Automated Business-to-Business Integration of a Logistics Supply Chain Using Semantic Web Services Technology

Chris Preist[1], Javier Esplugas-Cuadrado[2], Steven A. Battle[1], Stephan Grimm[3], and Stuart K.Williams[1]

[1] Hewlett-Packard Laboratories, Filton Road, Stoke Gifford, Bristol, BS34 8QZ, UK
{chris.preist, steve.battle, skw}@hp.com
[2] Hewlett-Packard Espanola SL, Jose Echegaray nº8, La Rozas, Spain. 28230
javier.esplugas.cuadrado@hp.com
[3] Forschungszentrum Informatik(FZI), Haid-und-Neu-Strasse 10-14, 76131 Karlsruhe, Germany
grimm@fzi.de

Abstract. In this paper, we present a demonstrator system which applies semantic web services technology to business-to-business integration, focussing specifically on a logistics supply chain. The system is able to handle all stages of the service lifecycle – discovery, service selection and service execution. One unique feature of the system is its approach to protocol mediation, allowing a service requestor to dynamically modify the way it communicates with aprovider, based on a description of the provider's protocol. We present the architecture of the system, together with an overview of the key components (discovery and mediation) and the implementation.

1 Introduction

The demonstrator system presented in this paper uses semantic web services technology to tackle the problem of business-to-business integration (B2Bi). It was developed as one of four case studies used in the European Union Semantic Web-enabled Web Services (SWWS) project. Increasingly, when two companies wish to do business with each other, they establish a means of exchanging messages via the internet. This connection can be used for a variety of purposes – placing orders, invoicing, making payments, initiating and tracking shipment of goods and providing customer service information, among many others. The aim of such a connection is to allow automated or semi-automated processing of many of the transactions which take place between the two companies, and thus reduce costs and increase speed.

However, to set up such a relationship requires a large initial investment of time and money. A team of developers need to reconcile the business processes of the two organizations and design a set of messages and permissible message sequences that can flow between them. This can be a formidable task. To ease this, standards bodies have developed standard sets of messages and guidelines for how they should be used. Three key standards in the world of B2Bi are EDIFACT, AnsiX12 and Roset-

Y. Gil et al. (Eds.): ISWC 2005, LNCS 3729, pp. 987 – 1001, 2005.

taNet. By agreeing on one such standard, two organizations can ease the integration task. However, these standards have reasonable flexibility in them, both in terms of message content and message sequencing. This means that even having agreed a standard, significant effort is required to agree and implement exactly how it is used. As a result of this, even a standards-based B2Bi connection can take six months to set up.

Semantic Web Services technology [1, 2] uses the tools of the semantic web to describe both the purpose of a service and the behaviour of its provider during service execution. This has the potential to significantly speed up this integration process; If one business partner gives a description of how to interact with it, it is possible to use mediation technology [3] to adapt the interaction of the other business partner so as to be compatible. Ideally, this would be fully automated, reducing integration time from months to minutes.

Prior to integration, the selection of a business partner can also be time consuming. By providing semantic descriptions of the services a business offers, then discovery techniques [4, 5, 6] can support this process. This is particularly important when selection is needed rapidly, such as the emergency replacement of a link in a supply chain. Our system supports automated discovery and selection of a service provider, and description-driven mediation which allows automated integration. The paper is structured as follows. In section 2, we introduce a motivating example, in the domain of logistics, and show how semantic web technology can be used to support it. In section 3, we present the architecture of the system we have developed, and show how it is used in the logistics domain. In section 4, we present the discovery module, and in section 5 we present the mediation modules. In section 6, we discuss the implementation. In section 7 we discuss limitations of the current implementation, lessons learned and related work. We then present the business value of the system, and conclude.

2 The Logistics Example

To motivate this work, we use an example scenario. We consider a manufacturing company in Bristol, UK which needs to distribute its goods internationally. It does not maintain its own transportation capability, but instead outsources this to other companies, which we refer to as *Freight Forwarders*. These companies provide a service to the manufacturing company – they transport crates on its behalf. However, the manufacturing company still needs to manage relationships with these service providers. One role within this company, which we refer to as the *Logistics Coordinator*, is responsible for doing this. Specifically, it carries out the following tasks;

1. Commissioning new service providers, and agreeing the nature of the service they will provide. (E.g. locating a new freight forwarder in Poland, and agreeing that it will regularly transport crates from Gdansk to Warsaw.)
2. Communicating with service providers to initiate, monitor and control shipments. (E.g. informing the Polish freight forwarder that a crate is about to arrive at Gdansk; receiving a message from them that it has been delivered in Warsaw, and they want payment.) This is done using one of the messaging standards, EDIFACT.

3. Coordinating the activity of service providers to ensure that they link seamlessly to provide an end-to-end service. (E.g. making sure the shipping company plans to deliver the crate to Gdansk when the Polish transport company is expecting it. Informing the Polish company when the shipping company is about to drop it off.)
4. Communicating with other roles in the company to coordinate logistics with other corporate functions. (E.g. sales to know what to dispatch; financial to ensure payment of freight forwarders.)

In our scenario, we consider a specific logistics supply chain from Bristol, UK to Warsaw, Poland (Fig 1). It consists of three freight forwarders: The first is a trucking company, responsible for transporting crates from the manufacturing plant in Bristol to the port of Portsmouth, UK. The second is a shipping company, responsible for shipping crates from Portsmouth to the Polish port of Gdansk. The third is another trucking company, which transports crates to the distribution warehouse in Warsaw. We assume that the Logistics Provider communicates with the Freight Forwarders using the EDIFACT standard, and is already successfully using this logistics chain.

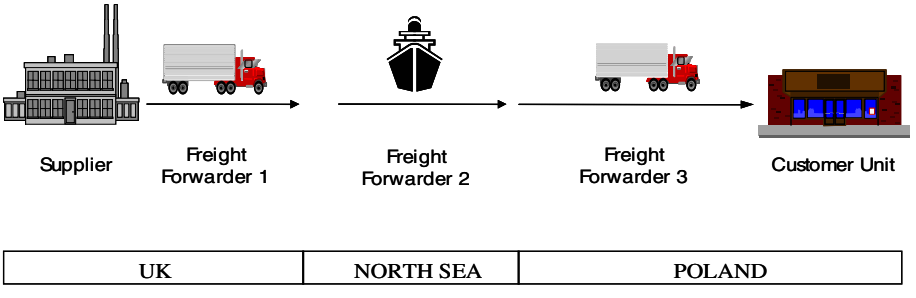

Fig. 1. Example Logistics Supply Chain

However, at some point a problem arises; the shipping company is temporarily unavailable and a new freight forwarder must be used for one shipment. At this point the Logistics Coordinator must;

1. Locate a new shipping service provider able to meet its needs.
2. Agree a service definition with it as to what exactly it should do. (When the crate will be transported, to where, how much it will cost, etc.)
3. Perform B2B integration with the provider, to ensure messages can flow between them. We assume the new provider communicates using RosettaNet.
4. Initiate and monitor the shipment via the logistics supply chain.
5. Coordinate the three freight forwarders to ensure a seamless end-to-end service, resulting in the crate being shipped from Bristol to Warsaw.

Semantic Web Services technology can be deployed throughout this lifecycle to automate or semi-automate what currently takes significant time and effort.

1. Service Discovery can be used to locate potential service providers, based on them advertising descriptions of their service capabilities.

2. Service Definition allows the refining of a service description to specify exactly what the provider and requestor agree the service should do.
3. Message and Protocol Mediation allow a new provider to be integrated and communicated with, even though it uses a different messaging standard.

We now describe how our scenario can be automated using semantic web services. 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 can provide, and asks for providers able to ship between Portsmouth and Gdansk. 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 Gdansk. 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 Gdansk 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 freight forwarder 2, and ends with the sending of a Proof of Delivery and Invoice by freight forwarder 2 when the crate arrives in Gdansk.

3 Overall System Architecture

The overall system architecture used in the demonstrator system is provided by the generic SWWS Technical Architecture. The underlying conceptual model is provided by the SWWS Conceptual Architecture [7]. Here, we summarise the technical architecture and relate it to the specific actors in our B2B scenario.

In Agent Technology 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. Initially, we will present the macro-architecture for our community. There are three possible roles that a software entity can have; service requestor agent, service provider agent and discovery provider agent.

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 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. In our

scenario, the Logistics Coordinator takes the role of service requestor agent in relationship with each of the Freight Forwarders.

A service provider agent is able to provide a service on behalf of an organisation. In our scenario, service provider agents represent the three freight forwarder companies used, as well as additional companies which could potentially be used by the logistics provider. It has a service offer description in some domain ontology (ideally, the same as the requestor agent), which gives an abstract description of services it can provide. In our scenario, for example, this would state that a company can ship crates from UK ports to Baltic ports. It also has a means to generate more concrete descriptions of the precise services it can deliver. (For example, a specific shipment of crate42 from Portsmouth to Gdansk on 25/03/05.) 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 our scenario, 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 contains descriptions of service offers, together with references to provider agents able to provide these services. These service offer descriptions are all expressed 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. In our scenario, the ontology defines concepts relevant to logistics and transportation, and the descriptions the discovery provider contains are descriptions of transportation services the freight forwarders are able to offer.

We illustrate the macro-architecture by describing the interactions which can take place between the different agents. These interactions are roughly in order of the service lifecycle progression [8] adopted by the conceptual architecture.

1. *Provider agent registering a capability with the discovery provider.*

Initially, any service provider agent must register its service offer descriptions with the discovery provider using a simple message exchange protocol. It does this in terms of the ontology used by the discovery provider, and hence may require ontology mediation. In our scenario, each Freight Forwarder will register abstract descriptions of the services it can provide.

2. *Requestor agent finding possible providers.*

Discovery takes place through a simple message exchange protocol between a service requestor agent and a discovery agent. The requestor agent sends a message containing a service requirement description, and the discovery agent responds with a message containing a list of URIs of service provider agents. These correspond to those provider agents with offer descriptions which match the service requirement description, according to the discovery agent's algorithm. In our scenario, the Logistics Coordinator will send a description of the shipment it requires – that it is from Portsmouth to Gdansk, it must arrive by 27th March, etc. It will receive back a list of all freight forwarders which have advertised a service capability compatible with these requirements, as e.g. one that covers all the Baltic Sea area with its shipping services.

3. 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 service requirement description to each provider agent it is considering using. The provider agent replies with a list of (almost) concrete service descriptions of the services it is able to provide which meet the needs of the requestor. The requestor can select one of these, with the provider confirming the selection to the requestor. The confirm message contains a URI reference where the description of the choreographies, which will be used during service delivery, can 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.

In our scenario, the Logistics Coordinator sends a description of the shipment it requires to one or more of the Freight Forwarders located at the previous stage. They respond with specific detailed descriptions of relevant shipment services – for example, one may state that the crate can be carried on a ship departing on 24th March at 3pm, with a cost of 30 euros. A given freight forwarder may provide several options at this stage. The Logistics Coordinator reviews these, and makes a selection (either automatically using stored preference information or, more likely, by providing the best options to a user who makes the final decision.)

4. Service Delivery
Service delivery starts when one party (depending on the choreography used) sends an initiating message. The choreography used at this stage will correspond to the sequence of messages specified by the RosettaNet or EDIFACT standard. 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 engaging in a message exchange with the provider which satisfies the requirements of the choreography described. Exactly how this is done will be described in section 5.

Having described the macro-architecture, we now turn to the micro-architecture. We look at two of the three roles that software entities can have – requestor agent and provider agent – and present a micro architecture for each. The micro architecture of the discovery service provider agent will be covered in section 4. Figure 2 illustrates our architecture for the service requestor agent. The application logic is responsible for decision making with regard to which service to select and how to make use of it. Normally, this will be integrated with back-end systems within the organisation which the service requestor agent represents. In our demonstrator, we provide a user interface to allow a user to make the decisions that would be made by such a system.

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, which exchanges appropriate messages to do this. The message format and contents are prepared and passed to the transport routines for transmission via an appropriate transportation protocol. At points where a decision is required – namely, when one or more provider is chosen after discovery and when a service is selected – it is made by the application logic.

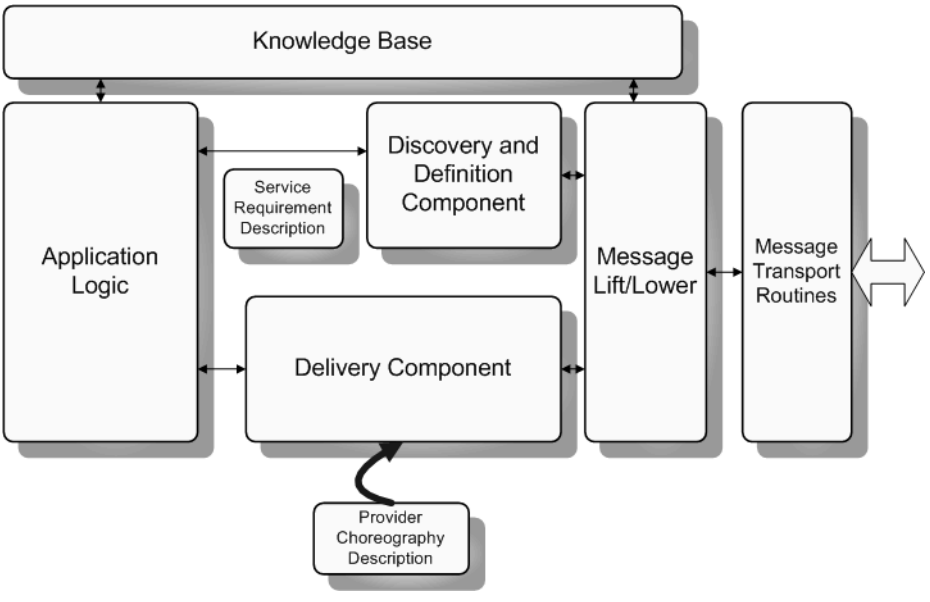

Fig. 2. Service Requestor Agent Micro-Architecture

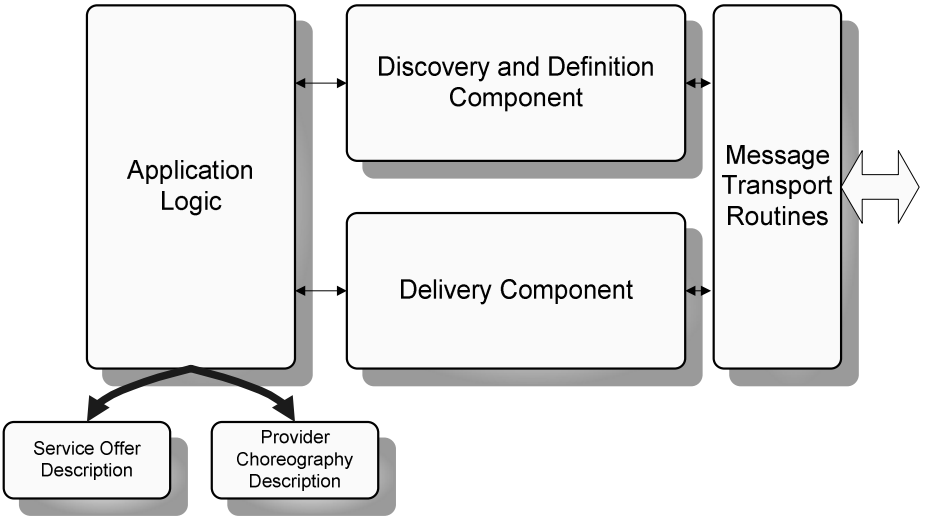

Fig. 3. Service Provider Agent Micro Architecture

When a service has been defined, the application logic initiates the delivery process by using the delivery module. The delivery module 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. The application logic can request the execution of one of these actions. This will result in the delivery module initiating an exchange of messages with the service provider. When an exchange terminates (either through successful completion or some failure) the application logic is informed of this. The delivery module 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. It informs the application logic of the actions and updates the knowledge base with relevant data from the messages. Details of this process are given in section 5.

We now turn our attention to the provider agent (figure 3). In our architecture we assume that protocol mediation takes place within the requestor, so the provider can be simpler. The application logic module is responsible for deciding which services to offer a given requestor and also for the provisioning of the service itself. This will usually be provided by back-end systems belonging to the provider's organisation.

Initially, the application logic prepares a service offer description and registers this with the discovery service provider. From that point on, in our architecture, the provider agent is reactive. The service definition module can receive a service requirement description from a requestor. The application logic then prepares a set of possible services which satisfy the requirement, and this is sent to the requestor. If the definition module receives a selection message from the requestor, it returns the URI of the choreography description which it obtains from the application logic. As the provider agent does not need to perform mediation, service delivery is carried out by a hard-wired protocol description which interacts with the application logic when business actions are required.

4 Service Description and Discovery

We now describe the service discovery functionality in more detail. The approach we use is inspired by that of [5]. During discovery and service selection, the business-level description of the service plays a key part. It gives a detailed description of the service in terms of the domain in which it provides value to the user, using some domain ontology. In our logistics domain, this will be a description of what goods are to be transported, where they will be transported from, which vehicle is being used, when the vehicle will depart, what its destination is, when it is expected to arrive, and other relevant terms of service such as insurance liability, cost and payment conditions, etc. At the end of the service selection stage, a concrete service description should be agreed between the requestor and provider, and effectively forms an informal 'contract' between the two parties. An example is the following:

$$
\begin{aligned}
&\mathit{Contract} \equiv \\
&\quad \mathit{Shipping} \sqcap \exists\, \mathit{startLocation}.\{\mathit{Portsmouth}\} \sqcap \exists\, \mathit{endLocation}.\{\mathit{Gdansk}\} \sqcap \\
&\quad \exists\, \mathit{dateOfDeparture}.{=}_{\text{2005-03-24}} \sqcap \exists\, \mathit{dateOfArrival}.{=}_{\text{2005-03-26}} \sqcap \\
&\quad \exists\, \mathit{item}.\{\mathit{SmallCargo\#typeA}\} \sqcap \exists\, \mathit{vehicle}.\{\mathit{CargoShip\#34}\} \sqcap \exists\, \mathit{price}.{=}_{90} \sqcap \\
&\quad \exists\, \mathit{currency}.\{\mathit{Euro}\} \sqcap \exists\, \mathit{meansOfPayment}.\{\mathit{EuroCreditTransfer}\}
\end{aligned}
$$

This concrete service description states that an item of small cargo will be carried on cargo ship 34 from Portsmouth to Gdansk, leaving on the 24th March and arriving on the 26th, and payment of 90 € will be made by credit transfer. It is expressed as an OWL-DL concept whose properties are restricted to specific values, allowing a unique configuration of the service. The terms used in this description are defined in a logistics domain ontology and in more generic ontologies for geography and vehicles.

As it stands, such a concrete description of a service is clearly inappropriate for advertising or discovery, as requests and adverts would have to include many such classes covering all acceptable service parameter configurations. Instead, requestors and providers abstract from concrete parameter information, switching to less specific class descriptions. In such abstract service descriptions they specify the set of concrete services that they are willing to accept. For example, a freight forwarder may advertise the following capability, using an OWL-DL based description approach for abstract service descriptions explained in [6].

$$S_p \equiv Shipping \sqcap \exists\, startLocation.EUPort \sqcap \exists\, endLocation.BalticPort \sqcap \forall\, item.Container \sqcap \forall\, vehicle.Ship \sqcap \exists\, meansOfPayment.(Cheque \sqcup BankTransfer)$$

This states that the service provider offers shipping services from EU ports to Baltic ports, can carry containers, and can accept payments by cheque or bank transfer. By using concepts and subconcepts to restrict the description appropriately, the service provider can give a precise view of the service it offers. It registers this with the discovery agent. Similarly, a requestor can describe the kind of service it needs;

$$S_r \equiv Shipping \sqcap \exists\, startLocation.\{Portsmouth\} \sqcap \exists\, endLocation.\{Gdansk\} \sqcap \exists\, dateOfArrival.\leq_{2005\text{-}03\text{-}27} \sqcap \exists\, item.CargoContainer \sqcap \forall\, meansOfPayment.(CreditCard \sqcup BankTransfer)$$

This requests the shipping of a cargo container from Portsmouth to Gdansk, to arrive by the 27th March at the latest. Payment can be made by credit card or bank transfer. Hence, by using OWL-DL concepts, we can give descriptions of various granularities, from abstract service requests/offers to specific agreed parameter values in contracts.

When the discovery agent receives a service request, it returns the set of all service advertisements which intersect with the request. An advert and a request intersect if they specify at least one common concrete service. The discovery agent uses an internal DL reasoner (RACER [9]) to check for intersection. Full details of the inferencing mechanism are given in [6]. The list of services returned includes URIs referencing the service providers, allowing the requestor to make direct contact. A requestor then makes contact with one or more of them to select and agree a concrete service. In some domains, negotiation of parameters may be necessary at this stage [10]. However, in our domain it is adequate for a provider to offer a list of relevant concrete services to the requestor, and allow them to select one. Again, this functionality can be provided by using a DL reasoner, this time internally to the service provider.

5 Mediation During Service Execution

Mediation is essential in our scenario to allow the rapid integration of a new freight forwarder into a logistics chain. We now present an overview of the approach taken. For a detailed description, see [11]. Communication is required during the execution of the service, as the shipment is initiated and progresses, to coordinate the behaviour of the service requestor and provider. In our scenario, we assume that the logistics coordinator usually communicates with freight forwarders using EDIFACT, but must now use RosettaNet with its new provider.

Our approach to mediation is based around the insight that, even though there may be several different communications protocols used to communicate about a given task, it is often the case that the underlying models of the task that are implicit in these protocols are very similar to each other. In the case of the logistics domain, analysis of the EDIFACT, ANSI X12 and RosettaNet protocols found that the set of actions carried out to execute the task, and the sequencing constraints on them, are identical. Hence, an *abstract protocol* can be identified and abstracted from the specific communications protocols [12]. In our system, the application logic communicates with the mediation component in terms of the actions within the abstract protocol – it informs the mediation component when it wishes to initiate such an action, and is informed by the mediation component when the other party carries out an action. The abstract protocol is represented as concurrent processes described by finite state machines, which can be used by the mediation component to determine what actions are permitted at any given stage in the process. The actions used are given in Table 1.

Each action in the abstract protocol maps to some exchange of messages in a specific standard such as RosettaNet or EDIFACT. This mapping will vary from standard to standard. We refer to this mapping as a *concrete protocol* relating to a specific standard. For example, in RosettaNet, the informReadyForCollection action maps to a sequence consisting in the Logistics Coordinator sending an Advanced Shipment Notification message (with up to 3 re-sends if no response within half an hour), followed by it receiving a response from the Freight Forwarder. In EDIFACT, however, it maps to a three-way exchange consisting of a DESADV message, responded to by an EDIFACT::ACK, followed by a re-send of the DESADV message. A concrete protocol is represented as concurrent processes described by finite state machines, which are used by the mediation component to manage the exchange of messages when an action takes place. The finite state machines are encoded in RDF, with transitions between states encoded procedurally in JavaScript.

When the application logic wishes to initiate an action, it informs the mediation component. The mediation component checks that this action is permissible in the current state of the abstract protocol, and if it is, it executes the appropriate state machine within the concrete protocol. This will result in the sending and receiving of messages. On termination, the mediation component informs the application logic of the success or otherwise of the action. When the mediation component receives a message from the other party which does not correspond to an action it has initiated, it pattern-matches against the action mappings in the concrete protocol to identify which action the other party is initiating. (If the protocol is well-designed, this should be unique.) It then executes the appropriate concrete protocol to respond to this action, and informs the application logic to allow the service requestor to respond.

Table 1. Communicative acts involved in the execution of a logistics service

Communicative Act	Direction	Communicative intent
informReadyForCollection	LC to FF	Inform the FF that the shipment is available for collection.
requestShipmentStatus	LC to FF	Request an update of the shipment status from the FF.
informShipmentStatus	FF to LC	Inform the LC of the shipment status
informReadyToDeliver	FF to LC	Inform the LC that the FF is ready to deliver the shipment.
informShipmentDelivered	FF to LC	Inform the LC (and provide proof) that the FF has infact delivered the shipment.
requestPayment	FF to LC	Request payment for delivering the shipment from the LC.

In addition to dealing with the message sequencing, the concrete protocol also contains data mappings for the syntax of the messages, showing how the different fields in the message correspond to different concepts in the domain ontology. When a message is received, content within that message is 'lifted' into an RDF knowledge base to become an instance of a concept in the logistics ontology. The application logic is able to read and assert information in this knowledge base as necessary. When a message needs to be transmitted by the mediation component, it 'lowers' appropriate concept instances within this knowledge base into an XML syntax appropriate to the chosen standard. The technology used to do this is described in [13]. Using this mediation technology, a requestor can communicate with different providers using different standards, while allowing the application logic to be encoded in terms of business actions. All it need do is insert the appropriate concrete protocol into its mediation component. Because we assume that the requestor is 'semantically enabled' (i.e. its internal logic uses RDF) the mediation component can be part of it. If it were not, mediation could take place as a semantically enabled intermediary agent using similar techniques. These alternative design decisions are discussed in [11].

During service execution, the freight forwarders' behaviour must be coordinated. For example, when the first is about to deliver the crate to Portsmouth docks, the second freight forwarder must be informed. This is achieved through a combination of the business logic and the mediation component. The actions involved are straightforward, and can be encoded as part of the business workflow. However, the messages involved are in different protocols, so require mediation. The first trucking company sends notification in EDIFACT. The mediation system recognizes that this message corresponds to an 'informReadyToDeliver' action, which the workflow identifies as requiring an 'informReadyForCollection' exchange with the shipment company. This is initiated, and the mediation component generates the appropriate RosettaNet messages. Specific data, such as the estimated time of delivery, are transferred from one message to the other through a process of lifting/lowering to/from the RDF database.

6 Implementation

The demonstrator system 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 Discovery Service is a self-contained web service that can be deployed remotely and accessed via a standard web service interface. The generic discovery service is linked to a repository containing OWL-DL service descriptions compliant to an early form of WSMO (http://www.wsmo.org/). Reasoning is performed by the RACER DL reasoner. The Freight Forwarders provide web service interfaces for the exchange of messages involved in service specification. Service execution requires the exchange of EDIFACT or RosettaNet messages, which takes place over a standard http port, as specified by either the EDIFACT or RosettaNet standard. The logistics coordinator interacts with the discovery service via its web service interface, and the freight forwarders both for service specification, via their web service interfaces, and service execution, via a standard http port using RosettaNet or EDIFACT messages in XML. The components within the logistics coordinator are implemented in JAVA, with the RDF knowledge base provided by HP's JENA semantic web application framework (http://jena.sourceforge.net/). Transformation of XML messages into RDF, and vice-versa, was carried out using a combination of XML Schema and the JENA rules engine. As noted above, the application logic is provided by a user interface allowing the user to make decisions the application logic would. If the system were used in a real environment, this functionality would be provided by a business workflow system integrated with the corporate IT systems.

7 Analysis and Related Work

The system presented in this paper is a demonstrator, not a deployed application. For this reason, certain simplifications have been made which need to be revisited. The first issue is that of representation; OWL does not have sufficient support for concrete domain predicates so that date ranges cannot properly be expressed and reasoned with. However RACER does support this feature and extensions to OWL such as OWL-E [14] provide a solution to this problem. Secondly, the system is not as secure as is necessary. The use of JavaScript in the choreography descriptions provides a security loophole; the system should provide a restricted JavaScript environment with a limited set of methods and reduced functionality. The messages sent during service execution should be packaged using S/MIME, to ensure non-repudiation. Thirdly, the system is not as robust as required – for example, conversations are not currently persistent objects, and hence will be lost if either party crashes. These issues require enhancements of the system, but do not invalidate the underlying design, and we are confident that they can be carried out straightforwardly.

If the system is to be deployed, it needs to be integrated with the internal workflow and decision support systems of the various service requestors and providers. Cur-

rently, these decisions are made by the user, using a bespoke user interface geared around the specific scenario described in this paper. The ideal approach to integration would be to have all internal systems re-engineered to communicate with each other (and the SWS components) using RDF data in a shared enterprise knowledge base – however, this is unlikely to be acceptable in the short term! Bespoke translation of relevant data into/out of RDF using the lifting tool would be more straightforward.

The approach followed for discovery based on semantic service descriptions works in a relatively small and closed environment, where parties refer to well defined and settled domain ontologies when specifying their service descriptions. However, it does not scale to an open environment in which parties use arbitrary ontological vocabularies that are not connected. This would require ontology mediation during discovery.

The approach adopted in this demonstrator is strongly influenced by architectural work in multi agent systems. Adept [15] was one of the first multi-agent systems to use a service agreement between provider and requestor. The role of contracts between requestors and providers has been incorporated in a semantic-web framework in the SweetDeal system [16]. Our approach to representing contracts is not as expressive as that used in SweetDeal, and it appears that the non-monotonicity they provide is not required in our domain. Trastour et. al. [8] describe the B2B lifecycle in terms of transformations of a DL contract, which has strongly influenced our approach.

Trastour et. al. [17] augment RosettaNet PIPs with partner-specific OWL constraints to determine if parties have compatible processes, and automatically propose modifications if not. This is a simple application of semantic web technology which can ease, but not automate, the linking of two business partners using RosettaNet.

Work on semantic web services provides approaches to service discovery (e.g. [4]) and generation and execution of composite services (e.g. [18]), however the majority of this work focuses on one specific part of the service lifecycle and so does not provide an integrated solution which can be applied to real problems. WSMX [19] is an exception to this, in that it provides an execution framework for semantic web services throughout the lifecycle. While promising, it does not yet provide protocol mediation capabilities or rich business-level descriptions. IRS-II [20] also supports a service lifecycle, but focuses on the composition of simple web services rather than choreography of complex business services.

8 Business Value of the System and Conclusions

If the system, enhanced as described above, were deployed in a business context it would have significant benefits. Specifically;

- By performing service discovery using detailed descriptions of capabilities, it is possible to rapidly locate many freight forwarders able to offer the service required. Because the service description is structured and detailed, it eliminates the many 'false positives' that a yellow-pages style service (or UDDI) would give. This will save a substantial amount of time, and therefore money, during the search for providers. Furthermore, it will increase the visibility of each provider, so will benefit them provided they offer a competitive service.

- By providing semi-automated assistance during the service selection phase, the system replaces a large number of phone calls with a simple selection of contract by the user. This again reduces time. Because it significantly reduces the effort needed to check out each possible freight forwarder, it allows a user to make a wider search of the options and therefore is likely to result in a better final choice.
- By allowing the service requestor to adapt its protocol to communicate with the service provider, the system dramatically reduces the time and effort required to integrate the two parties. Since integration can take months currently, this results in a very substantial cost saving. Furthermore, it means that the choice of service provider becomes far more flexible, as the time and effort of integration no longer results in lock in. This makes it easier to open up logistics chains to competitive tender where appropriate, with the resultant possibility of reducing costs further.

While the demonstrator is focussed around logistics, and so is equipped with ontologies appropriate to this domain, the software developed could be applied to other domains of B2B interaction and integration if given the appropriate ontologies and knowledge. For example, it can be used in purchasing, order management, billing and other areas of supply chain management. The protocol mediation, data mediation and discovery components are designed to be used independently of each other, and can be applied outside the domain of B2B in other semantic web service applications, providing further potential value.

In this paper, we have presented a demonstrator system using semantic web services technology which allows a requestor to discover logistics service providers, select appropriate logistics services, coordinate the services to form a composite service chain, and communicate with the service providers using arbitrary protocols through dynamic mediation. As far as we are aware, this is the first system implemented which manages the full service lifecycle of a realistic business example. The demonstrator itself is not of product quality, and needs augmenting to be more secure and robust before deployment. However, we believe our results demonstrate the feasibility of this approach to B2Bi problems in general, and expect the use of dynamic integration via semantic descriptions to become an important industrial technique in the near future.

Acknowledgements. Thanks to Zlaty Marinova, Dan Twining, Peter Radokov, Silvestre Losada, Oscar Corcho, Jorge Pérez Bolaño and Juan Miguel Gomez for work on the system implementation, and all on the SWWS project for stimulating discussions.

References

1. McIlraith, S. and Martin, D.: Bringing Semantics to Web Services. IEEE Intelligent Systems, 18(1) (2003) 90-93
2. Paolucci, M. and Sycara, K.: Autonomous Semantic Web Services. IEEE Internet Computing, (September 2003) 34-41
3. Fensel, D. and Bussler, C.: The Web Service Modeling Framework WSMF. Electronic Commerce: Research and Applications, 1 (2002) 113-117
4. Paolucci, M., Kawamura, T., Payne, T.R. and Sycara, K: Semantic Matching of Web Service Capabilities. Proc. International Semantic Web Conference (2002) 333-347

5. Trastour, D., Bartolini, C. and Gonzalez-Castillo,J.: A Semantic Web Approach to Service Description for Matchmaking of Services. In Proceedings of the Semantic Web Working Symposium, Stanford, CA, USA, July 30 - August 1, 2001
6. Grimm, S., Motik, B. and Preist, C.: Variance in eBusiness Service Discovery. Proc. of the ISWC Workshop on Semantic Web Services, 2004.
7. Preist, C.: A Conceptual Architecture for Semantic Web Services. Proc. 3rd International Semantic Web Conference (2004) 395-409
8. Trastour, D., Bartolini, C. and Preist, C.: Semantic Web Support for the B2B E-commerce pre-contractual lifecycle. Computer Networks 42(5) (August 2003) 661-673
9. Haarslev,V. and Moller, R.: Description of the RACER System and its Applications. Proc. International Workshop on Description Logics (DL-2001), Stanford, USA, 2001.
10. He, M., Jennings, N.R. and Leung, H: On Agent Mediated Electronic Commerce. IEEE Transactions on Knowledge and Data Engineering 15(4) (2003) 985-1003
11. Williams, S.K., Battle, S.A. and Esplugas Cuadrado, J.: Protocol Mediation for Adaptation in Semantic Web Services. HP Labs Technical Report HPL-2005-78
12. Esplugas Cuadrado, J., Preist, C. and Williams, S.: Integration of B2B Logistics using Semantic Web Services. Proc. Artificial Intelligence: Methodology, Systems, and Applications, 11th International Conference, (2004)
13. Battle, S.A.: Round Tripping between XML and RDF. Poster Proc. of ISWC 2004.
14. Jeff Z. Pan and Ian Horrocks. OWL-E: Extending OWL with Expressive Datatype Expressions. IMG Technical Report, School of Computer Science, the University of Manchester, April 2004
15. Jennings, N.R., Faratin, P.,Johnson,M.J., O'Brien,P. and Wiegand, M.E.: Using Intelligent Agents to Manage Business Processes. Proceedings of the First Int. Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (1996) 345-360
16. Grosof, B. and Poon, T.: 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. Trastour, D., Preist, C. and Coleman, D.: Using Semantic Web Technology to Enhance Current Business-to-Business Integration Approaches. Proc. 7th Enterprise Distributed Object Computing Conference, 2003, p222-231
18. McIlraith, S. and Son, T.C.: Adapting Golog for Composition of Semantic Web Services. Proc. 8th International Conference on Knowledge Representation and Reasoning, 2002, p482-493
19. Oren, E., Wahler, A., Schreder, B., Balaban, A., Zaremba, M., and Zaremba, M.: Demonstrating WSMX – Least Cost Supply Management. Proc. Workshop on WSMO Implementations, 2004.
20. Motta, E., Domingue, J., Cabral, L., and Gaspari, M.: IRS-II: A Framework and Infrastructure for Semantic Web Services. Proc. 2nd International Semantic Web Conference, 2003, p306-318