Model-driven design, simulation and implementation of service compositions in COSMO

Dick Quartel*

Telematica Instituut,
PO Box 589, Enschede, The Netherlands
E-mail: dick.quartel@telin.nl
*Corresponding author

Teduh Dirgahayu and Marten van Sinderen

Department of Computer Science,
University of Twente,
PO Box 217, Enschede, The Netherlands
E-mail: t.dirgahayu@ewi.utwente.nl E-mail: m.j.vansinderen@ewi.utwente.nl

Abstract: The success of software development projects to a large extent depends on the quality of the models that are produced in the development process, which in turn depends on the conceptual and practical support that is available for modelling, design and analysis. This paper focuses on model-driven support for service-oriented software development. In particular, it addresses how services and compositions of services can be designed, simulated and implemented. The support presented is part of a larger framework, called COSMO (COnceptual Service MOdelling). Whereas in previous work we reported on the conceptual support provided by COSMO, in this paper we proceed with a discussion of the practical support that has been developed. We show how reference models (model types) and guidelines (design steps) can be iteratively applied to design service compositions at a platform independent level and discuss what tool support is available for the design and analysis during this phase. Next, we present some techniques to transform a platform independent service composition model to an implementation in terms of BPEL and WSDL. We use the mediation scenario of the SWS challenge (concerning the establishment of a purchase order between two companies) to illustrate our application of the COSMO framework.

Keywords: service composition, service modelling, service design, model-driven design, COSMO, conceptual framework, simulation, BPEL transformation

Reference to this paper should be made as follows: Quartel, D., Dirgahayu, T. and van Sinderen, M. (xxxx) 'Model-driven design, simulation and implementation of service compositions in COSMO', *Int. J. Business Process Integration and Management*, Vol. X, No. Y, pp.000–000.

Biographical notes: Dick A.C. Quartel is Senior Scientific Researcher at the NOVAY Institute since 2007. His research interests include service-oriented enterprise architecture, BPM, requirements management, model-driven architecture, service composition and interoperability. Previously, he worked as an Assistant Professor at the EEMCS Faculty of the University of Twente where he still supervises PhD students. He participated in national and European research projects and has published many papers in journals and at various conferences and workshops. He is a PC Member of several conferences and workshops. He holds an MSc (1991) and PhD (1998) in Computer Science from the University of Twente.

Teduh Dirgahayu is a PhD candidate in Computer Science at the University of Twente (UT), The Netherlands. He holds an MSc in Telematics from the UT. His research interests include design methods, architectures for distributed systems, business process modelling, service-oriented computing and model-driven engineering. His PhD research is focused on the interaction design of service compositions, from business interaction specifications at higher abstraction levels down to executable implementations. It includes the development of a transformation tool to transform a specification at an implementation level to an executable implementation in BPEL. He participated in the Freeband/A-MUSE project (BSIK 03025).

Marten J. van Sinderen holds an MSc in Electrical Engineering and a PhD in Computer Science. He is currently an Associate Professor at the Faculty of Electrical Engineering, Mathematics and Computer Science of the University of Twente (UT) and Research Manager of the area Service Architectures and Health Applications at UT's Centre for Telematics and Information Technology. His research interests include context-aware systems, service-oriented architectures,

model-driven design and enterprise interoperability. He was a Project Manager of the Dutch Freeband/A-MUSE project (BSIK 03025) on model-driven design of context-aware services and currently leads the Dutch GenCom/U-Care project (IGC0816) on tailorable and adaptive homecare services.

1 Introduction

Service-orientation, model-driven development and semantic interoperability currently receive much attention from the research community and industry. Each of these paradigms aims at facilitating the development of ICT systems, in particular automated business processes and enterprise systems. Instead of considering these paradigms separately, we believe their combined application may offer additional benefits. This idea has been the starting point for the A-Muse project (http://a-muse.freeband.nl), in which we develop a methodology, comprising architectures, methods, techniques and tools, to facilitate the development of business processes and (mobile) applications.

To provide a conceptual basis for this methodology, the COnceptual Service MOdelling (COSMO) framework (Quartel et al., 2007) has been developed. The choice and definition of concepts in this framework has been guided by above-mentioned paradigms. Following the service-oriented paradigm, we want business processes and their supporting applications to be described in terms of the services they offer and new business processes and applications to be developed by composing existing services. For this purpose, COSMO defines a set of modelling concepts that capture the relevant properties of services, where relevance depends on the purpose of a service model, such as the specification, composition or discovery of services. Following the model-driven development paradigm, we want to model and relate services at successive abstraction levels. For this purpose, COSMO defines a small set of generic concepts that can be applied at multiple abstraction levels, in this way limiting the number of concepts that have to be used and making it easier to define the transformation of an abstract model into a concrete model including the assessment of its correctness. Following the semantic interoperability paradigm, we want service models to capture the semantics of the modelled services in terms of some conceptual model or ontology. For this purpose, COSMO provides (meta-)concepts to model the subject domain of a service, such that the effect or value that is established by some service can be modelled in terms of elements from this subject domain.

A quality of COSMO as compared to other conceptual frameworks, such as OWL-S (Martin et al., 2004) and WSMO (de Brujin et al., 2005), is the expressiveness of its behaviour modelling concepts. Furthermore, its abstract interaction concept and constraint-oriented modelling style allows one to model the interacting behaviour of services at higher abstraction levels, which facilitates separation of concerns and early analysis of service properties. Another quality of COSMO is the integration of behaviour and information modelling through a 'loose' coupling between behaviour and information modelling concepts. On the one hand, this integration allows one to model clearly how the behaviour of some service affects the status of its information (subject domain) model. And on the other hand, the loose coupling allows one to use different information modelling languages and supporting analysis techniques and tools to model a service's subject domain.

The objective of this paper is to demonstrate the application of COSMO for service composition. Whereas, in previous work, we have focused on the identification and definition of *conceptual* support, we address in this work the *practical* support that has been developed to apply the COSMO framework. The *practical* support presented in this paper consists of:

1 reference models and guidelines for designing service compositions

2 tool support for constructing and simulating service designs

3 transformation techniques to map a service composition design onto a BPEL/WSDL implementation.

The proposed reference models, guidelines, techniques and tools are illustrated through the elaboration of a service composition example.

The paper is further structured as follows. Section 2 introduces the basic modelling concepts from COSMO that are used in this paper. Section 3 describes different types of service models that can be constructed from these concepts. Section 4 describes the service composition example that is used in this paper. Section 5 discusses reference models and guidelines for designing service compositions, based on the service model types from Section 3. Section 6 presents tool support for constructing and simulating service models. Section 7 describes techniques to transform a service composition design to an implementation in BPEL/WSDL. Section 8 discusses related work. And Section 9 presents our conclusions and future work.

2 Basic concepts

We define a *service* as the establishment of some effect through the interaction between two or more systems. This definition is based on a study of existing service definitions (Quartel et al., 2007) and captures two main characteristics of a service. First, a service involves *interaction* between systems, typically service users and providers. This interaction represents (part of) the external behaviour of the systems involved and abstracts from their internal functioning. Second, the interaction should provide some

value to the systems. This value is called the *effect* of the service.

The COSMO framework defines concepts to model services according to the definition given above, i.e., modelling their interacting behaviour and effect. These concepts have been classified into distinct groups, where each group represents a certain aspect of the services we want to model. The following aspects have been identified: structure, information, behaviour, goal and quality.

In this section, we present concepts for modelling the information and behaviour aspect. The goal aspect is considered here as far as it can be modelled using information and behaviour concepts. Furthermore, we focus on basic concepts, which represent elementary service properties and thereby determine the expressive power of the framework. Basic concepts can be combined into composite concepts to facilitate the modelling of frequently occurring compositions of service properties.

For a more detailed explanation of the concepts and the presentation of the meta-models that define the relationships among the concepts, we refer to Quartel et al., (2007).

2.1 Information concepts

The effect of a service refers to elements in the subject domain of the systems involved in the service. The subject domain of a system comprises the entities and phenomena in the real world that are *identifiable* by the system. We use an *information model* to model a system's subject domain. This information model consists of *individuals* that represent the entities and phenomena from the subject domain, *classes* that represent the types of the entities and phenomena and *properties* that represent the possible relations between classes and individuals. An example of an information model is presented in Figure 6 in Section 4. This model does not include individuals and the valuations of their properties, which together we call the state of a system.

Various languages can be used to represent the concept of individual, class and property. Since these concepts underlie description logics (Baader et al., 2003), we often use OWL-DL (McGuinnes and van Harmelen, 2004) to represent information models in combination with SPARQL (Prud'hommeaux and Seaborne, 2007) to represent pre- and post-conditions on the state of an information model.

In this paper, we use UML class diagrams to represent information models. The concept of class maps to a UML class, the concept of property to a UML association and the concept of individual to a UML class instance (or object). To represent pre- and post-conditions on the state of an information model, a natural choice is OCL (OMG-OCL, 2006). However, to enable execution of these conditions during the simulation of service compositions (see Section 6), we have to map these OCL conditions onto Java. We have implemented this OCL-to-Java mapping using the Octopus tool (http://www.klasse.nl/octopus/index.html), but its explanation is beyond the scope of this paper. Therefore, we have chosen in this paper to represent conditions on the state of an information model directly in Java. Furthermore, by using proper naming conventions Java expressions may be easier to understand than OCL expressions.

2.2 Behaviour concepts

The behaviour of a service comprises the interactions between the systems involved in the service and the relationships between these interactions. To represent this behaviour, we use the concepts of interaction and causality condition. In addition, we consider the concept of action as a useful abstraction of an interaction. The ISDL language (http://isdl.ctit.utwente.nl) is used to express these concepts. ISDL supports the modelling of the behaviour concepts of COSMO, such that each behaviour concept can be mapped directly, i.e., one-toone, onto an ISDL concept. Instead, languages like BPMN (OMG-BPMN, 2006) and UML activity diagrams do not support the COSMO interaction concept properly, since their notion of interaction is based on message passing.

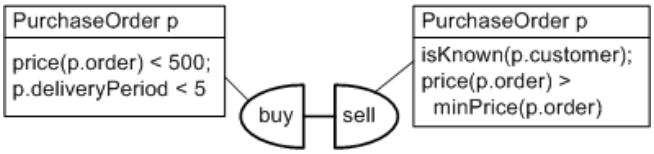
Modelling activities

An *interaction* represents an activity in which two or more systems produce some common result in cooperation. The interaction concept only considers the possible result that can be produced and abstracts from how this result is achieved. Consequently, an interaction is considered an atomic activity that either occurs and establishes the same result for all involved systems, or does not occur for any of the systems and therefore does not establish any result.

Each system may have different expectations on or responsibilities in the establishment of the interaction result. This is modelled by defining an interaction as the composition of two (or more) interaction contributions, one for each system involved. An *interaction contribution* represents the participation of a system in the interaction, by defining the constraints this system has on the possible interaction result, where the constraints represent the expectations or responsibility of the system.

For example, Figure 1 depicts a purchase interaction between a customer and a retailer. Interaction contributions buy and sell represent the participation of the customer and retailer in this interaction, respectively. The associated text boxes define the information attribute of the interaction contribution. This information attribute consists of a declaration part (upper part of the text box) and a constraint part (lower part of the text box). The declaration part defines the type of the interaction result by referring to some class of an information model and the attribute name. The constraint part defines the result constraints that must be satisfied by the interaction result, which are represented by Java expressions. In this case, both the customer and retailer want to establish a purchase order as the interaction result. The customer wants the pay a maximum price of 500 (e.g. euro) and wants the order to be delivered within five (say) days. The retailer, however, is only willing to accept orders from known customers and requires a minimum price. The information model underlying this example is explained in Section 4.

Figure 1 Purchase order interaction



The purchase interaction can only occur if the constraints of both the customer and the retailer can be satisfied. Where multiple results are possible that satisfy the constraints, only a single result (individual) is established. Since the interaction concept abstracts from how to select the result, the result is assumed to be selected non-deterministically.

An *action* represents an activity that is performed by a single system. Similar to an interaction, an action has an information attribute defining the type of the action result and the constraints on this result. The action concept is also used to model an interaction from a so-called *integrated perspective*. This perspective abstracts from the distribution of constraints over the systems involved, thereby considering these systems as a single (virtual) system. Accordingly, the action constraint is defined as the conjunction of the interaction contribution constraints. As opposed to the integrated perspective defined by the action concept, we say that, the interaction concept defines a distributed perspective of the same activity. An action is graphically expressed as an oval or circle. Examples of an action representing an integrated interaction are presented in Section 5.

Modelling related activities

Relations between activities can be modelled in different ways, e.g., in terms of state transitions or temporal relations. We define relations in terms of causality relations. A causality relation defines for each activity a so-called causality condition, which defines how this activity depends on other activities. An activity is enabled, i.e., allowed to occur, if its causality condition is satisfied. Three basic conditions are distinguished:

1 an enabling condition to define that some activity must have occurred before another can occur

2 a disabling condition to define that some activity must not have occurred either before or simultaneously with another activity

3 a start condition to define that an activity is allowed to occur from the beginning of a behaviour and is independent of other activities.

These basic conditions can be combined using conjunction and disjunction operators to represent more complex causality conditions. In this way common workflow operators such as and/or-join and and/or-split can be modelled. Figure 2 depicts some frequently used relations between activities.

Figure 2 Activity relations

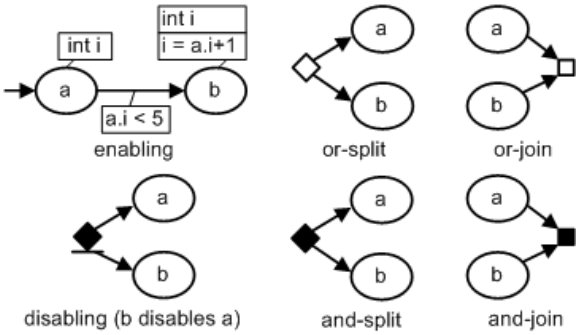
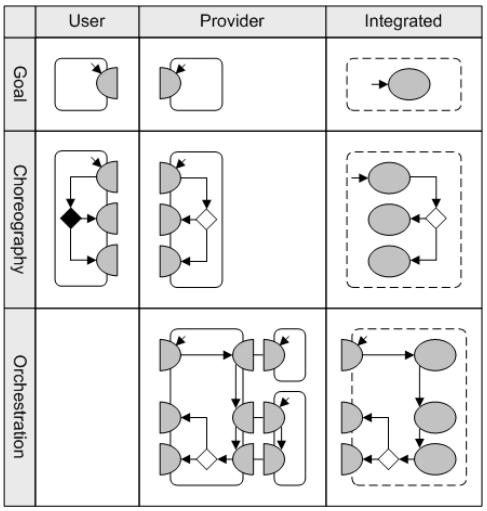


Figure 2 also shows that some action b that is enabled by another action a may refer to information attribute i of a. In this case the result constraint of b defines that the value its information attribute i is equal to the sum of the value of the information attribute i of a (i.e., a.i) and the value one. In addition, a so-called causality constraint is added to the enabling relation of b, which defines that b is only enabled if the value of the information attribute i of a is smaller than five. Compared to a result constraint, a causality constraint is only concerned with the attributes (results) of actions that enable b and not with the attributes of b itself. The conjunction of the causality condition that a must have occurred and the causality constraint a.i < 5 defines a precondition for the occurrence of b. On the other hand, the result constraint i = a.i + 1 defines a post-condition on the occurrence of b.

To represent multiple related activities, the *behaviour* concept is introduced. A behaviour is associated with some system and defines the activities that are performed by this system, including the relationships between these activities. The activities that can be defined are actions and/or interaction contributions. A behaviour is graphically expressed as a rounded rectangle. Examples of behaviours are presented in the following sections.

3 Types of service models

Figure 3 Types of service models



The COSMO framework distinguishes different types of service models. These models vary in abstraction level and the role of the system that is being modelled. Figure 3 gives an overview of these model types.

3.1 System roles

The systems involved in a service are not modelled explicitly, but instead the roles they play in this service are modelled. The reason for this is that the same system may play different roles in different services. The role of a system is represented by the behaviour concept.

In this paper, we consider two generic roles: the user and provider roles. Typically, the user role invokes or requests the provider role to accomplish some effect or offer some value. Both roles represent partial, but complementary definitions of the service. The user role defines the service from the perspective of the user, i.e., the user's participation and is called the *requested service*. Similarly, the provider role defines the perspective of the provider and is called the *offered service*. We denote a system as a service user when it performs a user role and a service provider when it performs a provide role.

Apart from a user and provider perspective, we can also model services from a so-called integrated perspective. This perspective represents the joint behaviour of the offered and requested service, thereby abstracting from the distinction between a user and provider role. This more abstract model of a service is called the *integrated service* model. For this purpose, the action concept is used to represent joint (integrated) interactions.

3.2 Abstraction levels

We distinguish three generic abstraction levels to model a service: as a single interaction, as a choreography of related interactions, or as an orchestration of other (sub-)services.

Single interaction

At the highest abstraction level, a service is modelled by a single interaction. For example, at this level the interaction in Figure 1 may represent a purchase service between some buyer and seller. The interaction result defines the effect of the service as a whole. The interaction contributions define constraints on this effect, representing the effect (or value) that is desired or requested from the service by the buyer and the effect that can be produced or offered by the seller.

We use this abstraction level to represent the *goal* of a service, which is considered synonym to the effect of the service (Lamsweerde, 2001). For example, interaction contributions buy and sell can be interpreted as representing the goals the buyer and seller have in mind when using the service. The goal of a service user is called *requested goal* and the goal of a service provider is called *offered goal* or capability.

The goal abstraction proves useful in modelling the embedding of a service in compositions of services and business processes, since it is represented as a single activity. This is further explained in Section 5. Furthermore, we consider goal models to be useful for concept-based (ontology-based) approaches to service discovery.

Choreography

A service can in general not be implemented as a single interaction, but has to be refined into multiple related, more concrete interactions. This abstraction level, called choreography, is illustrated in Figure 4(i) with a purchase service involving four systems: a buyer that wants to order articles, a bank that settles the payment with the buyer, a shipper that delivers the articles and a retailer that handles the order by requesting a bank to settle the payment and asking a shipper to deliver the articles after a response from the bank has been received.

Each of the systems plays a different role in the purchase service, which can be a user role, a provider role or a combination of both. From the perspective of each single role, the choreography exposes certain information that is irrelevant to this role. For example, the buyer is not interested in the interactions between the bank and retailer or between the retailer and shipper. Similarly, the retailer is not interested in the interactions between the buyer and bank and between the buyer and shipper. These roles actually represent local perspectives on the choreography. The local perspective of some role R can be made explicit by introducing a composite role that comprises the other roles and hides (abstracts from) their (internal) interactions. The composite role can be considered as the environment of role R. Figure 4(ii) and (iii) depict the local perspective for the buyer and bank (with its composite role indicated by a dashed shape), respectively.

Consequently, a local perspective considers only two roles, the role one wants to isolate and the role of the environment. Typically, one of these roles is characterised as the user role and the other as the provider role. For example, the seller and retailer in Figure 4(ii) and (iii) may be assigned the provider role, since they are invoked by the other role.

The choreography level is typically used to model, refine and relate the interfaces of the systems that are involved in a service.

Orchestration

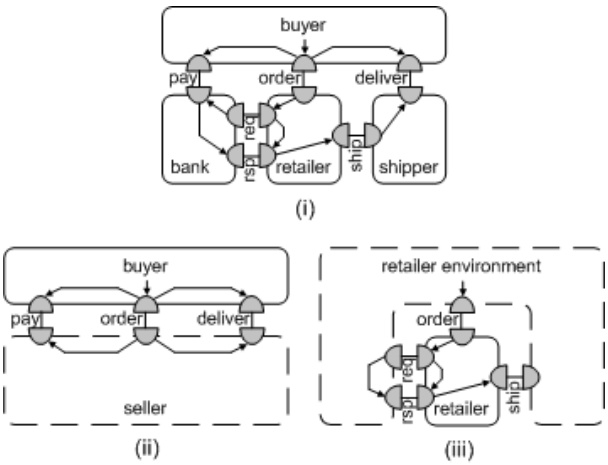
This abstraction level considers the implementation of the service that is offered by a service provider as a composition of other services. We assume these services either exist or can be implemented directly (e.g., as a web-service) or again as a composition.

A commonly used technique to implement a service provider is through orchestration. In this case, the service provider is decomposed into a central coordinator (the orchestrator) that interacts with multiple other service providers. This technique typically uses the local perspectives on a choreography as a starting-point. For example, each of the services provided by the bank, retailer and shipper in Figure 4 could be implemented by an

orchestration. Examples of orchestrations are presented in Section 5.3.

Other techniques may be thought of to implement choreographies, e.g., a distributed protocol that implements all interactions between the systems in Figure 4(i). However, such techniques are considered outside the scope of this work.

Figure 4 Example choreographies



4 Example

The concepts introduced in the previous sections constitute the conceptual basis for our proposed models, techniques and tools. These will be illustrated with a single service composition example.

The example is derived from the mediation scenario of the SWS challenge as described at http://www.sws-challenge.org. This scenario concerns the establishment of a purchase order between two companies Blue (customer) and Moon (manufacturer), which have different requirements concerning:

1 the interaction process, i.e., the order and content of message exchanges

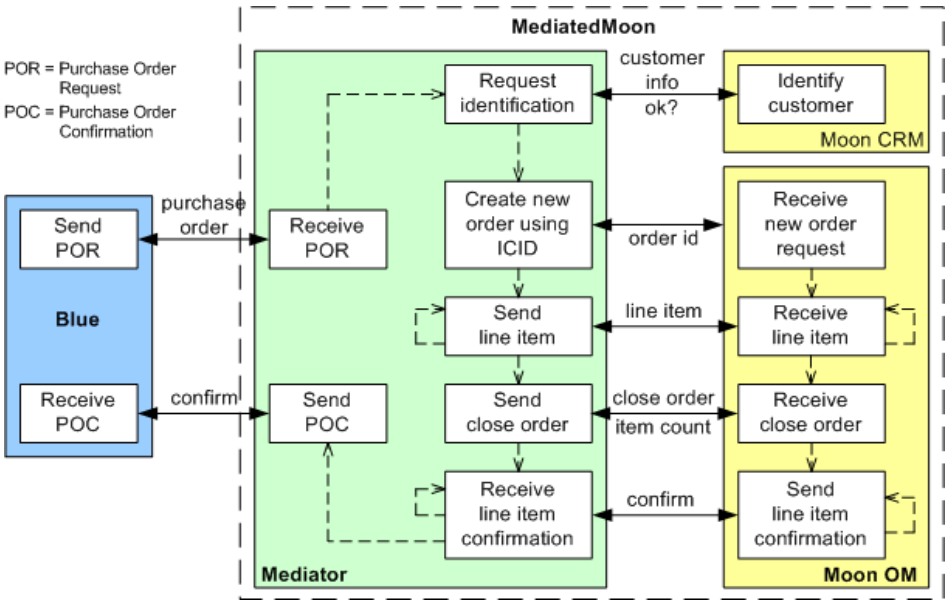
2 the information model, i.e., the vocabulary that is used to describe the messages.

The challenge of this scenario is to develop a mediator that resolves the difference in the requirements of both companies. For simplicity, however, we will only consider differences in requirements on the interaction process. This suffices for the purpose of this paper.

The composition of the mediator and Moon's customer relation management (CRM) and Order Management (OM) system constitutes a service provider, called MediatedMoon (MM), which should offer a service that is interoperable with the service requested by Blue. The mediator acts as an orchestrator in this composition and is responsible for offering a service that is interoperable with the service requested by Blue through invoking the existing services provided by Moon's CRM and OM systems.

Figure 5 depicts the interaction process between the mediator, Blue and Moon's CRM and OM systems. Blue wants to send a request for a purchase order as a single message and also receive a confirmation as a single message. The purchase order contains customer information, e.g., the company name and address and order information, e.g., the order lines, order status and maximal delivery period.

Figure 5 Mediation process (see online version for colours)



Source: http://www.sws-challeng.org

Instead, the CRM system of company Moon first wants to receive a message containing the customer information, because only orders from known customers are accepted. Subsequently, the OM system of Moon handles the order, for which it assumes the following message exchanges:

1 the receipt of a request to create a new order, followed by the sending of a new order id

2 the receipt of individual order lines

3 the receipt of a close order request

4 the sending of confirmations for each order line that has been received, indicating that the order line is accepted or rejected, in case the article is out of stock or cannot be delivered in time.

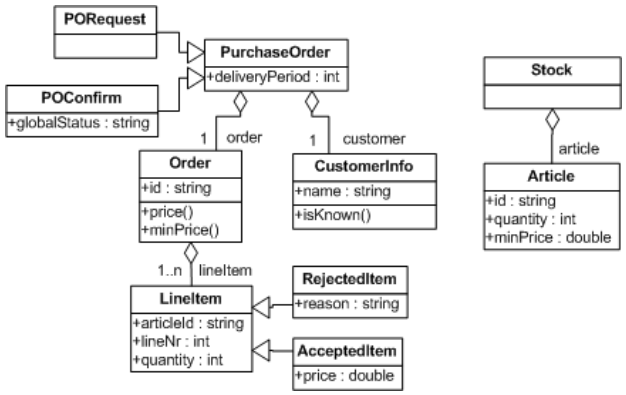
The actual price of each order line is determined by Moon, based on some discount percentage that may differ per customer. Furthermore, Moon only accepts orders with a minimal total price.

Given the requirements of Blue and Moon, the main tasks of the mediator are:

- the splitting of a purchase order request from Blue into the verification of the customer information and the handling of the order information by the CRM and OM systems of Moon, respectively
- the splitting of a single order into multiple order lines, preceded by the creation of a new order and followed by the closing of the order
- the combination of the order line confirmations received from Moon into a single confirmation that is sent to Blue.

Figure 6 depicts the information model of Blue, Moon and the mediator, which is derived from the WSDL and complementary informal descriptions of the services provided by Blue, Moon and the mediator. Based on the explanation given above, the interpretation of the information model should be straightforward.

Figure 6 Information model

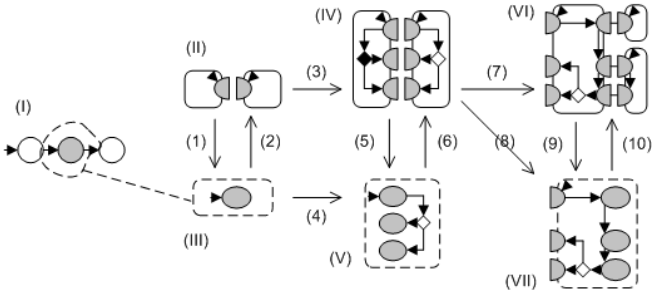


5 Modelling and design

The types of service models presented in Section 3 can be used as reference models for service composition. During a service composition process, various design scenarios may be followed involving different design steps, in which some or all of these reference models are used as templates to model the composed service. A discussion of all possible design scenarios is however unfeasible within the scope of this paper. Instead, we illustrate in this section, the application of the reference models for the mediation example and discuss how these reference models can be used in a service composition process.

In order to organise the discussion of this section we use the schema in Figure 7, which illustrates the reference models and the design steps that can be made. In addition, these reference models and design steps have been given numbers by which they will be referred to in the rest of this section, where Roman numbers are used for models, e.g., (II) and Arabic numbers for steps, e.g., (1).

Figure 7 Reference models and designs steps



5.1 Goal models

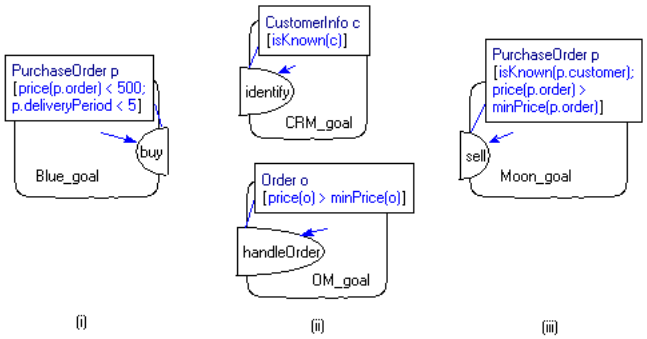
Figure 8(i) depicts the requested goal model of company Blue and (ii) the offered goal models of the CRM and OM systems of company Moon. The requested goal model represents that Blue wants to establish a purchase order, with a maximum price of 500 (euro) and a maximum delivery period of five (days). The offered goal models of Moon's CRM and OM systems represent that Moon is willing (or capable) to accept any purchase order as long as the customer is known and the price exceeds some minimum price, respectively. In addition, Figure 8(iii) depicts the goal (capability) of Moon as a whole, which is formed by the conjunction of the goals of its CRM and OM services.

These goal models (II) can be used for different purposes:

1 To check if the requested and offered goals are interoperable. Goals are interoperable if their result constraints match, such that the conjunction of these constraints allows one or more results to be established. Interoperability at goal level is a necessary condition for interoperability at choreography level, since a choreography refines a goal.

2 To discover an offered service for some requested service (or vice-versa). A goal model seems a suitable abstraction to be used as starting point for service discovery. Based on a goal model, a first selection of services can be made that match at least the goal of the user (or provider). Subsequently, one can determine from this selection the services that are interoperable at choreography level (see Section 5.2).

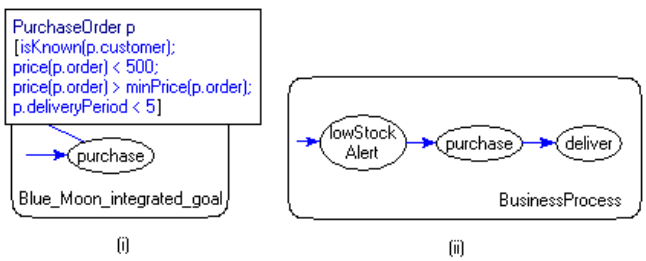
Figure 8 Requested and offered goal models (see online version for colours)



The requested and offered goal models of Figure 8 can be integrated (Step 1) into a joint goal model (III) as depicted in Figure 9(i). This joint goal model abstracts from the distribution of constraints (responsibilities) over the service user and provider. This allows one to model a service as a joint (inter)action and use this action as a building block to model some task in a business process (I), as depicted in Figure 9(ii).

Alternatively, one may start with the definition of a business process model (I) and decide that some task has to be supported by some external service. Subsequently, one can decompose (2) this task into an interaction, i.e., a distributed goal model (II), to define the goal that is requested by the entity that coordinates the business process and the capability that is (assumed) to be offered by the external service. In this way, the design step is used to decide about the responsibility of the coordinator and the external service in performing the task.

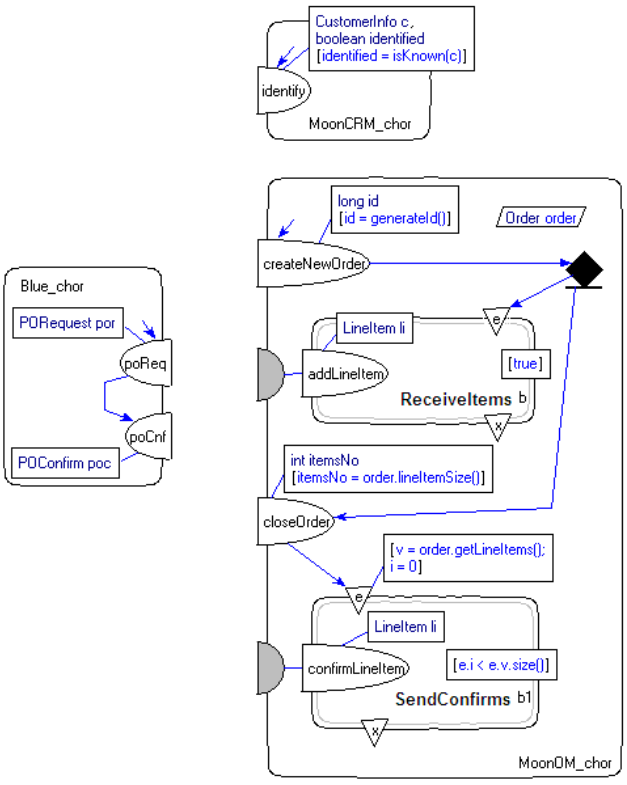
Figure 9 Integrated goal and embedding in business process (see online version for colours)



5.2 Choreography models

Figure 10 depicts models of the choreography requested by Blue, the choreography offered by Moon's CRM and the choreography offered by Moon's OM system. For brevity, some of the information attributes and constraints have been omitted. In addition, the definition of sub-behaviours ReceiveItems and ConfirmItems have been omitted, which represent the reception and confirmation of a single order line item, respectively. The double-lined rounded rectangles represent the repetitive instantiation of these behaviours, in order to model that multiple line items may be received and confirmed. The number of repetitions is determined by the repetition constraints 'true' and 'e.i < e.v.size()'. In this case, new order line items can be received until interaction closeOrder occurs and disables the ReceiveItems subbehaviour. These line items are stored in the global behaviour variable order, which is represented by a parallelogram symbol. Confirmations are sent until all line items from order have been confirmed, as represented by constraint 'e.i < e.v.size()', where e.i and e.v denote parameters i and v of entry point e, respectively. Parameter i represents the index of the next line item to be confirmed, which is increased inside sub-behaviour ConfirmItems.

Figure 10 Requested and offered choreography models (see online version for colours)



The purpose of the requested and offered choreography models (IV) is

1 To design the abstract service interfaces by modelling the interactions that have to be implemented. These interactions (and thus, the interfaces) could again be modelled at different abstraction levels. Here, we assume an abstraction level at which each interaction can be implemented by a single (web service) operation. The refinement of these interactions into operations is described in Section 7.1

2 To check the interoperability of the requested and offered choreography models. Choreographies are interoperable if their interactions have matching constraints (post-conditions) and relationships (preconditions), such that a service execution can reach the final interaction(s).

The requested and offered choreography models can be obtained from the corresponding goal models through refinement (3) or alternatively by a discovery request that produces besides a goal model also a choreography model that conforms to the goal model. Conformance holds if the goal model is a proper abstraction of the choreography model (reverse of 3).

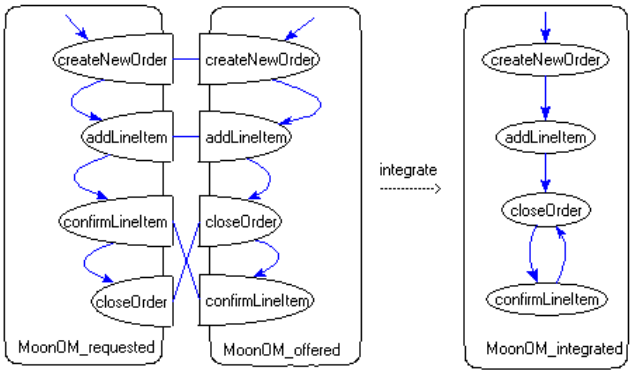
In this example, however, the requested and offered choreographies do not match. To bridge the gap between these choreographies, service provider MM as presented in Section 4 has to be designed. This design is presented in Section 5.3.

Analogously to the integration of goal models, a requested and offered choreography model with matching interactions can be integrated (5) into a joint choreography (V). Figure 11 depicts this for a simplified version of a requested and offered OM choreography. The joint choreography model can be used for different purposes:

1 To analyse the interoperability of the relationships between the interactions in the requested and offered choreography. The joint model facilitates, e.g., reachability and deadlock analysis, because it defines the conjunction of the relationships (causality conditions) of the interactions. For example, the joint model of Figure 11 clearly reveals a mutual dependency (deadlock) between joint actions closeOrder and confirmLineItem.

2 To refine the joint goal model (4). Based on this refinement, the interfaces of the service user and provider may be designed by decomposing the joint model (6). This scenario postpones the distribution of responsibilities over the user and provider, as compared to a scenario using the design steps (2) and (3). For example, this allows one to decompose only a subset of the actions in the joint model and making each of the remaining actions either a complete responsibility of the user or of the provider.

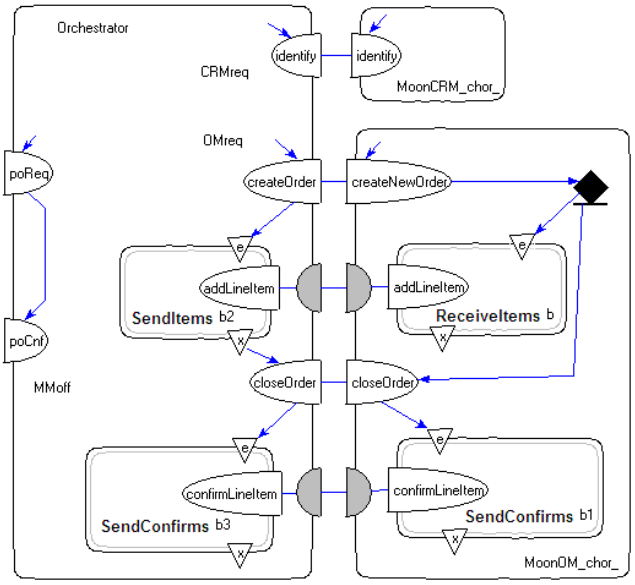
Figure 11 Integrated (joint) choreography model (see online version for colours)



5.3 *Orchestration models*

Figure 12 depicts the structure of an orchestration model (VI) that implements service provider MM. A starting point for the design of the mediator is the definition of a model that consists of three parts: an offered choreography MMoff that matches the requested choreography of Blue and two requested choreographies (CRMreq and OMreq) that match the offered CRM and OM choreographies of Moon. As an initial match, one could simply take the 'mirror' of each of the choreographies that have to be matched.

Figure 12 Structure of MM (see online version for colours)



The specification of choreography MMoff determines what should be implemented by MM. The orchestration model should be a correct refinement (7) of this choreography and can therefore be used as a reference for assessing conformance.

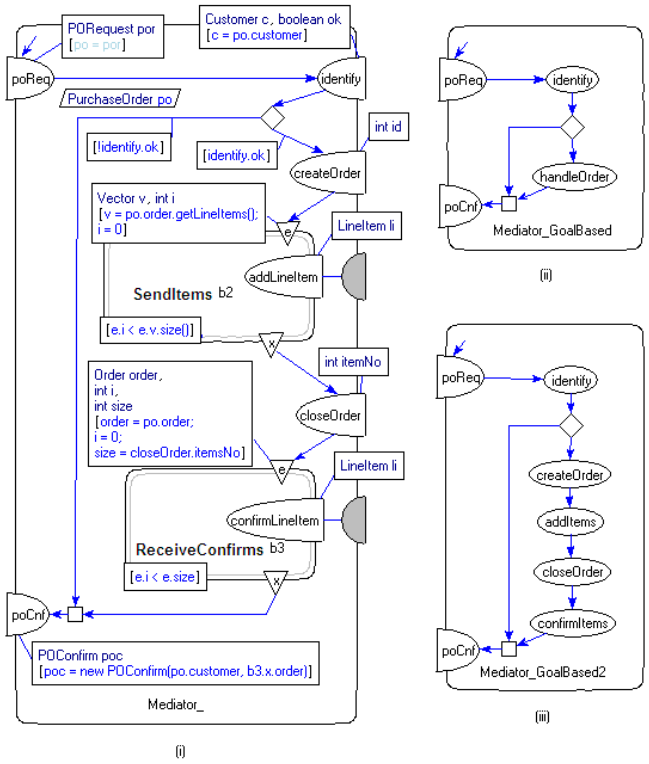
In subsequent design steps, these choreographies MMoff, CRMreq and OMreq have to be related and their interaction constraints may have to be relaxed or narrowed to satisfy the requirements of Blue and Moon. Figure 13(i) depicts the resulting model of the orchestrator. Again, for brevity, the definition of the sub-behaviours and some information attributes has been omitted. However, it should be easy to recognise the mediation tasks as described in Section 4.

As an intermediate design step, the offered choreography MMOff could be refined (8) into a so-called goal-based orchestration model (VII), as depicted in Figure 13(ii) and (iii). This model refines the choreography by modelling joint actions that represent sub-goals in performing the mediation process. The purpose of this model is to focus first on what has to be done by the orchestration, while abstracting from how this is done. Subsequently, these goals may be refined (10) into choreography models. This step may 'reuse' design steps (2), (3) and (4), which illustrates recursion in the design process, since goal and choreography models and the

associated design steps can be re-used in the design of an orchestration model.

Alternatively, one could choose to derive a goal-based orchestration model from the detailed orchestration in Figure 13(i). The purpose of this abstraction step (9) would be to facilitate the analysis of the orchestration model, for the same reasons as explained in Section 5.2.

Figure 13 Orchestration models (see online version for colours)



5.4 Consistency between models

In the foregoing sections, we have introduced and explained the use of a number of reference models for designing service compositions. Since reference models may serve different purpose, various design scenarios are possible that may involve several instances of the reference models.

A concern when using multiple models in a design process is the consistency between these models. Two types of consistency relations have been identified: *interoperability* (of service models at the same abstraction level) and *conformance* (between service models at different abstraction levels). Techniques to describe and analyse these relations using the COSMO concepts have been presented in Quartel and van Sinderen (2007).

6 Editing and simulation

To support the design and analysis of the behaviour models presented in the previous section, we have developed an integrated editor and simulator for the ISDL language. For the creation of information models, we use existing tools. For example, we use Protégé (http://protege.stanford.edu/) to create OWL models and Eclipse plug-ins to create UML models. In this case, we have used the Omondo EclipseUML (http://www.omondo.com/), which automatically generates Java code from class diagrams, including the reverse engineering of these diagrams from Java code.

6.1 Editing support

The editor, called Grizzle (http://isdl.ctit.utwente.nl), enables one to create and manipulate the graphical representation of an ISDL model. The main features of this editor are listed below.

- *Syntax checking*. The graphical syntax is validated on request of the user or before performing certain operations on the model. The source of a syntax error is indicated both textually and by highlighting the corresponding part in the graphical representation.
- *Model organisation*. A behaviour model may consist of multiple (sub-)behaviour definitions. These definitions can be organised over different sheets, which are implemented by separate tabs in the editing window.
- *Ecore XMI export*. The abstract syntax of ISDL is defined by a meta-model in the Ecore meta-modelling language of EMF (www.eclipse.org/emf). Grizzle can export an ISDL model as an instance of this meta-model (in Ecore XMI format).
- *Petri net export*. A mapping from ISDL to Petri nets has been defined. Currently, monolithic behaviour definitions can be exported to Petri nets in a format that can be read by CPNTools (http://wiki.daimi.au.uk/cpntools.wiki).
- *Language profiles*. The editor supports ISDL dialects, which are specialisations or extensions of ISDL. Typically, a dialect introduces specialised or extra language elements as compared to the basic ISDL language. Which language elements can be used in some dialect, including the syntax rules, are defined in a so-called language profile. When creating a new behaviour model, the user can select one of the available profiles.
- *Stereotyping*. Most graphical language elements can be annotated with stereotypes (Quartel et al., 2005). A stereotype consists of a name and zero or more properties and a set of constraints, where a stereotype property consists of a name-value pair. This stereotype information can be used, e.g., to specialise the represented concepts by adding specific modeling information, which is an alternative to defining an ISDL dialect with specialised language elements.
- *Splitting and joining behaviours*. The editor supports two transformations on behaviour models:
  - joining two interacting behaviour models into a single integrated behaviour model by transforming interactions into joint actions and the inverse, i.e.

- splitting a single integrated behaviour model into two interacting behaviour models by transforming joint actions into interactions.

These transformations support the vertical design steps in Figure 7 as explained in Section 5. Figure 14 shows two screen shots that illustrate the splitting transformation.

- *WSDL import*. In order to support modelling the behaviour of web-services, a user may 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. Accordingly, a behaviour model of a single operation, port type or complete web service is generated, including the attributes of the operations. Section 7.1 presents the language elements that are used for representing web service operations.
- *Simulation front-end*. The editing interface is also used as a front-end for simulating behaviour models. The simulator is explained in the following section.

6.2 Simulation support

The ISDL simulator, called Sizzle (http://isdl.ctit.utwente.nl), allows a user to analyse the possible executions of some behaviour. The simulation of a behaviour B consists of the following steps:

1 The *preparation* step. This step determines the activities of B that are enabled, based on the current simulation status. An action is enabled if its causality condition is satisfied and at least a single result value can be established. As compared to an action, an additional requirement for the enabling of an interaction contribution is that the causality conditions of all other interaction contributions involved in the same interaction are satisfied. If the condition of one of these contributions is not satisfied, the interaction cannot be executed and thus none of its contributions occur.

2 *The user interaction step*. This step colours the activities in the editing window. Enabled activities are coloured green and executed activities are coloured yellow. The user can select one of the enabled activities for execution.

3 The *execution* step. This step executes the selected activity and updates the simulation status accordingly. After this step, the simulation returns to Step 1.

Figure 15 depicts the editor in simulation mode. The editing window displays the behaviour model and the colouring of activities. The left window shows the simulation tab, which displays the simulation tree. This simulation tree represents the hierarchy of super/subbehaviours and those activities that have been executed.

Figure 14 Splitting of an integrated behaviour (see online version for colours)

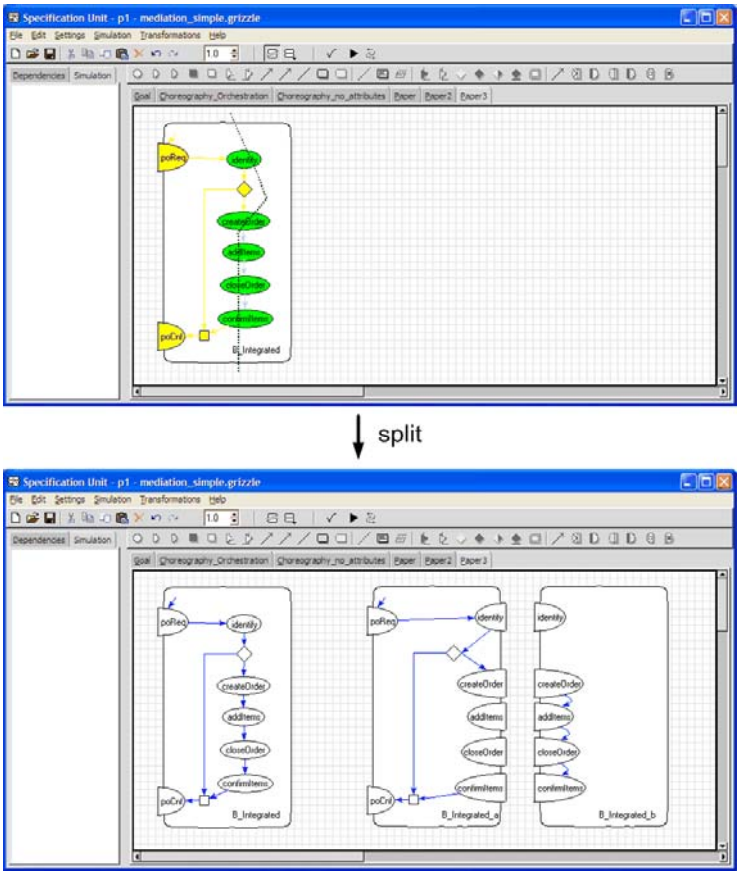
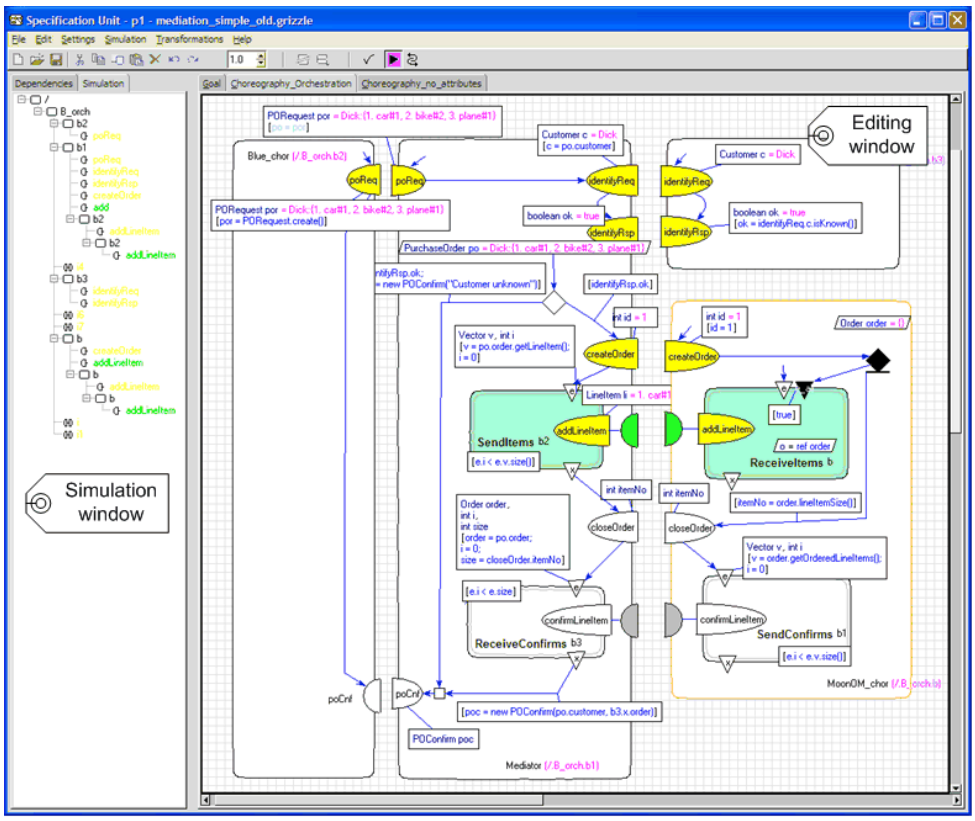


Figure 15 Behaviour simulation (see online version for colours)



The main characteristics of the simulator are listed below.

- *Causality semantics*. The simulator implements the causality-based semantics of ISDL faithfully. The simulation status maintains the causal dependencies between activities to enforce, e.g., that some activity can only refer to results of activities that have directly or indirectly caused the execution of this activity. In contrast, an interleaving semantics would allow an activity to refer to any activity that temporally preceded the execution of this activity. For an elaboration of the causality semantics of ISDL, we refer to Quartel et al. (2002).
- *Interaction semantics*. Considering the establishment of the interaction result, an interaction represents a kind of negotiation in which the interaction contributions represent the negotiation constraints. The following basic types of negotiation are distinguished:
  - value matching, in which all involved contributions propose a single result value of some type
  - value passing, in which one contribution proposes a single result value of some type and all other contributions accept all or multiple values of this type
  - value generation, in which none of the contributions propose a particular result value, but instead accept all or multiple values of some type.

  The latter type of negotiation is difficult to implement without making assumptions about the Java types that are used to represent the interaction result type. Therefore, in case of value generation, the simulation user is required to propose a value when selecting an interaction for execution. Subsequently, the validity of this value is assessed against the constraints.
- *Transformation to the basic ISDL profile*. Simulation is performed on the basic ISDL profile, which comprises the basic COSMO concepts. Composite or specialised concepts used by other profiles are transformed to the basic profile before simulation. In fact, a requirement on the introduction of a new profile is that its concepts can be mapped to the basic profile. This enforces consistency and compliance of a new profile with the COSMO framework.
- 'Live' simulation of web-services. The simulator provides hooks in the simulation process to execute application code upon execution of an activity. When modelling an orchestration of web services (as discussed in Section 7), 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, stubcode is linked to an interaction contribution that represents a web service invocation. This code is generated automatically based on stereotype information that has to be defined for the

interaction contribution, such as the web service's endpoint address and port type name.

In addition, the simulator allows external web-clients to invoke a web-service operation that is modelled by some interaction contribution. For this purpose, a web service proxy is automatically generated and deployed in an application server, again using stereotype information that has to be defined for the interaction contribution. 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. For this purpose, the simulator executes the interaction contribution that models the establishment of the invocation result.

7 Implementation

The design of an orchestration as described in Section 5 results in a platform independent model, since no assumptions are made on the technology that will be used for implementing the orchestration. This section explains how an orchestration model represented in ISDL can be transformed to a web services platform. This platform uses BPEL for specifying the implementation of the orchestrator and WSDL for specifying the services that are offered by the orchestrator and the orchestrated service providers.

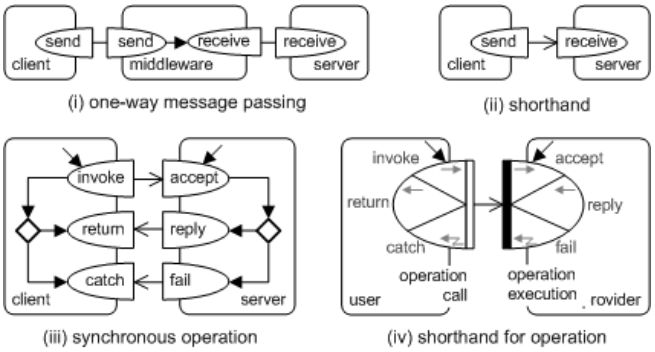
7.1 Interaction refinement

The abstract interaction concept of COSMO cannot be mapped directly onto interaction mechanisms supported by communication middleware. Most middleware assumes the less expressive interaction concept of message passing. This implies that an abstract interaction has to be refined to a composition of message passing interactions.

Figure 16(i) models message passing in terms of the COSMO interaction concept. A shorthand notation for this model construct that hides the role of middleware is depicted in Figure 16(ii). This shorthand is used in Figure 16(iii) to model an operation as a composition of three types of message passing: the sending of an invocation and the return of either an invocation result or a fault message. Figure 16(iv) depicts a shorthand notation for an operation, where the use of the reply-return part and the fail-catch part is optional, i.e., either one or both parts can be omitted.

The concepts of message passing and operation are used as target refinements of the abstract interaction concept. In case of the mediation example, an abstract interaction in the orchestration model is typically refined into a single operation. Rules for assessing the correctness of this refinement are presented in Quartel and van Sinderen (2007).

Figure 16 Message passing and operations

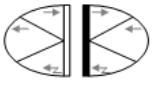


7.2 Behaviour transformation

A designer may freely structure the behaviour of an orchestration model. However, not every structure can be mapped onto structures that are supported by an implementation language. Therefore, we define a set of behaviour implementation patterns (Dirgahayu et al., 2007) that are common to and thus, can be transformed to, various implementation languages. In addition, we define how frequently used behaviour modelling patterns, such as the workflow patterns in van der Aalst et al. (2003), can be transformed to these implementation patterns. In this way, behaviour models that comply to these patterns can be transformed to most implementation languages, including BPEL. This transformation consists of two main steps: the recognition of behaviour patterns and their mapping to BPEL constructs.

Besides constraining its structure, a behaviour model has to be annotated with information that is required as input to the transformation. This information may concern choices in the mapping of abstract behaviour constructs onto concrete BPEL constructs or extra design information that is needed at the platform specific level. Figure 17 depicts the annotations that are used for the transformation to BPEL/WSDL. These annotations are added as stereotype information to a behaviour model. The 'process' annotation is used to indicate which behaviour definition represents the complete orchestration behaviour and thus has to be mapped onto a BPEL process. The other annotations deal with the mapping of operations and the information model (see Section 7.3) onto WSDL. We have implemented a transformation that can generate a complete BPEL/WSDL model from a properly annotated behaviour model.

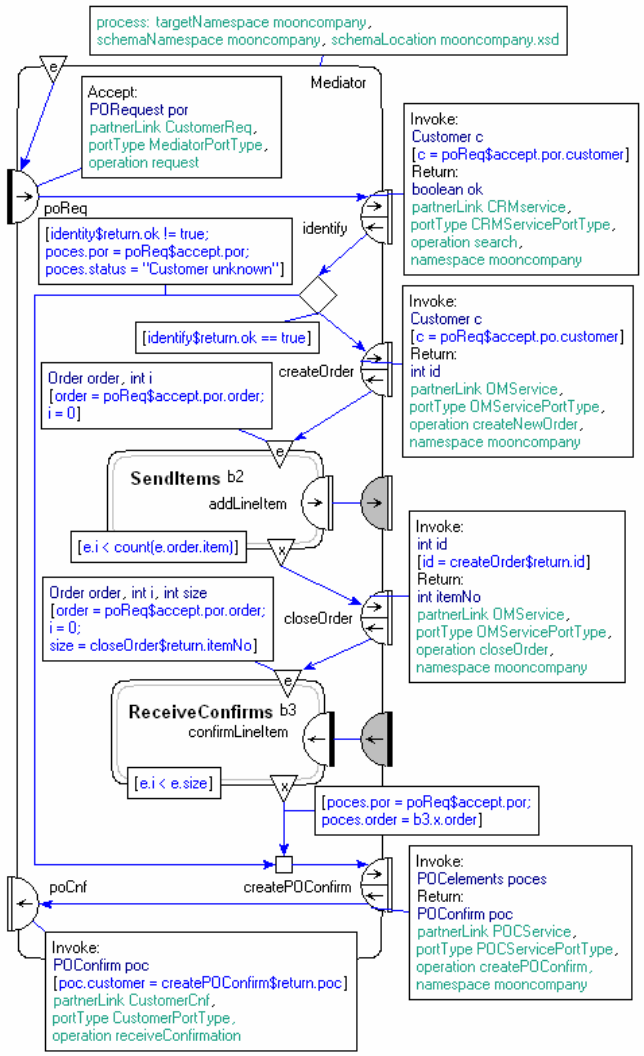
Figure 17 BPEL/WSDL annotations

Model element	Annotation
behaviour	process targetNamespace schemaNamespace schemaLocation
	partnerLink portType operation namespace*

* only for operation call

Figure 18 illustrates the annotation of the refined behaviour model of the mediator in Figure 13(i).

Figure 18 Annotated behaviour model of the mediator (see online version for colours)



7.3 Information transformation

To transform an information model, we use an existing tool named hyperModel (http://www.xmlmodeling.com/hypermodel), which can generate an XML schema from a UML model complying with the UML2 Profile for XML Schema (http://www.xmlmodeling.com/documentation/specs). If an XML schema of the information model is available, its location (URL) should be set as the value for the schemaLocation annotation (see Figure 17).

Besides the information model, the expressions representing result constraints in terms of this information model have to be transformed. BPEL uses XPath (http://www.w3.org/TR/xpath) as its default query and expression language. Instead, for the design and simulation of the orchestration model, we have used Java to express interaction results and constraints. The following techniques may be used to transform Java expressions:

1 Restrict the use of Java expressions to ones that can be transformed to XPath expressions. For example, constraint 'c = poReq$accept.por.customer' of operation identify can be transformed to an XPath expression, because it only involves assignment and references to object attributes.

2 Implement Java expressions as separate services. This implies that the orchestration model should be refined accordingly to include these services and their invocation.

We apply both techniques in combination. Where a Java expression cannot be transformed to an XPath expression it is implemented as a separate web service.

Some BPEL engines, e.g., ActiveBPEL Engine (http://www.active-endpoints.com) and Oracle BPEL Process Manager (http://otn.oracle.com/bpel), allow BPEL processes to contain Java classes or keywords. We do not consider the use of this feature, because it would cause a portability problem for the BPEL processes that are generated by our transformation. Each BPEL engine handles Java expressions in a BPEL process differently.

8 Related work

Many, if not most, publications on service composition present particular approaches and techniques to solve the composition problem, i.e., how to find a composition of existing services that satisfies some service requirement. Descriptions and classifications of service composition approaches, such as static vs. dynamic, model-driven, ontology-driven, declarative, automated vs. manual, context-based and workflow vs. planning approaches, can be found in Dustdar and Schreiner (2005), Rao and Xu (2005), Milanovic and Malek (2004), Alamri et al. (2006). In general, the applicability of these approaches is limited because of the assumptions that are made. In addition, many approaches are technology and platform dependent.

In order to facilitate modelling, reasoning and analysing services and service compositions at more abstract (platform independent) levels, we developed the COSMO framework. There are a number of related activities in developing conceptual frameworks, such as WSMO (de Brujin et al., 2005), OWL-S (Martin et al., 2004), W3C's web services architecture (W3C, 2004) and SeSCE (Colombo et al., 2005). Among these frameworks, we consider OWLS and WSMO as the most prominent ones, both in terms of their expressiveness and the extent of usage and reference by the research community.

The relationship between the frameworks referred above and COSMO has been discussed in Quartel et al. (2007). We consider COSMO particularly strong in modelling the interacting behaviour of services and in its capability to integrate existing languages for modelling the information aspect. Furthermore, service behaviours can be modelled and related at distinct abstraction levels, using the same (small) set of concepts. To exploit this strength, we developed practical support to apply the framework for

designing, simulating and implementing service composition behaviours, as presented in this paper.

WSMO uses BPMO (Business Process Modelling Ontology) (Belecheanu et al., 2007), which extends and restricts the use of BPMN (OMG-BPMN, 2006), to facilitate the graphical modelling of the behaviour of web services. Adopting from BPMN, BPMO uses message flows for modelling interaction between web services. A BPMN message flow represents a message sent from a sender to a receiver. The message passing concept does not, however, allow for the modelling of choreographies or orchestrations at high abstraction levels, like in COSMO. WSMO is mainly supported by two integrated development environments (IDEs), namely WSMO studio (http://www.wsmostudio.org/) and Web Service Modeling Toolkit (http://sourceforge.net/projects/wsmt). These IDEs consist of several tools, e.g., for creating WSMO elements and reasoning ontologies. All WSMO elements are specified as ontologies in the Web Service Modeling Language (WSML) (http://www.wsmo.org/wsml). To our knowledge no support is currently available from these IDEs for the analysis and simulation of the behaviour of web services. Two alternatives for executing a WSMO web service specification are

1 to execute the specification on the Web Service Modeling Execution (WSMX) environment ([http://www.wsmx.org)

2 to execute it as extended BPEL (Filipowska et al, 2007).

A drawback of both alternatives is that they use non-standard implementation languages.

Currently, only a few tools are available that support the modelling and implementation of service behaviour in OWL-S. Elenius et al. (2005) and Scicluna et al. (2004) have developed an editor that supports the visual creation and modification of OWL-S specifications. Composite processes are modelled using a notation based on UML Activity diagrams. Since the interaction concept underlying OWL-S and UML Activity diagrams is based on message passing, the abstract modelling of service behaviour is not supported. Narayana et al. (2002) have defined an execution semantics for process models in OWLS through a mapping onto Petri nets. This mapping allows for the analysis of process models, such as simulation, reachability analysis and deadlock detection. Concerning the implementation of OWL-S specifications, Gannod et al., (2005) present an approach and tool to facilitate the generation of OWL-S groundings, i.e., mappings of service operations and processes to concrete and executable realisations, e.g., in BPEL and WSDL. A model transformation from (parts of) OWL-S to BPEL is reported in Bordbar et al. (2007).

At the platform specific level several integrated tools are available for specifying and simulating BPEL processes. ActiveBPEL Designer (http://www.active-endpoints.com) and Oracle BPEL Designer (http://otn.oracle.com/bpel) use graphical representations of BPEL constructs to facilitate the specification task, whereas Intalio BPMS Designer (http://bpms.intalio.com) uses the BPMN notation. Graphical representations of BPEL constructs require designers to be knowledgeable in BPEL before they can model a service composition in ActiveBPEL Designer or Oracle BPEL Designer. Instead, the BPMN notation is platform independent. However, to be able to simulate a model of a service composition in Intalio BPMS Designer, designers have to add BPEL-specific information to the model. In our COSMO tools, the simulator can simulate a model of a service composition without having to add BPEL-specific information, except when it is used for simulating 'live' web services. BPEL-specific information is mandatory in the transformation step only. The use of conceptual frameworks, such as COSMO, allows business analysts or architects who are experts in service composition in the business domain to model and analyse service compositions at a platform independent level, thereby requiring little or no knowledge about the concrete technology that is used to implement services, such as BPEL.

9 Conclusions and future work

In this paper we have demonstrated the application of the COSMO framework for service composition. This framework provides a conceptual basis to model and reason about services at multiple, related abstract levels. In order to support the practical application of the COSMO framework, we have presented:

- reference models and guidelines (design steps) for designing service compositions
- an editor for constructing service models at different abstraction levels
- a simulator for analysing the behaviour of services and service compositions
- a transformation to map a service orchestration onto an implementation in BPEL/WSDL.

These contributions facilitate the specification, design, analysis and implementation tasks in a service development process. For analysis, only simulation is considered in this paper. Techniques to analyse the conformance and interoperability between service models have been presented in Quartel and van Sinderen (2007). These techniques use Petri net tools for reachability and deadlock analysis and OWL reasoners to match interaction constraints. For this purpose, a mapping from COSMO to Petri nets (for the behaviour aspect) and OWL (for the information aspect) has been defined.

In general, one may want to use different languages for specifying, designing, analysing or implementing service models. For example, for the purpose of analysis the choice of language will depend on the type of analysis one wants to perform. Therefore, in future work, we want to support mappings between COSMO and additional languages, such as BPMN for specification and Promela to support model

checking. The role of COSMO in these mappings is to serve as a common semantical meta-model that allows one to relate models produced in different languages.

Furthermore, we want to complement the COSMO framework with concrete techniques for service composition and discovery at a platform independent level. We do this by studying and extending existing approaches and techniques and describing them in terms of the concepts and facilities that are offered by the COSMO framework. Another topic for future work is the development of (transformation) support for mapping orchestrated services expressed in COSMO onto additional service platforms that are based on industry standards. Examples of these service platforms are OSGi (http://www.osgi.org) and SCA (http://www.osoa.org/display/Main/Service+Component+Architecuture+Home).

Acknowledgements

This work is part of the Freeband A-Muse project (http://amuse.freeband.nl), which is sponsored by the Dutch Government under contract BSIK 03025.

References

Alamri, A., Eid, M and El Saddik, A. (2006) 'Classification of the state-of-the-art dynamic web services composition', *International Journal of Web and Grid Services*, Vol. 2, No. 2, pp.148–166.

Baader, F., Calvanese, D., McGuinnes, D., Nardi, D. and Patel-Schneider, P. (2003) *The Description Logic Handbook: Theory, Implementation and Applications,* Cambridge University Press, ISBN 0521781760.

Belecheanu, R., Cabral, L., Domingue, J., Gaaloul, W., Hepp, M., Filipowska, A., Kaczmarek, M., Kaczmarek, T., Nitzsche, J., Norton, B., Pedrinaci, C., Roman, D., Stollberg, M. and Stein, S. (2007) 'Business process ontology framework', *Deliverable 1.1. Project IST 026850 SUPER*.

Bordbar, B., Howells, G., Evans, M. and Staikopoulos, A. (2007) 'Model transformation from OWL-S to BPEL via SiTra', *Model Driven Architecture – Foundations and Applications*, pp.43–58, Lecture Notes in Computer Science 4530, Springer Berlin/Heidelberg, ISSN 0302-9743 (Print), 1611-3349 (Online).

de Bruijn, J. et al. (2005) 'Web Service Modeling Ontology (WSMO)', W3C member submission 3 June 2005, available at http://www.w3.org/Submission/WSMO/.

Colombo, M., Di Nitto, E., Di Penta, M., Distante, D., Zuccalà, M. (2005) 'Speaking a common language: a conceptual model for describing service-oriented systems', *Proceedings of the 3rd International Conference on Service-Oriented Computing,* ICSOC 2005, pp.48–60, Lecture Notes in Computer Science 3826, Springer Berlin/Heidelberg, ISSN 0302-9743 (Print), 1611-3349 (Online).

Dirgahayu, T., Quartel, D. and van Sinderen, M. (2007) 'Development of transformations from business process models to implementations by reuse', *3rd International Workshop on Model-Driven Enterprise Information Systems,* Funchal, Portugal, pp.41–50. INSTICC Press, ISBN 978-989-8111-00-5.

Dustdar, S. and Schreiner, W. (2005) 'A survey on web services composition', *International Journal of Web and Grid Services*, Vol. 1, No. 1, pp.1–30, ISSN 1741-1106.

Elenius, D., Denker, G., Martin, D., Gilham, F., Khouri, J., Sadaati, S. and Senanayake, R. (2005) 'The OWL-S editor – a development tool for semantic web services', *Proceedings of the Second European Semantic Web Conference*, ESWC 2005, pp.78–92, Lecture Notes in Computer Science 3532, Springer Berlin/Heidelberg, ISSN 0302-9743 (Print), 1611-3349 (Online).

Filipowska, A., Haller, A., Kaczmarek, M., van Lessen, T., Nitzsche, J. and Norton, B. (2007) 'Process ontology language and operational semantics for semantic business processes', *Deliverable 1.3. Project IST 026850 SUPER*.

Gannod, G.C., Brodie, R.J. and Timm, J.T.E. (2005) 'An interactive approach for specifying OWL-S groundings', *Proceedings of the 2005 Ninth IEEE International EDOC Enterprise Computing Conference*, EDOC 2005, 15–19 September, Enschede, The Netherlands, pp.251–260, ISBN 0-7695-2441-9.

Lamsweerde, A. (2001) 'Goal-oriented requirements engineering: a guided tour', *Proceedings of the 5th IEEE International Symposium on Requirements Engineering* (RE'01), pp.249–263, ISBN 0-7695-1125-2.

Martin, D., et al. (2004) 'OWL-S: semantic markup for web services – W3C member submission 22 November 2004', available at http://www.w3.org/Submission/OWL-S.

McGuinnes, D.L. and van Harmelen, F. (2004) 'OWL web ontology language overview – W3C recommendation 10 February 2004', available at http://www.w3.org/TR/owl-features/.

Milanovic, N. and Malek, M. (2004) 'Current solutions for web service composition', *IEEE Internet Computing*, Vol. 8, No. 6, pp.51–59, ISSN 1089-7801.

Narayanan, S. and McIlraith S. (2002) 'Simulation, verification and automated composition of web services', *Proceedings of the 11th International Conference on World Wide Web*, Honolulu, Hawaii, USA, pp.77–88, ISBN 1-58113-449-5.

OMG-OCL. (2006) 'Object Constraint Language – Version 2.0', formal/06-05-01.

OMG-BPMN (2006) 'Business Process Modeling Notation Specification', dtc/06-02-01.

Prud'hommeaux, E. and Seaborne, A. (2007) '*S*PARQL query language for RDF – W3C proposed recommendation 12 November 2007', http://www.w3.org/TR/rdf-sparql-query/.

Quartel, D.A.C., Ferreira Pires, L. and van Sinderen, M.J. (2002) 'On architectural support for behaviour refinement in distributed systems design', *Journal of Integrated Design & Process Science*, Vol. 6, No. 1, pp.1–30, ISSN 1092-0617.

Quartel, D.A.C., Dijkman, R.M. and van Sinderen, M.J. (2005) 'Extending profiles with stereotypes for composite concepts', *The 8th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, 2–7 Oct 2005, Montego Bay, Jamaica, pp.232–247. Lecture Notes in Computer Science 3713, Springer Verlag, ISSN 0302-9743 ISBN 978-3-540-29010-0.

Quartel, D.A.C., Steen, M.W.A., Pokraev, S.V. and van Sinderen, M.J. (2007) 'COSMO: a conceptual framework for service modelling and refinement', *Information Systems Frontiers*, Vol. 9, Nos. 2–3, pp.225–244. ISSN 1387-3326 (Print), 1572-9419 (Online).

Quartel, D.A.C. and van Sinderen, M.J. (2007) 'On interoperability and conformance assessment in service composition', *Proceedings of the Eleventh IEEE International EDOC Enterprise Computing Conference*, EDOC 2007, 15–19 Oct 2007, Annapolis, MD, USA, pp.229–240, IEEE Computer Society Press, ISSN 1541-7719 ISBN 978-0-7695-2891-5.

Rao, J. and Su, X. (2005) 'A survey of automated web service composition methods', *Semantic Web Services and Web Service Composition*, pp.43–54, Lecture Notes in Computer Science 3387, Springer Verlag, ISSN 0302-9743 (Print), 1611-3349 (Online).

Scicluna, J., Abela, C. and Montebello, M. (2004) 'Visual modeling of OWL-S services', *Proceedings of the IADIS International Conference WWW/Internet*, October 2004, Madrid, Spain, IADIS.

van der Aalst, W.M.P., ter Hofstede, A.H.M, Kiepuszewski, B. and Barros, A.P. (2003) 'Workflow patterns', *Distributed and Parallel Databases*, Vol. 14, No. 1, pp.5–51, ISSN 0926-8782.

W3C (2004). 'Web services architecture, W3C Working Group note 11 February 2004', available at http://www.w3.org/TR/ws-arch/.

Websites

ActiveBPEL Designer,
available at http://www.active-endpoints.com.

ActiveBPEL Engine,
available at http://www.active-endpoints.com.

A-Muse project, available at http://a-muse.freeband.nl.

Carlson, D. UML2 Profile for XML Schema, available at http://www.xmlmodeling.com/documentation/specs.

CPNTools – Computer Tools for Coloured Petri Nets, available at http://wiki.daimi.au.uk/cpntools//cpntools.wiki.

Eclipse Modeling Framework (EMF) project, available at http://www.eclipse.org/emf.

hyperModel, available at
http://www.xmlmodeling.com/hypermodel.

Intalio BPMS Designer, available at http://bpms.intalio.com/.

ISDL (Interaction System Design Language), available at http://isdl.ctit.utwente.nl.

Octopus tool, available at
http://www.klasse.nl/octopus/index.html.

Omondo EclipseUML, available at http://www.omondo.com/.

Oracle BPEL Designer, available at http://otn.oracle.com/bpel.

Oracle BPEL Process Manager, available at
http://otn.oracle.com/bpel.

OSGi Alliance, available at http://www.osgi.org.

Protégé, available at http://protege.stanford.edu/.

Service Component Architecture (SCA), available at http://www.osoa.org/display/Main/Service+Component+Architecture+Home.

SWS Challenge, available at http://www.sws-challenge.org.

W3C. XML Path Language (XPath) Version 1.0, available at http://www.w3.org/TR/xpath.

Web Service Modeling Language, available at http://www.wsmo.org/wsml/.

Web Service Modeling Toolkit, available at http://sourceforge.net/projects/wsmt.

WSMO (Web Service Modeling Ontology) Studio, available at http://www.wsmostudio.org/.

WSMX (Web Service Modeling Execution) environment, available at http://www.wsmx.org.