6

Goals and Vision

Combining Web Services with Semantic Web Technology

Chris Preist

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

Summary. This chapter introduces the combination of the formerly described Web Services and Semantic Web technologies to Semantic Web Services. It outlines the vision and goals in the Semantic Web Services area and clarifies terminology in this field. It defines an abstract Semantic Web Service architecture and introduces a life cycle of the relationship between a requester and a provider party. This motivates the subsequent chapters for description, discovery, mediation and invocation of semantically annotated services in the web.

6.1 Semantic Web Services Vision

As we have seen from previous chapters, the technologies provided by the Semantic Web are working towards a web which is machine-interpretable; a web where computer algorithms are able to process and reason with information which currently is only available in a human readable form. Web Services technologies, on the other hand, are working towards an environment where organisations can make some of their abilities accessible via the Internet. This is done by 'wrapping' some computational capability with a Web Service interface, and allowing other organisations to locate it (via UDDI) and interact with it (via WSDL). Web Service technology provides a standard and widely accepted way of defining these interfaces.

The Semantic Web Services vision [1, 2] is to combine these two technologies, and through this to enable automatic and dynamic interaction between software systems. Web Service technology allows the description of an interface in a standard way, but says nothing (in machine-interpretable form) about what the software system does, or what sequence of messages is used to interact with it. We can overcome this lack using Semantic Web technology. We can annotate software being offered via Web Service interfaces with machine interpretable descriptions describing what the software does (namely the service it provides a potential user) and how it does it. Furthermore, with ontologies able to describe the services that can be provided, we can bring about 'advertising' of services in a way which is both rich and machine-interpretable. This allows more sophisticated discovery of services than is currently possible with UDDI.

Combining these technologies enables many new things to be done. Services' as varied as protein analysis, bookselling, translation and animation rendering could be advertised and discovered automatically on the Internet. A company needing a service could locate a provider they were previously unaware of, set up a short-term business relationship and receive the service in return for a payment. All this could be done automatically and at high speed. Furthermore, several services could be combined into a more complex service, possibly automatically [3]. If one of the component services is unavailable, a replacement could be rapidly found and inserted, so the complex service can still be provided.

Note that we used the word enable' in the previous paragraph. Semantic Web Services technology is an enabling technology, so it is necessary but not sufficient to bring about the vision we have just described. It provides a means for describing services, and also infrastructural capabilities to discover services and to enable interoperation. However, it does not provide the reasoning to decide which service you want, which provider is best', how to negotiate the parameters of a service and what actions to take when using a service. If a service is simple and used in a straightforward way, this reasoning will also be simple. However, in some of the more ambitious scenarios, complex reasoning such as negotiation or dynamic planning will be necessary. Hence, Semantic Web Services alone will not bring about this brave new world – it can do so only in conjunction with other computer science disciplines.

6.2 Example Scenarios

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

6.2.1 Scenario A: Overdraft Noti cation Service

A bank provides an overdraft notification service' to its customers to help them manage their account, and to warn them 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. If this is about to happen, the customer is warned via an email, text 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 text or voice message to the customer.

This scenario is described in [4] in more detail.

6.2.2 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 companys requirements. However, less business-critical supplies, such as stationery, anti-static foot straps or reference books, 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 stationery equipment needed over the next month. It looks in a directory of suppliers the company considers acceptable for those which are able to supply stationery. The suppliers provide purchasing websites 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, allowing programs to interact with them as well as people. 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.

6.2.3 Scenario C: Provision of a Logistics Supply Chain

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 for some reason unavailable and it needs to rapidly locate and agree 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 requester 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.

This scenario is described in [5] in more detail.

6.2.4 Scenario D: Free Stock Quote Web Service

A small-time investor has a software package to keep track of his/her share portfolio. He/she is able to receive updated share prices via Web Services technology. When he/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 /per month to use. The other gives prices delayed by 30 minutes and is free. The investor chooses the latter, because he/she does not engage in real-time trading, and the software package then updates his/her portfolio information whenever he/she is online.

6.3 Key Concepts in Semantic Web Services

We now introduce some key concepts in Semantic Web Services, and show how these inter-relate. In each case, we illustrate this with examples from the scenarios introduced in Sect. 6.2. The work presented in this section follows the Semantic Web–enabled Web Services conceptual architecture [6].

6.3.1 Notion of Service

First, 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. A window cleaner performs the service of cleaning windows; a hairdresser performs the service of cutting hair. In Scenario A above, the bank provides the overdraft warning service to its customer; in Scenario B, the stationery supplier provides the service of sale and shipment of stationery 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 requester*. 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 from 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. In particular, 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 requester may want to talk about a hypothetical service which will carry crate 246, departing from Portsmouth sometime on 11/12/04 and arriving in St Petersburg before 17/12/04.)

6.3.2 Service Representation

One goal of Semantic Web Services is to bring about a computational machine-readable representation of the service, in terms of the value it provides. This is referred to as the *service description*. Usually, a service description will describe an abstract service, in which case it can be referred to as an *abstract service description*. Less often, a concrete service description is used to describe a concrete service.

To describe services, the Semantic Web approach uses techniques based on knowledge representation, a discipline which has developed a set of formal languages and techniques for describing knowledge in a way which permits reasoning with it. When describing a service, there are two key design decisions which must be made initially. First, what formal language is going to be used to describe it? Should it be described using horn clause logic, description logic, non-monotonic logic or some other approach? Different formalisms can be used, and this will be further discussed in Chap. 7. Secondly, what specific concepts and relations are going to be permitted in descriptions, and what is the meaning of these? This involves the creation or selection of an ontology, which provides a structured ontological vocabulary: a set of concepts and relations which can be used to describe things in the domain of interest. It is important that the terms the ontology provides allow a specification of the actions the service consists of, and/or the outcomes it brings about, in the terminology of the domain of value of the service.

When two parties describe services, they make different choices with regard to the language and ontology used. 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 *mediation*, specifically ontology mediation. Other forms of mediation may also be necessary, and we will discuss this further below.

6.3.3 Agents

Having discussed the representation of services, we need to consider the online representation of the service requester and provider. If the providing and receiving of a

service is to be automated, then these two parties 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 requester agent representing a service requester.

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', who 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 from the party they represent, they will make use of technology developed by the Multi Agent Systems community. For example, they may use negotiation algorithms [8] to allow them to agree details of services and prices; they may use distributed planning [9] to allow complex service composition and they may use utility theoretic reasoning to decide between possible alternative courses of action [10] as a service is delivered.

Another consequence of agent' being a role that a software component takes is that it can behave as a requester 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 requester 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 requester (or provider) and the service requester agent (or provider agent) which represents it. However, in practice this is not necessary in our subsequent discussions and we will use service provider/requester' to refer to the agent also.

6.3.4 Communication

Choreography

When a service is provided online, there must be some interaction between the provider and the requester. This interaction will require some exchange of messages. The exchange of messages within an interaction must follow certain constraints if they are 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 book, 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

As we discussed above, the first technical goal of Semantic Web Services is to provide machine-readable descriptions of service, to allow them to be reasoned with by different parties. The second 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 D, messages are exchanged to build up a stationery 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.

6.3.5 Orchestration and Service Composition

As explained above, choreography determines the constraints on the ordering of messages sent between the service requester 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 requester 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 requester. 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 requester 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, e.g., 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 requester 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 requester 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 [14]. 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 [11], is not to select the service providers in advance within the orchestration, but instead merely include descriptions of their required functionality. When the orchestration is executed, appropriate service providers are dynamically discovered and selected at run-time.

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 requester agent, and be passed between agents as a data structure. This approach is used to a great extent by the OWL-S virtual machine [12]. Rather than the service requester being responsible for generating an orchestration, any party can produce one, showing how several services can be combined to produce a more complex service. 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 [13], and a service requester has access to a means to interpret that process language, such as the OWL-S virtual machine, then any such service requester can make use of the complex service.

6.3.6 Mediation

When an interaction between two parties takes place, there may be further need for mediation. There are four forms of mediation which could be necessary: *data mediation*, *ontology mediation*, *choreography mediation* and *process mediation*. We will now briefly introduce each.

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.

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.

Protocol Mediation

Two components which are to interact with each other (such as a service requester and service provider) 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.

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.

Mediation of these four different kinds is only possible automatically if the messages and choreographies 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. Further discussion will be provided in Chap. 10.

Up to now, we have discussed interactions between service provider and service requester in general terms, without considering the underlying goals of the interaction. This is because there are several different goals an interaction can have behind it. As the relationship between service provider and service requester progresses, different goals are required. For that reason, we now turn to this relationship.

6.3.7 Life cycle

The life cycle of the relationship between service requester and service provider goes through four phases: *modelling*, *discovery*, *service definition* and *service delivery*. During discovery, a requester attempts to locate possible providers able to give it the service it requires. During service definition, the requester and provider interact to define the details of the service which is to be provided. During service delivery, different kinds of interactions can occur which are associated with the provision of the service.

Service Modelling Phase

At the outset of the discovery phase, a service requester 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 (e.g., 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 requester. 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 requester 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 ways of deciding whether two descriptions are compatible; these will be discussed in Chap. 8.

To illustrate this, consider Scenario A. The bank is looking for a service provider able to send a message to the customer. Let us say the customer has chosen to receive the message via text. The bank's requester agent creates a service requirement description stating that it wants to send a text message of length 112 characters to a number on Telefonica Movistar, a Spanish mobile network. A provider advertises a service offer description stating that it is able to send text messages of maximum length 120 characters to Telefonica Movistar numbers at a cost of € 0.1. These two are potentially compatible, so a match should be made during discovery.

There are also different architectures which can be used to carry out discovery. The most common is a centralised discovery service' which is contacted by the requester using a simple message exchange protocol.

Service De nition Phase

During discovery, a requester may identify several providers which are potentially able to meet their needs. From the set of providers identified, the requester 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 requester 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 so that it describes a specific service which meets the requester'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 requester 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 requester agent making a stationery purchase goes through the motions of preparing an order and receiving a quote with several providers.) The requester will complete the service definition phase with only 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 requester, 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 500 Kg.) 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 requester will inform the freight forwarder of the final crate weight in 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 [16].

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 requester to freely choose. Effectively, the provider gives a take it or leave it' description of the service it provides, and the requester simply selects one. This can be seen in Scenario D. There is no service definition con-

versation between the investment software and the service provider agents. Instead, the investor simply selects which to use based on his/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 requester 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,but detailed discussion is beyond the scope of this chapter.

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 text 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 text 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 requester 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 requester to get information about the progress of the shipment from the freight forwarder. This is an example of a monitoring choreography.
3. A cancellation/renegotiation choreography allows the service requester, 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.

6.4 Architecture for Semantic Web Services

Having introduced the concepts used in Semantic Web Services, we now consider an architecture which can be used to develop and deploy applications. Inevitably, by moving from a conceptual level to an architecture, certain design decisions will be made. We do not claim that this is the only way to make such decisions.

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 within a community. 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.

6.4.1 Macro-Architecture

Initially, we will present the macro-architecture for our community. In our community, there are three possible roles that a software entity can have: service requester agent, service provider agent and discovery provider agent. In general, an entity may have more than one role; however, for clarity we will consider each role separately.

To recap from the previous section, a service requester 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 the requirement it has been given. It has some model, in an ontology, of the domain of the service and also has some model 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 requester 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 service offer 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 life cycle progression introduced in the previous section.

1. Provider agent registering a service offer description

   A simple message exchange protocol is used between a provider agent and a discovery agent. 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 can't accept it, such as lack of memory. Prior to this, if the provider agent is not 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.

2. Requester agent finding possible providers

   Discovery takes place through a simple exchange protocol between a service requester agent and a discovery agent. The requester 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 requester agent.) The discovery agent responds with an informProviders message containing a list of URIs of service provider agents. These correspond to those provider agents which have offer descriptions stored within the discovery agent which match (using the discovery agent's algorithm) with the service requirement description.

3. Requester and provider agents define service

   Following discovery, the requester 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 requester and provider agents. This protocol is adequate for very many service definition tasks; however, in the general case this assumption is unrealistic and multiple protocols (and possibly protocol mediation) would be necessary. The FIPA standards [17] provide various possible negotiation protocols which could be used at this stage. Our simple protocol consists of two rounds of message exchange. Initially, the service requester agent sends a requestServices message to each provider agent. The message contains

a service requirement description. The provider agent replies with an inform-Services message, which contain (almost) concrete service descriptions of the services it is able to provide that meet the needs of the requester. If the requester 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 requester does not select one within a certain time window, sending no response to the provider, this is taken as cancelling. Note that this protocol does not allow negotiation–it simply allows the service provider to list a set of potentially interesting concrete services to the requester, and allows the requester to select one of these. Note also that our protocol does not capture Scenario B, intelligent procurement. The service selection protocol used in this example is a shopping-trolley protocol. While this is a natural protocol for human users (who are interested in browsing, and looking at one item at a time), it is less essential for software entities interacting with the service provider. We can imagine an alternative access protocol to shopping sites, similar to the one described here, where a requester agent submits a list of product descriptions it is interested in, receives a list back of relevant available products, selects a subset and places an order. However, in practice, many sites will continue to use the shopping-trolley metaphor meaning that protocol mediation will be important at this stage. For the purposes of the architecture presented here, we ignore this additional complexity and assume all parties use the same service specification protocol.

4. 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 off-line, 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 requester 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 appears 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 requester of a URI which references this description. The requester is then responsible for accessing this description, interpreting it and engaging in a message exchange with the provider which satisfies the requirements of the choreography described.

6.4.2 Micro-Architecture

Having described the macro-architecture, we now turn to the micro-architecture of the system. We look at two of the three roles that software entities can have – requester agent and provider agent – and present a micro-architecture for each. The micro-architecture of the discovery service provider agent will be covered in Chap. 8. Note that, unlike the macro-architecture, a micro-architecture is not normative within a community. The macro-architecture defines the exchange of messages between entities of different roles, and if the community is to function effectively, this must be agreed and adhered to (though the provision of protocol mediation within the macro-architecture allows some flexibility). The micro-architecture of each agent, however, need not follow some pre-agreed structure. The community can function perfectly well with any internal structure, provided the functionality the micro-architecture implements does indeed correspond to the requirements of the macro-architecture.

Figure 6.1 illustrates our architecture for the service requester agent. At the heart of the agent is the application logic, which is responsible for decision-making with

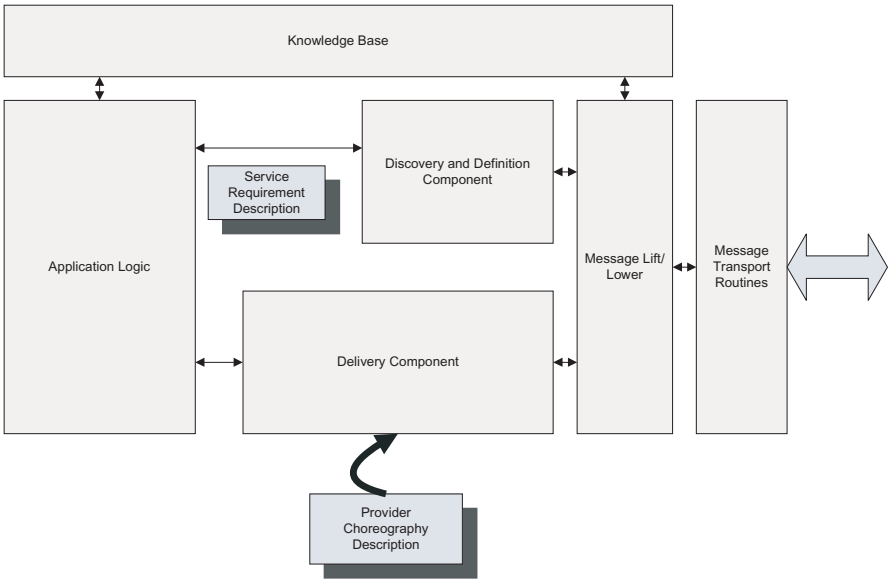

Fig. 6.1. Architecture of service requester agent

regard to which service to select and how to make use of it. This can include integration with other back-end systems within the organisation which the service requester 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 requester 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 for 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. 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 onto the domain ontology of the knowledge base, and also shows 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 through 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 requester. 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 Chap. 10.

We now turn our attention to the provider agent (Fig. 6.2). Because, in our architecture, we assume that protocol mediation takes place within the requester, 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 requester and also for the provisioning of the service itself. As in the case of the requester, 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 requester to submit a service requirement description. On receipt of this, the application logic prepares a set of possible services which satisfy the requirement, and this is sent to the requester through the definition component interface. If the definition component receives a selection message from the requester, 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 requester 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

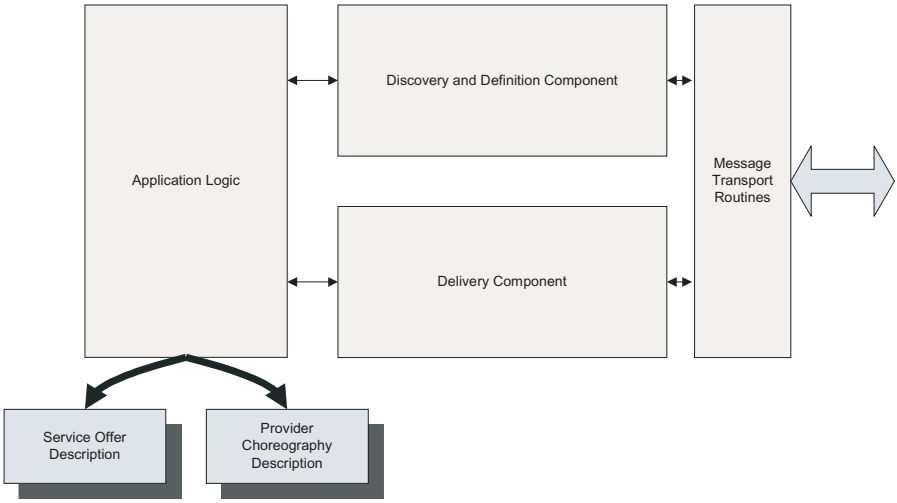

Fig. 6.2. Architecture of service provider agent

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.

6.5 Outlook

In this chapter, we have presented a conceptual model for Semantic Web Services which is driven from a requirements analysis of several scenarios. Using this conceptual model, we have developed a technical architecture which could be used to deploy applications of Semantic Web Services. We have introduced the key notions of discovery, service description, mediation and composition, and shown how they form part of a service life cycle within our conceptual model. Subsequent chapters will provide more details of these ideas, and present the techniques available to make them real.

The architecture presented in this chapter is one possible embodiment of the conceptual model, but others are possible. This particular embodiment has been implemented as part of the EU Semantic Web–enabled Web Services program, and has been used to create a demonstrator of Semantic Web Services technology in the domain of logistics supply chain management [5]. 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 chapter are a step in this direction.

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. *Proceedings 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. *Proceedings 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. *Proceedings of the 4th International Semantic Web Conference (ISWC-2005)*, Galway, Ireland, 2005.
6. C. Preist. A Conceptual Architecture for Semantic Web Services. *Proceedings 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, Lecture Notes in Artificial Intelligence, 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. E.H. Durfee. Planning in Distributed Artificial Intelligence. *Foundations of Distributed Artificial Intelligence*:231–245, John Wiley, 1996.
10. M. Barbuceanu and W. Lo. A Multi-Attribute Utility Theoretic Negotiation Architecture for Electronic Commerce. *Proceedings Fourth International Conference on Autonomous Agents (AGENTS 2000)*:239–246, 2000.
11. D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF *Electronic Commerce: Research and Applications*, 1:113–137, 2002.
12. M. Paolucci, A. Ankolekar, N. Srinivasan and K. Sycara. The DAML-S Virtual Machine *Proceedings of the 2nd International Semantic Web Conference (ISWC-2003)*, 290–305, Florida, USA, 2003.
13. `http://www.daml.org/services/owl-s/`
14. 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.
15. D. Trastour, C. Bartolini and C. Preist. Semantic Web Support for the B2B E-Commerce Pre-Contractual Lifecycle *Computer Networks*, 42(5):661–673, 2003.
16. 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.
17. The Foundation for Intelligent Physical Agents `http://www.fipa.org/`