Mediation Enabled Semantic Web Services Usage

Emilia Cimpian, Adrian Mocan, Michael Stollberg

Digital Enterprise Research Institute,
Institute for Computer Science, University of Innsbruck,
Technikerstrasse 21a, A-6020 Innsbruck, Austria
firstname.lastname@deri.org

Abstract. The Semantic Web services has become a challenging research topic in the last half of decade. Various frameworks offer means to semantically describe all the related aspects of Semantic Web services, but the solutions to the heterogeneity problems, inherent in a distributed environment as the Web, are still to be properly integrated and referred to from the main phases of the Web services usage. Both data and process heterogeneity, as well as the multitude of functionalities required and offered by semantic Web services' requesters and providers hamper the usability of Web services, making this technology difficult to use. This paper emphasizes the role of mediators in a Semantic Web services architecture, illustrating how the mediators can enable the Semantic Web services usages as discovery, invocation and composition.

1 Introduction

An intense research activity regarding Semantic Web, Web services and their combination, Semantic Web services, has been going on during the last years. But only the semantic descriptions attached to data or to the Web services deployed using today's technologies, does not solve the heterogeneity problem that may come up due to the distributed nature of the Web itself. As such, the heterogeneity existing in representing data, in the multitude of choices in representing the requested and the provided functionalities, and in the differences in the communication patterns (public processes) are problems that have to be solved before being able to fully benefit of the semantic enabled Web and Web services. Considering that these problems can not be avoided, dynamic mediation solutions that fully exploit the semantic descriptions of data and services are required.

This paper emphasizes the importance of the mediators in the usages of Semantic Web services, showing why the basic phases needed for Semantic Web services usages (discovery, invocation and composition) can hardly take place without the support of mediators. It also identifies different levels of mediation, illustrating what type of mediation is needed in a particular phase.

The discussion is held in the context of Web Service Modeling Ontology (WSMO) [4], a framework that offers all the necessary instruments to semantically describe the Web services and all the related aspects. One of the main reasons in choosing WSMO as the semantic framework for Web services is that it realizes the importance of mediators and treats them as first class citizens. WSMO offers specific means to semantically

describe concrete mediation solutions and to directly refer to them where needed (e.g. from ontologies or Web services).

The paper is structured as follows: Section 2 provides an overview of Semantic Web services definition, as an important aspect for the usability of the services; Section 3 describes how the discovery, invocation and composition can benefit from the use of mediators, and what type of mediation is needed in each of these phases; Section 4 presents an illustrative example, addressing all types of mediation previously identified, while Section 5 provides an overview of the related mediation work; finally, Section 6 concludes the paper.

2 Semantic Web Services Definition

Any Semantic Web service is accessible via its interface, which provides information on how a service can be invoked. As a consequence, we believe that the ability of a service to participate in complex interactions directly dependents on the expressivity of its interface and on its correctness (from the business logic point of view). Simultaneously, a service has to correctly and completely advertise its functionality (that is, what the service can provide), which will enable the service's discovery by potential requestors.

Web Service Modeling Ontology (WSMO[1]) provides an exhaustive definition of Semantic Web services [4].

Table 1: WSMO Web Service Definition

```
Class webService
  hasNonFunctionalProperties type nonFunctionalProperties
  importsOntology type ontology
  usesMediator type ooMediator, wwMediator
  hasCapability type capability
    multiplicity = single-valued
  hasInterface type interface
```

The WSMO service definition consists of the following elements:

non-functional properties - general information about the Web service, like `creator`, `format`, or `description` [12];
imported ontologies - external ontologies used in defining the service;
used mediators - different mediators needed for definig the service (for example, for importing ontologies);
capability - a functional description of what the service can do;
interface - the way of communicating with the requestor or with other services.

From the service's behaviour point of view, the important items from this definition are the capability and the interface, and we reproduce their definitions from [12].

[1] see `http://www.wsmo.org`

Table 2: WSMO Web Service Capability

```
Class capability
  hasNonFunctionalProperties type nonFunctionalProperties
  importsOntology type ontology
  usesMediator type ooMediator, wgMediator
  hasSharedVariables type sharedVariables
  hasPrecondition type axiom
  hasAssumption type axiom
  hasPostcondition type axiom
  hasEffect type axiom
```

Apart from the non-functional properties, the imported ontologies and the used mediators, the capability definition of a semantic Web service must contain the following information:

precondition - the information space of the Web service before its execution;
assumption - the state of the world before the execution of the Web service;
postcondition - the information space of the Web service after its execution;
effect - the state of the world after the execution of the Web service;
shared variables - variables that are shared between preconditions, postconditions, assumptions and effects.

Table 3: WSMO Web Service Interface

```
Class interface
  hasNonFunctionalProperties type nonFunctionalProperties
  importsOntology type ontology
  usesMediator type ooMediator
  hasChoreography type choreography
  hasOrchestration type orchestration
```

The semantic Web service's interface must contain information about the choreography and the orchestration of a service. The choreography offers indications about how a client should invoke the service, while the orchestration shows how the service can communicate with other services in order to achieve a common functionality.

3 Semantic Web Services Usability

The previous section provided some details on how WSMO currently defines semantic Web services. An important aspect that needs to be noted is the presence of the `usesMediator` attribute in all three of the presented definitions, which shows that in WSMO, mediation can be supported by all constituent elements of a WSMO service description.

This section will elaborate on *why* and *how* different types of mediation should be used for the actual usage of a service, during discovery, invocation and composition phases.

3.1 Web Service Discovery

The Web service discovery has the role of determining appropriate Web services for fulfilling a certain goal, out of a collection of services. There are numerous techniques for Web service discovery; WSMO addresses three possible discovery techniques: keyword-based discovery, discovery based on simple semantic descriptions of services, and discovery based on rich semantic descriptions of services [7].

While the keyword-based discovery can take place without the use of any mediation service, the last two (called semantic-based discovery techniques) can benefit from `data level mediation` and `functional level mediation`.

In this context the data mediation is considered to be the mediation between two ontologies. That is, the data mediator is able to transform instances expressed in terms of one ontology (considered to be the `source` ontology) in instances expressed in terms of the other ontology (`target` ontology) [10].

During the discovery, the data level mediation is used in case the requestor uses a different ontology than the ones used by the available Web services. For example, if a service's declared functionality is to provide accommodation in a certain city in Austria and train tickets between any location in Europe and that particular city, it may have in its internal ontology a concept called `train_ticket`. On the other hand, a requestor of a train ticket may have an equivalent concept called `travel_voucher`. Without the services of a data mediator able to map the `train_ticket` concept to the `travel_voucher` concept, the service could not be discovered. Of course, this is a very simple example of data mediation. Several data mediator tools are able to solve more complex mappings. For example the data mediator tool developed as part of the Web Service Execution Environment (WSMX) takes into consideration both syntactical and structural aspects, mapping concepts based on their names (syntactical aspect), attributes and relations they are involved in (structural aspects) [10].

The functional level mediation is used to state the logical relations between the service offer and the service request. Considering a request G[2] and an offer WS, there are five types of possible relations between their functionalities [16]:

1. **equal** - meaning that WS offers exactly the functionality required by G.
2. **plug-in** - meaning that WS provides all the functionalities required by G, and some extra functionalities (G is a plug-in of WS). In the previously described example, the relation between the functionality requested by the goal (booking a train ticket) and the one offered by the Web service (booking both a train ticket and an accommodation) is a plug-in relation.
3. **subsume** - meaning that G requires more functionalities that WS provides (G subsumes WS). An example for this case would be when a a requester asks for a com-

[2] In WSMO the requests are expressed as `Goals`; a goal has a similar definition as a Web service, expressing in a formal way both the requested capability and the requested interface

plete holiday package (flight tickets and accommodation) and the service offers only accommodation.
4. **intersecting** - meaning that WS offers part of what G requests, and some additional functionality as well. An example for this case would be when a a requester asks for flight tickets and accommodation and the service offers accommodation and car rental.
5. **disjoint** - meaning that the requested and provided functionalities are totally different.

For a Web service to be discovered as a candidate in fulfilling a certain goal, the relation between them has to be `equal` or `plug-in`.

We might assume that these functional relationships between the requested capability and the offered capability can be derived directly in the discovery process in an automatic manner (using reasoning techniques). There also could be cases when the human domain expert has to be involved and semi-automatically (or even manually) derive these functional relations. Such situations appears when there are dependencies between the matching functionality and the remaining, additional functionality, or when financial conditions have to be analyzed.

3.2 Web Services Invocation

After the discovery of a service able to fulfil a certain request, the actual invocation can take place. Since both the provider and the requester of a service express the way they want to communicate by using the interface (choreography) description prior to the discovery, it is quite possible that there are a number of mismatches between these descriptions.

Some of them can be solved by data mediation techniques (like the `train_ticket - travel_voucher` one illustrated in the previous section), but some of them can be communication specific, solvable only by using `process level mediation` techniques (we call this process mediation since a choreography represents the public processes of an entity).

[2] identifies five types of mismatches that can be automatically solved, considering that the choreographies are expressed conforming to Abstract State Machine (ASM) specifications [1][3] as illustrated in the following figurer.

Stopping an unexpected message (Figure 1. 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 1. 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.

Splitting a message (Figure 1. c)): in case one of the partners sends in a single message multiple information that the other one expects to receive in different messages.

[3] WSMO also uses ASM for representing the choreographies [15]

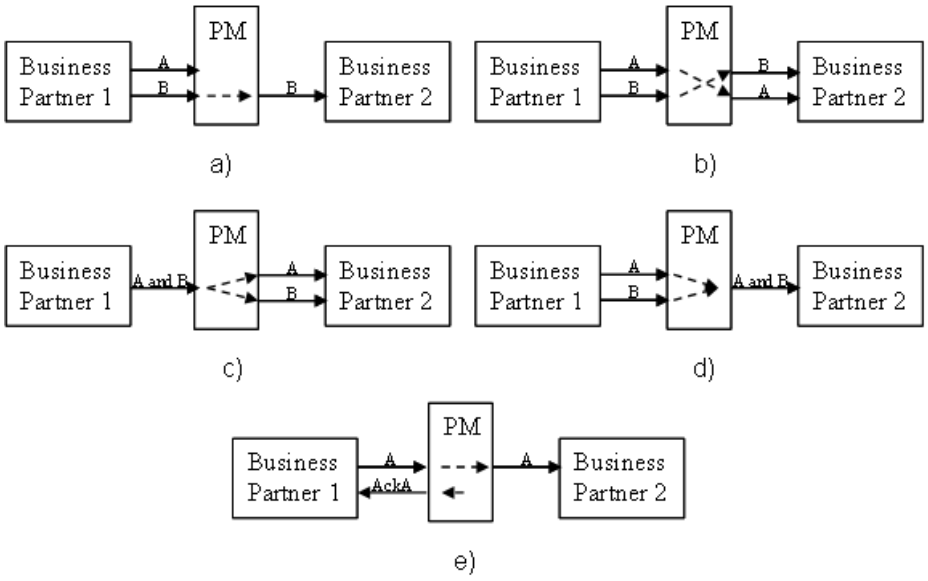

Fig. 1: Solvable Mismatches

Combining messages (Figure 1. 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 1. 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.

3.3 Web Services Composition

The Web service composition is the most complex action from the three phases described in this paper, and involves the previously two described actions.

Composition and Discovery - discovery is needed for composition in order to identify the services that need to be composed; any service that offers only part of the required functionality is a candidate for the composition.

Composition and Invocation - the composition of several services can be seen as a composition of several invocations; in order to compose different services, an execution environment needs to be able to communicate with all of them, sequentially or in parallel.

Similarly with the relation that should exist between a goal and a Web service (for the Web service to be discovered as a candidate for fulfilling the goal) the relation between the goal and the composition of services has to be `equal` or `plug-in`. In the following subsections, we will analyze the possible relations between the goal and the composed Web services, that would allow obtaining a valid composition. In any other

situation the composition is not correct or not complete (for example, if the required functionality subsumes the functionality offered by the composition of Web services, the composition is not complete - one or more other services need to be added to the composition)

For determining these relations between the functionality of the goal and the functionality of the potential services and the composed functionality, the functional mediation can be used. Also the data level mediation may be needed in case different ontologies are used for representing data.

The help of a process mediator is needed during the actual invocation of the composed services, if the behavior of the participants differ.

Exact Match The exact match between the functionality offered by the composition of the services and the one requested by the goal can appear only if

– all the composed services offere functionalities subsumed by the required functionality,

and

– the requested functionality is the exact reunion of the functionalities offered by the services.

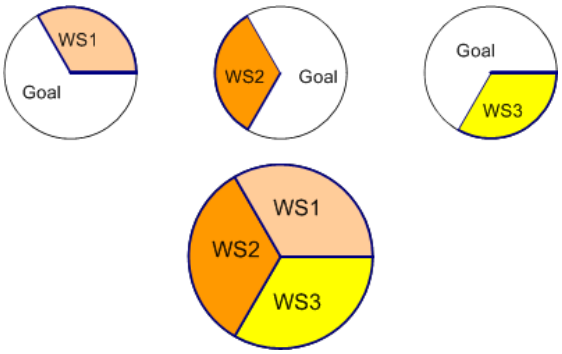

Fig. 2: Exact Match

Please note that the exact match between the goal's functionality and the functionality of the composed services still stands even if the functionalities of individual services overlap at some point.

An example of such a mach is the following:

G - requests train tickets between two locations in Austria, hotel reservation in the destination city and car hiring in the destination city;
WS_1 - offers train tickets between any two locations in Austria;
WS_2 - offers hotel accommodation in any city in Austria;
WS_3 - offers cars for rent in any city in Austria.

Plug-in match The plug-in match between the functionality required by the requestor and the functionalities offered by the composition of the services (i.e. composition subsumes the requested functionality) can appear only if

– all the composed services offere functionalities subsumed by the required functionality, or the intersection between the required functionality and the one offered by the services is not null,

and

– the requested functionality is a plug-in of the reunion of the functionalities offered by the services.

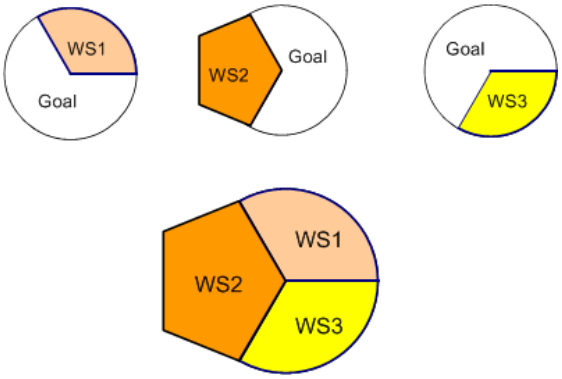

Fig. 3: Plug-in Match

Similarly with the previous case, the relation stands even if the intersection between the functionalities of individual services overlap at some point.

An example of such a mach is the following:

G - requests train tickets between two locations in Austria, hotel reservation in the destination city and car hiring in the destination city;
WS_1 - offers train tickets between any two locations in Austria;
WS_2 - and offers hotel accommodation in any city in Europe (not only in Austria);
WS_3 - offers cars for rent in any city in Austria.

4 Example

In order to illustrate how the mediation can be used in various stages of services' usages, we consider the following example: a client requesting train tickets between two Austrian cities, and hotel accommodation in the destination city. The available services are offering: WS_1- train tickets between any two cities in Europe, WS_2 - hotel accommodation in any city in Austria.

The lack of space does not allow us to give all the details regarding the goal, Web services, ontologies, and mismatches that may appear in such a scenario. The following sections illustrate some possible data and process mismatches, and the way these could be solved, and also the functional relations between the goal and WS_2 and the goal and the service composition [4]

4.1 Data Mediation

For illustrating the data mismatches and how the mismatches can be solved, we consider the following example.

The goal has in its ontology the concept `station` with the following definition:

```
concept station
 nonFunctionalProperties
   dc[5]#description hasValue "Station concept"
 endNonFunctionalProperties
 start_Location typeOf _boolean
 destination_Location typeOf _boolean
 name typeOf _string
```

where `start_Location` and `destination_Location` are two boolean attribute showing if the station represent the starting or the ending point of a trip, and name is the actual name of a station. For example, an instance of the station concept `S` having the `start_Location` set to true, and the `destination_Location` to false (assuming that internally there is an imposed condition on these attributes, that only one of them can be true) will be considered to be the starting point of a trip.

On the other hand the service WS_1 may have in its ontology the concept route, with the following definition:

```
concept route
 nonFunctionalProperties
   dc#description hasValue "Route concept"
   endNonFunctionalProperties
 from typeOf _string
 to typeOf _string
```

showing which are the names of departure and arrival stations.

Without the services of a data mediator, an execution environment would not be able to determine that from two instances of station an instance of route have to be created. For supporting this, a data mediator has to be able to create and execute rules similar with the following ones[6]:

```
Mapping(
 OG[7]#station
```

[4] All the examples are expressed using Web Service Modeling Language (WSML): `http://www.wsmo.org/wsml/`

[5] dc is the prefix we use to refer to Dublin Core non-functional properties set URL http://purl.org/dc/elements/1.1

[6] The rules are expressed in the Abstract Mapping Language [14] are generated and can be executed using the Web Service Execution Environment (WSMX) data mediation tool, available for download at `http://sourceforge.net/projects/wsmt`

[7] We use OG to denote the goal's ontology

```
  OS[8]#route
  classMapping( one-way station route))

Mapping(
  OG#destination_Location
  OS#to
  attributeMapping( one-way
    [(station) destination_Location => boolean]
    [(route) to => string]))
    valueCondition
      (station [(station) destination_Location => boolean] true)

Mapping(
  OG#start_Location
  OS#from
  attributeMapping( one-way
    [(station) start_Location => boolean]
    [(route) from => string]))
    valueCondition
      (station [(station) start_Location => boolean] true)

Mapping(
  OG#name
  OS#to
  attributeMapping( one-way
    [(station) name => string]
    [(route) to => string]))

Mapping(
  OG#name
  OS#from
  attributeMapping( one-way
    [(station) name => string]
    [(route) from => string]))
```

The first rule states the relations between `station` and `route`, the following two between the boolean attributes from the station and the `to` and `from` attributes from the route. The last two rules are showing the relation between the `name` attribute from the station and the `to` and `from` attributes.

4.2 Functional Level Mediation

For illustrating the functional level mediation we will describe the capabilities of the goal, WS_1 and WS_2 and also the capability of the services' composition. For simplicity reasons, we will present only the post-conditions, which describe the information space of the Web service after its execution; additionally we consider that all the involved parties use the same terminology.

```
goal G
 capability Gcapability
   postcondition Gpostcondition
     definedBy
       ?x[source_location hasValue ?cityS,
          destination_location hasValue ?cityD] memberOf ticket and
       ?cityS[locatedIn hasValue "Austria"] and
       ?cityD[locatedIn hasValue "Austria"] and
       ?h[locatedIn hasValue ?cityD] memberOf hotel and
       ?r[client hasValue ?p,
```

[8] We use OS to denote the service's ontology

```
          hotel hasValue ?h] memberOf Reservation and
       ?p memberOf person.[9]

webService ws1
 capability ws1capability
   postcondition ws1postcondition
     definedBy
       ?x[source_location hasValue ?cityS,
          destination_location hasValue ?cityD] memberOf ticket and
       ?cityS[locatedIn hasValue "Europe"] and
       ?cityD[locatedIn hasValue "Europe"].

webService ws2
 capability ws2capability
   postcondition ws2postcondition
     definedBy
       ?h[locatedIn hasValue ?city] memberOf hotel and
       ?city[locatedIn hasValue "Austria"] and
       ?r[client hasValue ?p,
          hotel hasValue ?h] memberOf Reservation and
       ?p memberOf person.

webService composedws
 capability composedwscapability
   postcondition composedwspostcondition
     definedBy
       ?x[source_location hasValue ?cityS,
          destination_location hasValue ?cityD] memberOf ticket and
       ?cityS[locatedIn hasValue "Europe"] and
       ?cityD[locatedIn hasValue "Europe"] and
       ?h[locatedIn hasValue ?city] memberOf hotel and
       ?city[locatedIn hasValue "Austria"] and
       ?r[client hasValue ?p,
          hotel hasValue ?h] memberOf Reservation and
       ?p memberOf person.
```

The functional relation between G and WS_2 has to illustrate the fact that WS_2 offers less than what the goal requests [10]:

```
source G
target WS2
nonFunctionalProperties
  dc#type hasValue subsume[11]
endNonFunctionalProperties
  definedBy
    ?h[locatedIn hasValue ?cityD] memberOf hotel and
    ?r[client hasValue ?p,
        hotel hasValue ?h]
    memberOf Reservation and
    ?p memberOf person.
```

The functional relation between the goal and the composition of services has to illustrate the fact that the composition offer more functionality then the one required by the goal.

---

[9] "?" denotes a variable in WSML.

[10] The relations are expressed based on [16]; this is still on-going work, and there is no tool available for generating them

[11] This non-functional-property is used to express the way the subsumption relation should be read, i.e G subsumes WS2 by ...

```
source G
target composedws
nonFunctionalProperties
  dc#type hasValue plug-in
endNonFunctionalProperties
  definedBy
    ?x[source_location hasValue ?cityS,
       destination_location hasValue ?cityD] memberOf ticket and
    naf[12] ?cityS[locatedIn hasValue "Austria"] and
    naf ?cityD[locatedIn hasValue "Austria"].
```

4.3 Process Level Mediation

For illustrating process mediation mismatches we will provide a piece of the goal choreography and and a piece of WS_1 choreography. Both of them are representing using WSMO definition of choreographies [15], which respects the ASM specifications.

In WSMO, the owner of any instance (that is, the concept that is instantiated) expected by an entity is part of an `in` list, and any owner of an instance that should be sent by an entity is part of an `out` list. Further more, the order in which the instances are expected is set by using transition rules, consisting of conditions and updates.

The goal choreography contains the following rules:

```
 /*
* the invocation starts with the creation of a date instance; no condition
* need to be fulfilled in order to create this instance
*/
do
  add(_#[
    year hasValue ?year,
    month hasValue ?month,
    day hasValue ?day
  ]memberOf tro#date)

/*
* after the date is created, the requestor creates an instance of station -
* the starting point of the trip
*/
forAll ?date with (?date[] memberOf
tro#date
  ) do
    add(_#[
      start_Location hasValue _boolean("true"),
      destination_Location hasValue _boolean("false"),
      name hasValue ?name
    ]memberOf tro#station)
endForAll

/*
* after the instance denoting the starting point of the trip exists, the
* requestor creates an instance denoting the destination point
*/
forAll ?station with (?station[
  start_Location hasValue _boolean("true"),
  destination_Location hasValue _boolean("false"),
  ] memberOf tro#station
  ) do
    add(_#[
      start_Location hasValue _boolean("false"),
      destination_Location hasValue _boolean("true"),
```

[12] naf stands for negation as failure in WSML [8]

```
      name hasValue ?name
    ]memberOf tro#station)
endForAll
```

where `station` and `date` are both part of the `out` list of the choreography. The choreography of WS_1 contains the following rules:

```
 /*
* the invocation starts when the service receives an instance of route
*/
do
  add(
    _#[ from hasValue ?from,
    to hasValue ?to
  ]memberOf too#route)

/*
* after the route is created, the service expects an instance of date - the
* date of the trip
*/
forAll ?route with (?route[] memberOf
too#route
  ) do
    add(_#[
      year hasValue ?year,
      month hasValue ?month,
      day hasValue ?day
    ]memberOf too#date)
endForAll
```

where `route` and `date` are part of the `in` list.
A graphical representation of the two choreographies is illustrated in Figure 4. a).

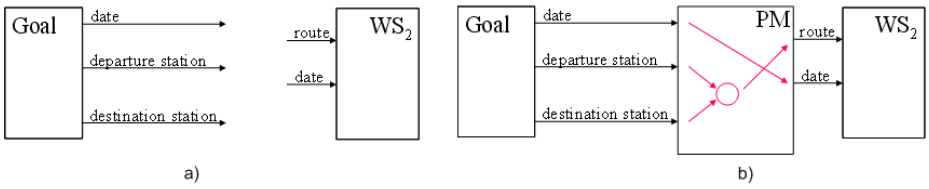

Fig. 4: Choreography Mismatches

What the process mediator should do in this case is represented in Figure 4. b). For example, the way a Process Mediator could work is [2]:

- when the instance of `date` is sent by the requestor, the process mediator should determine that this instance will be expected at some point in time (by checking the service's `in` list)but it is not expected by the service at the beginning of the conversation. As a consequence, the `date` instance should be stored for further use.
- when the first instance of `station` is received, the process mediator should invoke a data mediator in order to obtain the equivalent instance in terms of the service's ontology. The data mediator will return an instance of `route`, which conforming

with the service's choreography is expected at this point of the communication. The problem is that this instance is incomplete, does not have all the attributes instantiated as required, so the process mediator will store the `station` instance for further use.
- when the second instance of `station` is sent, the process mediator invokes a data mediator with both `station` instances, obtaining a correct instance of `route` from the service's point of view. As the `route` is expected, this instance is sent to the service.
- after the `route` is sent, the process mediator determines, based on the service's choreography that an instance of `date` is expected now. The `date` instance is retrieve from the internal storage and sent.

The WSMX process mediator prototype[13] is able to perform this kind a computations, and to address all the types of mismatches identify in [2]

5 Related Work

Data mediation represents an old research topic that was reshaped and re-explored in the semantic context. Semantic-based solution have been proposed that offer better, more-dynamic and Web oriented mediators in a more-effective and effort saving manner [9,11,3].

At the same time, processes mediation is still a poorly explored research field. 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(see for example Contivo[14] and CrossWorlds[15]).

As far as we know the functional mediation has not been directly addressed in any other work. However similar classification and functional relationships were explored in various discovery working groups [6,13,5] as prerequisites for the discovery engines.

Even if as future development this work aims in providing complete solution for this three areas of mediation, this paper focuses on analyzing how these complementary techniques can be integrated and used in the main steps of Semantic Web services usage. We are not aware of ay similar overview and work towards a complete mediation framework for Semantic Web services.

6 Conclusions

This paper emphasizes the importance of mediators in a Semantic Web services infrastructure, illustrating why and how the mediators can be used during Semantic Web services discovery, invocation and composition. These three phases, considered to be

[13] Available for download from `http://sourceforge.net/projects/wsmx/`
[14] `http://www.contivo.com`
[15] `http://www.sars.ws/hl4/ibm-crossworlds.html`

of highly importance for the Semantic Web services usage are explained in the paper, and the appropriate mediation technics are identified. These techniques refer to data mediation (tackling the terminology and representation mismatches), process mediation (addressing the public process mismatches, i.e., communication mismatches) and functional mediation (bridging various required and offered capabilities).

The paper also presents examples of mismatches that can appear in a Semantic Web services environment, and it proposes ways of solving these heterogeneity problems.

References

1. E. Börger and R. Stärk. *Logical Foundations of Artificial Intelligence*. Springer, Berlin, Heidelberg, 1987.
2. E. Cimpian and A. Mocan. WSMX Process Mediation Based on Choreographies. In *Proceedings of the 1st International Workshop on Web Service Choreography and Orchestration for Business Process Management at the BPM 2005, Nancy, France*, 2005.
3. J. Euzenat, D. Loup, M. Touzani, and P. Valtchev. Ontology alignment with ola. *Proc. 3rd ISWC2004 Workshop on Evaluation of Ontology-based Tools (EON), Hiroshima, Japan*, pages 59–68, 2004.
4. C. Feier, A. Polleres, R. Dumitru, J. Domingue, M. Stollberg, and D. Fensel. Towards intelligent web services: The web service modeling ontology (WSMO). *International Conference on Intelligent Computing (ICIC)*, 2005.
5. H.-C. Hsiao and C.-T. King. Neuron - A Wide-Area Service Discovery Infrastructure. *In Proceedings of the Internaltional Conference on Parallel Processing (ICPP'02)*, 2002.
6. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic Location of Services. In *Proceedings of the 2nd European Semantic Web Conference (ESWC 2005), Crete, Greece*, 2005.
7. U. Keller, R. Lara, and A. Polleres (eds.). WSMO Web Service Discovery. Deliverable D5.1, 2004. available at: http://www.wsmo.org/TR/d5/d5.1/.
8. H. Lausen, J. de Bruijn, A. Polleres, and D. Fensel. WSML - A Language Framework for Semantic Web Services. *W3C Workshop on Rule Languages for Interoperability*, April 2005.
9. A. Maedche, B. Motik, N. Silva, and R. Volz. Mafra - a mapping framework for distributed ontologies. *Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW)*, September 2002.
10. A. Mocan and E. Cimpian. Mapping creation using a view based approach. *1st International Workshop on Mediation in Semantic Web Services (Mediate 2005)*, December 2005.
11. N. Noy. Semantic Integration: a Survey of Ontology-based Approaches. *ACM SIGMOD Record*, 33(4):65–70, 2004.
12. D. Roman, H. Lausen, and U. Keller (eds.). Web Service Modeling Ontology (WSMO). Deliverable D2, 2005. available at: http://www.wsmo.org/TR/d2/.
13. B. Sapkota, L. Vasiliu, I. Toma, D. Roman, and C. Bussler. Peer-to-peer technology usage in web service discovery and matchmaking. *Sixth International Conference on Web Information Science and Engineering (WISE 2005)*, November 2005.
14. F. Scharffe and J. de Bruijn. A language to specify mappings between ontologies. In *Proc. of the Internet Based Systems IEEE Conference (SITIS05)*, 2005.
15. J. Scicluna, A. Polleres, and D. Roman (eds.). Ontology-based Choreography and Orchestration of WSMO Services. Deliverable D14, 2005. available at: http://www.wsmo.org/TR/d14/.
16. M. Stollberg, E. Cimpian, and D. Fensel. Mediating Capabilities with Delta-Relations. In *Proceedings of the First International Workshop on Mediation in Semantic Web Services, Amsterdam, the Netherlands*, 2005.