Chapter 1

A BROKER FOR OWL-S WEB SERVICES

Massimo Paolucci, Julien Soudry, Naveen Srinivasan and Katia Sycara
The Robotics Institute
Canegie Mellon University
5000 Forbes ave Pittsburgh, PA USA
paolucci,jsoudry,naveen,katia@cs.cmu.edu

Abstract Brokers are widely used in distributed information systems such as Multi-agent systems and distributed databases. Yet, there has not been a detailed analysis of Brokers' architecture and no general solution has been proposed on how the Brokers' tasks have to be accomplished. In this paper, we provide a detailed analysis of these tasks, and an implementation based on OWL-S. We show that while OWL-S is adequate to provide all the information that is needed by the Broker, the straightforward implementation of the Broker using OWL-S results in a paradoxical situation. We solve this paradox by extending the Process Modeling language of OWL-S. Finally, we propose a solution to a number of issues that arise in the brokered management of the interaction between Web services such as the abstraction from queries to capabilities required to solve that query, and management of the knowledge required by the Broker to control the multi-party interaction.

Keywords: OWL-S, OWL, Brokers, Semantic Web services

Introduction

Brokering is a natural coordination and mediation mechanism that we often encounter in our daily life. The most striking example of Brokers are stock Brokers that mediate between the stock market and its investors. Brokers play a role when there is a need to facilitate the interaction between two or more parties. For example, if two parties want to communicate, but they do not share a common language, Brokers may provide translation services, or if the two parties do not trust each other, a Broker may provide a trusted intermediary (e.g. an es-

crow service for commercial transactions). Furthermore, Brokers may provide anonymization for one (or both) of the parties, by mediating the transaction.

Not surprisingly, Brokers are one of the main discovery and synchronization mechanisms among autonomous agents (Decker et al., 1996; Wong and Sycara, 2000). Examples include the Open Agent Architecture Facilitator (Martin et al., 1999) which mediates between OAA agents that collaborate toward the solution of a problem. Furthermore, Brokers have been widely used in many agents applications such as integration of heterogeneous information sources and Data Bases (Lu and Mylopulos, 2002), e-commerce (Jennings et al., 2000), pervasive computing (Chen et al., 2004) and more recently in coordinating between Web services in the IRS-II framework (Motta et al., 2003). Finally, theoretical studies (Decker et al., 1996; Wong and Sycara, 2000) show analyze trade-offs that result from the use of Brokers. On one side architectures based on Brokers are centralized and bound to have bottlenecks and single points of failure. On the other side, Brokers, can perform a range of coordination activities such as load balance between different agents, and anonymization where the Broker acts as a proxy of an agent effectively hiding the provider or the requester of a given functionality.

Because of their mediation and coordination propertie as well as their wide applicability, Brokers are a natural candidate component for the Web services infrastructure. The SOAP (Mitra, 2003) specification and WS Architecture group (WSA) (Booth et al., 2004) specifications have provisions for intermediaries, but their role is limited to message routing. Furthermore, WSA assumes the existence of policy enforcing guards and auditing guards that act as Brokers that verify that the transactions performed by the agents are consistent with current policies. However, Brokers with rich functionality of discovery and mediation, are not part of the Web services infrastructure.

In the current Web services infrastructure, the only component that is devoted to discovery is UDDI (UDDI, 2000). However, UDDI is a registry that does not perform any mediation between the requester and the discovered service provider. A number of services that satisfy a particular request could be found using UDDI, but then it is up to the requester to decide which web service to use and how to interact with the selected provider. The DAML-S/OWL-S Matchmaker (Paolucci et al., 2002) provides automated semantic matching of service requests to capability advertisements, but, after the matchmaker returns a list of candidate providers, the selection of the most suitable service is done by the requester, which can use the DAML-S Virtual Machine (Paolucci et al., 2003a) to invoke the selected service.

In this paper, we provide an analysis of the requirements of a Broker that performs both discovery and mediation between agents and Web services. We show that such a Broker performs very complex reasoning tasks that include (1) the interpretation of the capability advertisements of service providers; (2) the interpretation of the requesters' queries that must be fulfilled by a service provider; (3) finding the best provider based on the requester's query; (4) invocation of the selected provider on behalf of the requester, interacting with the provider as necessary to fulfill the query, and (5) returning the query results to the requester. The accomplishment of these tasks requires ontologies to describe capabilities of Web services, their interaction patterns and the domain they operate on, and a logic that allows reasoning on those ontologies. Furthermore, we will provide a description of an implementation of a Broker using OWL-S (DAML-S Coalition: et al., 2002), a Web services description language based on OWL (Dean et al., 2004) and the Semantic Web (Berners-Lee et al., 2001).

The implementation of the Broker also highlights some of the challenges of the automatic interaction between Web services. The first overall challenge concerns the selection of a suitable service provider to fulfill the requester's query. The requester's query is a particular instantiation of an input to an invocation of a service (e.g. "what is the five day forecast in Pittsburgh"?), while advertisements of Web services express the capabilities of the Web service; in other words the class of queries that it can answer (e.g. "I provide a service that gives weather information"). Therefore, the Broker cannot match directly the query against its advertisement store, rather the Broker needs first to transform the query into the capabilities requested to answer it, only then the matching can be effected. How this transformation can be automatically generated is a big challenge.

The second major challenge for the Broker is the management of the interaction between the provider and the requester. Ideally, the Broker may try to act as the provider and present to the requester the same interaction protocol of the provider. Therefore the Broker may forward to the requester the information coming from the provider, and conversly forward to the provider the information that it receives from the requester. The problem is that in a Brokered system neither the requester that addresses a Broker nor the Broker itself know a priori which is a suitable provider. Hence they do not know what information the Broker needs to proceed with the transaction, nor what information the Broker will report to the requiester. It is impossible to specify how a requester can formulate its query to the Broker nor how the Broker can medi-

ate interactions between a requester and a provider whose interaction protocol is known only at a later stage.

In this paper, we present an approach to the development and implementation of a Semantic Broker, namely a Broker that performs semantic discovery and mediation among Web service requesters and providers . The Broker uses OWL-S to perform its functions. Our approach gives solutions to the three challenges presented above. The solution necessitates an extension to OWL-S to address the problem of the initial lack of knowledge of the provider's Process Model on the part of the requester and of the Broker.

The rest of the paper is organized as follows. In Section 1, we present an overview of OWL-S; in Section 2, we provide a detailed analysis of the Broker, exploring its interaction protocol and the reasoning tasks that it has to accomplish. In Section 3, we show how OWL-S provides the information that the Broker needs to perform its tasks. In Section 4, we explore the problems that emerge from describing the Broker with OWL-S, and we describe the *exec* extension of OWL-S. In Section 5, we describe the basic features of our implementation and provide details on how we address the reasoning problems of the Broker. At last, in Section **??**, we conclude.

1. OWL-S

OWL-S is a Semantic Web services description language that enriches Web services descriptions with semantic information from OWL (Dean et al., 2004) ontologies and the Semantic Web (Berners-Lee et al., 2001). OWL-S is organized in three modules: a Profile that describes capabilities of Web services as well as additional features that help to describe the service. A Process Model that provides a description of the activity of the Web service provider from which the Web service requester can derive information about the service invocation. A Grounding that describes of how abstract information exchanges described in the Process Model is mapped onto actual messages that the provider and the requester exchange.

The role of the OWL-S Profile is to support the requester in (a) discovering suitable providers, and (b) selecting among them. The OWL-S Profile achieves the first goal, i.e. provider discovery, by prescribing ways for representing Web service capabilities. The Service Profile fulfills the second goal, i.e. service selection, by providing for the representation of additional information about the service, such as information describing provenance and quality or cost specifications of the Web service.

A Web service capability is the description of the service functionality, i.e. what the service does. For example, the capability of a bookseller, such as Barnes and Noble, is to *sell books*. The capability of a Web service can be viewed in two ways: first as a service category within an ontology of services (e.g. *selling books is-a selling products*) or as a transformation of a set of inputs to a set of outputs (e.g. selling books transforms the inputs "book title" and "book author" to the output "book invoice"). OWL-S Profile describes capabilities of Web services by the transformation that they produce. Besides transforming inputs into outputs at an information level, invoking a Web service can produce effects in the real world and need preconditions to be satisfied. For example, invoking the book selling service and buying a book produces the effect in the real world that the requester's credit card gets charged; a precondition for the invocation of the book selling service is that the requester has a valid credit card.

In addition to capabilities, an OWL-S Profile provides provenance information that describes the entity (person or company) that deployed the service; and non-functional parameters that describe features of the services such as quality rating for the service. These pieces of information help a requester discriminate among different services with similar capabilities. For example, a requester may prefer a bookseller that has a Dunn and Bradstreet quality rating.

In order to make its capabilities known to service requesters, a service provider advertises its capabilities with infrastructure registries, or more precisely middle agents (Wong and Sycara, 2000), that record which agents are present in the system. UDDI registries are an example of a middle agent, with the limitation that it can make limited use of the information provided by the OWL-S Profile. The OWL-S/UDDI Matchmaker (Paolucci et al., 2002; Paolucci et al., 2003b) is another example, which combines UDDI and OWL-S. Finally, the Broker defined in this paper is another example of a middle agent that performs both discovery and mediation.

The second module of OWL-S is the Process Model. The Process Model has two aims: the first one is to show how the provider achieves its goals, and the second to provide the requester-provider interaction protocol. The first goal is achieved by allowing the provider to make public a description of its computation, to the extent that the provider feels comfortable to do so. The requester-provider interaction protocol is derived by the processes that the provider performs by locating when the provider needs information, and what type of information, and where it sends information, and what type of information.

A Process Model defines a set of concurrent threads of execution. Each thread is an ordered collection of processes. OWL-S distinguishes between two types of processes: composite processes and atomic processes. Atomic processes correspond to operations that the provider can perform directly. Composite processes are used to describe collections of processes (either atomic, or composite) organized on the basis of some control flow structure. For example, a sequence of processes is defined as a composite process whose processes are executed one after the other. Other control constructs supported by OWL-S are conditional expressions, non-deterministic choices between alternative control flows, and spawning of new concurrent threads. Finally, OWL-S includes looping constructs like while and repeat-until.

The last module of OWL-S is the Grounding that describes how atomic processes which provide abstract descriptions of the information exchanges with the requesters, are transformed into concrete messages or remote procedure calls over the net. Specifically, the OWL-S Grounding is defined as a one to one mapping from atomic processes to WSDL (Christensen et al., 2001) input and output message specifications. From WSDL it inherits the definition of abstract message and binding, while the information that is used to compose the messages is extracted through the execution of the process model during service invocation.

Therefore, the Web services philosophy of interaction between a service requester and a service provider is that a requester would need to know the information that a service provider requires at different stages of the interaction. For example, in industrial standards, the requester-provider interaction is governed by knowledge of the provider's Web services Description (WSD) given in WSDL, and in Semantic Web services, the requester-provider interaction presupposes knowledge on the part of the requester of the Process Model (plus WSD) of the provider.

2. Overview of the Broker

Brokers have been widely applied in many different applications and domains, therefore, not surprisingly, there are many different definitions of what a Broker is. We adopt the definition of the Broker protocol based on (Decker et al., 1996), and graphically summarized in Figure 1.1. Any transaction involving a Broker requires three parties. The first party is a requester that initiates the transaction by requesting information or a service to the Broker. The second party is a provider which is selected among a pool of provider as the best suited to resolve the problem of the requester. The last party is the Broker itself.

The protocol in Figure 1.1 can be divided in two parts: the advertisement protocol, and the mediation protocol. In the advertisement protocol, the Broker first collects the advertisements of Web services that are available to provide their services. These advertisements, shown in Figure 1.1 by straight thin lines, are used by the Broker to select the best provider during the interaction with the requester. The mediation protocol, shown in Figure 1.1 using thick curve lines, requires (1) the requester to query the Broker and wait for a reply while the Broker uses its discovery capabilities to locate a provider that can answer the query. Once the provider is discovered, (2) the Broker reformulates the query for that provider, and finally queries it. Upon receiving the query, (3) the provider computes and send the reply to the Broker and finally (4) the Broker replies to the requester.

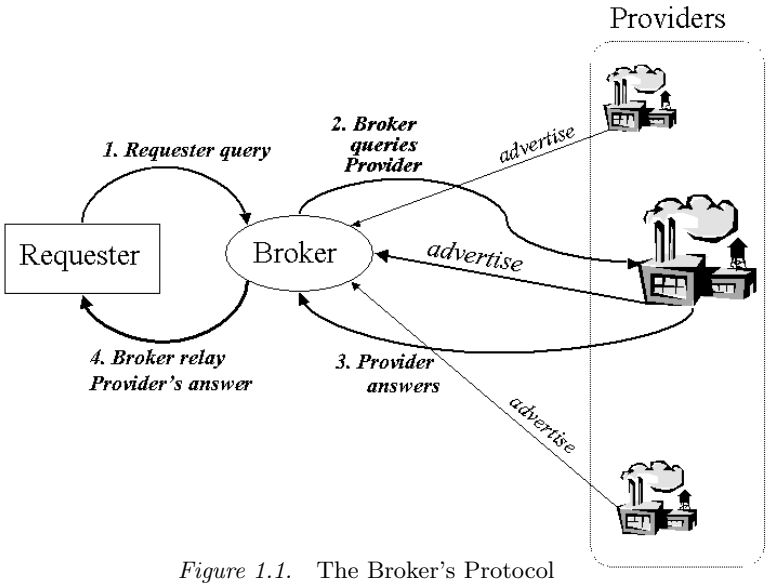

Figure 1.1. The Broker's Protocol

In general, the execution of the protocol may be repeated multiple times. For example, the requester may have asked the Broker to book a flight from Pittsburgh to New York. Since there are multiple flights between the two cities, the provider may ask the Broker, and in turn the requester, to select the preferred flight. These interactions are resolved with multiple loops through the protocol. For example, the Broker translates the list of flights retrieved by the provider for the requester, through steps (3) and (4) of the protocol, and then translates the message with

the selected flight from the requester to the provider, via steps (1) and (2) of the protocol. The only exception is that step (1) does not require any discovery since the provider is already known.

In some cases, the Broker may be able to answer the request of information from the provider directly; therefore, it does not have to involve the requester in the interaction. For example, the requester's query may request a seat on the cheapest flight from Pittsburgh to New York. When the provider reports all flights between the two cities, the Broker selects the cheapest one and responds directly to the provider without asking anything to the requester. Following the protocol in Figure 1.1, these interactions are the result of the inner loop produced by the steps (3) and (2). The provider sends message (3) and the Broker responds directly with (2) without contacting the requester.

The protocol described above shows that the Broker needs to perform a number of complex reasoning tasks for both the discovery and mediation part of its interaction. The discovery task requires the Broker to use the query to describe the capabilities of the desired providers that can answer that query, and then match those capabilities with the capabilities advertised by the providers. During the mediation process, the Broker needs to interpret the messages that it receives from the requester and the provider to decide how to translate them, and whether it has enough information to answer directly the provider without further interaction with the requester. In the next two sections, we will analyze these reasoning tasks in more detail.

2.1 Discovery of Providers

The task of discovery is to select the provider that is best suited to reply to the query of the requester. Following the protocol, providers advertise their capabilities using a formal specification of the set of capabilities they posses, i.e. the set of functions that they compute. These capability specifications implicitly specify the type of queries that the provider can answer.

The discovery process requires two different reasoning tasks. The first one is to abstract from the query of the requester to the capabilities that are required to answer that query. The second process is to compare the capabilities required to answer the query with the capabilities of the providers to find the best provider for the particular problem.

The first problem, the abstraction from the query to capabilities, is a particularly difficult one. Capabilities specify what a Web service or an agent does, or, in the case of information providing Web services, what set of queries it can answer. For example, the capability of a Web

service may be to provide weather forecasting, or sell books, or register the car with the local department of transportation. Queries instead are requests for a very specific piece of information. For example, a query to a weather forecasting agent may be to provide the weather in Pittsburgh, while a query to a book-selling agent may be to buy a particular book. Because of their difference, queries and capabilities are also expressed in very different formats. The task of the Broker therefore is to abstract from the particular query, to its semantics, i.e. what is really asked. Finally, the Broker must identify and describe in a formal way the capabilities that are needed to answer that query.

The second task of the discovery process is to match the capabilities required to answer the query with the advertisements of all the known providers. Since it is unlikely that the Broker will find a provider whose advertisement is exactly equivalent to the request, the matching process can be very complicated, because the Broker has to decide to what extent the provider can solve the problems of the requester.

2.2 Management of Mediation

The second reasoning task that the Broker has to accomplish is to transform the query of the requester into the query to send to the provider. This process of mediation has two aspects. The first one is the efficient use of the information provided by the requester to the Broker; the second one is the mapping from the messages of the requester to messages to the provider and vice versa.

Since the requester does not *a priori* know which is the relevant provider, the (initial) query it sends to the Broker and the query input that the (selected) provider may need in order to provide the service may not correspond exactly. The requester may have appended to the query information that is of no relevance to the provider, while the provider may expect information that the requester never provided to the Broker. In the example above, we considered the example of a requester that asks to book the cheapest flight from Pittsburgh to New York. However, besides the trip origin and destination, the selected provider may expect date and time of departure. In the example, the requester never provided the departure time, and the provider has no use for the "cheapest" qualifier. It is the task of the Broker to reconcile the difference between the information that the requester provided and the information that the provider expects, by (1) recognizing that the departure time was not provided, and therefore it should be asked for, and (2) finding a way to select the cheapest flight among the ones that the provider can find.

The other type of inference on the message passing that the Broker has to perform is the mapping between ontologies and terms used by the two parties. For example, the requester may have asked for information on IBM whereas the provider expects inputs in terms of International Business Machine Corporation. A more complicated mismatch may be at the level of concepts and their relations in the ontologies used for inputs and outputs of the provider *vis a vis* the ontological information used by the requester. For example, the requester may have asked for the weather in Pittsburgh, but instead the provider can report only the weather at major airports. The task of the Broker in this case is to infer which is the most appropriate airport, and use it in the query to the provider. Therefore, instead of asking for the weather in Pittsburgh, the Broker asks the provider for the weather at PIT, where PIT is the code of the Pittsburgh International Airport.

Finally, the Broker has the non-trivial task of translating between the different syntactic forms of the queries and replies. The examples that we discussed above assume semantic mismatches between the different messages that the Broker has to interpret and send. These messages have to be compiled in an appropriate syntactic form, and despite their semantic similarity, the messages could be realized in very different ways. The task of the Broker is to resolve syntactic differences, and to formulate messages that all the parties can understand.

In conclusion, the Broker performs a number of complex reasoning tasks that range from discovery to the interpretation, translation and compilation of messages. To accomplish these tasks, the Broker needs the support of a formal framework that allows complex reasoning about agents, what they do and how to interact with them. Furthermore, the Broker needs a way to translate the semantics of the information that it wants to communicate, into the syntactic form that the provider or the requester expects.

3. OWL-S Support of Broker's Reasoning

The OWL-S language and ontology provides constructs to support the Broker in both discovery and mediation between Web services. The OWL-S Profile supports the discovery process by providing a representation of capabilities of Web services and agents. The OWL-S Process Model and service Grounding provide support for the interaction between the Broker and the requester and provider of the service.

The discovery process requires a representation of capabilities of provider services and a representation of the capabilities that are required to answer the query. These capabilities are represented in OWL-S by the

service Profile. In addition to the representation of capabilities, the analysis above showed that the Broker needs an abstraction process from the query to the capabilities needed to answer it, and a matching mechanism from the capabilities required for the query to the capabilities of the providers to select the best service provider to answer the query.

A number of capability matching algorithms for OWL-S based Web services have been proposed (see (Benatallah et al., 2003; Noia et al., 2003; Li and Horrocks, 2003; Paolucci et al., 2002)) which exploit OWL ontologies and the related logics to infer which advertisements satisfy a request for capabilities. These algorithms can be used to solve the second problem: the matching from the capabilities required for the query to the capabilities of the providers.

The first problem, the abstraction from the query to the capabilities, is more complicated. First of all, there is no explicit support in OWL-S for queries, nevertheless, it is easy to use the OWL Query Language (OWL QL) (Committee, 2002; Fikes et al., 2003) which relies on the same logics required by OWL-S. The transformation is still an open problem, which, to our knowledge, has not been addressed in the literature. In the next section, we will propose an abstraction algorithm to transform queries into capabilities.

After selecting a provider, the Broker has access to the provider's Process Model from which it can derive the provider's interaction protocol by extracting what information the provider will need, in what order, and what information it will return. For the rest of the interaction the Broker acts as the provider's direct requester. However, this relation is not straightforward. Since the Broker acts on behalf of the requester, it must somehow transform the requester's initial query (and all subsequent messages) into a query (or a sequence of queries) to the provider. This transformation is necessary since the requester does not have direct access to the Process Model of the provider, but interacts with the provider only through the Broker. We show how this transformation can be done in Section 5.1.

Furthermore, since the requester initiated its query without having access to the provider's Process Model (since the provider was not known at the time of the requester's query initiation), the Broker needs to infer what additional information it needs from the requester. Once it has done that, it then uses this knowledge to construct a new Process Model. This new Process Model is presented by the Broker to the requester, not as the Process Model of the selected provider but as the Process Model of the Broker. This makes sense since the requester interacts only with the Broker. The new Process Model indicates to the requester what information is needed and in what order. How the Broker infers the

additional information it needs from the provider and how it constructs the new Process Model is presented in Section 5.2.

The Service Grounding provides a mapping from the semantic form of the messages exchanged as defined in the Process Model, to the syntactic form as defined in the WSDL input and output specifications. The Grounding provides to the Broker the mapping from the abstract semantic representation of the messages to the syntactic form that these messages adopt when they become concrete information exchanges. The Broker uses this mapping to interpret the messages that it receives and compile the messages that it sends to the requester or to the provider.

4. A Process Model for the Broker

Every interaction between agents using OWL-S must be effected in accordance with the provider's Process Model. Interactions with a Broker are no exception. Since, from the point of view of the requester, the Broker is the provider, it expects the Broker to publish a Process Model that is to be used during the interaction. In this section, we show that the Broker's Process Model pushes the boundaries of the current specification of OWL-S.

4.1 The Broker's Paradox

A requester interacts with the Broker using the Broker's Process Model. The Broker's Process Model should specify how the requester can submit its query, but it should also allow the requester to provide any additional information that the Broker needs to interact with the provider. Since, to the requester, the Broker is a (representative of) the provider, the Process Model of the Broker should contain the crucial elements of the Process Model of the provider. However, since the Broker is unaware of the provider until it has discovered and selected the provider based on a requester's query, the Broker is faced with a challenge: it must publish a Process Model that depends on the provider's Process Model, but the provider is not known until the requester reveals its query. On the other hand, the requester cannot query (interact with) the Broker until the Broker publishes its Process Model. The result is a paradoxical situation in which the Broker cannot reveal its Process Model until it receives the query of the requester, but cannot receive the query from the requester until it publishes its Process Model.

Essentially, the Broker's paradox is due to the fact that the discovery of the provider depends on the requester's query, while the rest of the interaction between the requester and the Broker depends on the provider selected. Ultimately, the Broker paradox results from an inflexibility of

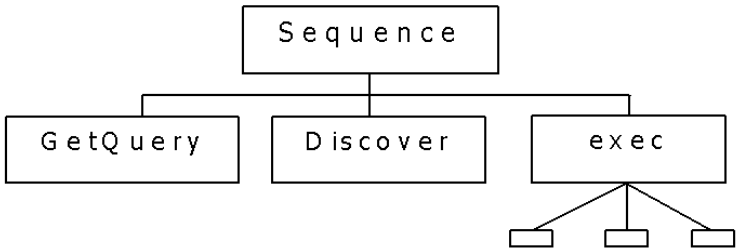

Figure 1.2. Broker's Process Model

the OWL-S specification of service invocation, which requires the specification of the Process Model before the interaction, and does not allow any means to modify the Process Model during the interaction.

4.2 Extending OWL-S Process Model

The solution of the Broker's Paradox that we propose requires an extension of the specification of the OWL-S Process Model to allow the flexibility to dynamically modify an agent's Process Model during the interaction. As a result, the Broker can provide an initial, provider-neutral, Process Model to the requester, and then modify it consistently with the requirements of the Process Model of the provider. The changes are then adopted by the requester in its interaction with the Broker.

To implement this solution, we propose to extend the OWL-S Model Processing language by adding a new statement, that we call *exec*. The *exec* statement takes as input a Process Model and executes it. Therefore, the Broker can compile a new Process Model, return it as an output of one of its processes, and then use the *exec* statement to turn the new Process Model into executable code that specifies the Broker's new interaction protocol.

The provider-neutral Process Model of the Broker is shown in Figure 1.2. It shows that the Broker performs a sequence of three operations, where the first operation is GetQuery in which the Broker gets the query from the requester. The second operation is Discover in which the Broker uses its discovery capabilities to find the best provider. The result of the Discover process is a new Process Model that depends on the provider found. Finally, the Broker performs the *exec* operation which passes

control to a new Process Model. This change of control is shown in the figure by the three small rectangles that display processes that will be run as a consequence of the *exec*.

The use of the *exec* solves the Broker's Paradox by removing the inflexibility of the OWL-S Process Model. The *exec* operation allows the separation of service discovery from service invocation and interaction. First the discovery is completed, then the interaction, which depends on the discovered provider, is initiated through the *exec*.

One important question that is left unanswered is whether there is a clever way to use OWL and OWL-S that does not require the extension of the language that we propose. Unfortunately, such an extension does not exist, because neither OWL nor OWL-S provides a way to transform a term into a predicate of the logic, which is the essential step that is performed by the *exec*.

5. Implementation

We have implemented a prototype of a Broker that makes use of OWL-S with the *exec* extension described above to mediate between agents and Web services. We based our implementation of the Broker on the OWL-S Virtual Machine (OWL-S VM) (Paolucci et al., 2003a), which is a generic OWL-S processor that allows Web services and agents to interact on the basis of the OWL-S description of the Web service and OWL ontologies. In the implementation of the Broker, we extended the OWL-S VM to include the semantics of the *exec*. Furthermore, we developed the reasoning that allows the Broker to perform discovery and to mediate the interaction between the provider and the requester.

In this section, we analyze how we implemented the different aspects of the Broker. We will first discuss the implementation of the discovery process, and then we will analyze the modification of the interaction protocol that allows the Broker to mediate between the provider and the requester. Finally, we will discuss the use of the OWL-S VM in the implementation that allows us to actually mediate between the two parties.

5.1 Supporting Discovery

The Broker expects from the requester a query in OWL-QL format (Fikes et al., 2003), where the predicate corresponds to a property in the ontology, the terms in the query are either variables, or instances that are consistent with the semantic type requirements of the predicate.

The discovery process takes as input the query of the requester and generates as output the advertisement of a provider (if any is known

to the Broker) that can answer the query. The discovery process has three steps. First the Broker abstracts from the query to the capabilities that are required to answer that query, thus constructing a service request. Second, the Broker finds appropriate providers by matching the capabilities requested with the capability advertisements by the providers. Third, the Broker select the most appropriate provider among the providers that match the capabilities requested. The matching of the service request against the advertised capabilities was implemented using the OWL-S matching engine reported in (Paolucci et al., 2002) and (Paolucci et al., 2003b).

The automatic abstraction from the requester's query to a service request is, to our knowledge, an unexplored problem. The abstraction process must respect the constraints of the OWL-S discovery process, namely generation of an OWL-S service profile where the service inputs and outputs reflect the semantic content of the query. The abstraction procedure that we implemented distinguishes between variables, and the terms that are instantiated in the query. Since the result of the query should be an instantiation of the variables, ideally, the selected provider agent would take as inputs the instantiated terms and return as output an instantiation for the variables.

---

1 set V = set of variables in the query

2 set T= set of instantiated terms in the query

3 set I= abstraction of each term in T to its immediate class

4 use predicate definition in the ontology to abstract variables in V to their class

5 set O= abstraction of each variable in V to its class

6 generate a service request with input I and outputs O

Figure 1.3. The abstraction algorithm

The instantiation algorithm follows the 6 steps listed in Figure 1.3. In the steps 1 and 2, terms from the query are extracted distinguishing between variables and instantiated terms. In step 3, the set of inputs of the service request is derived by abstracting the instantiated terms to their immediate class. For instance, if one term were Pittsburgh, it would be abstracted to City (assuming the presence of a location ontology). Step 4 is needed to handle variables. In OWL-QL variables

are of class Variable, but there is no constraint on the type that they have to assume. We use the definition of the predicate in the ontology to constrain the type of the values of the variable to the most restrictive class of values that they can be assigned to. In step 5, we use the abstraction in step 4 to generate the set of outputs O. Finally, in step 6, the service request is generated by specifying the inputs and the outputs.

5.2 Supporting Mediation

After the Broker has selected a provider, it must mediate between the provider and the requester. The mediation process depends on the Process Model of the provider which specifies what information is required and when. In theory, the Broker may just present to the requester the Process Model of the provider and limit mediation to message forwarding. But this solution is unacceptable, since it ignores the information that the requester already provided to the Broker. For example, the requester may ask the Broker to book a trip to Pittsburgh. The Broker may find a Travel Web service that asks for departure and arrival location. The task of the Broker is to recognize that arrival location information has already been specified so the Broker needs to ask the requester for the departure location only.

```
1 KB= knowledge from query
2 I= input of process
3 for i ∈ I
4     select k from KB with the same semantic type of I
5     if k exists
6         remove i from I
```

Figure 1.4. Algorithm for pruning redundant information

The algorithm for pruning redundant information is shown in Figure 1.4. It hinges on removing from processes inputs that should be provided by the requester, but that can be filled by the information the Broker already has. First, the Broker records the information provided by the query in a KB (step 1), and the inputs of the process (step 2). Next for each input i, the Broker looks in the KB for information that it can use in place of i. If any is found, i is removed from the inputs of a process.

For example, suppose that the requester's query asking for the booking of a trip used `ArrivalLocation=Pittsburgh` to indicate the destination of the travel. Furthermore, suppose that the Process Model requires two inputs of type `DepartureLocation` and `ArrivalLocation`. Our algorithm would generate a Process Model in which the Broker asks only for first input (the departure city), while the second input `ArrivalLocation` will be pruned by the algorithm because it has the same semantic type of the information provided in the query.

5.3 Managing Message Passing

The last aspect of the Broker is to instantiate a message passing mechanism that allows consistent data transfer between the provider and the requester. The architecture of the Broker is shown in Figure 1.5. To interact with the provider and the requester the Broker instantiates two ports: a server port for interaction with the requester (since the Broker acts as a provider vis a vis the requester) and a client port for interaction with the provider (since the Broker acts as a client vis a vis the provider). The functionalities of the server port are described using OWL-S. Specifically, the Broker exposes to the requester its Process Model, Grounding and WSDL specification. The client (requester) uses these descriptions to instantiate an OWL-S Virtual Machine to interact with the Broker. Since the provider-neutral Process Model exposed by the Broker makes use of the *exec* extension, the OWL-S Virtual Machine used by the requester also implements the axioms that implement the execution semantics of *exec*. The client port is also implemented as an OWL-S Virtual Machine that uses the Process Model, Grounding and WSDL description of the provider to interact with it.

The reasoning of the Broker happens in the `Query Processor` (see Figure 1.5) that is responsible for the translation of the messages between the two parties and for the implementation of the algorithms in Figures 1.3 and **??**. Specifically, the `Query Processor` stores information received from the query in a `Knowledge Base` that is instantiated with the information provided by the requester. Furthermore, the `Query Processor` interacts with the `Discovery Engine`, which provides the storage and matching of capabilities, when it receives a capability advertisement and when it needs to find a provider that can answer the query of the requester.

6. Conclusion

Despite the wide use of Brokers in different aspects of distributed systems, and despite the many uses Brokers can have in the discovery and

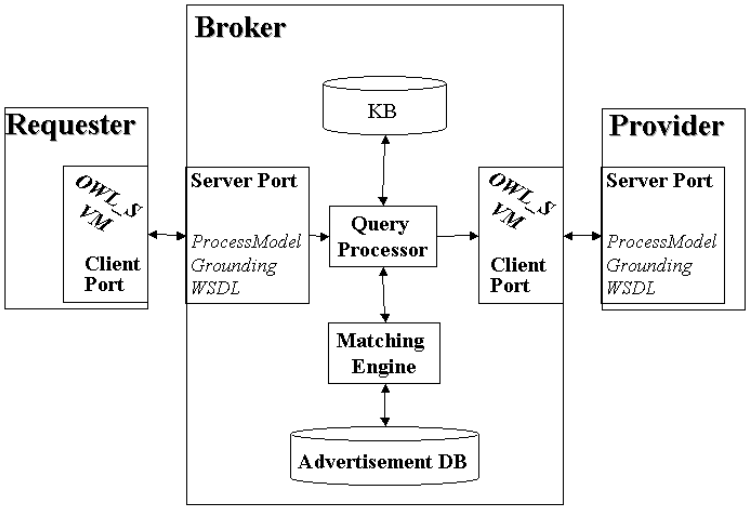

Figure 1.5. Broker Architecture

mediation of Semantic Web services, no detailed analysis of what tasks a Broker should perform to support the interaction has been proposed. One contribution of this paper is to provide such as analysis. In the course of this analysis, a few challenges were uncovered, and solutions for these challenges were presented.

The first challenges is the "Broker's paradox", namely that the Broker cannot publish a Process Model that is based on a yet unknown provider before it receives a request query, and the requester cannot send a query until it knows the Broker's process Model. This paradox arises from the OWL-S Web service interaction specification that requires a declarative specification of a process model to guide the requester and provider interaction. To address the Broker paradox, we extended the OWL-S Process Modeling language with an *exec* operation that allows the dynamic modification of the Broker's Process Model during its execution to include Process Models of dynamically discovered new parties.

A second set of challenges derives from the management of the mediation between the provider and the requester. To address these challenges, we developed a method for abstracting from a service query to a service request. We proposed an algorithm to address this issue. Furthermore, we used the knowledge provided by the requester during the interaction with the provider.

Crucially, the issues emerging with the mediation between the provider and the requester are not unique to Web services Brokering, rather they

comes up in web services composition as well. In the context of Web service composition, a planner may issue a goal that it wants to subcontract. The task of the Web service is first to abstract from the specific goal to a capability description of a provider that can solve the goal, then use its current knowledge, and the goal, to interact with the provider. In current research, we are looking to integrate our work in the context of Brokering to automated composition.

Acknowledgments

We would like to thank Khalid El-Arini for his contribution to early development of this work. This work has been 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. .

References

Benatallah, B., Hacid, M.-S., Rey, C., and Toumani, F. (2003). Request rewriting-based web service discovery. In *Proceeding of the Second International Semantic Web Conference*, Sanibel Island, Fl, USA.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004). Web services architecture.

Chen, H., Finin, T., and Joshi, A. (2004). Semantic web in the context broker architecture. In *Proceedings of the IEEE Conference on Pervasive Computing and Communications (PerCom)*, Orlando, Florida, USA.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

Committee, D. J. (2002). DAML query language (DQL) abstract specification.

DAML-S Coalition:, Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., Martin, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., and Sycara, K. (2002). DAML-S: Web Service Description for the Semantic Web. In *First International Semantic Web Conference*.

Dean, M., Schreiber, G., Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2004). Owl web ontology language reference.

Decker, K., Sycara, K., and Williamson, M. (1996). Matchmaking and brokering. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*. The AAAI Press.

Fikes, R., Hayes, P., and Horrocks, I. (2003). OWL-QL - a language for deductive query answering on the semantic web. Technical Report KSL-03-14, Technical Report Knowledge Systems Laboratory, Stanford University.

Jennings, N. R., Faratin, P., Norman, T. J., O'Brien, P., and Odgers, B. (2000). Autonomous agents for business process management. *International Journal of Applied Artificial Intelligence*, 14(2):145–189.

Li, L. and Horrocks, I. (2003). E-commerce: A software framework for matchmaking based on semantic web technology. In *Proceeding of the Twelfth International Conference on World Wide Web*, Budapest, Hungary.

Lu, J. and Mylopulos, J. (2002). extensible information broker. *International Journal on Artificial Intelligence Tools*, 11(1):95–115.

Martin, D., Cheyer, A., and Moran, D. (1999). The Open Agent Architecture: A Framework for Building Distributed Software Systems. *Applied Artificial Intelligence*, 13(1-2):92–128.

Mitra, N. (2003). Soap version 1.2 part0: Primer.

Motta, E., Domingue, J., Cabral, L., and Gaspari, M. (2003). Irs-ii: A framework and infrastructure for semantic web services. In *Second International Semantic Web Conference*, Sanibel Island, Florida, USA.

Noia, T. D., Sciascio, E. D., Donini, F. M., and Mongiello, M. (2003). A system for principled matchmaking in an electronic marketplace. In *Proceeding of the Twelfth International Conference on World Wide Web*, Budapest, Hungary.

Paolucci, M., Ankolekar, A., Srinivasan, M., and Sycara, K. (2003a). The daml-s virtual machine. In *Second International Semantic Web Conference*, Sanibel Island, Florida, USA.

Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K. (2002). Semantic matching of web services capabilities. In *First International Semantic Web Conference*.

Paolucci, M., Sycara, K., and Kawamura, T. (2003b). Delivering semantic web services. In *Proceeding of the Twelfth International Conference on World Wide Web*, Budapest, Hungary.

UDDI (2000). The UDDI Technical White Paper. Technical report, OASIS.

Wong, H.-C. and Sycara, K. (2000). A Taxonomy of Middle-agents for the Internet. In *ICMAS'2000*.