v0.1/20050119/>

[Mocan, 2005] A. Mocan (ed.): WSMX Data Mediation, WSMX Working Draft v0.2, available at <http://www.wsmo.org/2005/d13/d13.3/v0.2/>, 2005.

[Peltz, 2003] C. Peltz: Web Service Orchestration. A Review of emerging technologies, tools, and standards. Hewlett Packard, CO., January 2003.

[Roman et al., 2004] D. Roman, H. Lausen and U. Keller (eds): Web Service Modeling Ontology (WSMO) WSMO Working Draft v1.1, available at <http://www.wsmo.org/2004/d2/v1.1/>, 2004.

[Roman et al., 2005] D. Roman, J. Scicluna, Cristina Feier (eds.): Ontology-based Choreography and Orchestration of WSMO Services, WSMO deliverable D14 v0.1. available at <http://www.wsmo.org/TR/d14/v0.1/>, 2005.

[Solanki and Abela, 2003] M. Solanki and C. Abela: The Landscape of Markup Languages for Web Service Composition, May 2003.

[Zaremba, 2005] M. Zaremba (ed.): WSMX Architecture, WSMX Working Draft v0.2, available at <http://www.wsmo.org/2005/d13/d13.4/v0.2/>, 2005

Acknowledgement

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, SWWS, Esperanto and h-TechSight; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate programme.

The editors would like to thank to all the [members of the WSMO working group](#) for their advises and inputs to this document.



[webmaster](#)