Model-driven Service Integration using the COSMO framework[1]

Dick A.C. Quartel[1], Stanislav Pokraev[1], Teduh Dirgahayu[2], Rodrigo Mantovaneli Pessoa[2] and Marten van Sinderen[2f]
[1] Telematica Instituut, P. O. Box 589, 7500 AN, Enschede, The Netherlands,
{Dick.Quartel, Stanislav.Pokraev}@telin.nl
[2] Center for Telematics and Information Technology (CTIT), University of Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands
{T.Dirgahayu, MantovaneliR, M.J.vanSinderen}@ewi.utwente.nl

Abstract. In this paper, we describe an approach for solving the integration problem in the Purchase Order Mediation scenario of the Semantic Web Service Challenge[2]. The key feature of our approach is that service models are employed at different abstraction levels to develop end-to-end integration solutions from business requirements to software implementation.

Keywords: Service mediation, MDA, model transformations.

1 Introduction

The goal of the SWS Challenge is to explore the trade-offs of various existing technologies that aim at automation of mediation, choreography and discovery of Web Services. For that reason, the SWS Challenge defines a number of scenarios providing a standard set of problems, based on industrial specifications and requirements.

In this paper, we present a model-driven approach for solving the integration problem described in the Purchase Order Mediation scenario of the SWS Challenge. Model-driven techniques are used to abstract the integration problem and solution to a higher (platform-independent) level. This way, the problem and solution can be captured in a technology independent manner enabling more active participation of business domain experts.

This paper is structured as follows: Section 2 briefly presents our integration framework. Section 3 shows how we use the framework to solve the integration problem from the Purchase Order Mediation scenario. Section 4 compares our work with the solutions provided by the other SWS Challenge participants. Finally, section 5 presents our conclusions and future work.

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

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

2 Integration Framework

We approach the design of a mediator as a *composition* problem: each service that is requested by some system has to be composed from one or more services that are provided by one or more other systems. Fig. 1 illustrates this for the case of two systems A and B. Mediator M offers a mediation service that matches requested service S1 of A by composing services S3 and S4 that are offered by B. The mediator should provide such a mediation service for each service that is requested by A (and B).

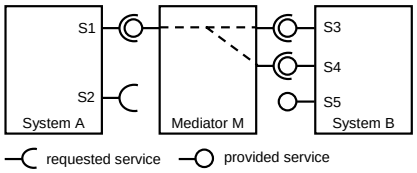

Fig. 1. Service mediation as service composition.

To support the design, implementation and verification of mediators we have developed an integration method. Our method uses the COSMO framework [11] to model and reason about services. The method defines a number of steps to build end-to-end integration solutions and to verify their correctness. In the following sections, we briefly describe the COSMO framework and the steps of our method.

2.1 The COSMO framework

We define a *service* as "*the establishment of some effect (or value) through the interaction between two or more systems*". Based on this definition, the COnceptual Service MOdeling (COSMO) framework defines concepts to support the modeling, reasoning and analysis of services.

We distinguish four service aspects, i.e., *information*, *behavior*, *structure* and *quality*, representing categories of service properties that need to be modeled. The *structure* aspect is concerned with modeling the *systems* that provide or use services, and their *interconnection structure*. The interconnection structure comprises (amongst others) the *interfaces* at which services are offered. The *behavioral* aspect is concerned with the *activities* that are performed by systems as well as the *relations* among these activities. The *information* aspect is concerned with modeling the information that is managed by and exchanged among systems. The *quality* aspect is concerned with modeling the non-functional characteristics of services. These qualities often play an important role in the selection of services. Examples of quality aspects are the "cost" associated with a service or the "response time" of a service.

Besides service aspects, we distinguish three generic abstraction levels at which a service can be modeled, namely, *goal*, *choreography* and *orchestration* level. A model at goal level describes a service as a single interaction, where the interaction result represents the effect of the service as a whole. A model at *choreography* level refines the model at goal level by describing the service as a set of multiple related, more concrete interactions. A model at *orchestration* level describes the

implementation of the service using a central coordinator that invokes and adds value to one or more other services.

Finally, we distinguish different roles of the systems involved in a service: the *user*, *provider* and *integrated role*. The integrated role abstracts from the distinction between a user and provider by considering interactions as joint actions, thereby focusing on what the user and provider have in common.

This paper mainly considers choreographies and orchestrations from the behavior and information aspect, and by distinguishing between a user and provider role. Furthermore, services are modeled close to the level at which they are described using WSDL, while abstracting from technology details. Therefore, and for brevity, we only explain COSMO's *operation* concept below and its notation using ISDL [10]. For an overview and explanation of the COSMO concepts, we refer to [11].

Fig. 2 (i) and (ii) depict the operation concept and its interpretation in terms of a flow chart-like notation, respectively. An operation represents a composition of three instances of message passing: the sending (invoke) and receipt (accept) of an invocation, followed by either the sending (reply) and receipt (return) of the invocation result, or the sending (fault) and receipt (catch) of a fault message. The use of the reply-return and the fail-catch message passing instances are optional, i.e., either one or both parts may be omitted; e.g., to model one-way operations.

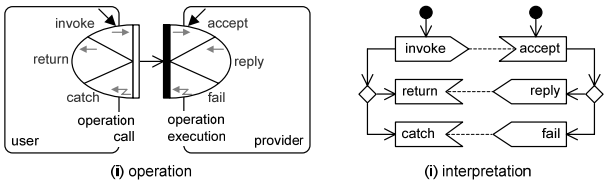

Fig. 2. The operation concept.

2.2 Integration method

The steps of our integration method are depicted in Fig 3. For the sake of readability, we consider two systems, but the same steps apply to the case of multiple systems.

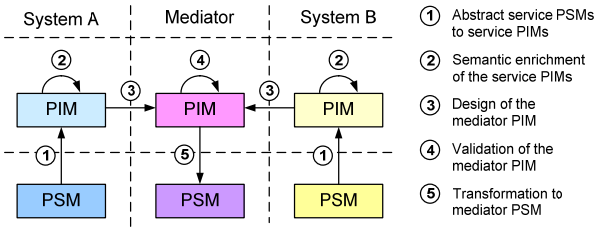

Fig. 3. The steps of the integration method

In Step 1 of our method, we derive the platform-independent models (PIMs) of the services to be integrated by abstracting from all technical details in the platform specific models (PSMs). Next, in Step 2 we increase the coverage and precision of the PIMs by adding semantic information that cannot be derived from the original service

descriptions (PSMs). In Step 3, we solve the integration problem at PIM level, which enables the more active participation of domain experts. In addition, the abstract nature of the integration solution allows one to reuse it for different implementation technologies. Besides, the semantically enriched service models allow some integration tasks to be fully or partially automated. Next, in Step 4 we verify the correctness of the integration solution using various analysis techniques. Finally, in Step 5 the service PIM is transformed to a PSM solution by mapping the integration solution to a specific service computing platform.

3 Application of the integration framework

This section presents the application of our framework to the Purchase Order Mediation scenario of the SWS Challenge. For this purpose, the integration method is made concrete by deciding on, amongst others, the type of PSMs that are considered, the languages to be used at PIM level, and related to these choices the transformations and analysis techniques that are needed, c.q. have to be developed.

Step 1: Abstract service PSMs to Service PIMs. In the first step, we derive the platform independent information and behavior models of the services of Blue and Moon, which are specified in WSDL. ISDL [10] is used to represent the service behavior, and UML class diagrams are used to represent the information models.

This step is automated using the WSDL import function of the Grizzle tool [4]. Grizzle is an integrated editor and simulator for ISDL, and uses Java to represent and execute operation parameter constraints. Once a WSDL document is imported, a behavior model is generated that represents the user or provider role of the web service, in terms of operation calls or operation executions, respectively. In addition, an information model is generated consisting of Java classes that represent the information types that are refered to by the operations in the behavior model. The transformation from WSDL to ISDL and Java is implemented using JAXB and JAX-WS [5]. We use an EclipseUML tool [3] to represent (and manipulate) the information model using UML class diagrams.

Step 2: Semantic enrichment of PIMs. The WSDL descriptions of the example scenario define the services that are provided by Blue, Moon and the Mediator, in terms of their operations and the types of the input and output messages of these operations. However, WSDL does not define the interaction protocols of the involved systems, i.e., the possible orderings of the operations. Therefore, to derive the complete PIMs of Moon and Blue, we have to use and interpret the provided textual descriptions. This is a manual process.

Firstly, the behavior models that were generated in Step 1 are completed by defining relations between operations. These relations can be derived from the scenario description. This includes the explicit modeling of the repetitive process of adding and confirming line items. Fig. 4 depicts the enriched model of the service requested by Blue and the service provided by Moon OM.

Secondly, the information model may be enriched by interpreting the scenario description. A WSDL description defines the syntax of the messages that are exchanged, but does not provide information about their semantics. This semantics can be made explicit by defining new classes, properties and relations among classes.

Furthermore, the meaning of classes and their properties may be defined by a mapping onto some domain-specific ontology, e.g., the Universal Data Element Framework [15]. The benefits of these types of semantic enrichment can however, only be fully exploited when using an ontology language (such as OWL [16]) that allows one to explicitly model and reason about the semantics of classes and their properties. The semantic analysis of information models is considered in the next version of our integration method.

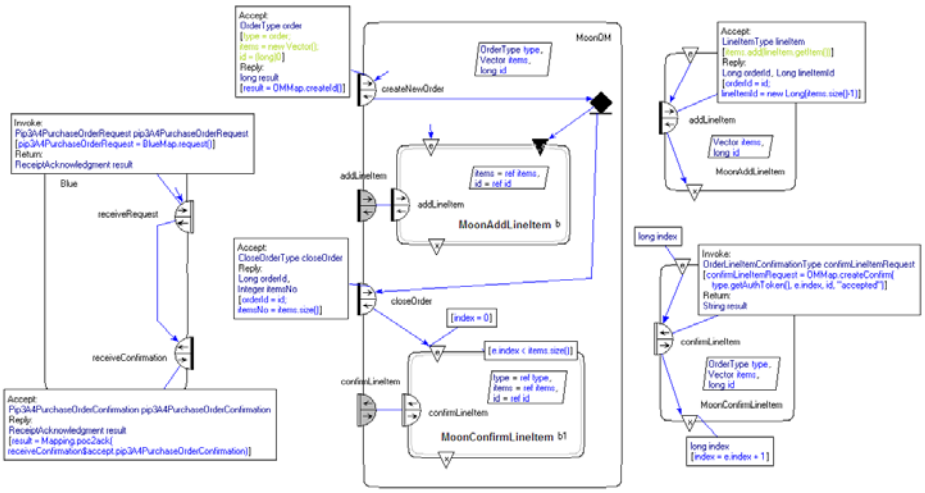

Fig. 4. Enriched behavior models of Blue and Moon OM

Step 3: Design of the mediator PIM. In this step, we design the behavior and information model of the Mediator. The information model of the Mediator is constructed from the union of the information models of Blue and Moon. For the same reason as explained at the end of the previous section, this information model is not enriched to define the relationships between the classes and properties from the information models of Blue and Moon, except for informal annotations that may explain these relationships using natural language. The information model is extended, however, with classes to represent status information of the Mediator, such as the set of order line items that have been confirmed so far.

The construction of the behavior model of the Mediator requires the definition of (i) the services provided and requested by the Mediator, (ii) the composition of these services by relating the operations of the services, and (iii) the data transformations among the parameters of the operations.

In the example scenario, the Mediator provides one service that must match the service requested by Blue. The service provided by the Mediator can initially be defined as the 'complement' of the service requested by Blue. The complement of a service is obtained by changing each operation call into an operation execution, and vice versa, while keeping the same parameters. In addition, the relations among the operations and the parameter constraints may (initially) be retained. Likewise, the services that are requested by the Mediator can be obtained by taking the complement of the services that are provided by Moon. These retained relations and parameter constraints may be refined in the next design steps, respectively. For example, the

relation between operations receiveRequest and receiveConfirmation has to be implemented by the orchestration of the services of Moon. As another example, the disabling relation (represented by the black diamond on top of a horizontal bar in Fig. 4) between addLineItem and closeOrder will be replaced by an enabling relation, since the order should be closed only after all line items have been added.

The design of the Mediator behavior can now be approached as the search for a composition of the requested services that conforms to the provided service. The structure of this composition is defined by the (causal) relations among the operations. Most of these relations can be found by matching the input that is required by each operation to the output that is produced by other operations. For example, operation search of Moon's CRM service requires as input a search string that can be matched to some element of the customer information that is part of the purchase order information received by operation receiveRequest. This implies that a relation should be defined between receiveRequest and search. Fig. 5 depicts the design.

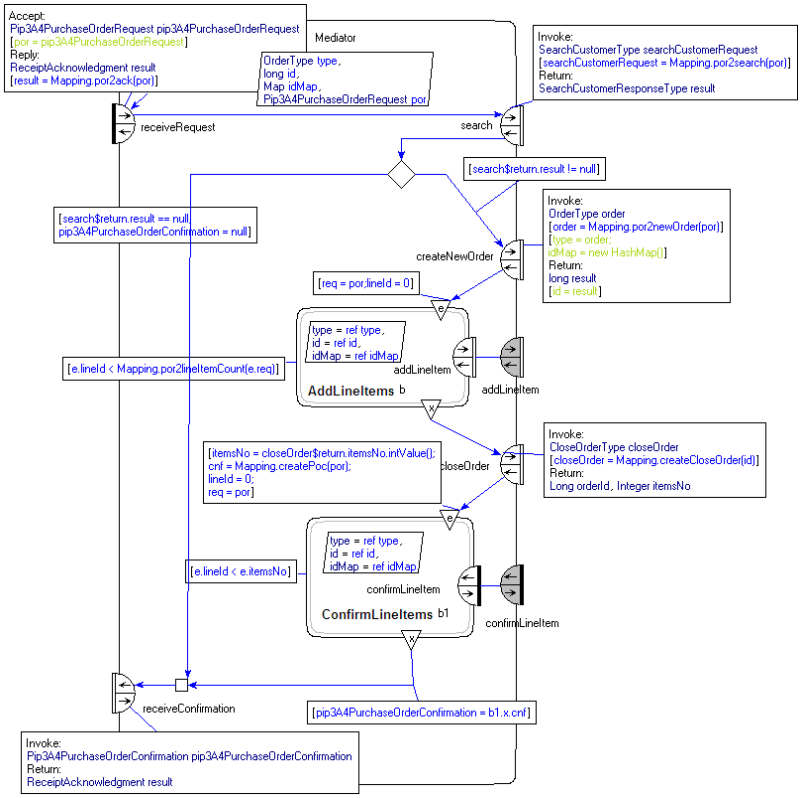

Fig. 5. Design of the mediator

Matching inputs and outputs is however insufficient to find all relations. For example, although operation receiveRequest and operation search provide information that matches the input required by operation createNewOrder, the information that is provided by receiveRequest should be used instead. This hidden assumption has to be

made explicit in the behavior model. Furthermore, specific processing logic may have to be designed manually. For example, the process of receiving confirmations from Moon's OM system depends on information from operations receiveRequest (the items to be confirmed), createNewOrder (the order id) and addLineItem (the item id used by Moon), and depends on internal status information of the Mediator, i.e., the knowledge that operation closeOrder has occurred and the set of confirmations that has been received so far. Even when these information requirements are given, the relations involved in the repetitive processing of confirmations can not be derived easily, and have to be designed explicitly.

The definition of the data transformations among operation parameters can be approached as a refinement of the relations among operations defined in the preceding step. These relations define for each operation on which other operations it depends, and therefore which output parameters can be used in the generation of its input parameters. The data transformations then define how the value of each input parameter is generated from the values of the output parameters and, in some cases, some internal state information of the Mediator. This involves the definition of translations between the information models of Blue and Moon. However, these translations only need to address those parts of the information models of Blue and Moon that are related via the relations defined in Step 2.

To express data transformation functions we have defined a *Domain-Specific Language* (DSL) using the Eclipse TCS [14]. In our DSL, a data *transformation* is specified as a set of *relations* among two or more objects. In a *from* clause a number of variables are bound by evaluating queries on a source object. In a *create* clause a new (target) object is created and its properties are set to the values of the variables bound in the *from* clauses. Likewise, an *update* clause takes an existing object and only sets its properties. Optionally, a *conditions* clause defines when a relation can be executed. An example of a data transformation definition is shown below.

```
transformation Blue2Moon {
  relation POR2LineItemType (lineItem: LineItem, por:Pip3A4PurchaseOrderRequest, index:int) {
    from por {
      orderId   = orderId;
      articleId = purchaseOrder/productLineItem[index]/productIdentification/globalProductIdentifier;
      quantity  = purchaseOrder/productLineItem[index]/orderQuantity/requestedQuantity/productQuantity;
    }
    create lineItem {
      orderId   = orderId;
      articleId = item/articleId;
      quantity  = item/quantity;
    }
  }
}
```

Once all relations are defined, we automatically generate a Java class Mapping that implements the actual data transformations. For that purpose, we use OpenArchitectureWare [8]. For example, the relation between operations receiveRequest and search has been implemented by the method por2search() as described in the text box associated with operation search. This method gets as argument the value of behavior item (variable) Pip3A4PurchaseOrderRequest por. This value is assigned after operation receiveRequest has received the purchase order request from Blue.

Step 4: Validation of the mediator PIM. In this step, the design of the Mediator is validated by means of (i) *assessment of the interoperability* between the services of Blue, the Mediator and Moon, and (ii) *simulation* of the interacting behavior of these services. The interoperability assessment method has been presented in [9]. In short, the method checks whether each individual interaction can establish a result and whether the service composition as a whole can establish a result.

The simulation of behaviors is supported by the Grizzle tool [4]. Simulation allows a designer to analyze the possible orderings of operations occurrences, as well as the information results that are established in these operations. In addition, the simulator provides hooks in the simulation process to execute application code upon execution of an operation. This enables us to perform real web service invocations and incorporate the results that are returned by web services during the simulation. For this purpose, stub-code is linked to a modeled web-service operation *call*. This code is generated automatically based on stereotype information that has been retained during the WSDL import, such as the web service's end-point address and port type name. Furthermore, the simulator allows external web-clients to invoke a modeled web-service operation *execution*. A web service proxy is automatically generated and deployed in an application server, using aforementioned stereotype information. This proxy is responsible for handling the reception of the invocation request and the return of its result. In between, the proxy delegates the calculation of the invocation result to the simulator, which indicates to the user that the operation is enabled and waits till the user requests the simulation of this operation.

The support for real, also called 'live', web service invocations, allows one to use the simulator as an orchestration engine in which an orchestration can be executed by simulating its ISDL model. This means that that the simulator provides, in principle, an implementation for the Mediator. However, this simulator does not support important properties of an execution environment, such as performance, monitoring, etc. Therefore, in the next step we transform the Mediator design to a BPEL process.

Step 5: Derivation of the mediator PSM. In this final step the service PIM of the mediator is transformed into a PSM. In our approach, we do not assume a particular execution platform. For example, the service PIM can be transformed to a WS-BPEL specification, EJB, or .Net application. In this section, we present an abstract architecture of possible execution platforms. Fig. 6 depicts this architecture.

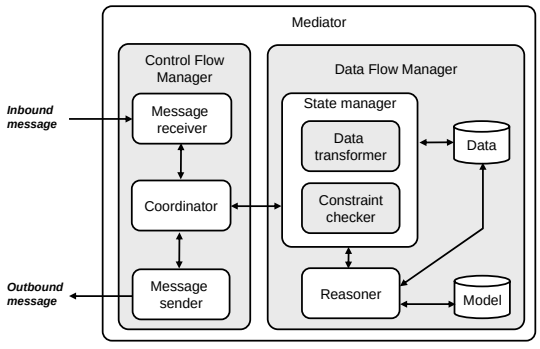

Fig. 6. The architecture of the abstract execution platform

The abstract architecture of the Mediator consists of two main components: a *Control Flow Manager* and a *Data Flow Manager*. The *Control Flow Manager* is responsible for sending and receiving messages in a particular order as well as for querying and updating the state of the Mediator. The *Data Flow Manager* in turn, is responsible for managing the state of the Mediator and for performing the necessary data transformations and constraint checking.

The *Control Flow Manager* consists of three subcomponents: a *Message receiver*, a *Message sender* and a *Coordinator*. The *Message receiver* is responsible for receiving all inbound messages and the *Message sender* for sending all outbound messages. The *Coordinator* executes the behavior specified in the behavioral model of the Mediator, i.e., based on the current state it activates and deactivates the *Message receiver* and *Message sender*. When a message is received, the *Coordinator* interacts with the *Data Flow Manager* to update the state of the Mediator. When a message is to be sent, the *Coordinator* interacts with the *Data Flow Manager* to obtain the data required to construct the outbound message.

To derive the *Control Flow Manager* we adopt and extend the approach described in [2]. Our transformation is divided into three successive steps: *pattern recognition*, *activity replacement* and *model realization*. Fig. 7 depicts these steps.

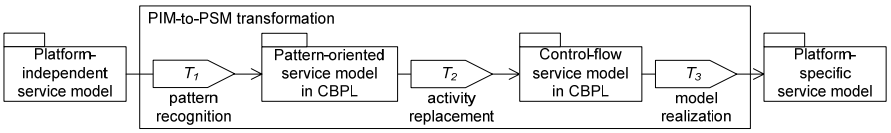

Fig. 7. Transforming the service PIM of the mediator to a service PSM

The first step recognizes the control flows in a service PIM and then transforms the service PIM to a pattern-oriented service model in the *Common Behavioral Patterns Language* (CBPL). Each CBPL pattern represents a control flow that is common to most execution languages, i.e., *sequence*, *concurrence*, *selection* and *iteration*. A *sequence* contains one or more activities to be executed in succession. A *concurrence* contains two or more activities that can be executed independently. A *selection* contains one or more cases to be selected, where a case contains an activity to be executed when its condition holds. An *iteration* contains an activity to be executed repeatedly as long as its condition holds.

The second step replaces data transformations and constraint checking in the pattern-oriented service model with operations for interacting with the *Data Flow Manager*. This steps results in a control-flow service model that represents the *Control Flow Manager* in CBPL.

The last step maps the control-flow service model onto a service PSM. A service PSM contains information that is not present in the service PIM. Examples of such information are the XML namespaces of the exchanged messages or the WSDL port types and operations of the services to be integrated. To provide the required platform-specific information we annotate the elements of the service PIM. This information is maintained during the first and second steps and is used in the last step.

The *Data Flow Manager* consists of two components: a *State manager* and an optional *Reasoner*. The *State manager* is responsible for updating the state of the Mediator (after receiving a message) and for querying that state (before sending a

message or when checking a constraint). In some cases, data in the received message may have to be transformed before updating the state. For that purpose, the *State manager* uses the *Data transformer* component. Likewise, in some cases the *State manager* uses the *Data transformer* to construct new messages. The *Data transformer* is in fact the component that implements the *mapping relations* specified in the information model of the Mediator. The *Constraint checker* queries the state of the mediator and determines whether a constraint holds or not.

To take full advantage of the formal specification of the information model of the Mediator, the *Data Flow Manager* may contain a *Reasoner* component. The *Reasoner* uses the formal knowledge specified in the information model of the Mediator in conjunction with the facts about the current state of the Mediator to infer new state information, i.e., it makes all implicit knowledge about the state more explicit. In addition, the *Reasoner* can be used by the *Data transformer* and the *Constraint checker* as an intelligent query engine and constraint solver.

In our solution, we use the ActiveBPEL engine to realize the *Control Flow Manager* and an external web service to realize the *Data Flow Manager*. The ActiveBPEL engine executes the WS-BPEL specification generated from the PIM model of the Mediator. The web service stores and retrieves data, exchanged in the messages between Blue, Moon and the Mediator, and performs the data transformations defined in Step 3.

4 Related approaches

Several approaches and solutions have been proposed within the SWS Challenge. Here we briefly discuss the approaches reported at the past edition of the workshop held in Tenerife, Spain. The proposed approaches were based on the WSMO, jABC and FOKUS frameworks. For a more detailed comparison please refer to [6].

The DERI approach [12] follows the Web Services Modelling Ontology (WSMO) framework. It consists of four main components – ontologies, goals, web services and mediators. Data mediation is achieved through the design and implementation of adapters specifying mapping rules between ontologies. During runtime, the approach considers specific mediator services which perform data transformations at entity instance level. The mediator interaction behavior is described by means of Abstract State Machines, consisting of states and guarded transitions. A state is described within an ontology and the guarded transitions are used to express changes of states by means of transition rules. However, this implicit behavior specification may be neither intuitive nor trivial to make sure that the expectations implied by the designed transition rules match the expected operation message exchange patterns.

The jABC solution [13] uses SLGs (Service Logic Graphs) as choreography models, allowing the designer to model the mediator in a graphical high level modeling language by combining reusable building blocks into (flow-)graph structures. These basic building blocks are called SIBs (Service Independent Building Blocks) and have one or more edges (branches), which depend on the different outcomes of the execution of the functionality represented by the SIB. The provided model driven design tools allow the modeling of the mediator in a graphical high level modeling language and support the derivation of an executable mediator from

these models. More recently [7], the approach has focused on how to apply a tableau-based software composition technique to automatically generate the mediator's interaction behavior. This uses a LTL (Linear Time Logic) planning algorithm originally embedded in the jABC platform. However, the applicability of automated synthesis of the mediator's business logic is still limited considering the kind of assumptions being made. In comparison with the jABC approach, the approach presented in this paper does not cover automated synthesis of the mediator logic as it intentionally leaves the planning task to the business domain expert.

The core concept of the FOKUS [1] approach is the integration of ontology mappings into BPEL processes. The approach addresses the data mediation by applying semantic bridges to mediate between different information models and representations. Semantic bridges are described as a set of description logic-based axioms relating entities in business information models that are defined in different ontologies but have a similar meaning. The description logic-based data model provided by ontologies in conjunction with semantic bridges allows for applying automatic semantic matching and reasoning mechanisms based on polymorph representations of service parameters. The interaction behavior of the mediator has been manually designed and addressed by using a BPEL engine as the coordinating entity. Some BPEL enhancements were developed to integrate semantic bridges and to support data flow specifications in terms of rules. These enhancements were implemented as external functions that can be plugged into BPEL engines. Thus, in contrast to our approach, the presented approach designs the mediation solution at technology level. It relies strongly on the existing Web standard BPEL and cannot easily be used with alternative technologies.

5 Conclusions and future work

In this paper, we presented a model-driven method for the semantic integration of service-oriented applications. The key feature of the proposed method is that semantically enriched service models are employed at different levels of abstraction to develop end-to-end integration solutions from business requirements to software realization. Therefore, the integration problem is solved at a higher level of abstraction by business domain experts and then (semi-)automatically transformed to a software solution by adding technical details by the IT experts. This way, the same business integration solution can be reused to implement different IT integration solutions using different implementation technologies. In addition, our framework provides a means to define domain-specific languages (DSLs). This way, business domain experts can solve integration problems using concepts that are closer to their domain, thereby, abstracting from complex service representation techniques and the syntax of data transformation definitions.

Currently, we focus on techniques to automate parts of the composition process of the Mediator. In particular, we consider backward-chaining techniques to discover causal relations among the activities performed by the mediator. In our approach, we start with the activities that send messages and recursively search for activities that provide the information required to construct these messages. The search is performed using the mappings defined in the information model of the mediator. In order to

support semantic matching in this search, we use OWL to express the information model and facilitate reasoning about the mappings.

References

1. Barnickel N, Weinand R, Flügge M. Semantic System Integration - Incorporating Rule based Semantic Bridges into BPEL Processes, In: Proceedings of the 6th International Workshop on Evaluation of Ontology-based Tools and the Semantic Web Service Challenge (EON-SWSC-2008), Tenerife, Spain, June 2008.
2. Dirgahayu T, Quartel D and van Sinderen M. Development of Transformations from Business Process Models to Implementations by Reuse, In: 3th International Workshop on Model-Driven Enterprise Information Systems, 2007, pp. 41-50.
3. ElipseUML. http://www.eclipsedownload.com/.
4. ISDL. http://ctit.isdl.utwente.nl.
5. JAX-WS and JAXB. http://java.sun.com/webservices/technologies/index.jsp.
6. Mantovaneli Pessoa R, Quartel D and van Sinderen M. A Comparison of Data and Process Mediation Approaches, In: Proceedings of the 2nd International Workshop on Enterprise Systems and Technology (I-WEST 2008). INSTICC Press, May 2008, pp. 48-63, Enschede, The Netherlands.
7. Margaria T, Bakera M, Raffelt H, and Steffen B. Synthesizing the mediator with jabc/abc, In: Proceedings of the 6th International Workshop on Evaluation of Ontology-based Tools and the Semantic Web Service Challenge (EON-SWSC-2008), Tenerife, Spain, June 2008.
8. openArchitectureWare, http://www.openarchitectureware.org/.
9. Quartel D and van Sinderen M. On interoperability and conformance assessment in service composition. In: Proceedings of the Eleventh IEEE International EDOC Enterprise Computing Conference (EDOC 2007), 2007, pp. 229-240.
10. Quartel D, Dijkman R, van Sinderen M. Methodological support for service-oriented design with ISDL. In: Proceedings of the 2nd International Conference on Service Oriented Computing, 2004, pp. 1-10.
11. Quartel D, Steen M, Pokraev S and van Sinderen M. COSMO: a conceptual framework for service modelling and refinement. In: Information Systems Frontiers, 9 (2-3), 2007, pp. 225-244.
12. Roman D, Keller U, Lausen L, de Bruijn J, Lara R, Stollberg M, Polleres A, Feier C, Bussler C, and Fensel D. Web Service Modeling Ontology, In: Applied Ontologies, 2005, vol. 1, pp. 77-106.
13. Steffen B, Margaria T, Nagel R, Jörges S, and Kubczak C. Model-Driven Development with the jABC. In: Proceedings of Haifa Verification Conference, LNCS N.4383. Springer Verlag, 2006.
14. Textual Concrete Syntax, http://wiki.eclipse.org/TCS.
15. UDEF. http://www.opengroup.org/udefinfo/.
16. World Wide Web Consortium(W3C), OWL Web Ontology Language Reference, W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-ref/.