Model-driven development of mediation for business services using COSMO

Dick A.C. Quartel[a], Stanislav Pokraev[a], Teduh Dirgahayu[b], Rodrigo Mantovaneli Pessoa[b], Maarten W.A. Steen[a] and Marten van Sinderen[b]

[a]*Telematica Instituut, Enschede, The Netherlands;*
[b]*CTIT, University of Twente, Enschede, The Netherlands*

(Received 17 February 2009; final version received 28 February 2009)

Although service-oriented architectures offer real benefits when pursuing business integration and flexibility, there are still no satisfactory solutions to accomplish cooperation between services of existing systems that have no perfect match. In the case of incompatible services, a 'mediator' may be introduced which resolves semantic and syntactic interoperability problems by intervening in the cooperation between systems. Building mediators is currently often a manual process, resulting in dedicated IT-driven solutions, with no concern for re-use of process, models or code. This paper presents a framework to guide the development of mediators, with the following objectives: (i) uncover and capture the actual interoperability problem that needs to be solved; (ii) allow the involvement of non-IT (i.e., business) experts in the development of the solution; (iii) support evolution of the solution and re-use of results in case of changing interoperability requirements; (iv) facilitate automation of parts of the process. The framework is based on service-oriented and model-driven techniques. Available tool support for the different elements in the framework is indicated.

Keywords: service mediation; interoperability; service composition; COSMO; model-driven development.

1. Introduction

Re-use and composability are considered as important benefits of the service-oriented paradigm. These benefits do however not come for free. Re-usable services need to be identified, specified and, possibly, re-engineered. For this purpose, standardization guidelines may be developed that reflect best-practices and put general quality principles like generality, orthogonality and parsimony into practice. Composition techniques need to be able to apply knowledge about existing services, in order to find combinations of services that match some service request, and select the best among alternatives. The idea behind standardization of services is to facilitate the composition process by reducing the search and solution space. The realization of this idea is however difficult and takes time. Instead, the composition and integration of services from proprietary and legacy systems is currently common practice.

Over the past years, service composition has emerged as an active research area, which has resulted in various approaches and techniques ([9],[29],[18],[1]). However, the applicability of automated approaches is still limited considering the kind of assumptions being made. Furthermore, many approaches are defined at a technology level and cannot easily be used with alternative technologies.

This paper contributes to the area of service composition by presenting a framework for service mediation. We approach mediation as a service composition problem, where multiple systems have to cooperate using non-interoperable services. In order to resolve the differences between these services, a mediator is designed. Two types of mediation are considered: (i) information mediation to resolve differences between the

information models being used, and (ii) process mediation to resolve differences between the interaction protocols being assumed by the systems.

Nowadays, building mediators is mostly a manual process performed by IT experts that consult business domain experts only at the requirements elicitation phase. Often, such projects fail due to miscommunication and misinterpretation of these requirements, or the resulting solutions come at a high price because of the manual labour required to build and maintain them. To address these issues we propose a framework for building mediation solutions by using model-driven techniques. Model-driven techniques are used to lift the design of the mediator from technology to (platform-independent) model level, in order to clearly capture the semantics of the problem in terms of the problem domain rather than solution techniques. This facilitates the involvement of business domain experts in the design process. Furthermore, it enables reuse of the mediator design in case of changing implementation technology.

This paper is further structured as follows. Section 2 analyses the mediation problem and describes our requirements for a mediation framework. Section 3 presents two example mediation scenarios, where the second scenario is an adaptation of the first to reflect the case of changing business requirements. Section 4 presents our mediation framework, including a method for composing mediators. Section 5 demonstrates the method by applying it to the first scenario. Section 6 shows how the method deals with change by applying it to the second scenario. Section 7 discusses related work. And section 8 presents our conclusions and future work.

2. Mediation

This paper addresses the problem of integrating existing systems, in particular business processes and enterprise applications. Following the service-oriented paradigm, we assume that such systems are defined in terms of the services they provide to and request from their environment, e.g., using WSDL. Furthermore, we assume these services can not be changed.

2.1 Definition

Unless systems have been designed with cooperation in mind, it is unlikely that their services will match perfectly. We distinguish two types of mismatches:

- *information mismatches*, which occur when systems use different information models to describe the messages that are exchanged by their services;
- *process mismatches*, which occur when systems use services that define different messages or different orderings of message exchanges.

Figure 1 illustrates two common categories of basic information mismatches: (i) *concept interpretation* mismatches, where the same symbol is used to refer to different, possibly related, concepts, and (ii) *concept representation* mismatches, where different symbols are used to refer to the same or related concepts.

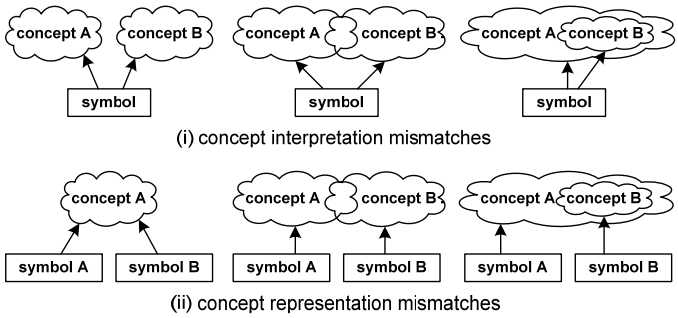

Figure 1. Examples of information mismatches

Figure 2 illustrates some common examples of basic process mismatches. In practice, combinations of these mismatches will occur.

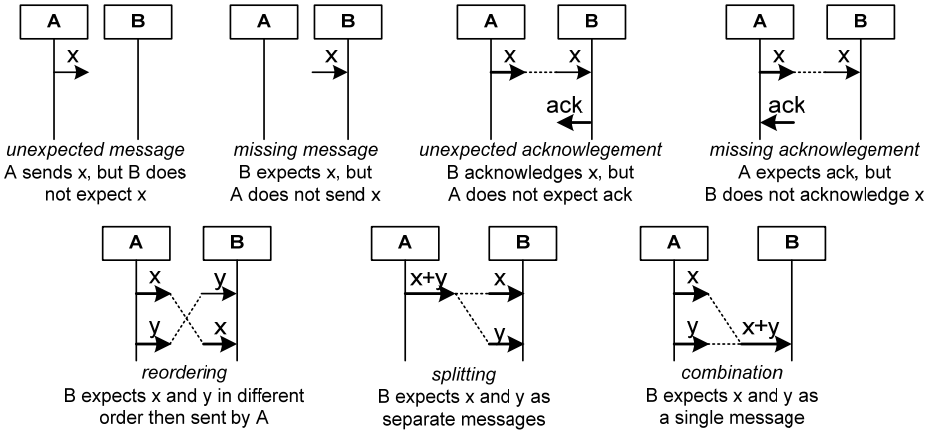

Figure 2. Examples of process mismatches

Service mediation aims at resolving information and process mismatches. Webster's defines mediation as "to act as intermediary agent in bringing, effecting, or communicating" and "to interpose between parties in order to reconcile them". Correspondingly, we define service mediation as "*to act as an intermediary agent in reconciling differences between the services of two or more systems*". The need for an intermediary, further on denoted as mediator, is imposed by the assumption that the mediated services can not be changed. The definition abstracts, however, from whom will perform the mediator role, e.g., some of the existing systems or a 'third' system.

2.2 Approach

We approach the design of a mediator as a composition problem: each service that is requested by some of the involved systems has to be composed from one or more services that are provided by the other systems and, possibly, by the same system. This corresponds to fixed public process composition as described in [5], with the composition (integration) process acting as a mediation broker. Process-based integration, combined with the service-oriented paradigm, is better suited for mediation than earlier approaches, such as hub-and-spoke and point-to-point [6].

Figure 3 illustrates our approach for the case of three systems A, B and C. We assume that A requests some service S1 that has to be provided using services S3 and S5 of systems B and C. The first step defines the provided service S1' that should match the requested service S1. This service is provided by some (virtual) system Q, which comprises (conceptually) systems B and C. The interaction between S1 and S1' defines the choreography of the message exchanges between systems A and Q. The second step refines the provided service S1' into an orchestration, comprising mediator M and systems B and C. Mediator M should orchestrate, i.e., compose, the use of services S3 and S5 such that it offers service S1' to A. The mediator should offer such a mediation service for each service that is requested by systems A, B and C.

A mediation service as defined above provides interoperability for each individual service that is requested by some system. This may however not guarantee interoperability in scenario's where multiple of these requested services have to cooperate. Therefore, our approach allows one to model this cooperation and validate whether it satisfies the goals for integration.

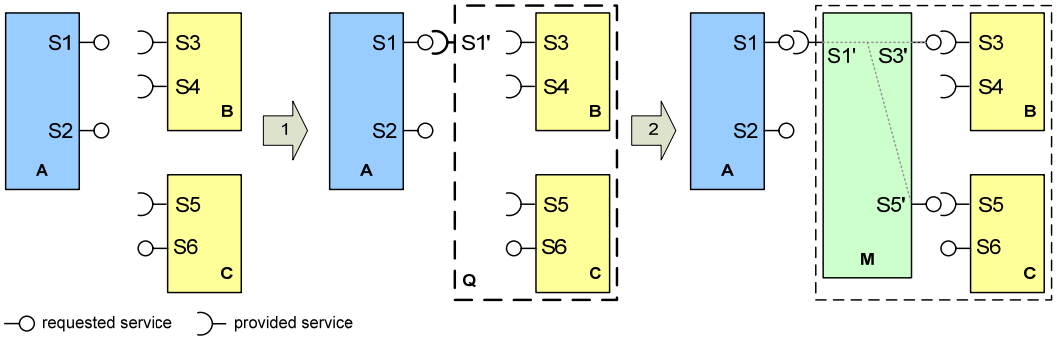

Figure 3. Service mediation as service composition

2.3 Requirements

To address some of the shortcomings of existing integration approaches, we define the following requirements for our approach:

1. The approach should allow one to design the integration solution in terms of the problem domain rather than the implementation technology.
2. The approach should enable the formal verification of the integration solution.
3. The approach should facilitate changes in the implementation technology. This means that if the implementation technology changes, it should be possible to re-use the same abstract solution defined by the domain experts.
4. The approach should facilitate changes in business requirements. This means that if the business requirements change, only the abstract solution specification has to be updated to reflect the new business requirements.

3. Example scenarios

To illustrate and validate our approach we present two reference scenarios that are defined in the Semantic Web Service Challenge (SWSC) [31]. This challenge provides a standard set of integration problems, based on industrial specifications and requirements, and provides a platform to test, discuss and evaluate solutions.

3.1 Scenario 1

A manufacturing company called *Moon* uses two back-end systems to manage its order processing: a *Customer Relation Management (CRM)* system and a *Stock Management (SM)* 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 a proprietary data model and interaction protocol that differ from the ones used by RosettaNet. The objective is to build a *Mediator* that enables *Moon* and *Blue* to cooperate. Figure 4 depicts the scenario.

The interaction between both systems is initiated by Blue who sends a PIP 3A4 *Purchase Order Request* message (M1). PIP 3A4 enables a buyer to issue a purchase order and to obtain a quick response (M13) from the provider that acknowledges which of the purchase order product line items are *accepted*, *rejected*, or *pending*. Both messages must be synchronously confirmed by an *Acknowledgement of Receipt* message (M2 and M14).

According to the RosettaNet standard a Purchase Order Request is sent using a single message. However, in order for Moon to be able to process a purchase order, several steps have to be made. First, the customer needs to be identified by sending a search string to Moon's CRM system (message M3), which replies by sending a customer object that matches the search string (message M4).

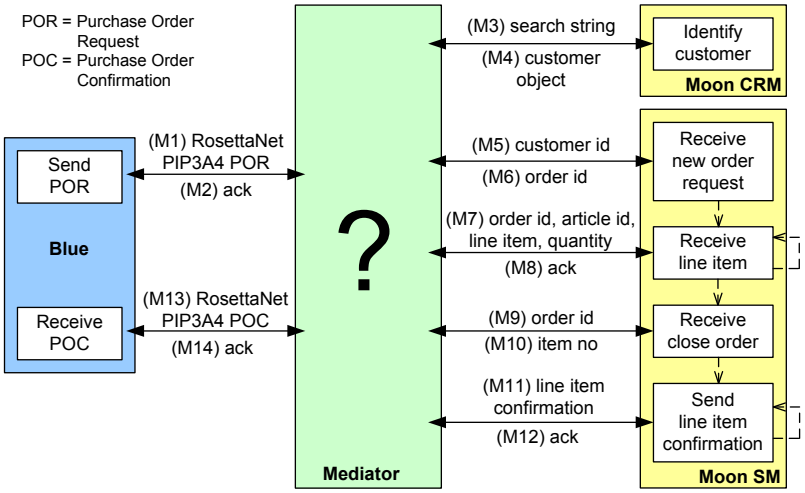

Figure 4. Mediation scenario 1

Next, the creation of a new order is requested by sending the customer object (message M5) to Moon's SM system, which returns the id of the newly created order (message M6). After a new order is created, Moon's SM system expects all order lines to be added one by one (message M7). These messages are acknowledged synchronously by sending the order id and an item id (message M8). Once all order lines have been added, Moon SM is requested to close the order (message M9), and returns the number of items that has been received (message M10). Subsequently, Moon's SM system confirms the status of each order line one by one (message M11), which is acknowledged synchronously (message M12) by the mediator.

After all order lines have been confirmed a RosettaNet PIP3A4 Purchase Order Confirmation message (M13) is sent to Blue and confirmed synchronously by an Acknowledgement of Receipt message (M14).

3.2 Scenario 2

The second scenario aims at validating how well an integration method supports change in the integration requirements. Company Moon decides to integrate also its *Production Management (PM)* system. The Mediator can use the PM to order products to be scheduled for production when they are not available from the SM system. Figure 5 depicts this scenario.

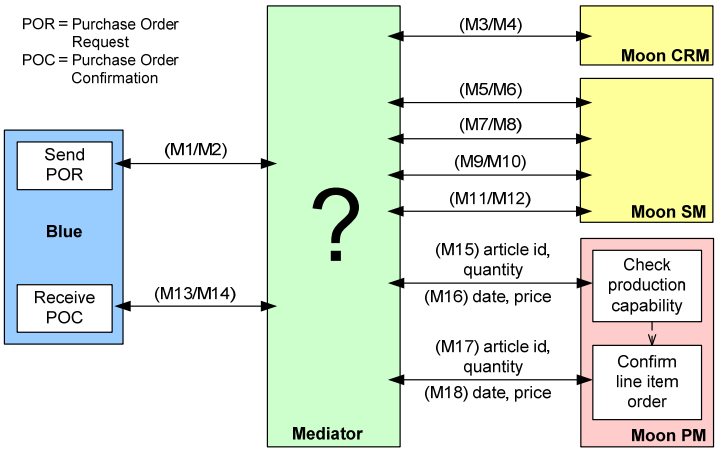

Figure 5. Mediation scenario 2

In case the SM system reports that an item is not available, the PM system will be used to check whether that item can be produced. This is done by sending a message

(M15) to the PM system, which responds synchronously by sending a message (M16) containing the price and the availability date. If the price and the availability date meet the expectations of customer Blue (as specified in message M1) the item will be ordered by sending a message (M17) to the PM system and be confirmed synchronously (M18).

In addition, the possibility is created in this scenario to define a shipment address at the line item level. If present, this address should be used instead of the one defined at the purchase order level. This implies that a purchase order from Blue may result in the creation of multiple orders at Moon, one for each distinct shipment address.

4. Mediation framework

We have developed a mediation framework to support the design, implementation and validation of mediation services. This framework consists of the following elements:

- a conceptual framework for modelling and reasoning about services, called COSMO [25];
- languages to express service models using COSMO, which currently include ISDL [13],[28], OWL [17], SPARQL [23] and Java;
- techniques to analyse the interoperability and conformance of service models [27];
- transformations between service design and service implementation level [8];
- tools supporting the editing, analysis and transformation of service models [24];
- a method for developing mediation services.

This paper focuses on our method for service mediation. The method uses and relates the other elements of the framework listed above, which are only explained here as far as required for a proper understanding of the paper. This includes a brief description of how services are modelled in the second part of this section.

4.1 Method

Figure 6 illustrates the steps that constitute our method for service mediation. For convenience, the integration of two systems is considered, but the same steps apply to the case of multiple systems.

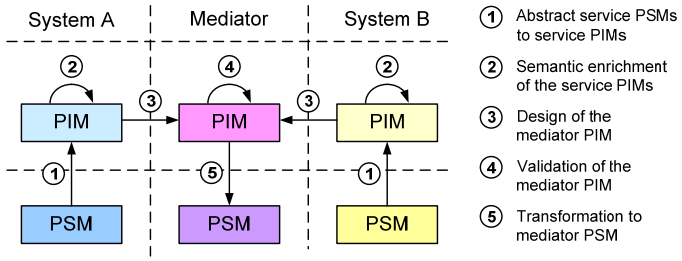

Figure 6. Method for service mediation

In general, the services of systems that have to be integrated are described at implementation (technology) level, e.g., using WSDL. The method starts with "lifting" these service descriptions to design level, by abstracting from implementation specific information. Such information may unnecessarily complicate the design of an integration solution, and therefore hinder the participation of business domain experts that are knowledgeable about the integration requirements at business level, but do not (want to) know how these requirements are implemented at IT level. In terms of the MDA (Model Driven Architecture) this means that we transform the service PSMs (Platform Specific Models) of the systems being integrated to their respective service PIMs (Platform Independent Models).

Subsequently, the service PIMs may be semantically enriched by adding information that could not be derived (automatically) from the service PSMs. For example, a service PSM may be complemented with some text document that describes part of the service in natural language. Alternatively, interviews or even code inspection may be used to obtain information that is missing from the service PSMs. The purpose of semantical enrichment is to make models precise and complete, which in turn is necessary to enable formal reasoning about and, potentially, the (semi-) automated generation of the integration solution.

The next steps represent the design, validation and implementation of the integration solution, i.e., the mediator PIM. The design step can be split into two parts: (i) the design of an information model, and (ii) the design of a behaviour model for the mediator. The purpose of the information model is to enable information mediation, by defining a mapping between the vocabularies of the systems being integrated. The purpose of the behaviour model is to enable process mediation by defining a mapping between the services that are requested and the services that are provided by the systems being integrated (cf. section 2.1).

The validation step is used to analyse whether interoperability is obtained by the proposed integration solution. This step could be omitted in case one would support the automated composition of mediators. But for now this seems an ideal that can not been realized yet. In the final step, the mediator PIM is transformed to an implementation, the mediator PSM.

4.2 The COSMO framework

We define a *service* as the establishment of some effect (or value) through the interaction between two or more systems. The COSMO framework defines concepts to support the modelling, reasoning about and analysis of services. These concepts are structured along three axes as depicted in Figure 7.

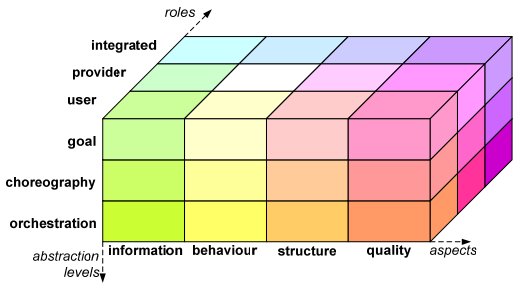

Figure 7. The COSMO framework

The horizontal axis distinguishes four aspects, i.e., *information*, *behaviour*, *structure* and *quality*, representing categories of service properties that need to be modelled. This classification corresponds to aspects found in frameworks for enterprise architectures like GRAAL [10] and ArchiMate [15].

The vertical axis distinguishes three global abstraction levels at which a service can be modelled:

- a *goal* models a service as a single interaction, where the interaction result represents the effect of the service as a whole;
- a *choreography* refines a goal by modelling a service as a set of multiple related, more concrete interactions;
- an *orchestration* implements a service using a central coordinator that invokes and adds value to one or more other services.

We note that these abstraction levels should not be treated as absolute levels, but can again be considered in more or less detail, resulting in 'sub-levels' of abstraction.

The diagonal axis distinguishes the 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 behaviour and information aspect, and by distinguishing between a user and provider role.

4.3 Service modelling concepts

COSMO uses a small number of basic concepts to model the information aspect – using the concepts of *class*, *property* and *individua*l – and the behaviour aspect of services – using the concepts of *action*, *interaction* and *causality relation*. These basic concepts can be used at different abstraction levels to manage the complexity of service design and validation. For this purpose, methods and techniques have been defined to assess the conformance and interoperability of service models.

In this paper, however, services are modelled 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 and its notation using ISDL. More information on COSMO will be provided as needed when discussing the validation of service models later on. But for a proper explanation we refer to [25].

Figure 8(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 interactions: 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.

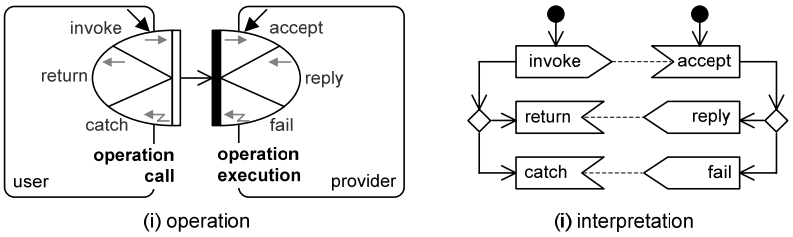

Figure 8. Operation concept

Figure 9 depicts an ISDL model of the choreography between Moon's SM system – behaviour MoonSM – and a user of this system – behaviour User. This choreography consists of four two-way operations (cf. section 3.1). In case of the user only the individual operations are modelled. In case of the SM system also the relationships between the operations are modelled. The processes of receiving and confirming line items by the SM system are represented by the repetitive instantiation of sub-behaviour b of type SM_AddLineItem and sub-behaviour b1 of type SM_ConfirmLineItem (expressed as double bordered rounded rectangles). Grey coloured operation calls and executions represent *delegated* operation calls and executions; since behaviour MoonSM delegates the receipt of a line item and the sending of a confirmation to its sub-behaviours SM_AddLineItem and SM_ConfirmLineItem, respectively.

A textbox defines the parameters associated with an operation, including the constraints on these parameters (between square brackets). For example, the constraint rsp = createAddResponse(addLineItem.req) of operation call addLineItem represents

that the value of response parameter rsp is defined by function createAddResponse, with as argument the request parameter req.

A triangle pointing inside or outside a behaviour represents an entry or exit point, respectively. Also entry and exit points may have parameters, which are represented in textboxes. For example, the repetitive behaviour instantiation b has as entry parameters a list of items and an order id. Upon instantiation these parameters are initialized by an empty list and the order id established in operation createNewOrder, via constraints items = createList() and orderId = getOrderId(createNewOrder.req). Furthermore, these parameters may be used to define the repetition constraint of a repetitive instantiation. For example, the repetition constraint lessThan(e.index, e.itemsNo) defines that sub-behaviour b1 is repeated until all items have been confirmed. Instead, the repetition constraint of b is defined to be always true. This repetitive behaviour terminates once the closeOrder operation is invoked. Since the SM system does not know when all line items have been added, a disabling relation (represented by a black diamond on top of a horizontal bar) is used to model that it is willing to execute both the closeOrder and addLineItem operation after an order has been created, but the occurrence of new addLineItem operation instances is disabled (disrupted) as soon as the closeOrder operation occurs.

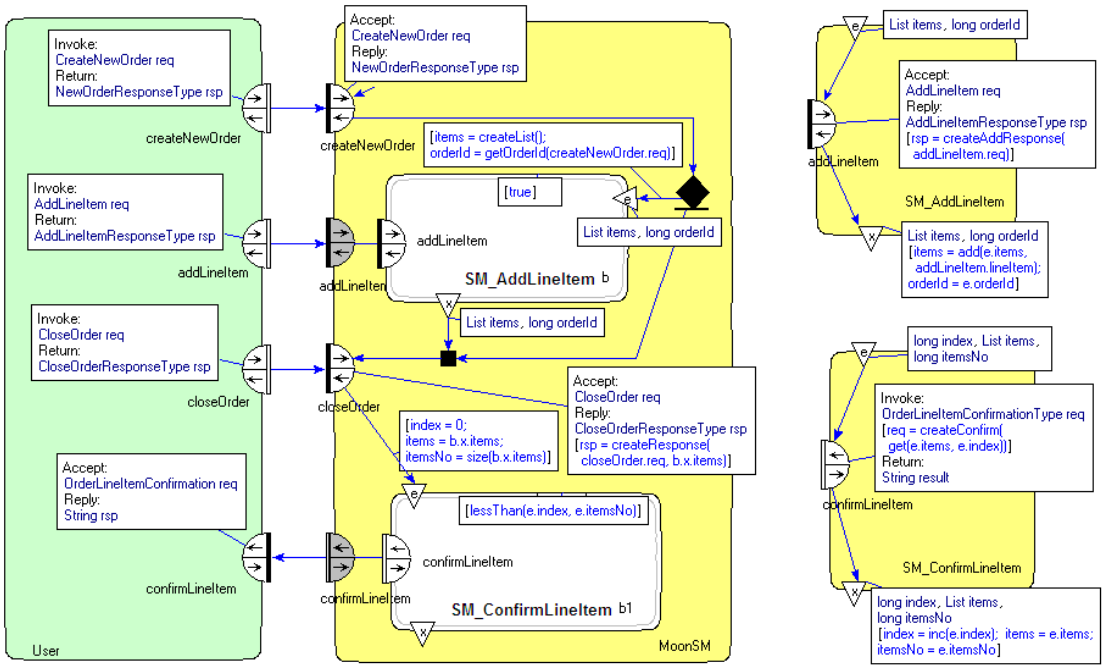

Figure 9. Moon SM choreography

5. Application of the framework

This section illustrates the application of the mediation framework to the scenario of section 3.1. For this purpose, the method of section 4.1 has to be made more 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. This means that in time the mediation framework may be populated with different instances of the mediation method, depending on the type of integration problems that have been addressed.

5.1 Step 1: Abstract from PSMs to PIMs

In this step, we derive the platform independent information and behaviour models of the services of Blue and Moon, which are specified by WSDL documents. Figure 10 illustrates this step. The behaviour models are represented using ISDL, and the information models are specified using UML class diagrams.

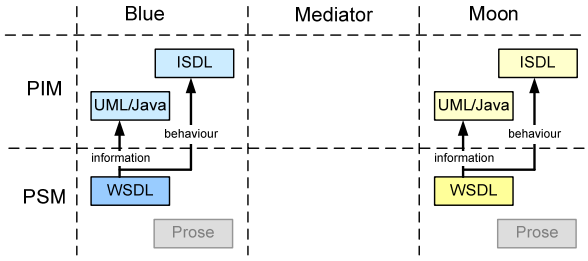

Figure 10. Abstract from PSMs to PIMs

This step is automated using the WSDL import function of the Grizzle tool ([24], [13]). This tool provides an integrated editor and simulator for ISDL. The WSDL import function enables a user to import a WSDL specification by providing the URL of this specification. The user can choose to either import a single operation, single port type or the complete WSDL definition. Furthermore, the user may choose whether the web service should be considered from a client or server perspective. Accordingly, a behaviour model is generated that represents the user (client) or provider (server) role of the web service, in terms of operation calls or executions, respectively. In addition, an information model is generated consisting of UML/Java classes that represent the information types that are referred to by the operations in the behaviour model. The transformation of WSDL to ISDL and UML/Java is implemented using JAXB and JAX-WS ([14]). The EclipseUML tool ([11]) is used to visualize and manipulate the information model. Behaviour User in Figure 9 illustrates a behaviour model that is obtained with the WSDL import function.

5.2 Step 2: Semantic enrichment of PIMs

The WSDL descriptions of the example scenario define the services that are provided by Blue and Moon 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, 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 textual descriptions that are provided with the integration case (the boxes labelled “Prose” in Figure 10). This is a manual process.

Firstly, the behaviour 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 modelling of the “loops” in the schema of Figure 4, representing the repetition of adding and confirming line items. For example, Figure 9 depicts the enriched model of the service provided by Moon SM.

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 provides no information about their semantics. This semantics can be made explicit by defining new classes and use these classes to relate the existing (generated) 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 [32]. The benefits of these types of semantical enrichment can however only be fully exploited when using a language that allows one to explicitly

model and reason about the semantics of classes and their properties. [26] discusses the use of OWL for this purpose, and explores its potential for automated reasoning and composition. In this paper, we focus on the model-driven aspect of our integration method, and will discuss its combination with semantic-web technology in a forthcoming paper.

5.3 Step 3: Design of the mediator PIM

In this step we design the behaviour and information model of the Mediator. The information model 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. [26] discusses a formal approach in defining such relationships using OWL. 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 behaviour model of the Mediator requires the definition of:
(1) the services provided and requested by the Mediator;
(2) the composition of these services by relating the operations of the services;
(3) the information mapping relations among the parameters of the operations.

Step 3.1: Provided and requested services. In the example scenario, the Mediator provides one service that must match the service requested by Blue. This service 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. Analogously, the services that are requested by the Mediator can be obtained by taking the complement of the services that are provided by Moon. Figure 11 depicts the resulting skeleton of the Mediator. For illustration purposes, we use simplified behaviour models in the remainder of this section; replacing parameter type names by the message identifiers used in section 3 and omitting some of the constraints and entry and exit point parameters.

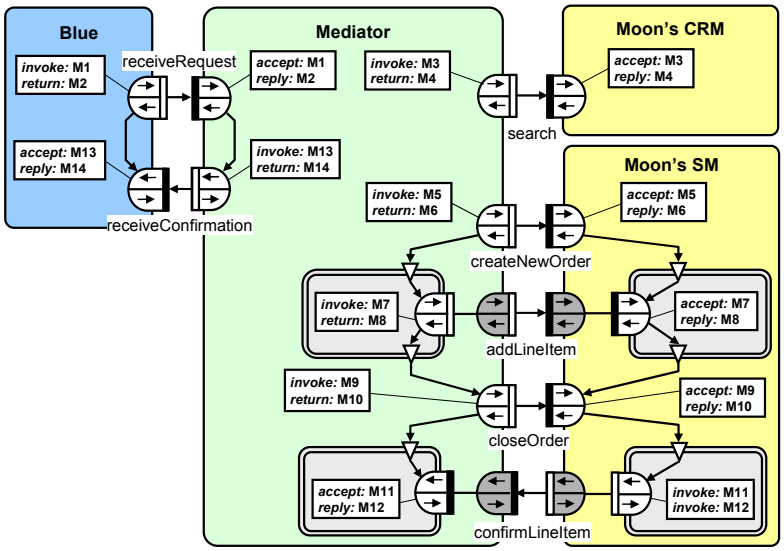

Figure 11. Skeleton of the Mediator

The 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 between addLineItem and closeOrder has already been replaced by an enabling relation, since the order should be closed only after all line items from message M1 have been added.

Step 3.2. Composition of services. The design of the Mediator behaviour can now be approached as the search for a composition of the services requested from Moon, which conforms to the service provided to Blue. 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 information that is required by each operation to the output information that is produced by other operations. For example, operation search of Moon's CRM service requires as input a search string (message M3) that can be matched to some element of the customer information that is part of the purchase order information received by operation receiveRequest (message M1) This implies that a relation should be defined between receiveRequest and search; see Figure 12(i).

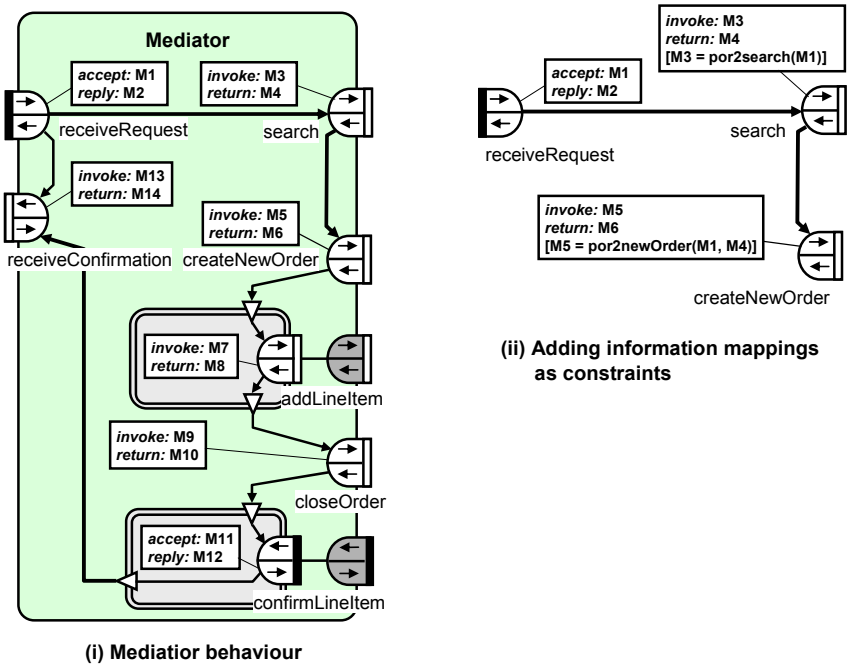

Figure 12. Design of the Mediator

Matching input and output information is however insufficient to find all relations. For example, although both operations receiveRequest and search provide information that matches the input required by operation createNewOrder, the information that is provided by receiveRequest should be used. This hidden assumption has to be made explicit in the behaviour model.

Furthermore, specific processing logic may have to be designed manually. For example, the process of receiving confirmations from Moon's SM 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.

Step 3.3: Information mapping relations among parameters. The definition of the information mapping relations (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 referred to (i.e., used) in the generation of its input parameters. The information mappings then define how the value of each input parameter is generated from the values of the output parameters and, possibly, some internal state information of the Mediator. This involves the definition of mappings between the vocabularies used by Blue and Moon. These mappings only need to address those parts of the vocabularies that are related via the causal relations defined in step 3.2

To define the mappings we use the approach presented in [12]. A mapping system MS is a triple (S, T, M), where S is the source information model, T is the target information model and M is the mapping between S and T, i.e., a set of assertions $Q_s \rightarrow Q_t$, where Q_s and Q_t are conjunctions of queries over S and T, respectively, with the same set of variables x. Thus, a mapping is equivalent to the axiom: $\forall x: Q_s(x, y_s) \rightarrow Q_t(x, y_t)$. This mapping system is independent of the language used to represent the information models of Blue and Moon. In this paper, we use UML class diagrams that are derived from the XML schema in WSDL documents. But for instance, in case OWL would be used, the correspondence between classes and properties from the information models of Blue and Moon can be expressed as a function of subsumption, e.g., $Q_s \subseteq Q_t$. This would allow one to use an OWL reasoner to check the information model of the Mediator for consistency.

To facilitate the use of the mapping approach, we have defined a Domain-Specific Mapping Language (mapping DSL), including a textual and graphical editor. The mapping DSL provides a means for defining the mapping relations between the information models of Blue and Moon and for automatically deriving the information model of the Mediator and the Java classes that implement the information mappings (used at runtime to transform the exchanged messages between Blue and Moon). Figure 13 presents the metamodel of the mapping DSL.

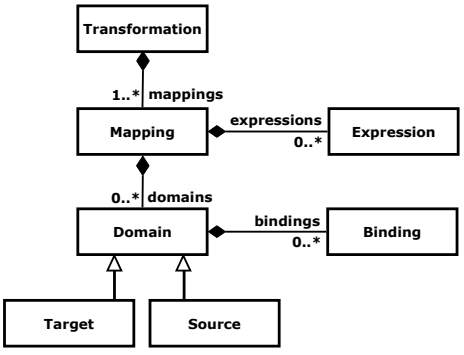

Figure 13. Metamodel of the mapping DSL

A `Transformation` consists of one or more `Mapping`s. A `Mapping` defines an assertion $Q_s \rightarrow Q_t$ where Q_s is defined as conjunction of `Bindings` in a number of `Source domains` and Q_t is defined in a Binding in the `Target domain`. In addition, a `Mapping` may contain zero or more expressions which bind variables by invoking other mappings or custom functions. Figure 14 illustrates the mappings between the class *Pip3A4PurchaseOrderRequest* from the information model of Blue and the classes *SearchCustomer*, *Order* and *LineItem* from the information model of Moon.

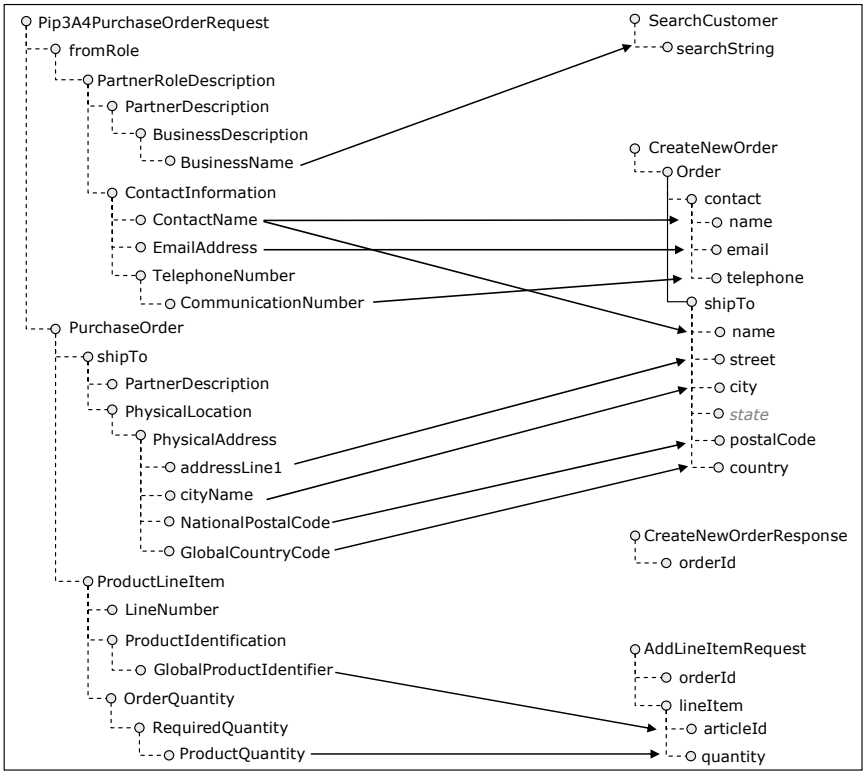

Figure 14. Illustration of information mapping

Using the mapping DSL we formally define these mappings as follows:

```
transformation blue2moon {
  mapping por2search {
    target moon:SearchCustomerType SearchCustomer {
      contactName="searchString";
    }
    source por:Pip3A4PurchaseOrderRequestType req {
      contactName = "fromRole/PartnerRoleDescription/PartnerDescription/
                     BusinessDescription/businessName/FreeFormText";
  } }
  mapping por2newOrder {
    target moon:OrderType createNewOrder {
      authToken = "authToken";
      name = "contact/name";
      phone = "contact/telephone";
      mail = "contact/email";
      addressLine1 = "shipTo/street";
      city = "shipTo/city";
      postcode = "shipTo/country";
      country = "shipTo/country";
    }
    source por:Pip3A4PurchaseOrderRequestType por {
      name = "fromRole/PartnerRoleDescription/PartnerDescription/
              ContactInformation/contactName/FreeFormText";
      phone ="fromRole/PartnerRoleDescription/ContactInformation/
              telephoneNumber/CommunicationsNumber";
      email = "fromRole/PartnerRoleDescription/ContactInformation/EmailAddress";
      addressLine1 = "fromRole/PartnerRoleDescription/PartnerDescription/
                      PhysicalLocation/PhysicalAddress/addressLine1/FreeFormText";
      city="fromRole/PartnerRoleDescription/PartnerDescription/
                      PhysicalLocation/PhysicalAddress/cityName/FreeFormText";
      postcode="fromRole/PartnerRoleDescription/PartnerDescription/
                      PhysicalLocation/PhysicalAddress/NationalPostalCode";
      country= "fromRole/PartnerRoleDescription/PartnerDescription/
                      PhysicalLocation/PhysicalAddress/GlobalCountryCode";
    }
    source moon:SearchCustomerResponseType rsp {
      authToken = "authToken";
  } }
..
```

These mappings are used to constrain the operation parameters in Figure 12(i). For example, the value of the parameter of operation search (message M3) is defined by the mapping M3 = por2search(M1), and the value of parameter operation createNewOrder is defined by the mapping M5 = por2newOrder(M1, M4). Figure 12(ii) links these constraints to the corresponding behaviour relations.

5.4 Step 4: Validation of the mediator PIM

In this step, the design of the Mediator is validated by means of:

- assessing the interoperability between the services of Blue, Mediator and Moon;
- simulating the interacting behaviour of these services.

Interoperability assessment.

A method for interoperability assessment has been presented in earlier work [27]. This method consists of two techniques.

The first technique checks whether each individual interaction can establish a result. This check is based on the abstract interaction concept of COSMO, which allows complex negotiations to be modelled in which the involved systems may define their own, possibly conflicting, constraints on the interaction result. In this case, the interactions are operations, which have been designed such that the parameter types at the sending and receiving side are the same, and the parameter values are completely determined by the sending side. However, when modelling the services of Blue and Moon at a higher abstraction level, i.e., as goals, Blue and Moon may impose different constraints. We refer to [27] for a detailed explanation on how this technique can be used at various abstraction levels during a service design process.

The second technique checks whether the service composition as a whole can establish a result. For this purpose, the interacting behaviour among Blue, the Mediator and Moon is viewed from an integrated perspective. This perspective views operations as joint actions, such that parameter constraints are defined as the conjunction of the parameter constraints of the involved operation call and execution, and causal relations are defined as the conjunction of the causal relations of the involved operation call and execution. Figure 15 illustrates the integrated view of the mediation solution.

Subsequently the integrated view is transformed to a Coloured Petri Net. From this net we construct the corresponding occurrence graph to perform reachability analysis, using the CPNTools [7]. This analysis allows us to check whether operations can be reached, and in a certain order.

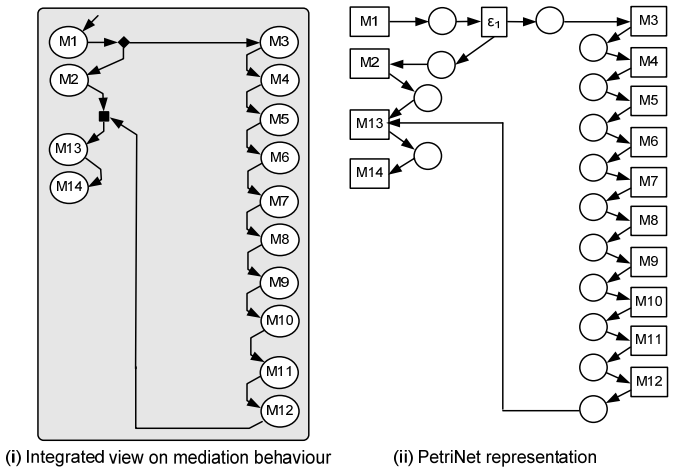

Figure 15. Integrated view on the mediation solution

Simulation

The simulation of ISDL behaviours is supported by the Grizzle tool ([24], [13]). Simulation allows a designer to analyse the possible orderings of operation occurrences, as well as the information results that are established in these operations. In addition, the Grizzle 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 modelled 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. Figure 16 depicts an example of this information.

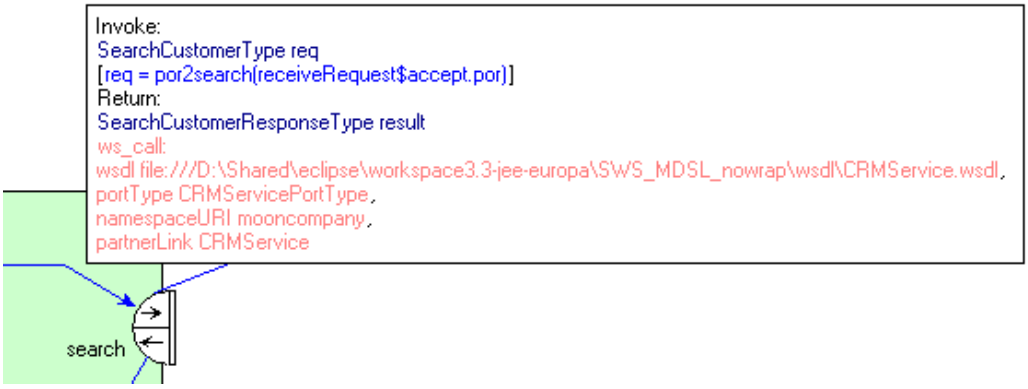

Figure 16. Example stereotype information

Furthermore, the simulator allows external web-clients to invoke a modelled web-service operation *execution* (cf. Figure 8(i)). A web service proxy is automatically generated and deployed in an application server, again using forementioned stereotype information. This proxy is responsible for handling the reception of the invocation request and the return of the invocation 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 the simulator also provides an implementation for the Mediator. However, this simulator lacks important properties of an execution environment, such as performance, monitoring, etc. Therefore, we transform the Mediator design towards a BPEL process in the next step.

5.5 Step 5: Derivation of the mediator PSM

In this final step an implementation is derived for the Mediator design. In our approach we do not assume a particular execution platform. For example, the platform-specific service model of the mediator can be transformed to a WS-BPEL specification, EJB, or .Net application. Figure 17 depicts an abstract architecture of a possible execution platform.

The 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.

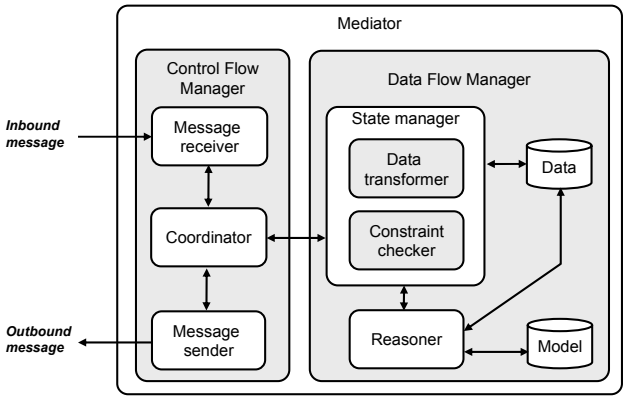

Figure 17. Abstract architecture of an execution platform

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 behaviour specified in the behaviour 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 use the approach described in [8]. This approach distinguishes three steps: behaviour *pattern recognition*, behaviour *pattern realization* and *activity transformation*.

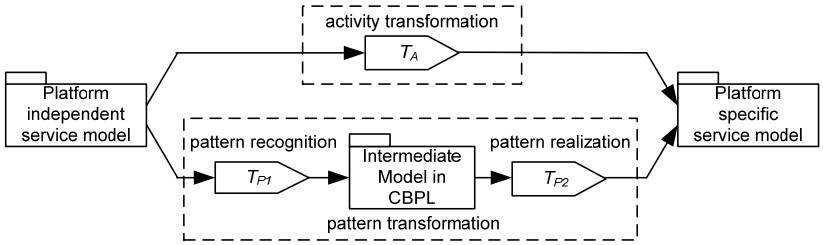

Figure 18. Transformation steps

To decouple the pattern recognition and pattern realization steps (and in this way provide support for building reusable transformations), an intermediate language is defined, i.e., the *Common Behavioural Patterns Language (CBPL)*. Each recognized pattern is mapped onto a CBPL pattern, which is again mapped onto a composition of four basic patterns: *sequence*, *concurrence*, *selection* and *iteration*.

The *Data Flow Manager* consists of two components: a *State manager* and a *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. Similar to the *Data transformer*, the *Constraint checker* queries the state of the mediator and provides an answer whether a constraint holds or not.

To take full advantage of the formal specification of the information model of the Mediator, e.g., in OWL, 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.

A platform-specific service model contains information that is not present in the platform-independent service model. Examples of such information are the XML namespaces of the exchanged messages or the endpoints of the service operations. To provide the required platform-specific information we annotate the elements of the platform-specific service model. For the SWSC case, we have selected WS-BPEL as platform to implement the *Control Flow Manager*. A transformation has been developed that transforms an orchestration model in ISDL, via CBPL, to a BPEL specification that can be executed on a standard WS-BPEL engine. For information on this transformation we refer to [24].

6. Dealing with changed requirements

In order to show how our mediation method copes with changed integration requirements, this section applies the method to the second mediation scenario as described in section 3.2.

6.1 Step 1: Abstract from PSMs to PIMs

In this step, we reuse the transformation described in section 5.1 and derive the information and behavior PIM of the PM system, which concern messages M15 – M18 in Figure 5. The PIMs of the other systems can simply be re-used.

6.2 Step 2: Semantic enrichment of PIMs

This step is identical to step 2 from section 5.2. In this case there is no need to enrich the behaviour PIM of the PM system, since its service operations checkProductionCapability and confirmOrder can be invoked independently of each other.

6.3 Step 3: Design of the mediator PIM

In this step, we update the information and behaviour models of the Mediator to reflect the changes in the business requirements. Similar to step 3 from section 5.3, we first obtain the new services requested by the Mediator by 'complementing' the services provided by the PM system. Next we reuse the relations between the 'old' services defined for scenario 1, and add relations between the new services and the old services. Finally, we re-use the 'old' information mappings and add new mappings to relate the information models of the Mediator and the PM system.

Figure 19 depicts the resulting behaviour PIM of the Mediator. This picture shows that the addition of the PM system is addressed by extending the 'old' PIM model – the parts in grey represent the re-used parts. The extra requirement concerning the use of shipment addresses at line item level is solved by re-using the 'old' mediation solution in a repetitive process, i.e., the 'old' solution is defined in a separate behaviour, called MoonSM, which is invoked one or more times depending on the number of shipment addresses in the purchase order requested by Blue. Action split is added to split the original order in multiple sub-orders, one per shipment address. Action assemble combines the confirmation of these sub-orders in a single confirmation which is returned to Blue.

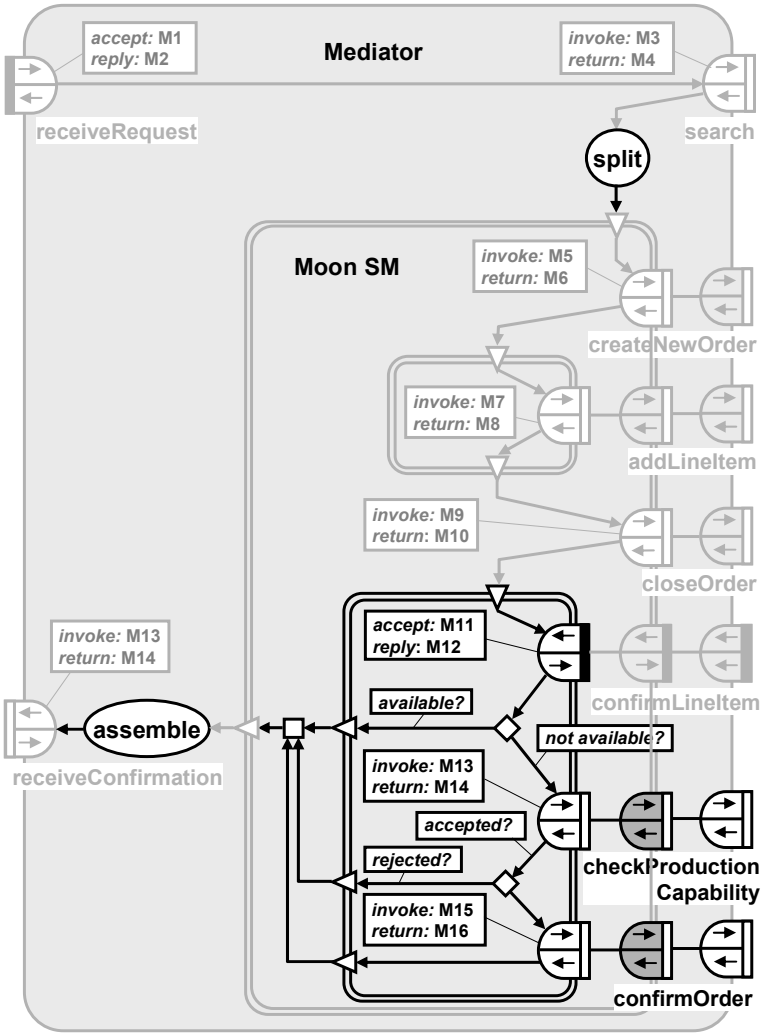

Figure 19. Mediation solution for scenario 2

6.4 Step 4: Validation of the mediator PIM

After updating the service PIM of the Mediator to address the new requirements, we analyse whether the integration solution still enables the integrated systems to interoperate. The same techniques as described in section 5.4 can be used for this purpose.

6.5 Derivation of the mediator PSM

In this final step, we re-use the transformation described in section 5.5 to derive a platform specific model for the Mediator in terms of WS-BPEL. This step is automated and only requires us to add the endpoint address of the PM system.

7. Related and future work

Several approaches and solutions have been proposed within the SWS challenge. Here we briefly discuss the approaches based on the WSMO, SWE-ET, jABC/jETI and FOKUS frameworks.

The DERI approach [19] follows the Web Services Modelling Ontology (WSMO) framework. It consists of four main components – ontologies, goals, web services and mediators. The main difference between WSMO and our work is that our framework has less concepts while providing comparable expressive power. Both solutions, however, differ with respect to the way of process modelling. WSMO describes the

mediator interaction behaviour by means of Abstract State Machines. A state is described by a WSMO ontology, the domain ontology constitutes the underlying knowledge representation and each transition rule defines a state transition where the condition is defined as an expression in logic, which must hold in a state before the transition is executed. For the purposes of the SWS Challenge, the provided solution assumes that the invocation order is unimportant. This is not the case though: the operations of system Moon should be invoked in a particular order.

The joint team of Politecnico di Milano and CEFRIEL [4] focuses more on the modelling of the Mediator's internal logic, which is defined by a BPMN model. A coarse WebML skeleton is automatically generated from the BPMN model and manually refined by the designer. The WebML process model, specified as a graph of (web)pages, differs quite significantly from our approach. Pages consist of connected units, representing the publishing of atomic pieces of information, and operations for modifying the underlying data or performing arbitrary business actions. Units are connected by links, to allow navigation, parameter passing, and computation of the hypertext from one unit to another. The method was not natively meant to face mediation problems, but showed to adapt rather well to this class of problems.

The jABC approach [30] uses *Service Logic Graphs* as choreography models, allowing the designer to model the mediator in a graphical high-level modelling language by combining reusable building blocks into (flow-)graph structures. These basic building blocks are called *Service Independent Building Blocks (SIB)* 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 modelling of the mediator in a graphical high-level modelling language and support the derivation of an executable mediator from these models. More recently [16], the approach has focused on how to apply a tableau-based software composition technique to automatically generate the mediator's interaction behaviour. This uses a *Linear Time Logic* (*LTL*) 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, our approach does not cover automated synthesis of the mediator logic as it intentionally leaves the planning task to the business domain experts.

The core concept of the FOKUS [2] approach is the integration of ontology mappings into WS-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 behaviour 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 WS-BPEL and cannot easily be used with alternative technologies.

9. Conclusions and future work

In this paper, we have presented a framework for developing mediation services as a means to integrate non-interoperable systems. The framework combines model-driven and service-oriented techniques. Model-driven techniques are used to lift the design of a mediation solution from technology to (platform-independent) model level, in order to clearly capture the semantics of the integration problem and proposed solution, and to facilitate the involvement of business domain experts by abstracting from implementation details. In this way we could address the first requirement on our mediation approach (cf. section 2.3).

Following the service-oriented paradigm, the systems that have to be integrated are assumed to be defined in terms of the services they provide to and request from their environment. The integration problem is then approached as a service composition problem, where a mediator must be found that orchestrates and enhances the existing services provided by one system in such a way that it matches the service requested by another system. The service concept helps in addressing the first requirement of our approach, since it provides a unifying concept to bridge the gap between business (services) and IT (services).

A method has been presented to guide the development of a mediator. Tool support is provided for each of the steps in this method, including the modelling, analysis and 'live' simulation of the mediation solution. The presented interoperability analysis and simulation techniques address the second requirement on our mediation approach, i.e., to enable the formal verification of the integration solution.

In addition, transformations between model and implementation level have been developed, with web services (WSDL and BPEL) being assumed as implementation technology. Since these transformations can be reused when the integration problem changes, they limit the impact of changes at model and implementation level. For example, a modification in the WSDL specifications is 'lifted' automatically to model level, and a re- design of the mediator is 'pushed' automatically to implementation level by running the transformations on the modified artefacts. This also enabled us to quickly deal with 'last minute' changes in solving a surprise scenario for the SWSC, and thus to address the third and fourth requirement on our approach.

A fifth requirement – not addressed in this paper – is that our approach should enable the *semantic* integration (or composition) of services. Currently, the composition of the mediation solution is mainly a manual process. The use of semantic web technology enables, in principle, the automated reasoning about the mediator design, in particular the information modelling part. We have applied this in the development of a general technique to assess the interoperability of systems, based on OWL, and have used it in other work [27] to validate the interoperability that is offered by the mediator. [26] also discusses techniques based on OWL to automate parts of the composition process of the mediator. Our ongoing and future work will focus on the elaboration of these techniques, and the development of tool support to make them practically applicable. Here, automation is not a goal in itself. In fact, we do not think semantic web technology can be used (yet) to develop fully automated techniques for building mediators. However, semantic web technology can be helpful in automating techniques that support the designer in re-using design information that is present in existing models.

In general, service composition and mediation have emerged as an active and productive research area. Various approaches and techniques have been presented, such as static vs. dynamic, model-driven, declarative, automated vs. manual, context-based, and workflow vs. planning approaches ([9],[29],[18],[1]). Currently, we

investigate the use of existing AI planning techniques [20] for automated construction of the behaviour of the Mediator. In particular, we focus on the use of 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.

Acknowledgement

This work is partly funded by the Dutch Ministry of Economic Affairs, the Dutch Tax Administration and BiZZdesign, through the BServed project (http://bserved.telin.nl).

References

[1] Alamri A, Eid M and El Saddik A. Classification of the State-of-the-art Dynamic Web Services Composition. In: *International Journal of Web and Grid Services*, Vol. 2, No. 2, 2006, pp. 148-166.

[2] Barnickel N, Weinand R and Fluegge 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*, Tenerife, Spain, June 1-2, 2008.

[3] Battle S. Gloze: XML to RDF and back again. In: *Proceedings of 2006 Jena User Conference*, 2006. http://jena.hpl.hp.com/juc2006/proceedings.html.

[4] Brambilla M, Celino I, Ceri S, Cerizza D, Della Valle E and Facca F. A Software Engineering Approach to Design and Development of Semantic Web Service Applications, In: *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, LNCS 4273, 2006, pp. 172-186.

[5] Bussler C. Semantic Web Services: Reflections on Web Service Mediation and Composition. In: *Proc. of the Fourth Int. Conf. on Web Information Systems Engineering (WISE)*, 2003, p. 253.

[6] Busser C. B2B Integration – Concepts and Architecture. Springer. ISBN 3-540-43487-9.

[7] CPNTools - Computer Tools for Coloured Petri Nets. http://wiki.daimi.au.uk/cpntools//cpntools.wiki.

[8] 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.

[9] Dustdar S and Schreiner W. A survey on web services composition. In: *International. Journal of Web and Grid Services*, Vol. 1, No. 1, 2005, pp. 1-30.

[10] van Eck P, Blanken H, Wieringa R. Project GRAAL: Towards Operational Architecture Alignment. International Journal of Cooperative Information Systems 13(3), 2004, pp. 235-255.

[11] ElipseUML. http://www.eclipsedownload.com/.

[12] Haase P and Motik B. A mapping system for the integration of OWL-DL ontologies. In: *Proceedings of the First international Workshop on interoperability of Heterogeneous information Systems* (Bremen, Germany, November 04 - 04, 2005). IHIS '05. ACM, New York, NY, 9-16.

[13] ISDL. http://ctit.isdl.utwente.nl.

[14] JAX-WS and JAXB. http://java.sun.com/webservices/technologies/index.jsp.

[15] Jonkers H, Lankhorst M, van Buuren R, Hoppenbrouwers S, Bonsangue M, van der Torre L. Concepts for Modelling Enterprise Architectures. International Journal of Cooperative Information Systems, vol. 13, no. 3, 2004, pp. 257-287.

[16] 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*, Tenerife, Spain, June 1-2, 2008.

[17] McGuinnes D and van Harmelen F. *OWL Web Ontology Language Overview – W3C Recommendation 10 February 2004*. http://www.w3.org/TR/owl-features/.

[18] Milanovic N and Malek M. Current Solutions for Web Service Composition. In: *IEEE Internet Computing*, Vol. 8, No. 6, 2004, pp. 51-59.

[19] Mocan A, Moran M, Cimpian E and Zaremba M. Filling the gap - extending service oriented architectures with semantics. In: *IEEE International Conference on e-Business Engineering (ICEBE)*, 2006, pp. 594-601.

[20] Peer J. Web service composition as AI planning - a survey. Technical report, Univ. of St. Gallen, Switzerland, 2005.

[21] Pellet. http://pellet.owldl.org/.

[22] Protégé. http://protege.stanford.edu/overview/protege-owl.html.

[23] Prud'hommeaux E and Seaborne A. *SPARQL Query Language for RDF - W3C Proposed Recommendation 12 November 2007*. http://www.w3.org/TR/rdf-sparql-query/ .

[24] Quartel D, Dirgahayu T and van Sinderen M. Model-driven design, simulation and implementation of service compositions in COSMO. To appear in: *Int. J. of Business Process Integration and Management.*

[25] 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.

[26] Quartel D, Pokraev S, Mantovaneli Pessoa R, van Sinderen S. Model-driven Development of a Mediation Service *Proceedings of the Eleventh IEEE International EDOC Enterprise Computing Conference (EDOC 2008)*, 2008, pp. 117-126.
[27] 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.
[28] Quartel D, Dijkman R, van Sinderen M. Methodological support for service-oriented design with ISDL. Proceedings of the 2nd Internatiation Conference on Service Oriented Computing, 2004, pp. 1-10.
[29] Rao J and Su X. A Survey of Automated Web Service Composition Methods. In: *Semantic Web Services and Web Service Composition*, LNCS 3387, 2005, pp. 43-54.
[30] Steffen B, Margaria T, Nagel R, Jörges S and Kubczak C. Model-Driven Development with the jABC. In: *Proceedings of Haifa Verification Conference*, LNCS 4383, 2006, pp. 92-108.
[31] SWS challenge. http://sws-challenge.org.
[32] UDEF. http://www.opengroup.org/udefinfo/.