A Conceptual Model and Technical Architecture for Semantic Web Services

Chris Preist
Digital Media Systems Laboratory
HP Laboratories Bristol
HPL-2005-220
December 21, 2005*

semantic web, web services, architecture, conceptual model

In this paper, we present an architecture for the deployment of Semantic Web Services. We present a conceptual model for semantic web services, linked with concepts in existing work and illustrated with four scenarios. The conceptual model includes a lifecycle for service interaction, mediation and composition of services. We then present an embodiment of the model in an architecture. We present both the 'macro-architecture' of how the different entities interoperate, and the 'micro-architecture' defining the internal functionality of the different entities. We then discuss implementation of this architecture.

* Internal Accession Date Only
Approved for External Publication

A Conceptual Model and Technical Architecture for Semantic Web Services

Chris Preist

HP Laboratories, Bristol, UK
chris.preist@hp.com

Abstract. In this paper, we present an architecture for the deployment of Semantic Web Services. We present a conceptual model for semantic web services, linked with concepts in existing work and illustrated with four scenarios. The conceptual model includes a lifecycle for service interaction, mediation and composition of services. We then present an embodiment of the model in an architecture. We present both the 'macro-architecture' of how the different entities interoperate, and the 'micro-architecture' defining the internal functionality of the different entities. We then discuss implementation of this architecture.

1 Introduction

The Semantic Web Services vision ([1],[2]) is to combine semantic web and web service technologies, and through this to enable automatic and dynamic interaction between software systems. In this paper, we present an architecture for the deployment of Semantic Web Services. Firstly, we introduce a set of four diverse motivating scenarios. We then present a conceptual model for semantic web services, linking the concepts in with existing work and using elements from the four scenarios to illustrate the concepts. The conceptual model includes a lifecycle for service interaction, mediation and composition of services. Based on this general conceptual model, we then present a specific embodiment of it in an architecture. We present both the 'macro-architecture' of how the different entities interoperate, and the 'micro-architecture' defining the internal functionality of the different entities. We then discuss implementation of this architecture.

The real value that Semantic Web Services can enable is best illustrated through some example scenarios. In this section, we introduce four scenarios. Initially, we present a 'storyboard' for each. In subsequent sections, these will be used to illustrate different features of Semantic Web Services.

1.1 Scenario A - Overdraft Notification Service [4]

A bank provides an 'overdraft notification service' to warn customers when they are at risk of going overdrawn. Software at the bank monitors the behaviour of a customer's account, and keeps track of when regular payments are made into or out of it. Based on the expected future transactions in an account, and the current balance, banking software can predict if the customer is likely to go overdrawn. In this case, the customer is warned via an email, txt or voice message. To send the warning message, the bank's

software component uses some message-sending service. It does not have a pre-selected provider of this service, but instead automatically makes a decision at the time a message must be sent. To do this, it looks in a directory of available service providers and the message services they offer, and selects one based on factors such as cost, reliability and the preferences of the customer receiving the message. It then sends the message to the provider of that service, which in turn sends a txt or voice message to the customer.

1.2 Scenario B - Intelligent Procurement

A large manufacturing company makes regular purchases of supplies from a variety of on-line companies. Supplies essential for manufacturing, such as components, are purchased through a fixed supply chain from providers who have been carefully vetted to meet the companies requirements. However, less business-critical supplies, such as stationary, can be purchased from any reputable supplier. This provides opportunity for shopping around to get the best deal. A software agent acting on behalf of the company is given a list of stationary equipment needed over the next month. It looks in a directory of suppliers the company considers acceptable for those which provide stationary. The suppliers provide purchasing web sites which use a 'shopping trolley' model similar to Amazon's - a customer browses a catalog, places items it wants onto a list, and goes to a checkout to get a quote for the total package, including postage. They provide a Web Service front end to these portals. The software agent visits several such sites simultaneously. It interacts with them, discovering if they have the specific items in stock, and builds up a 'shopping trolley' of purchases. On reaching 'checkout', it receives a quote for each as to the total cost of the bundle, including volume discounts. Based on these quotes, it selects the cheapest and completes the transaction with that supplier, cancelling the other requests. The supplier then ships the order.

1.3 Scenario C - Provision of a Logistics Supply Chain [5]

A company requires the transport of a crate from Bristol to Moscow. It already has long-term contracts in place for land transportation of crates from Bristol to Portsmouth, and from St Petersburg to Moscow. However, its usual supplier of shipping services is unavailable and it needs to rapidly find a replacement freight forwarder. A software agent acting on behalf of the company has detailed information about the transportation task which must be carried out. It contacts a discovery agent which has access to descriptions of services various organisations are able to provide, and asks for providers able to ship between Portsmouth and St Petersburg. The discovery agent responds with a list of possible freight forwarders likely to be able to meet these requirements. The software agent then selects one or more of the possible freight forwarders, and sends a more detailed description of the task it requires to be performed, including the date the shipment will arrive at Portsmouth, and the date it must reach St Petersburg. The freight forwarders respond with lists of services they can offer which meet these requirements. For example, one forwarder may say that it has a ship leaving Portsmouth on the required day which will arrive in St Petersburg the day before the deadline. It will also give the cost of placing a crate on that ship. The requesting agent then selects one of the proposed services (possibly by interacting with a user to make the final decision) and informs

the provider of the decision. Effectively, the two parties enter into an agreement at this point.

As the shipment takes place, it is coordinated by an exchange of messages between the two parties. The messages use an industry standard, RosettaNet, which describes the format and order of the messages. The exchange starts when the crate is about to arrive in Portsmouth, with a RosettaNet Advanced Shipment Notification being sent by the requestor to the freight forwarder, and ends with the sending of a Proof of Delivery and Invoice by the freight forwarder when the crate arrives in St Petersburg.

1.4 Scenario D - Free Stock Quote Web Service

A small-time investor has a software package to keep track of her share portfolio. It is able to receive updated share prices via Web Services technology. When she connects to the internet, the software searches for services able to provide share prices. It locates two possible services, and asks the user to select one. One service gives prices delayed by 1 minute, and requires a subscription of 10 euros/month to use. The other gives prices delayed by 30 minutes, and is free. The investor chooses the latter, because she does not engage in real-time trading, and the software package then updates her portfolio information whenever she is online.

2 Key Concepts in Semantic Web Services

We now introduce some key concepts in Semantic Web Services, and show how these inter-relate. The work presented in this section follows the Semantic Web-enabled Web Services conceptual architecture [6].

2.1 Notion of Service

Firstly, let us define the key concept of *service*. Intuitively, one party provides a service to another when the first party does something for the benefit of the second. A service may be freely given, but is often done for payment. In Scenario A above, the bank provides the overdraft warning service to its customer; in Scenario B, the stationary supplier provides the service of sale and shipment of stationary to the manufacturing company; in Scenario C, the freight forwarder provides the service of transferring a crate from one port to another. Formally, we can summarise this by saying that a service is the performance of some actions by one party to provide some value to another party. Note that it makes sense to talk about a service in a certain domain (In Scenario C, the domain would be transport and logistics). We refer to this as the *domain of value* of the service. We call the party which performs the service the *service provider* and the party which receives the benefit of the service the *service requestor*. Services can be considered at different levels of abstraction. A *concrete service* is a specific performance of actions at a given time by one party for another. (In Scenario C, a concrete service would be the shipping of crate 246 on the ship departing Portsmouth at 9.25 on 11/12/04 and arriving in St Petersburg at 22.00 on 14/12/04.) However, often when we are reasoning about services, we do not want to be so specific. When discussing a

hypothetical service to be performed in the future, we cannot be specific about all of its details. Hence, we use an abstraction. An *abstract service* corresponds to some set or class of concrete services, and allows us to discuss these hypothetical future services without being precise about all aspects of them. (In Scenario C, the service requestor may want to talk about a hypothetical service which will carry crate 246, departing Portsmouth sometime on 11/12/04 and arriving in St Petersburg before 17/12/04)

2.2 Agents

If the providing and receiving of a service is to be automated, then the service requestor and provider must have some online presence. We refer to the software components which represent the parties as *agents*, with a service provider agent representing a service provider and a service requestor agent representing a service requestor.

These software components are agents in a very precise sense; they act as representatives online on behalf of some party. (This is the same sense of the word 'agent' as in 'estate agents', which act on behalf of a house seller.) Hence, the agent property is a role the component takes, rather than some intrinsic property of the component. Hence, these software entities are not necessarily agents in the sense used in Multi Agent Systems research [7]. Often, they will be reactive not proactive, and will be hardwired to follow some pre-determined process. For example, a set of Web Services provided by Amazon make up a service provider agent able to sell books on behalf of Amazon. However, as these entities become more sophisticated, and take on further tasks, they will make use of technology developed by the Multi Agent Systems community.

Another consequence of 'agent' being a role that a software component takes is that it can behave as a requestor agent at one time, and a provider agent at another. This can be seen in Scenario A; the bank's software system acts as a service provider agent to its customer, providing the service of account warnings. However, to get the warning message to its customer, the bank's system behaves as a service requestor and enters into an arrangement with a provider of a message delivery service.

If we are to be precise, we need to make a clear distinction between the service requestor (or provider) and the service requestor agent (or provider agent) which represents it. However, in practice this is not necessary in our subsequent discussions and we will use 'service provider/requestor' to refer to the agent also.

2.3 Communication

Choreography When a service is provided online, there must be some interaction between the provider and requestor. This interaction will require some exchange of messages. It must follow certain constraints if it is to make sense to both parties. In other words, the message exchange must proceed according to a certain communication protocol known to both parties. In the Semantic Web Services world, a communication protocol, which can be multi-party, is often referred to as a *choreography*. For consistency with this existing literature, we will adopt this terminology subsequently. When some exchange of messages takes place according to the constraints provided by some choreography, we refer to this as a *conversation* between two parties which satisfies the choreography.

Interactions about a service may involve more than two parties, playing different roles. In general, many multi-party interactions can be reduced to a set of two-party interactions. However, there are some cases where this is not possible. For the purposes of this paper, we focus only on two-party interactions; however, many of the concepts generalise straightforwardly to the multi-party case.

When two parties engage in a conversation, they must each have one or more communication endpoints to send and receive the messages according to some transport protocol. This is referred to as the *grounding* of the choreography. In many cases, service providers will interact via an interface specified in terms of Web Service operations. This is particularly the case for simple service provider agents with no internal state, where their choreography consists simply of a call/response interaction. The free stock quote service in Scenario D is of this type. However, there are other possibilities; the freight forwarder interacts using a complex set of RosettaNet messages with implicit state information in their sequencing, and these messages will be transported between the business partners using the RNIF standard.

Semantics in Choreography A key technical goal of Semantic Web Services is to describe the different choreographies, which parties can use to interact, in a machine-readable form. This form should represent not only the messages which are exchanged, but also provide some model for the underlying intention behind the exchange of messages on the part of both parties. In other words, it should represent the semantics of the message exchange. In Scenario C, messages are exchanged to build up a stationary order; semantic representation of this will show that a certain sequence of messages corresponds to adding an item to the order, another sequence corresponds to getting a quote for the order, and another sequence corresponds to a final agreement that the order will be processed and payment made. Doing this will allow software entities to reason about choreographies. For example, an entity could use an explicit model of a choreography to dynamically decide which action to take or message to send next.

2.4 Orchestration and Service Composition

As explained above, choreography determines the constraints on the ordering of messages sent between the service requestor and service provider. However, the constraints alone are not enough to determine exactly which message is sent when. This is the role of an *orchestration*. An orchestration is a specification, within an agent, of which message should be sent when. Hence, the choreography specifies what is permitted of both parties, while an orchestration specifies what each party will actually do.

The real power of orchestration becomes evident when we look at multiple simultaneous relationships between agents. So far in this discussion, we have focussed on a single interaction, with one agent taking the role of service requestor and the other the role of service provider. However, it is clear that in many circumstances an agent will be involved in multiple relationships; in some, it will be acting as a service provider, while in others it will be acting as a service requestor. For example, in scenario A, the bank's overdraft notification service agent acts as a provider to the bank's customer. However, it outsources the task of delivering the notification to other parties. Hence, it acts as a service requestor in relationship with these parties.

Often, such an agent will communicate with several service providers and coordinate the services they provide to produce some more complex service - as, for example, the logistics coordinator does in scenario C. This act of combining and coordinating a set of services is referred to as *service composition*. When a requestor agent is interacting simultaneously with many service providers, an orchestration can specify the sequencing of messages with all of these, including appropriate dependencies. The orchestration can be specified in several different ways. The most straightforward, and least flexible, is to make a design time choice of which service providers to use, and hard-code the integration logic in the service requestor agent. A more flexible way is to use a declarative workflow language to describe the process of integrating the interactions with the chosen service providers. This is the approach taken by BPEL [20]. This is more easily maintainable, but suffers from the drawback that if one of the chosen service providers is unavailable, then the overall service orchestration will fail. A more robust approach, advocated by WSMF [9], is not to select the service providers in advance within the orchestration, but to include descriptions of their required functionality. When the orchestration is executed, appropriate service providers are dynamically discovered and selected.

Having an explicit description of a service orchestration in terms of some process language has a further advantage. It means that the orchestration can exist independently of a specific requestor agent, and be passed between agents as a data structure. This approach is used to great effect by the OWL-S virtual machine [18]. Rather than the service requestor being responsible for generating an orchestration, any party can produce one. In particular, in the case where a single service provider offers a variety of services, it is more appropriate for the provider to take responsibility to show how they can be combined in different ways. If this is done in some agreed standard process language, such as the OWL-S process model [19], and a service requestor has access to a means to interpret that process language, such as the OWL-S virtual machine, then any such service requestor can make use of the complex service. Furthermore, such orchestrations could be composed dynamically [3].

2.5 Mediation

When an interaction between two parties takes place, there may be further need for mediation [9]. There are four forms of mediation which could be necessary;

Data Mediation A message or fragment of data represents the information it carries in some specific syntactic format. Different service providers may expect different syntactic formats for their messages, even though the information carried is equivalent. Data mediation consists of transforming from one syntactic format to another [16].

Ontology Mediation When two parties describe services, they make different choices with regard to the vocabulary of terms, and therefore ontology, used to do so. As a result, if one party is to reason with a description produced by the other party, then some additional reasoning will be necessary to translate between the two approaches. This additional reasoning is termed ontology mediation [15].

Protocol Mediation Two components which are to interact may each have been designed with a particular interaction choreography in mind. Unless agreement was reached

between the two designers (either directly, or indirectly through the adoption of a standard) then it is unlikely that the two choreographies will be identical. Protocol mediation is mediation which reconciles these two choreographies, by translating a message sequence used by one into a different message sequence used by the other to accomplish the same end [17].

Process mediation Behind any interaction, each party has some internal process which manages the reasoning and resources necessary to bring about that interaction. (In many domains of application, this will correspond to a business process.) In some cases, even though the two parties are able to interact via some protocol, there may be some difference between their processes which means this interaction will not succeed. Process mediation is mediation which reconciles the differences in such processes. This is the hardest form of mediation, and may in many cases be impossible without engaging in process re-engineering [10].

Mediation of these four different kinds is only possible automatically if the messages and choreograpies are annotated semantically. It is key to enabling service interaction to take place automatically, and so forms a core part of the Semantic Web Services research programme.

2.6 The Service Interaction Lifecycle

The lifecycle of the relationship between service requestor and provider goes through four phases; *modelling*, *discovery*, *service definition* and *service delivery*;

Service Modelling Phase Initially, a service requestor prepares a description of the service it is interested in receiving. Because it is unlikely that all details of the service will be known at the outset (for example, the provider of the service is not known, and the cost of the service may not be known) the description will be of an abstract service. This abstract service description makes up the service requirement description of the service requestor. Similarly, service providers create abstract service descriptions representing the service they are able to provide. This is referred to as the service offer description. Note that both the service requirement description and the service offer description are simply descriptions of a service, and hence use the same concepts and relations in the description. However, in each case, the service description plays a different role. In the first case, it describes a service which is being looked for, and in the second case it describes a service which is being provided.

Service Discovery Phase If the requirement description of a requestor and the offer description of a provider are in some sense compatible, then there is a match and the two parties could go on to the service definition phase. There are different formal ways of deciding whether two descriptions are compatible ([11],[12],[13],[14]). To illustrate this, consider Scenario A. The bank is looking for a service provider to alert the customer. Let us say the customer has chosen to receive the message via txt. The bank's requestor agent creates a service requirement description stating that it wants to send a txt message of length 112 characters to a number on Telefonica Movistar. A provider advertises a service offer description stating that it is able to send txt messages of maximum length 120 characters to any Spanish network at a cost of 0.1 euro. These two are potentially compatible, so a match should be made during discovery.

Service Definition Phase During discovery, a requestor may identify several providers which are potentially able to meet their needs. From this set, they may contact one or more of these and enter into a service definition conversation with them. Selection of which to contact may simply be random, or may involve some analysis of the service providers and choice of which appear in some sense 'best'. (Recall in Scenario D, the investor chose the cheaper but older service for stock quotes, because low cost was more important than having immediate information.) If service definition fails with those selected, the requestor has the option to later contact others which were not initially selected, and try with those.

The service definition phase involves taking an abstract service description of a provider and refining it to describe a specific service which meets the requestor's needs. One way of conceptualising this is to think of the abstract service as having attributes which must be instantiated. In Scenario C, the shipment service would have attributes including weight of crate, departure and arrival ports, departure and arrival times, and price. The selection of the values these attributes take is the role of the service definition phase. Sometimes, it is not necessary to specify a specific value, but some constraint on a value is adequate. (In Scenario C, the arrival time might be specified as between 18.00 and 22.00.) This process takes place through a conversation governed by a service definition choreography.

When a requestor enters into service definition phase with several possible providers, it will often be in an attempt to explore what options the different parties provide in order to select the best. (Recall in Scenario B, the service requestor agent making a stationary purchase goes through the motions of preparing an order and receiving a quote with several providers.) The requestor will only complete the service definition phase with one of them, terminating the conversations with those it has not selected.

If the service definition phase is successfully completed between two parties, they have agreed a service to be delivered by the provider to the requestor, and can enter into the service delivery phase. Some of the attributes may not be fully defined, merely constrained. (In Scenario C, the freight forwarder may specify that the crate must be lighter than 500Kg.) In this case, it means that one party (usually the provider) will allow the other to make a selection of attribute value during the delivery phase. (In Scenario C, the requestor will inform the freight forwarder of the final crate weight within the advance shipment notification message it sends just before dispatch.)There may be a formal representation of the agreed service description, which can form part of a contract between the two parties [21].

In many cases, a service definition conversation will not be necessary. The description of the service by the provider will define fixed values for all the attributes the provider cares about. The only flexibility in the description will be where a provider is willing to allow a requestor to freely choose. Effectively, the provider gives a 'take it or leave it' description of the service it provides, and the requestor simply selects one. This can be seen in Scenario D. There is no service definition conversation between the investment software and the service provider agents. Instead, the investor simply selects which to use based on her preferences.

In some cases, the conversation will involve iterative definition of the service, selecting from options to create a complete description piece-by-piece. This can be seen

in Scenario B, where the shopping trolley metaphor is used during service definition. Through an exchange of messages between the two parties, the requestor browses the wares, selects some, gets a final quote and agrees (or not) to purchase them.

Less often, the conversation may involve negotiation of certain parameters, such as price. Negotiation involves the iterative relaxing of constraints on values until some agreement is reached. Negotiation is an important area of agent technology research [8], but detailed discussion is beyond the scope of this paper.

Service Delivery Phase When the definition of a service has been agreed, then service delivery can take place. It may be immediate, as in Scenario A where the txt message is sent as soon as the bank confirms its selection. Alternatively, it may take place a while after service definition has been completed, as in Scenario C where the agreement to carry a crate in a certain ship may be made days or weeks before the actual voyage. Service delivery may take place entirely off-line, with no communication, as in Scenario A where the txt message is sent by the provider without any further exchange of messages. Alternatively, it may involve communication between the two parties. If communication takes place, this is again governed by an interaction choreography. Several different types of interaction can occur during service delivery, and each is governed by a choreography:

1. The service delivery choreography covers the exchange of messages associated directly with the delivery of the service. In some cases, the service is provided directly by this exchange of messages, as in Scenario D where the stock quote data will be carried within a reply message from the quotation Web Service. In other cases, the exchange of messages is linked with activities occurring in the real world, as in Scenario C where the messages initiate and control (to a limited extent) the movement of a crate from Portsmouth to St Petersburg.
2. A monitoring choreography covers the exchange of messages which allow the service requestor to receive information regarding the progress of the service from the provider. In Scenario C, there is a RosettaNet message exchange, 'Shipment Status Message', which allows the service requestor to get information about the progress of the shipmen from the freight forwarder. This is an example of a monitoring choreography.
3. A cancellation/renegotiation choreography allows the service requestor, in certain circumstances, to cancel or alter the service which they are receiving from the provider. In Scenario B, we can imagine (as in Amazon) that the purchaser has the option to review, modify or cancel their order through an exchange of messages, provided the order has not entered the dispatching process.

3 Architecture for Semantic Web Services

In multi agent systems research, a distinction is made between a micro-architecture and a macro-architecture. A micro-architecture is the internal component-based architecture of an individual entity. A macro-architecture is the structure of the overall community, considering each entity within it as a black box. It is also helpful to consider this distinction in Semantic Web Services. In an open community, it is necessary to standardise

the macro architecture to some extent, but the micro architecture can be more flexible, with differences in design between various community members.

3.1 Macro-Architecture

In our community, there are three possible roles that a software entity can have; service requestor agent, service provider agent and discovery provider agent. In general, an entity may have more than one role; however, for clarity we consider each separately.

To recap from the previous section, a service requestor agent acts on behalf of an individual or organisation to procure a service. It receives a service requirement description from its owner, and interacts with other agents in an attempt to fulfil this. It has some model, in an ontology, of the domain of the service and also of the kind of actions that can be taken (through message exchange) in this domain. A service provider agent is able to provide a service on behalf of an organisation. It has a service offer description in some domain ontology (ideally, the same as the requestor agent), which describes at an abstract level the kind of services it can provide. It also has a means to generate more concrete descriptions of the precise services it can deliver. Furthermore, it has a formal description of the message protocol used to deliver the service. This includes mappings from the content of messages into concepts within the domain ontology. It also includes mappings from message exchange sequences into actions. In Scenario C, a field in the initial Advance Shipment Notification (ASN) message might map onto the 'weight' attribute of the 'crate' concept within the domain. The sequence consisting of one party sending the ASN and the other party acknowledging receipt may correspond to a 'notify shipment' action in the domain ontology.

A discovery provider agent has access to descriptions of service offers, together with references to provider agents able to provide these services. These descriptions are all in some domain ontology associated with the discovery provider agent. Within this ontology is a 'service description' concept which effectively acts as a template for the descriptions of services that the discovery provider can contain. We illustrate the macro-architecture by specifying the interactions which can take place between the different agents. These interactions are roughly in order of the lifecycle progression introduced in the previous section.

Provider Agent Registering a Service Offer Description The provider agent sends a register message to the discovery agent, containing a service offer description in the ontology of the discovery agent and a URI for the provider agent. The discovery agent replies with an accept message if it is able to accept and store the description, reject otherwise. It will only reject a description if the description is not a valid concept in its ontology, or there is some practical reason it cant accept it, such as lack of memory. Prior to this, if the provider agent isn't aware of the ontology used by the discovery agent, it can send a requestOntology message to the discovery agent. The agent replies with an informOntology message containing the section of the ontology relevant to the service description. If this is a different ontology from that used by the service provider agent, then ontology mediation will be necessary. We assume this takes place within the provider agent. However, in general it could take place using a third party or within the discovery agent.

Requestor Agent Finding Possible Providers Discovery takes place through a simple exchange protocol between a service requestor agent and a discovery agent. The requestor agent sends a requestProviders message containing a service requirement description in the ontology used by the discovery agent. (As above, it can find out what this is using a requestOntology/informOntology exchange. It may then require ontology mediation, which we assume takes place within the requestor agent.) The discovery agent responds with an informProviders message containing a list of URIs of service provider agents. These correspond to those agents which have offer descriptions stored within the discovery agent which match (using the discovery agent's algorithm) with the service requirement description.

Requestor and Provider Agents Define Service: Following discovery, the requestor agent exchanges messages with one or more provider agents to define the service it will receive, and to select which provider agent to use. In our architecture, we assume a single simple service definition protocol is used by all requestor and provider agents. Our simple protocol consists of two rounds of message exchange. Initially, the service requestor agent sends a requestServices message to each provider agent. The message contains a service requirement description. The provider agent replies with an informServices message, which contain (almost) concrete descriptions of the services it is able to provide which meet the needs of the requestor. If the requestor wishes to select one of these, it replies with a selectService message containing the required service, and the provider responds with confirm. The confirm message contains a URI referencing where the description of the choreographies which will be used during service delivery are to be found. If the requestor does not select one within a certain time window, sending no response to the provider, this is taken as cancelling.

Service Delivery: Service delivery starts when one party (depending on the choreography used) sends an initiating message. Unlike previous stages, many different choreographies can be used depending on the domain of application of the service. In Scenarios A and D, the choreography is simply a single message exchange corresponding to 'do the service', with a reply being 'I have done the service and here is the result.' Scenario B is similar, except the response is 'I will do the service' (and it takes place offline, via mail.) In Scenario C, the choreography used at this stage will correspond to the sequence of messages specified by the RosettaNet standard.

Because of the large variety of choreographies which are possible during service delivery, it is at this stage that protocol mediation will play the largest role. This will particularly be the case where the choreography can be more complex, as in Scenario C. For the purposes of this architecture, we assume that any protocol mediation that is required will take place in the service requestor agent and use the choreography descriptions referenced by the provider agent. However, mediation can equally well take place within the provider or within a third party.

Given this assumption, then the macro-architecture is as follows. Each service provider has a description of the service delivery choreography associated with each service it can provide. At the end of the service definition protocol, as a parameter of the confirm message, it informs the requestor of a URI which references this description. The requestor is then responsible for accessing this description, interpreting it and engag-

ing in a message exchange with the provider which satisfies the requirements of the choreography described.

3.2 Micro-Architecture

We now look at two of the three roles that software entities can have - requestor agent and provider agent - and present a micro-architecture for each.

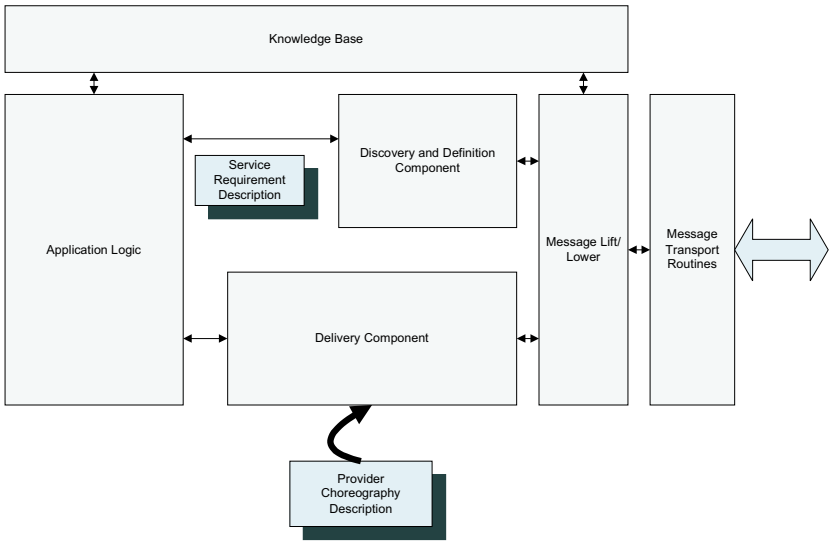

Fig. 1. Architecture of Service Requestor Agent

Figure 1 illustrates our architecture for the service requestor agent. At the heart of the agent is the application logic, which is responsible for decision making when selecting and using services. This can include integration with other back-end systems within the organisation which the service requestor agent represents. It may also access and assert information in the knowledge base. In other cases, the application logic will be provided by a user interacting through a user interface; the requestor agent effectively acts as an online proxy for the user, and relays any important decision making problems to them, acting on their choice.

The first role of the application logic is to define a service requirement description for the service it needs. When this has been done, it passes the description to the discovery and definition component. This component is responsible for managing the discovery and service definition choreographies, and sends appropriate messages to do this as described in the macro architecture above. The message format and contents are prepared using the messaging lift/lower component and passed to the transport routines for transmission via an appropriate transportation protocol. Often, but not exclusively,

these transportation routines will use WSDL Web Service technology for communicating with the service provider. At points where a decision is required - namely, when one or more provider is to be chosen to contact after discovery and when a service is to be chosen during the selection process - the decision is passed to the application logic to be made. The message lift/lower component performs data mediation. When it receives incoming messages, it translates their contents into semantic information according to an ontology, and stores these in the Knowledge Base. It generates the content of outgoing messages by using facts in the Knowledge Base to fill fields according to some message schema. For more details of the approach used, see [16]. When a service has been defined, the application logic initiates the delivery process by passing the URI identifying the delivery choreography to the delivery component. Unlike the discovery and selection component, which contains hard-wired logic for a single protocol, the delivery component is able to carry out protocol mediation. It accesses the description of the choreography given by the service provider. This shows how message contents map into the domain ontology of the knowledge base, and also how sequences of messages correspond to actions within this domain ontology. State machines describe the order in which actions can take place. The application logic can request the execution of an action. This will result in the delivery component initiating an exchange of messages with the service provider. The content of a message will be instantiated by accessing the knowledge base and 'lowering' the relevant information into the required message format using the lift/lower component. The message can then be passed to the transport routines for transmission. When a message is received as part of such an exchange, the contents of the message will be 'lifted' into the knowledge base using the lift/lower component and the delivery component will note the progress of the message exchange. When an exchange terminates (either through successful completion or some failure) the application logic is informed of this. The delivery component also handles messages from the provider which are not part of an exchange initiated by the requestor. These correspond to actions within the domain which the provider is initiating. The delivery component identifies which action they are initiating, 'lifts' the message content to the knowledge base, and informs the application logic. It replies (possibly after a decision from the application logic of how to respond) by 'lowering' content into a message, which is then passed to the transport routines. Full details of this process, and the architecture used, are given in [17].

We now turn our attention to the provider agent (Figure 2). Because, in our architecture, we assume that protocol mediation takes place within the requestor, the provider can be simpler. It also has an application logic component at its heart, which is responsible for deciding which services to offer a given requestor and also for the provisioning of the service itself. As in the case of the requestor, this will often involve integration with a variety of back-end systems belonging to the service provider's organisation. Initially, the application logic prepares a service offer description and registers this with the discovery service provider. It also prepares a choreography description associated with this service, and publishes it on the web, giving it a URI. From that point on, in our architecture, the provider agent is reactive. The service definition component has an interface (often, though not exclusively, provided by Web Service WSDL technology) which allows a requestor to submit a service requirement description. On receipt of this,

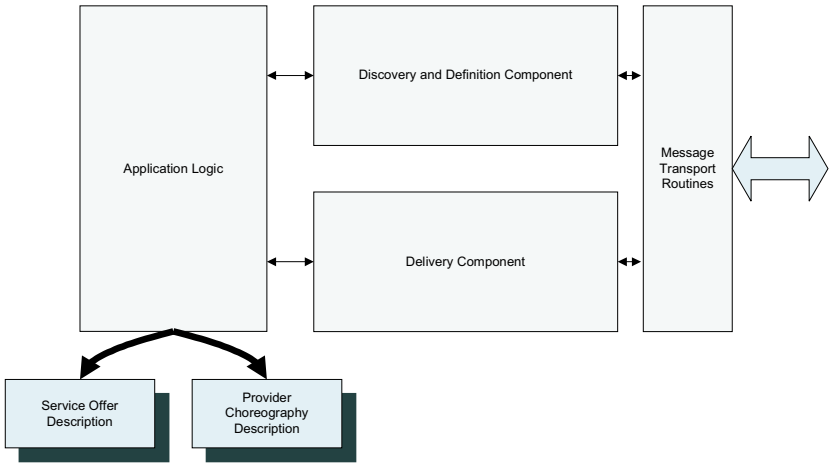

Fig. 2. Architecture of Service Provider Agent

the application logic prepares a set of possible services which satisfy the requirement, and this is sent to the requestor through the definition component interface. If the definition component receives a selection message from the requestor, it responds with a confirm containing the URI of the choreography description which it obtains from the application logic. The service delivery protocol is executed by the service delivery component, again via an interface which may or may not use WSDL Web Service technology. Unlike the requestor agent, the provider agent does not need to carry out protocol mediation so the protocol can be hard-wired in the component. Message contents are still lifted into the knowledge base, for access by the application logic. The application logic is informed of the progress of the conversation, requested to initiate internal actions to bring about the service, and also consulted if a decision is necessary during the execution of the protocol. In this way, the micro architectures of the two types of actor can animate the conversations required by the macro-architecture. The macro-architecture in turn embodies the concepts introduced in our conceptual model of Semantic Web Services.

4 Current Status and Conclusions

A demonstrator based on the architecture described above has been implemented as part of the EU Semantic Web-enabled Web Services project. It is implemented primarily in JAVA, as a distributed system with each requestor or provider agent as an independent entity. Different components internal to each agent access each other via Java RMI, to ease re-use of components beyond the demonstrator. Communication between agents takes place primarily through web service technology. To facilitate this, the agents are deployed on a web server platform consisting of Tomcat servlet container and Axis SOAP engine. The service provider agent makes use of HP's JENA semantic

web application framework (http://jena.sourceforge.net/) An RDF triple store acts as the knowledge base, and the JENA rules engine is used together with XML Schema to 'lift' data from XML into RDF. The demonstrator has been developed in a generic way, and has been used to implement the logistics supply chain scenario described above [5].

The architecture described above is one possible embodiment of the conceptual model, but others are possible. In particular, the architecture makes the following simplifying assumptions;

- We assume that service selection is done using the metaphor of choosing one from a list of options. Various other service selection and/or negotiation protocols could be used [22]. In general, it will be necessary to develop a protocol which is sufficiently flexible to capture a variety of scenarios, but tight enough to allow automated interoperability. It may be appropriate to use protocol mediation techniques at this stage in the service interaction lifecycle.
- We assume that when ontology mediation is necessary it takes place within the requestor/provider agent.
- We assume that protocol mediation takes place within the service requestor agent. (However, the software components developed have been designed to be deployed in alternative configurations.)

These simplifying assumptions were made due to constraints on the time and people available for the implementation effort of the SWWS project, and were made based on the scenarios we had chosen to focus on within SWWS. Specifically, analysis of the scenarios showed that no negotiation was necessary beyond the 'choice of alternatives' approach we used, and that mediation could be devolved to the requestors and providers. We believe that these assumptions will hold for many scenarios, but not all. If the architecture is to be more generic, work is necessary to generalise beyond these. The WSMX execution environment [23] shows promise to this end.

If Semantic Web Services are to be deployed effectively on a large scale, it will be necessary for the community to reach agreement about how to do this. A conceptual model and flexible architecture will be a necessary part of this agreement. We believe the ideas presented in this paper are a step in this direction.

Acknowledgements Thanks to Steve Battle, Oscar Corcho, Xavier Esplugas-Cuadrado, Stephan Grimm, Stuart Williams and all the team of the Semantic Web-enabled Services project.

References

1. S. McIlraith and D. Martin. Bringing Semantics to Web Services. *IEEE Intelligent Systems*, 18(1):90–93, 2003.
2. M. Paolucci and K. Sycara. Autonomous Semantic Web Services. *IEEE Internet Computing*, September 2003:34–41.
3. S. McIlraith and T.C. Son. Adapting Golog for Composition of Semantic Web Services. *Proc. 8th International Conference on Knowledge Representation and Reasoning*, 482–493, 2002.
4. J.M. Lopez-Cobo, S. Losada, O. Corcho, R. Benjamins, M. Nino and J. Contreras. Semantic Web Services for Financial Overdrawn Alerting. *Proc. of the 3rd International Semantic Web Conference (ISWC 2004)*, 782–796, Hiroshima, Japan, 2004.

5. C. Preist, J. Esplugas-Cuadrado, S.A. Battle, S. Grimm and S.K. Williams. Automated Business-to-Business Integration of a Logistics Supply Chain using Semantic Web Services Technology. *Proc. of the 4th International Semantic Web Conference (ISWC 2005)*, Galway, Ireland, 2005.
6. C. Preist. A Conceptual Architecture for Semantic Web Services. *Proc. of the 3rd International Semantic Web Conference (ISWC 2004)*, 395–409, Hiroshima, Japan, 2004.
7. M. Wooldridge and N.R. Jennings. Agent Theories, Architectures, and Languages: A Survey. *in Intelligent Agents, Proceedings of the ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, LNAI, Vol. 890, Pages 1-39, 1995.
8. N.R. Jennings, P. Faratin, A.R. Lomuscio, S. Parsons, C. Sierra and M. Wooldridge. Automated negotiation: prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.
9. D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF *Electronic Commerce: Research and Applications*, 1:113–137, 2002.
10. E. Cimpian and A. Mocan. Process mediation in wsmx. Technical report, 2005.
11. J. Gonzales-Castillo, D. Trastour, and C. Bartolini. Description Logics for Matchmaking of Services. In *Proc. of the KI-2001 Workshop on Applications of Description Logics*, volume 44. CEUR Workshop Proceedings (http://ceur-ws.org), 2001.
12. T. Payne K. Sycara M. Paolucci, T. Kawamura. Semantic mathcing of web service capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*, pages 333–347, 2002.
13. S. Grimm B. Motik C. Preist. Variance in e-Business Service Discovery. *Proc. 1st. Intl. Workshop SWS'2004 at ISWC 2004, Hiroshima, Japan, November 8, 2004, CEUR Workshop Proceedings, ISSN 1613-0073, online CEUR-WS.org/Vol-119/paper3.pdf.*
14. U. Keller R. Lara A. Polleres. WSMO Discovery. WSMO Working Draft D5.1v0.1, 2004. Available from http://www.wsmo.org/2004/d5/d5.1/v0.1/
15. O. Corcho, A. Gomez-Perez and M. Fernandez-Lopez *Ontological Engineering: with examples from the areas of Knowledge Management, e-commerce and the Semantic Web*. Springer-Verlag, 2004.
16. S.A. Battle. Round Tripping between XML and RDF *Poster Proc. of the 3rd International Semantic Web Conference (ISWC 2004)*.
17. S.K. Williams, S.A. Battle and J. Esplugas Cuadrado. Protocol Mediation for Adaptation in Semantic Web Services. HP Labs Technical Report HPL-2005-78
18. M. Paolucci, A. Ankolekar, N. Srinivasan and K. Sycara. The DAML-S Virtual Machine *Proc. of the 2nd International Semantic Web Conference (ISWC 2003)*, 290–305, Florida, USA, 2003.
19. http://www.daml.org/services/owl-s/
20. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trrickovic and S. Weerawarana. Business Process Execution Language for Web Services - Version 1.1. BEA Systems, IBM, Microsoft, SAP AG and Sibel Systems Whitepaper, 5 May 2003.
21. B. Grosof and T. Poon. SweetDeal: Representing Agent Contracts with Exceptions using Semantic Web Rules, Ontologies and Process Descriptions. *International Journal of Electronic Commerce*, 8(4):61–98, 2004.
22. The Foundation for Intelligent Physical Agents http://www.fipa.org/
23. M. Zaremba, M. Moran, T. Haselwanter, H. Lee and S. Han. WSMX Architecture http://www.wsmo.org/TR/d13/d13.4/