Business Level Service-Oriented Enterprise Application Integration

Stanislav Pokraev[1], Dick A. C. Quartel[2], Maarten W. A. Steen[1], Andreas Wombacher[2] and Manfred Reichert[2]

[1] Telematica Instituut, P. O. Box 589, 7500 AN, Enschede, The Netherlands
{stanislav.pokraev, maarten.steen}@telin.nl

[2] Center for Telematics and Information Technology (CTIT), University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
{D.A.C.Quartel, M.U.Reichert, A.Wombacher}@ewi.utwente.nl

Abstract. In this paper we propose a new approach for service-oriented enterprise application integration (EAI). Unlike current EAI solutions, which mainly focus on technological aspects, our approach allows business domain experts to get more involved in the integration process. First, we provide a technique for modeling application services at a sufficiently high level of abstraction for business experts to work with. Next, these business experts can model the orchestration as well as the information mappings that are required to achieve their integration goals. Our mediation framework then takes over and realizes the integration solution by transforming these models to existing service orchestration technology.

1 Introduction

In the networked economy the success of companies is largely determined by the speed and effectiveness with which they integrate their services into new (cross-organizational) business processes. However, almost every company has unique business processes supported by legacy systems which are difficult to integrate. Solving integration problems with traditional enterprise application integration (EAI) approaches often leads to expensive and inflexible solutions which do not always meet business requirements. Part of the problem is that integration is done at a technical level by IT specialists; e.g., by building data transformation components and customizing business process logic. Business domain experts are only consulted at the beginning, i.e., in the requirements elicitation phase.

Service-oriented architectures (SOA) provide an approach which allows integration solutions to be specified only by means of service interactions; i.e., technical details can be hidden from business domain experts. In addition, SOA solutions show a high degree of flexibility. Both service providers and service

consumers may change their software implementation; the solution will continue to work provided that both parties continue to adhere to the same service description.

To support SOA a lot of effort is currently being invested in the standardization of service description languages and protocols for service interactions such as WSDL, BPEL and WS-CDL. Unfortunately, these standards are still too technical for business domain experts to understand and to use them. Moreover, even properly educated and provided with sophisticated tools, business domain experts do not always have all information required to build integration solutions based on these technologies (e.g., port types, message formats, protocol bindings, etc.).

In this paper we describe a challenging integration scenario and show how it can be solved at a higher level of abstraction enabling the more active participation of business domain experts. The paper is organized as follows: Section 2 presents the integration scenario according to the Semantic Web Services (SWS) Challenge 2006[1]. Section 3 presents our integration approach. Section 4 sketches basic service modeling concepts needed for the further understanding of the paper. Section 5 shows how we deal with the given integration scenario in our approach. Section 6 relates our work to other research activities. Finally, Section 7 gives a summary and defines some future research directions.

2 Problem description

In order to illustrate the problem of integrating business processes we present an integration scenario according to the SWS Challenge. SWS Challenge provides a standard set of problems, based on industrial specifications and requirements.

A manufacturing company called *Moon* uses three backend systems to manage its order processing: a *Customer Relation Management System*, a *Stock Management System*, and a *Production Management System. Moon* has signed an agreement with a customer, called *Blue*, to exchange purchase order messages in *RosettaNet PIP 3A4* format. Currently, the back-end systems of *Moon* use proprietary data models and interaction protocols that differ from those of RosettaNet. The objective is to build a system, called *Mediator*, that compensates these differences and facilitates the conversation between *Moon's* and *Blue's* systems. The complete scenario is depicted in Fig. 1.

First, the *Mediator* receives a *Purchase Order Request* message from the customer *Blue*. Next, it communicates with the *Customer Relation Management system* to obtain *Moon's* internal customer ID. Then, it requests to create a new order by communicating with the *Stock Management system*. Next, all line items are submitted one by one and then the order is closed. If an article is not available the *Stock Management system* will reply that the respective line item is rejected. In this case the *Mediator* communicates with the *Production Management system* to obtain relevant information about the date and the price to manufacture the article. If this information meets the initial expectations of customer *Blue*, as specified in the RosettaNet message, the article is ordered. Finally, the mediator sends a *Purchase Order Confirmation* message back to Blue.

[1] http://sws-challenge.org/

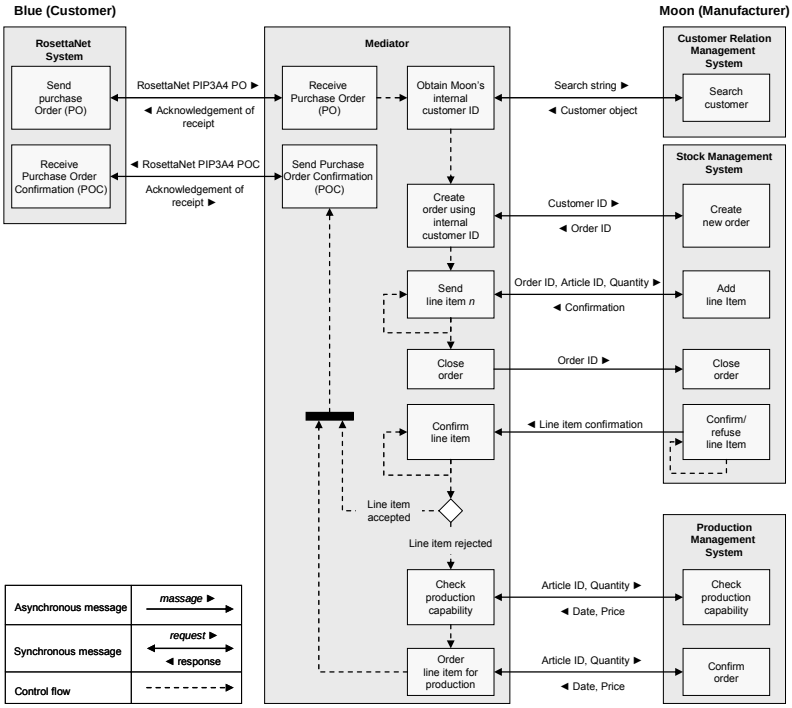

Fig. 1. Integration scenario

As one can see there are differences regarding data models and interaction protocols of Blue and Moon. We have classified respective problems into two groups – *data mismatches* and *behavior mismatches* – and have provided mediation patterns to cope with them (for details see [5][4]). This paper omits details of these mismatches and focuses on our overall integration methodology instead.

3 Integration framework

The service concept plays a central role in our approach. A *service* is a set of *related interactions* between a system and its environment that establish some *effect* that has *value* for the entities in this environment. Service orientation typically leads to a layered enterprise architecture model, where the service concept is the main link between the systems at the different layers (see [6] for details). In our approach we distinguish between the *business* and the *IT* world, and we divide these two worlds into service layers as depicted in Fig. 2:

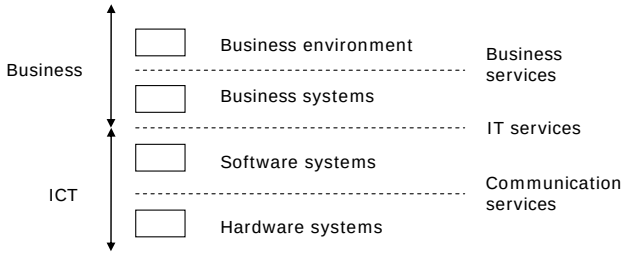

Fig. 2. Business and IT services

As mentioned, very often, integration solutions built at IT level do not always meet the business requirements of the involved companies. Part of the problem is that business domain experts do not speak the language of IT experts; usually, they are only consulted at the beginning of the project to identify requirements.

In our approach we "lift" IT service descriptions to a higher level of abstraction; i.e., closer to the business service layer, thus enabling the active participation of business domain experts. Once the integration solution is specified at the business service layer, it can be transformed (semi)automatically to an IT solution specification by applying a number of mapping rules. From this point on the IT experts can focus on the implementation of the missing parts of the integration solution. In addition, the formal link between the business and IT solution specifications enables reasoning tasks which, in turn, lead to more correct implementations of the solution. The approach is outlined in Fig. 3.

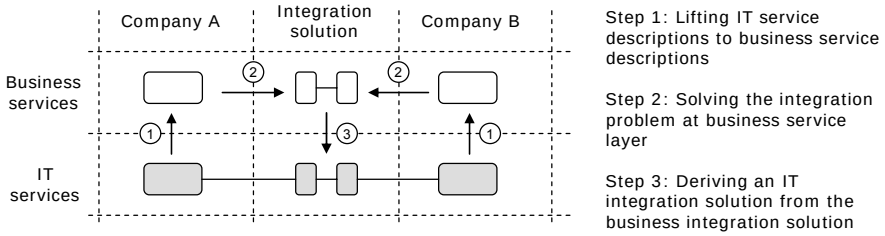

Fig. 3. Integration approach

4 Service modeling framework: basic concepts

In order to realize the sketched integration approach we have developed a framework, which provides concepts for modeling both services and integration solutions at the business level. Our framework provides modeling concepts that can be used in different application domains and at different abstraction levels. This reduces the number of required modeling concepts and thus makes it easier for business experts to apply them. The core concept of our framework constitutes the *interaction* concept, which is also the core concept in SOA. Our interaction concept supports a *constraint-oriented style* of service specification. This facilitates reasoning about the interoperability of involved systems by modeling their participation as separate constraints and by reasoning about satisfiability of the

logical conjunction of these constraints. Our framework has been presented in detail in [6]. This section summarizes the basic concepts needed for understanding the remainder of this paper.

At a high level of abstraction a service can be modeled as a single interaction between two or more systems (e.g. companies). The *interaction* represents an activity in which the involved systems produce some common result in cooperation. An interaction is defined by a composition of two or more *interaction contributions*, which represent the participation (or responsibility) of each system involved in the interaction. Consequently, an interaction is considered an atomic activity that either occurs and establishes the same result for all involved systems, or does not occur for any of the systems and therefore does not establish a (partial) result.

Fig. 4 models a sample procurement service as a single interaction between a customer and a retailer. Interaction contributions *buy* and *sell* represent the participation of the customer and retailer in this interaction, respectively. The associated text boxes define the constraints they each have on the interaction result. In this case, both the customer and the retailer want to establish an order as interaction result. The customer wants to order a notebook, whereas the retailer is willing to sell any article from its catalog. Furthermore, the customer wants the notebook to have certain properties, to be delivered within 5 days to a certain location, and to cost less than a maximum price in mind. The retailer has specified for each article a minimum price and delivery period. In addition, the retailer will only deliver articles in the region "Twente".

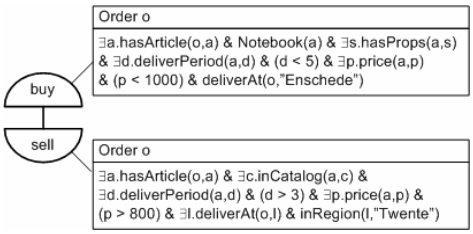

Fig. 4. Procurement interaction

In general, a service cannot be implemented as a single interaction, and we have to refine the abstract interaction into a structure of multiple smaller more concrete interactions. For representing related activities, we have introduced the *behavior* concept (graphically represented as rounded rectangles). Fig. 5 depicts a possible refinement of the procurement service from Fig. 4 into a number of interactions: *select* represents the selection of an article, *checkout* the establishment of the order comprising the selected article, *pay* the payment of the order (by credit card), and *deliver* the order delivery. In addition, the retailer offers the possibility to pay by bank transfer through the interaction contribution *pay2*. The retailer will allow credit card payments only if some *precondition* is satisfied, i.e., the price of the selected article is greater than 500. Note that the contributions *checkout*, *pay* and *pay2* refer to results established in the causally preceding contribution *select (*i.e., article and price of the article).

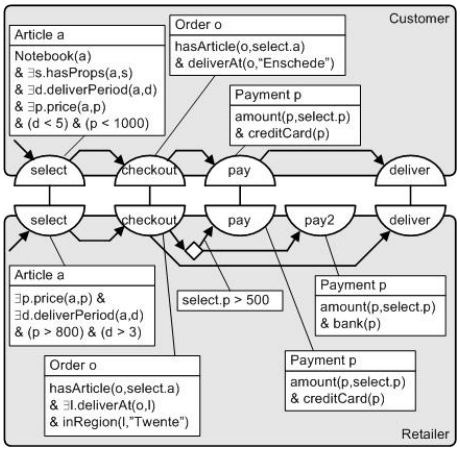

Fig. 5. Refinement of the procurement example

The effect of a service refers to elements in the subject domain of the systems involved in its execution. The subject domain of a system comprises the entities and phenomena in the real world that are *identifiable* by the system. For capturing these entities we use an *information model*. It consists of *individuals* that represent the entities and phenomena from the subject domain, *classes* that represent the types of the entities and phenomena and *properties* that represent the possible relations between individuals (*object properties*) or between an individual and a data value (*data properties*) . Further, we allow classes and properties to be defined as logical conjunctions, disjunctions or negations involving other classes and properties.

Fig. 6 depicts part of a simple information model for our procurement example. This model does not include individuals and the valuations of their data properties, which together we denote as the *state* of a system.

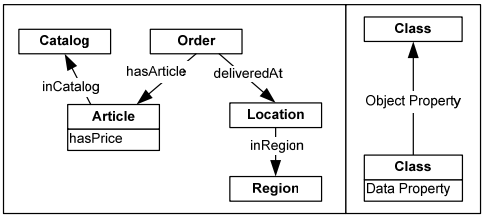

Fig. 6. Procurement information model

5 Solving the integration scenario

In this section we show how the approach described in Section 3 can be used to solve the integration problem described in Section 2. We assume the systems to be integrated all expose Web service interfaces and that we have a BPEL orchestration engine with XSLT support at our disposal to realize the integration at the IT level.

First, we derive the information and behavior models of the Blue and Moon systems using the WSDL descriptions of their services. Next, we define mappings between the classes and properties from the information models, and we define the integrated process model (the Mediator system). Once we have done all this we transform the Mediator specification into an executable BPEL specification and deploy it onto the BPEL orchestration engine. The concrete steps to be taken are illustrated in Fig. 7 and described in more detail in the remainder of this section. The left picture in Fig. 7 concerns the information models, and the right picture concerns the behavior models of Blue's and Moon's system.

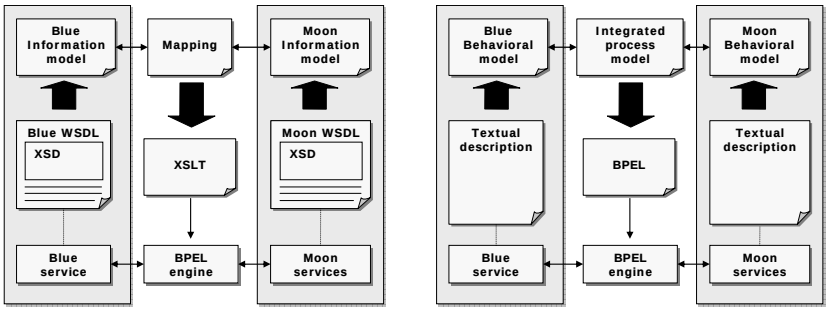

Fig. 7. Solving the integration problem from Section 2

5.1 Step 1a: Lifting data models to information models

In this step we use the XML schemas which define the message types of Blue's and Moon's services to derive information models. This is done by applying a number of rules including:

- xsd:element with at least one sub-element or attribute is transformed into a class
- xsd:element with neither sub-elements nor atributes is transformed into a property
- xsd:complexType is transformed into a class and xsd:simpleType into a property
- an XSD inheritance by extension is transformed to a subClassOf property
- an XSD inheritance by restriction is transformed to a class, defined by restricting the range of some Properties to a particular set of values
- xsd:sequence and xsd:all are transformed into a logical conjunction of two or more classes
- xsd:choice is transformed into a class defined as an expression containing logical conjunctions or disjunctions and negations of two or more classes

An XML schema defines only the *syntax* of the messages. Thus, some further work is required to define the *semantics* of their elements. For example, some hidden assumptions are made more explicit in the information models (e.g., by defining new classes and relations among them), or classes and properties are mapped to classes and properties from domain specific ontologies, thus defining their meaning. This is usually a manual process which requires domain specific

knowledge. Fig. 8 shows part of the information models of Blue and Moon that are relevant to understand our approach.

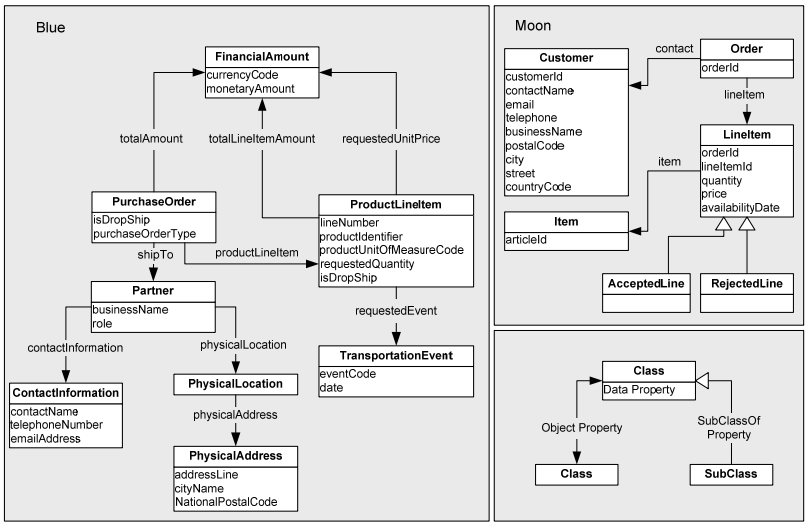

Fig. 8. Information models of Blue and Moon

5.2 Step 1b: Lifting interface descriptions to service behaviors

In this step we transform the WSDL descriptions that define the possible message exchanges between Blue's and Moon's services into behavior models. These behavior models are constructed from the concepts explained in Section 4.

A WSDL description defines the operations that can be invoked on some server that implements the service, and in this way, the types of messages that can be accepted and returned by the server. This description only concerns the involvement of the server. To model some service behavior completely we also want to model the involvement of the client, i.e., the sending of the request message and the acceptance of the reply message.

We model a two-way operation as a sequence of two instances of the interaction concept. The first instance models the operation invocation, and consists of two interaction contributions: one modeling the sending of the request message by the client and the other modeling the reception of the message by the server. The second instance models the operation return, which similar to the first interaction models the exchange of the reply message. In case of a one-way operation, the second interaction is absent.

A complete behavior model should also define the relationships between the executions of distinct operations. These relationships are not part of the WSDL descriptions, but have to be derived from informal textual descriptions as provided in Section 2. An example of how such relationships are modeled is given in Section 4.

For the purpose of structuring, distinct service or interface descriptions can be modeled by distinct instances of the behavior concept

5.3 Step 2a: Mapping information models

In this step we define mappings between classes, properties and individuals from Blue's and Moon's information models. We consider four types of mappings: *equivalence* - defines that two classes or properties have the *same* meaning; *more general* - defines that a class (or a property) has *more general* meaning than another class (or property); *less general* - defines that a class (or a property) has *more specific* meaning than another class (or property); *disjointness* - defines that two classes (or properties) have *different* meanings.

Creating mappings requires understanding of the meaning of the classes and properties in the information models of the systems being integrated and cannot be fully automated. However, tools and techniques exist that use sophisticated heuristic algorithms to discover possible mappings and propose these to the business domain experts. A good survey of semi-automatic schema matching is presented in [7]. Fig. 9 shows the mappings between Blue's and Moon's information models. In the figure we only show the equivalence relations.

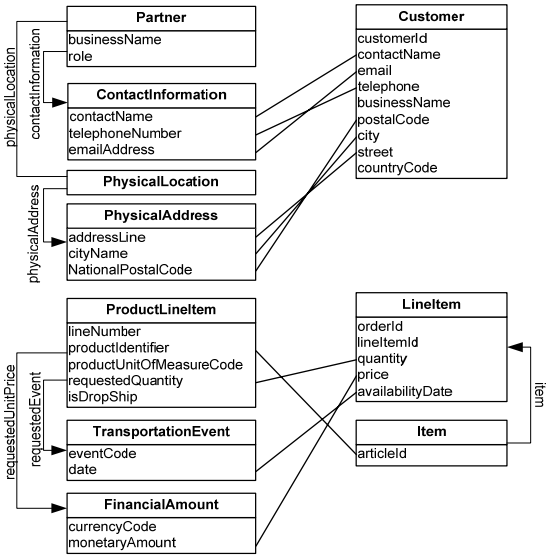

Fig. 9. Mapping Blue's and Moon's information models

5.4 Step 2b: Specifying the integrated behavior

In this step we define a *mediator*, which compensates the mismatches between Blue's and Moon's behavior models. This step can be performed in a manual (e.g.

using a modeling tool such as Grizzle[2]) or automated way. The model of the mediator is shown in Fig. 10.

The behavioral mismatches can be associated with specific solutions for each occurrence of a mismatch. Examples of such mismatches include different order of source and target messages, a source message that contains two or more target messages, etc. Based on these partial solutions the aim is to select only the relevant ones and compose them automatically to form the behavior of the mediator. In our approach we consider the partial solutions as little workflows and use them to compose more complex workflows. The composition is recursively applied until the composed workflow solves the mediation problem. We are currently implementing this approach based on annotated Finite State Automata [8].

Automated composition faces the challenge that the number of possible combinations is not constrained by a maximum number of possible compositions. Further, there is no constraint which enforces that a particular partial solution is applied only once. Therefore, the number of possible compositions of workflows explodes. Further, each composition of workflows increases the complexity of the resulting model significantly.

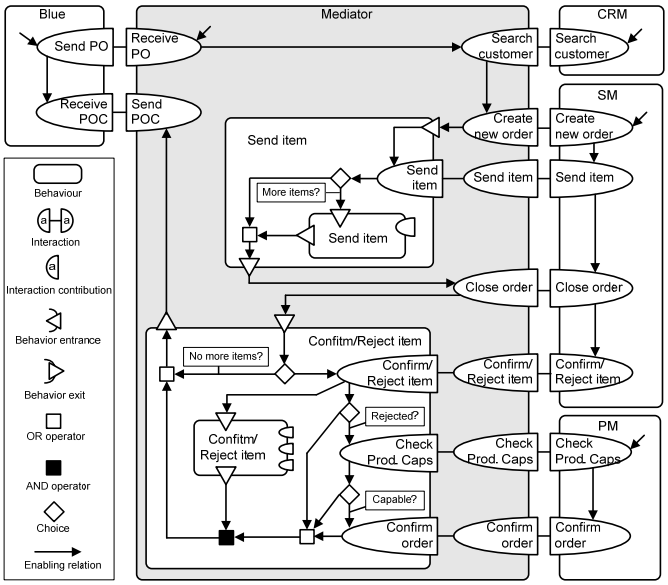

Fig. 10. The model of the mediator

5.5 Step 3a: Transforming information mappings to XSLT

In this step we use the information mapping defined in Section 5.3 to derive executable XSLT specifications for the exchanged messages. These XSLT

[2] http://isdl.ctit.utwente.nl/tools/grizzle/index.php

specifications are applied at runtime to transform messages sent by one system (source) to messages accepted by another system (target). The derivation of message transformation specifications is done by searching for elements in the source message that can be used to produce each element of the target message. The simplest scenario is as follows: If an element E_T from the target message corresponds to a property P_T in the information model of the target system and an element E_S from the source message corresponds to a property P_S in the information model of the source system and the mapping $P_R \equiv P_S$ is defined, then the value of E_S in the source message will be copied (using <xsl:copy>) as value of E_T in the target message.

In a more complex scenario, a number of properties can be composed to define a new, composite property. For instance, we can define a new property partnerCity in the information model of Blue as composition of the properties physicalLocation, physicalAddress and cityName and define a mapping partnerCity ≡ city from the information model of Moon. This mapping is used at runtime to copy the value of cityName as value of city only when cityName has an indirect relation (i.e., via physicalLocation and physicalAddress) to an individual of class Partner.

5.6 Step 3b: Transforming the Mediator's behavior to BPEL

In this step the behavior model of the mediator is transformed into a BPEL specification. In [2] a mapping has been defined between the behavior concepts presented in Section 4 and the BPEL language concepts. However, before this mapping can be applied a preparatory step is needed in which the behavior model of the mediator is annotated with marks and possibly restructured.

Marks are used to add implementation details, in this case BPEL specific information. For example, interaction contributions should be marked to indicate whether they have to be mapped onto an invoke, receive or reply activity in BPEL. Furthermore, information about partner links and invoked web services (e.g., namespace URI and endpoint address) may have to be provided.

Restructuring of the behavior model may become necessary to enable its mapping onto structured activities. For this purpose, separate behavior blocks can be used and marked to represent, for example, flow, switch, while, pick, scope and handler activities.

6 Related work

OWL-S[3] is an OWL ontology for describing services for the purpose of discovery, composition and delivery. In OWL-S a service is formally described by a *Service Model* that defines the steps required to execute a service. It describes a service in terms of its *inputs*, *preconditions*, *outputs*, *effects*, and, where appropriate, its components. The Service Model also describes the control flow in terms of the service's *state*, including initial activation, execution and completion.

The main difference between OWL-S and our work is that our conceptual framework allows for constraint-oriented service specifications while OWL-S enables only an imperative, and therefore prescriptive, specification style. In addition, OWL-S takes only the perspective of the service provider into

consideration, whereas we treat both participants in a service interaction equally. Our approach allows service requestors and providers to *explicitly* specify their assumptions about the environment of their systems. This in turn allows business domain experts to check whether their integration solutions satisfy these constraints and meet the requirements of both the service requestors and providers.

The Web Service Modeling Ontology (WSMO[1]) is a formal ontology for describing several aspects of Semantic Web Services. It consists of four main components – *ontologies*, *goals*, *web services* and *mediators*. Ontologies provide terminology and formal semantics of information that is used by the other components. A goal is a specification of the objectives of a service user. A web service is a specification of the functionality of the service provider. Mediators are used as connectors between ontologies, goals and web services.

The main difference between WSMO and our work is that our framework has less concepts while providing comparable expressive power. This makes it easier for business domain experts to learn and use. Furthermore, we feel that the behavioral semantics of WSMO choreographies and orchestrations are rather weakly specified.

7 Conclusion and Future Work

In this paper we presented a new approach for service oriented EAI. In our approach we used a novel service modeling framework that is domain-independent and provides concepts that can be applied at different abstraction levels enabling the more active participation of domain experts in designing the integration solution. The key concept in our framework (the interaction concept) supports *constraint-oriented style* of service specification. This makes our framework especially suitable for desiging integration solutions because the service requestors and providers can *explicitly* specify their assumptions about the environment of their systems as constraints and the system integrators can check if their solutions satisfy these constraints.

Our forthcoming work will focus on further validation of our approach in practice. In addition, we want to investigate the possibilities to extend existing business process integration tools such as Microsoft BizTalk, Oracle BPEL manager, and the IBM WebSphere Process Server to support our approach.

Acknowledgements

The presented work has been done in the Freeband Communication project A-Muse (a-muse.freeband.nl). Freeband Communication (www.freeband.nl) is sponsored by the Dutch government under contract BSIK 03025.

8 References

[1] Bruijn, J. de, et al. Web Service Modeling Ontology (WSMO), W3C Member Submission 3 June 2005, http://www.w3.org/Submission/WSMO.

[2] Dirgahayu, T. Model-Driven Engineering of Web Service Compositions: A Transformation from ISDL to BPEL. M.Sc. thesis, University of Twente, The Netherlands, July, 2005.
[3] Martin, D., et al. OWL-S: Semantic Markup for Web Services W3C Member Submission 22 November 2004, http://www.w3.org/Submission/OWL-S.
[4] Pokraev, S., Reichert, M. (2006). Mediation Patterns for Message Exchange Protocols. Open INTEROP-Workshop on Enterprise Modeling and Ontologies for Interoperability (EMOI06). In: Proc. CAiSE'06 Workshops, Luxembourg, June 2006, pp.659-663.
[5] Pokraev, S., Reichert, M., Steen, M.W.A., Wieringa, R.J. (2005). Semantic and Pragmatic Interoperability: A Model for Understanding. In: J. Castro, E. Teniente (Eds.) Proc. of the CAiSE'05 Workshops, Porto, Portugal, June 2005, pp. 377-382.
[6] Quartel, D. Steen, M.W.A., Pokraev, S. and van Sinderen, M. A Conceptual Framework for Service Modelling. In Proceedings of the Tenth IEEE International EDOC Enterprise Computing Conference, Hong Kong, China, pp. 319-330.
[7] Rahm, E. and Bernstein, P. A survey of approaches to automatic schema matching. VLDB journal, (10(4)):334-350, 2001.
[8] Wombacher, A., Fankhauser, P., Mahleko, B. and Neuhold, E. Matchmaking for Business Processes Based on Choreographies, Intl. Journal of Web Services, (1(4)): 14-32, 2004