

# Expressing WSMO Mediators in OWL-S\*

Massimo Paolucci, Naveen Srinivasan, and Katia Sycara

Carnegie Mellon University  
Pittsburgh, Pennsylvania, USA  
{paolucci,naveen,katia}@cs.cmu.edu

**Abstract.** The WSMO/WSML/WSMX initiative and the OWL-S initiative aim at similar goals: providing a support for Semantic Web services. Despite the similarity of the ultimate goal, the two approaches are very different. In this paper, we analyze one of such differences, namely the concept of mediators in WSMO, and we show how those mediators can be expressed in OWL-S. The result of this work shows that, given the focus of OWL-S, mediators should not be first class citizens in OWL-S, but that they can be expressed and discovered nevertheless.

## 1 Introduction

WSMO and OWL-S aim at representing Web services that make an essential use of ontologies from the Semantic Web, and the objective of both efforts is to support more effective discovery, composition and interoperation. Yet, the two efforts take very different approaches. WSMO stresses the role of mediation in order to support automatic interoperation between Web services, while OWL-S stresses action representation to support planning processes that provide automatic composition.

In WSMO's vision, mediators address interoperability problems that emerge when different Web services work together. But, while WSMO mediators represent a new concept in the field of Semantic Web services, the interoperability problems that they solve are not new. Indeed these problems are pervasive of every application and every aspect the Web services life cycle from discovery to composition and enactment.

Despite the importance of mediators in the WSMO ontology, the concept of mediator is still underspecified. As we shall show in the paper, virtually any program can be described as a WSMO mediator. While this is not surprising given that WSMO is in its early stages of development, it also makes very difficult to make any claim on the features as well as on the use of WSMO mediators. The first contribution of this paper is to provide a critical analysis of the WSMO

---

\* We would like to thank Rubén Lara and the participants of the *deri-wsmo-discussion* mailing list for their insights into WSMO's concept of mediators. The research was funded by the Defense Advanced Research Projects Agency as part of the DARPA Agent Markup Language (DAML) program under Air Force Research Laboratory contract F30601-00-2-0592 to Carnegie Mellon University.

mediators, as we understand it, with the objective to further the community understanding of this concept.

Nevertheless, the stress that WSMO poses on mediators provides a challenge for OWL-S: if OWL-S aims at supporting automatic composition of Web services, it needs to address the mediation problem. The second contribution of this paper is to identify how mediators are realized in OWL-S, and, to the extent that mediators are Web services, we also provide their description in OWL-S.

In the rest of the paper we assume some familiarity with the three modules of OWL-S, and specifically with the OWL-S Service Profile and Process Model, while in the following section 2 we will provide an introduction to the WSMO mediators. In section 3 we will discuss what we see as shortcomings of the definition of mediators in WSMO that are mostly due to the concept is still quite underspecified. In section 4, we will analyze the role of mediators in OWL-S and we will show how mediators can be represented in the language. In section 5 we will describe an example of broker, which can be represented as a stateful mediator, that performs mediation among Web services and the challenges that it poses to OWL-S. Finally, in section 6 we conclude.

## 2 WSMO Mediators

WSMO is an ontology for the description of Web services in the infosphere. The definition of WSMO hinges on the following four concepts: *Web services*, *Goals*, *Ontologies* and *Mediators*. The following list provide, to the best of our understanding, a succinct explanation of the meaning of these four concepts.

**Webservices** expose the interface of businesses on the Internet. Each Web service is characterized by a set of capabilities which specifies the functionalities provided by the Web service, and a set of interfaces that specify how the capabilities are fulfilled.

**Goals** “A goal specifies the objectives that a client may have when he consults a web service<sup>1</sup>”. Goals are characterized by a *PostConditions* that describe the information state that the client desires, and *Effects* that describe the state of the World that the client desires to achieve.

**Ontologies** provide a shared conceptualization and a formal specification of the domain. The use of ontologies achieves two objectives: first it provide a formal semantics to information exchanged by Web services facilitating the Web services interoperation; second, ontologies facilitate the interoperation between humans and Web services by specifying the precise terminology accepted by the Web services.

**Mediators** provide a general mechanism to reconcile the inevitable interoperability issues that emerge when Web services need to work together. Mediators are the core of our attention in this paper and we will analyze them in some detail.

---

<sup>1</sup> Cited from: D2v01. Web Service Modeling Ontology (WSMO), (<http://www.wsmo.org/2004/d2/v01/20040214>)

Mediators are defined to attack the alignment problems that inevitably emerge when interoperability of Web services is attempted. For example, Web services may use different ontologies, or different protocols or they may have been designed with different goals in mind. Through the solution of the interoperability problem, mediators are expected to tackle other two problems: namely reusability and scalability. Reusability is facilitated by the use of mediators because the same component may be used in very different context by defining a set of mediators that solve the interoperation problems generated in that particular context. Scalability instead is the result because instead of providing  $n^2$  direct mappings between different components, only mappings between mediators will be needed, and these mappings are expected to be far fewer reducing the interoperability complexity when compared with the direct mappings.

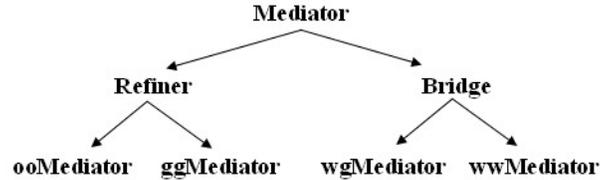


Fig. 1. WSMO taxonomy of mediators

Four types of mediators have been defined that are specialized to handle mediation between instances of the four basic concepts of WSML. The different types of mediators that are organized in the taxonomy shown in figure 1. Their main classification of mediators distinguishes two main types: *Refiners* and *Bridges*. Refiners specify “a new component as a refinement of an existing components. They support the reuse by minimizing the effort in generating new components from existing ones<sup>2</sup>.” Bridges instead enable different components to interoperate overcoming their interoperability problems. For instance bridges, such as *ww-mediator* presented below, may transform the outputs of a Web service into the inputs of another Web service.

Specifically, the *oo-mediators* and the *gg-mediators* described below are refiners, while the *ww-mediators* and the *wg-mediators* are bridges.

**OO-Mediators** provide translations and harmonization between different ontologies that are used by the Web services or any other WSMO component.

**GG-Mediators** provide a way to match goals at different levels of granularity.

For example a GG-Mediator may take the responsibility to refine the goal

<sup>2</sup> Cited from “D2v02. Web Service Modeling Ontology - Standard” (<http://www.wsmo.org/2004/d2/v0.2/20040306/>), section 5.

buy a ticket to the goal buy a train ticket upon recognizing that there is a subclass relation between the two concepts<sup>3</sup>.

**WG-Mediators** handle partial matches between goals of the clients and the functionalities provided by Web services.

**WW-Mediators** resolves the interoperability issues between Web services at all levels: data, process, and protocol. These interoperability problems are resolved both at the level of the single Web service choreography, as well as at the level of the orchestration of multiple Web services.

One common thread that characterizes all the WSMO mediators is that they can call each other. Typically, OO-Mediators are used at all levels since ontology mismatches are the most likely of all possible mismatches. But it is also possible to imagine that a WG-Mediator will invoke a GG-Mediator to decompose the goal and facilitate the matching with the capabilities of a Web service.

## 2.1 Expressing Mediators

WSMO provides a language to express the characteristics of Mediators. In the following we show two examples of Mediators. The first one an oo-Mediator, the second one a gg-Mediator.

```
ooMediator http://www.wsmo.org/.../owlCurrencyMediator.wsml
  namespace [...]
  non-functional-properties [...]
  sourceComponent
    ontology http://www.daml.ecs.soton.ac.uk/ont/currency.daml
  targetComponent
    ontology http://www.wsmo.org/.../resources/po.wsml
  mediationService [...]
```

**Fig. 2.** An example of specification of an OO-Mediator

Figure 2<sup>4</sup> shows *owlCurrencyMediator*, an example of oo-Mediator. The **namespace** and **non-functional-properties** have been abbreviated for readability reasons, but they provide a namespace mechanism equivalent to what is provided by XML and information on the authors of the mediator. Instead **mediationService**, which is unspecified in the original listing, a full wsml-webservice description. The properties **sourceComponent** and **targetComponent** which specify the source DAML+OIL ontology<sup>5</sup> and the target WSML ontology<sup>6</sup>. In this mapping the

<sup>3</sup> The example is taken from “D3.2 v0.1. WSMO Use Case Modeling and Testing” (<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040719/>), section 3.1.2.

<sup>4</sup> The example is taken from “D3.2 v0.1. WSMO Use Case Modeling and Testing” (<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040719/>), Listing 8

<sup>5</sup> <http://www.daml.ecs.soton.ac.uk/ont/currency.daml>

<sup>6</sup> <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040607/resources/po.wsml>

source ontology specifies concept such as `Country` and `Currency` while the target ontology specifies trading concepts such as `Buyer`, `Seller` and `price`. Considered the distance between the two ontologies, it is natural to expect that the resulting mapping is partial. But to the best of our knowledge such partiality is not specified anywhere in the definition and we can only speculate that it will be specified in the definition of the `mediationService`.

```

ggMediator
  http://www.wsmo.org/2004/d3/d3.2/v0.1/20040607/resources/ggm1.wsml
  namespace [...]
  non-functional-properties [...]
  sourceComponent
    goal http://www.wsmo.org/.../goal1.wsml
  targetComponent
    goal http://www.wsmo.org.../goal.wsml
  mediationService
    axiom GGReduction
    logical-expression
      "trainItinerary memberOf tc:itinerary[
        trip hasValue someTrip memberOf tc:trainTrip[
          start hasvalue LOC,end hasvalue LOC,],
        passenger hasValue _# memberOf loc:person
      ] and
      (LOC..locatedIn = austria or
       LOC..locatedIn = germany)."
```

**Fig. 3.** An example of specification of an GG-Mediator

A GG-Mediator is provided in figure 3<sup>7</sup> which specifies the mapping between `goal1` which corresponds to buy a ticket to a more specific goal which corresponds to "buy a train ticket". In this example the `mediationService` provides a restriction on the location that can be either in Austria or in Germany.

Other types of mediators are specified in a similar way, but since we could not find neither an example, nor a specification of the language, we do not venture to provide our own examples.

### 3 Problems with the WSMO Concepts of Mediator

Despite its importance, the concept of mediator in WSMO is still underspecified. Virtually any program can be re-casted as a mediator, or as composed of mediators, since any program produces a mapping from a data-type to another or a mapping from a functionality to another. Any attempt to define mediators

<sup>7</sup> The example is taken from "D3.2 v0.1. WSMO Use Case Modeling and Testing" (<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040719/>), Listing 11

only on the bases of generality or reusability does not quite hold. For example, the OO-Mediator in figure 2 is very specialized and do not generalize to any other mediation problem, though it may be argued that such a mediator may take advantage of more reusable mediators that map DAML+OIL to WSML. In general though, mediators that are generated with the WSMX tools<sup>8</sup> seem to generate very specific sets of rules that depend on the ontologies from which they are constructed. Part of the problem with the concept of mediator in WSMO is that the role of semantics and the Semantic Web have not been exploited. For example, it would be nice to see a way to derive OO-Mediators automatically as has been proposed by the C-OWL project [2]. Indeed, profound questions on what is the contribution of the Semantic Web provide to the interoperability problem remain unanswered.

Another aspect that is still unclear is whether mediators are client side components, or server side components or whether they are services that reside somewhere between the client and the server. Furthermore, it is universally recognized that the interaction between Web services is separated in three different problems: discovery, composition and invocation, but it is still unclear, where each type of Mediator plays a role. In this section we will try to address these problems.

### 3.1 Where do Mediators Reside?

It is still underspecified is whether mediators reside in the client, in the server or somewhere in between acting like a Middle Agents [14]. WSMO seems to collapse two different concepts in the same mediator concept. The first one is Mediators as internal modules or translation rules. The second is a view of Mediators as services which provide translation services to other Web services.

Architecturally speaking, these two models of mediation are very different: rules describe private knowledge of the Web service, and of its clients, as such they specify an internal module of the Web service infrastructure. On the other hand, if mediators are interpreted as Web services, they specify components of the Web services infrastructure; they are components of the overall infrastructure as UDDI servers are.

If Mediators are Web services, important issues need to be specified such as how are Mediators discovered; and how are mediators used. WSMO is still silent on these issues. The representations in figures 2 and 3 suggests that one possible way to perform the discovery of mediators is to exploit the **sourceComponent** and **targetComponent** in a way that is similar to the inputs and outputs in the OWL-S Profiles. The discovery algorithms that have been developed for OWL-S may be used in the context of WSMO.

As for the interaction, the problem is much deeper, and it depends on the WSMO choreography and orchestration languages that are still undefined. The simpler case here is to define stateless mediators that take a message from the client and send it to the server and vice versa. Such stateless mediator would

<sup>8</sup> “D13.3v0.1 Mediation” available at <http://www.wsmo.org/2004/d13/d13.3/v0.1/20040628>

not keep track of the interaction between the two parties. For example, the *owlCurrencyMediator* above seems to be of this type. A more complex case is when the mediator needs to maintain an history of the interaction. For example, an OWL-S to WSML mediator would have to satisfy this requirement since it will have to present to act as an OWL-S Web service on one side, and as a WSML web service on the other, which in turn means that it has to maintain the two services synchronized consistently with their process models.

We believe that in its evolution, the specification of WSMO will have to recognize the distinction between defining mediators as rule sets, and defining mediators as services in order to define a discovery mechanism and provide support for complex interaction protocols.

### 3.2 Mapping Mediators to Web services Problems

The mapping of Mediators to the problems of Web services is still quite under-specified. The use of Web services is accompanied by three problems: discovery, composition and invocation, but there is no clear specification of how the mediators help the solution of them. In this section we attempt to define a mapping between the Web Services problems and the WSMO mediators.

Composition and invocation of Web services seems to be addressed principally by WW-mediators who “are applied to resolve heterogeneities on the data, process, and protocol level between the Choreographies of Web Services that are ought to interact in a global interaction model”<sup>9</sup>.

The discovery problem presents a more complex picture, since GG-mediation and WG-mediation seem to be relevant during discovery when the goals of the requester need to be mapped to the goals of a Web service. Nevertheless, there seems to be a difference between the type of discovery that is supported by OWL-S matchmakers [13, 7, 3] and the functionality that is provided by mediations. Matchmakers attempt to check whether the goal of the requester matches the goal of one or more providers. When a match is not recognized the matchmaker fails. Similarly given two goals the mediators attempts to find map between them. But it is not quite clear what happens if such a mapping does not succeed. One solution would be to fail, in which case mediators behave similarly to a match in the OWL-S discovery. Alternatively, mediators could be used to reconcile the differences between goals, in which case the use of mediators during discovery may be disputable since every service would be discoverable. Essentially, the role of gg-mediators is still quite unclear, if they can be used for selection, then their use in discovery is quite straightforward; alternatively, if instead they are used for reconciliation, then their role may be more important in composition and discovery in WSMO would be still completely undefined.

While WW-mediation seems to be principally about composition and invocation, while WG and GG-mediation may be applied to discovery, OO-mediation

---

<sup>9</sup> Cited from D3.2 v0.1. WSMO Use Case Modeling and Testing (<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040719>), discussion on WW-Mediators in section 3.1.2

is pervasive in WSMO since differences between ontologies may emerge during discovery, composition and invocation.

## 4 Representing WSMO Mediators in OWL-S

If our hypothesis on the role of the different mediators in addressing the fundamental problems of Web services is correct, then it also provides an initial mapping between infrastructure that is assumed by OWL-S Web services and the mediators stipulated by WSMO. For instance, above we showed that gg-mediators and wg-mediators may provide the basic step toward a discovery mechanism. In such a case, the matching algorithms that have been defined to support discovery in OWL-S may also be used to define those WSMO mediators. Similarly, while OWL-S does not mandate the existence of ww-mediators or wg-mediators, it assumes the existence of composition mechanisms, such as planning [15, 10, 12, 9] that perform essentially the same role of the mediators.

We believe that mediators can be represented within OWL-S, and that such a representation can be used to support discovery and composition of mediators and Web services during orchestration. As an initial test of this hypothesis, in this section we provide an OWL-S description of three different mediators. The first one is a detailed description of an oo-mediator that translates F-Logics into OWL<sup>10</sup>; the second one is the gg-mediator in figure 3; the last one is a ww-mediator that has been described in [6].

### 4.1 OO Mediators

The first mediator translates from OWL ontologies [4] to F-Logic ontologies [8]. Although such a translation may be extremely complicated, an algorithm for such a mapping can be derived from [1]. Since we take the OWL-S point of view, we describe the inputs and outputs required by that algorithm ignoring, for explanation purposes, issues such as performance and completeness of the mapping. A scenario in which this mediator may be used in the following one. An agent which uses OWL as ontology description language may want to communicate with a Web service that instead of OWL uses F-Logic. At discovery time the agent would use a profile that specifies the transformation that it expects from the Web service, namely the translation from OWL to F-Logic. A profile describing that transformation is described in figure 4<sup>11</sup>.

<sup>10</sup> We thank Rubén Lara for proposing this example as a challenge for OWL-S.

<sup>11</sup> Note that our representation assumes the existence of a concept *FLogic* in an OWL ontology. In principle one may wonder how to discover a Web service that translates between a given OWL ontology and a given F-Logic ontology, or between statements derived from ontologies. OWL-S approach to this problem is the following: first the client identifies the logics used to represent the ontologies or the statements, in our examples OWL and F-Logic; second, the profile shown in figure 4 is used for discovery, then the found service is used for the translation.

```

<profileHierarchy:oo_mediator rdf:ID="Owl2FLogic_Translator">
[... ]
<profile:hasInput>
  <process:Input rdf:ID="OWL_Input">
    <process:parameterType rdf:datatype="&xsd;#anyURI">
      http://www.w3.org/2002/07/owl#Thing
    </process:parameterType>
  </process:Input>
</profile:hasInput>

<profile:hasOutput>
  <process:Output rdf:ID="FLogic_Output">
    <process:parameterType rdf:datatype="&xsd;#anyURI">
      http://www.wsmo.org/2004/d16/d16.2/v0.1/fLogic
    </process:parameterType>
  </process:Output>
</profile:hasOutput>
[... ]
</profileHierarchy:oo_mediator>

```

**Fig. 4.** The profile of an oo-mediator

This profile specifies that the input of the Web service is an OWL concept, while the output is an F-Logic statement formatted consistently with the WSMO serialization. Additional constraints could be defined in the preconditions and effects to specify additional features such as partial mapping and restrictions that guarantee that the object represented in the output is indeed the same thing of the input.

Assuming for sake of simplicity that the oo-mediator found is stateless, the process model for this Web service is quite trivial and it requires just an OWL object as input to be sent to it, and it returns F-Logic statements as output. The Web service described in figure 5 simply computes the mapping and sends it back to the client, who then takes the responsibility to send it to the F-Logic service. Alternatively, the OO-mediator may interact directly with the server providing a real translation service. The process model in the latter case is shown in figure 6. Here the mediator takes as input the message to translate and a recipient, and it returns as output an acknowledgment, and provides as effect the guarantee that the recipient will receive a message in F-Logic that is the translation of the input. Of course the condition in the effects will have to be expressed in SWRL or some other rule language.

The pattern described in this section can be used for other type of mediators. For instance the gg-mediator in figure 3 is represented by substituting the inputs and outputs of the Profile with concepts that correspond to WSMO goals. Additional restrictions may be added to represent additional information such as the restriction to travel in Austria or Germany.

```

<process:AtomicProcess rdf:ID="Owl2FLogic_Translator_Process">
  <process:hasInput>
    <process:Input rdf:ID="OWL_Input">
      <process:parameterType rdf:datatype="&xsd:anyURI">
        http://www.w3.org/2002/07/owl#Thing
      </process:parameterType>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:Output rdf:ID="FLogic_Output">
      <process:parameterType rdf:datatype="&xsd:anyURI">
        http://www.wsmo.org/2004/d16/d16.2/v0.1/fLogic
      </process:parameterType>
    </process:Output>
  </process:hasOutput>
</process:AtomicProcess>

```

Fig. 5. The process model of an oo-mediator

## 4.2 WW Mediator

In this second example we describe a mediation task that has been proposed in ([6] section 3.4.1). The scenario for this mediator is the following: two business parties *P* and *R* need to exchange purchase orders, yet *P* intends to send each line item in a separate message, while *R* expects the complete purchase order. The role of the mediator *M* in this case is to assemble the line items coming from *P* in a complete purchase order and send it to *R*.

This mediator can be expressed in OWL-S in two different ways. The first solution can be achieved internally to *P*. Specifically, *P* may have some knowledge of how to take line items coming from its purchase department and fuse them in a complete purchase order. Typically, these capabilities are described as planning operators, therefore the mediator would be automatically constructed through *P*'s internal planning process. This solution does not use OWL-S directly, yet the goal of composing one purchase order would come from *R*'s requirements expressed in the process model.

The second solution, which seems to be entailed by the example, requires the use of a ww-mediator<sup>12</sup>. In this case the discovery of the mediator is again driven by the requirements of the *R*'s process model (to receive a single purchase order) and *P*'s requirement to send all the line items separately. During the discovery, a Profile can be generated expressing the request for the mediator. Such Profile

<sup>12</sup> We assume that the type of mediator required here is a ww-mediator because it mediates between the Web service of *P* (or at least *P* as a Web service client) and the Web service of *R*. Nevertheless, an argument could be made that really the mediator needed is a wg-mediator which harmonizes the the goals of *P* (send each line item separately) with the goals of *R* (receive a single purchase order).

```

<process:AtomicProcess rdf:ID="Owl2FLogic_Translator_Process">
  <process:hasInput>
    <process:Input rdf:ID="OWL_Input">
      <process:parameterType rdf:datatype="&xsd:anyURI">
        http://www.w3.org/2002/07/owl#Thing
      </process:parameterType>
    </process:Input>
  <process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="F-LogicWS">
      <process:parameterType rdf:datatype="&xsd:anyURI">
        http://www.daml.org/services/owl-s/1.1B/Service.owl#Service
      </process:parameterType>
    </process:Input>
  <process:hasInput>
  <process:hasOutput>
    <process:Output rdf:ID="Acknowledgement">
      <process:parameterType rdf:datatype="&xsd:String">
      </process:parameterType>
    </process:Output>
  <process:hasEffect>
    receive(#F-LogicWS,msg) & F-Logic(msg) & translation(#OWL_Input,msg)
  </process:hasEffect>
</process:AtomicProcess>

```

**Fig. 6.** The process model of an oo-mediator revised

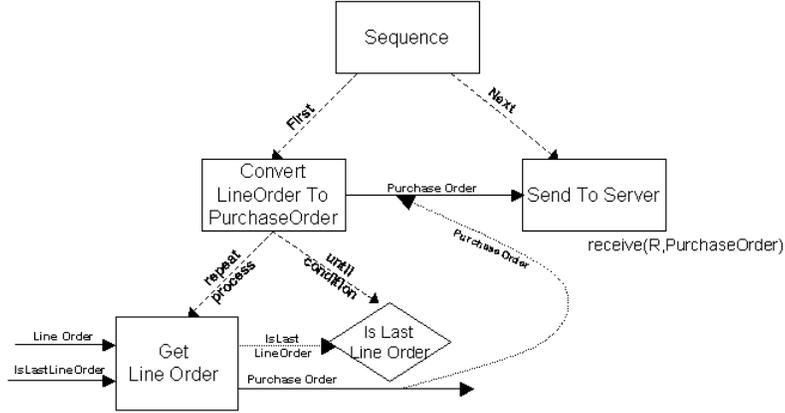
will specify that the expected inputs of the desired mediator are the line items, and the expected output is a complete purchase order.

The Process Model of such mediator is schematically<sup>13</sup> shown in figure 7. It consists of a sequence of two processes of which the first one **Convert LineItem to PurchaseOrder** is a loop that collects all the line items (exploiting the process **Get Line Item**) and compiles the purchase order. The second process, **Send To Server** is responsible to send the constructed purchase order to R. Although not shown, **Send To Server** may produce an acknowledgment of success or failure, but more importantly it specifies that the effect of the sending is that R receives the purchase order.

## 5 OWL-S Broker

The ww-mediator discussed in the previous section mediates between the Web service protocol of the seller, namely R, that expects one single message and the Web service protocol of the buyer, namely P, that expects to send multiple

<sup>13</sup> We skip the complete listing for both space and explanation clarity reasons.



**Fig. 7.** The Process Model the WW Mediator

messages. This is a specific case of a *broker* middle agent [5] that mediates between two different Web services.

The basic protocol of the broker is shown in figure 8. The broker collects the advertisements of providers available in the infosphere and store them for future discovery. Upon receiving a request of service (1) from a requester, the broker selects the best provider to address that request and (2) the broker passes the request to the selected provider. Upon receiving the request from the selected provider (3), the broker relays the information back to the requester.

The apparent simplicity of the broker's protocol is quite deceiving. The interaction between the provider and the requester may not be restricted to one single loop, but it may involve many exchanges. For instance, the requester may have asked for a ticket from Innsbruck to Frankfurt, but the provider, presumably the railway station, may want to know which train the requester intends to use. The request for additional information sparks a new loop through the protocol since rather than answering the requester, at step (4) the broker ask the requester for some additional information to send to the provider.

The second level of complexity stems from the fact that protocol used by the requester may not agree with the protocol used by the provider. For example the requester may have asked for a ticket on the 4:30 train from Innsbruck to Frankfurt, but the first asks for the departure and arrival stations and only later for the time of departure. This time around, the broker would have to recognize that the requester already provided the information and that it is useless to ask for the information again. Rather the broker can shortcut the protocol directly answer the provider.

The quick glimpse in the broker protocol shows that it summarizes all four WSMO mediators. The broker provides a ww-mediation to harmonize the pro-

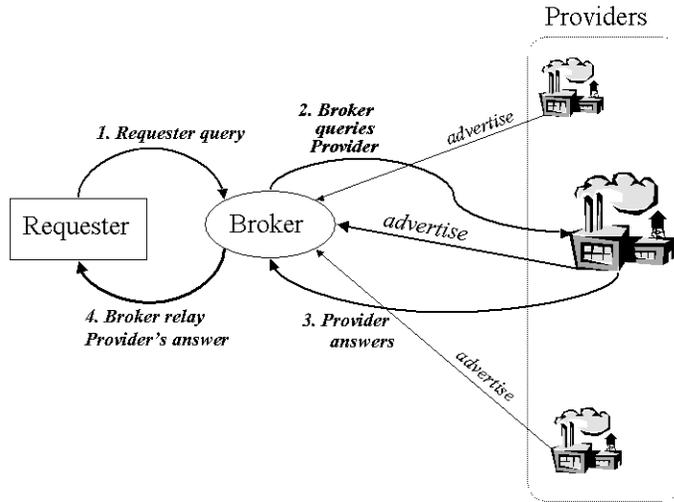


Fig. 8. The Broker's Protocol

tol of the provider with the protocol of the requester; furthermore, it performs oo-mediation to abstract the incoming query from the requester to the corresponding goal to be used select the best provider. Also, gg and wg-mediation are performed to link the selected goal to the provider's web service that actually interacts with the broker. Potentially, there are additional uses of mediators, for instance the oo-mediator may be used to reconcile different ontologies used by the provider and the requester.

Since the broker summarizes the all the WSMO mediators it becomes an interesting test case for OWL-S. An implementation of the broker is provided in [11] where details of the mediation processes are provided. Brokers challenge OWL-S' assumption that process model of the provider should be provided in advance since the requester cannot query the broker until a process model is provided, and the broker cannot provide a process model until the query is issued and a provider is selected. The solution of this dilemma is to support dynamic process modeling that allow the interaction protocol of a Web service to change mid-stream. Crucially, the limitation of OWL-S does not come from the representation of WSMO style mediators, but on the assumption that the Process Model of Web services is always prespecified. The experience of the broker suggests that OWL-S can represent the mediators that WSMO specifies, but it also shows that the flexibility to change interaction protocol mid stream will also be required for WSMO mediators.

## 6 Conclusions

In this paper we tackle the problem of representing WSMO mediators in OWL-S. Since the problems the mediators attempt to solve in the Web services infrastructure are not new, the issue of representing mediators has been decomposed into two problems: first finding which other components of the Web service infrastructure that is assumed by OWL-S provide the desired mediation mechanism, and second, to the extent that mediators can be casted into Web services, how they can be represent in OWL-S.

The contributions of this work are twofold. First we show that some components and algorithms that have been developed for OWL-S may provide different types of mediation. For instance, to the extent that gg-mediators can act as selectors among goals, the discovery algorithms that have been generated for OWL-S indeed specify how to do gg-mediation. Similarly, the planning algorithms used for automatic composition provide means for ww-mediation and wg-mediation. While, oo-mediation has not been the focus of any research in OWL-S Web services, its scope is more general since it relates to every aspect of reasoning with and exploiting the Semantic Web. Any contribution toward the solution of oo-mediation will be immediately reflected in OWL-S Web service representation.

The second contribution of this paper is the representation of WSMO mediators as OWL-S Web service descriptions. As we discussed, it is still quite unclear how WSMO mediators are reflected in the architecture of Web services. Specifically, there is still an ambiguity of whether WSMO mediators are a component of the client, of the server or they act as middle agents that mediate between the client and the server. Nevertheless, we show that to the extent that mediators can act as Web services they can also be represented in OWL-S.

Finally, we describe how to represent a broker in OWL-S, which can be considered as a stateful mediator. On the opposite of other mediators that concentrate on the translation task that they have to produce, stateful mediators should also manage the interaction protocol between the client and the server and provide a translation that is contextualized and the internal reasoning becomes more complex. Nevertheless, we show that stateful mediators can be represented in OWL-S, as long as contingencies for the dynamic loading of Process Models are provided.

## References

1. M. Balaban. The F-Logic Approach for Description Languages. *Annals of Mathematics and Artificial Intelligence*, 15(1):19–60, 1995.
2. P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-owl: Contextualizing ontologies. In *Second International Semantic Web Conference*, Sanibel Island, Florida, USA, 2003.
3. J. Colgrave, R. Akkiraju, and R. Goodwin. External matching in UDDI. In *2nd International Conference of Web Services (ICWS 2004)*, pages 226–233, San Diego, CA, USA, 2004.

4. M. Dean, G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference, 2004.
5. K. Decker, K. Sycara, and M. Williamson. Matchmaking and brokering. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*. The AAAI Press, 1996.
6. D. Fensel and C. Bussler. The web service modeling framework (WSMF). *Electronic Commerce: Research and Applications*, (1):113–137, 2002.
7. T. Kawamura, J.-A. D. Blasio, T. Hasegawa, M. Paolucci, and K. Sycara. A preliminary report of a public experiment of a semantic service matchmaker combined with a UDDI business registry. In *1st International Conference on Service Oriented Computing (ICSOC 2003)*, Trento, Italy, 2003.
8. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object oriented and frame-based languages. *Journal of the ACM*, 42, 1995.
9. D. McDermott. Estimated-regression planning for interactions with web services. In *AI Planning Systems Conference (AIPS 2002)*, 2002.
10. S. McIlraith, T. C. Son, and H. Zeng. Semantic web service. *IEEE Intelligent Systems*, 16(2):46–53, 2001.
11. M. Paolucci, J. Soudry, N. Srinivasan, and K. Sycara. Untangling the broker paradox in owl-s. In *Proceedings of AAAI 2004 Spring Symposium*, 2004.
12. M. Paolucci, K. Sycara, and T. Kawamura. Delivering semantic web services. In *Proceeding of the Twelfth International Conference on World Wide Web*, Budapest, Hungary, 2003.
13. N. Srinivasan, M. Paolucci, and K. Sycara. Adding OWL-S to UDDI, implementation and throughput. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, CA, USA, 2004.
14. H.-C. Wong and K. Sycara. A Taxonomy of Middle-agents for the Internet. In *ICMAS'2000*, 2000.
15. D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating daml-s web services composition using shop2. In *Proceedings of the Second International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, USA, 2003.