Inf Syst Front (2007) 9:225–244
DOI 10.1007/s10796-007-9034-7

COSMO: A conceptual framework for service modelling and refinement

Dick A. C. Quartel · Maarten W. A. Steen · Stanislav Pokraev · Marten J. van Sinderen

Published online: 3 May 2007

Abstract This paper presents a conceptual framework for service modelling and refinement, called the COSMO (COnceptual Service MOdelling) framework. This framework provides concepts to model and reason about services, and to support operations, such as composition and discovery, which are performed on them at design and run-time. In particular, the framework should facilitate the use of different service description languages tailored to different service aspects, such as the behaviour of a service and the information it manipulates, or design tasks, such as modelling, analysis and implementation. The idea is that models produced by these languages can be mapped onto the concepts of the framework, thereby facilitating one to relate these models, e.g., to verify consistency. Therefore, a requirement on the framework is to provide concepts that capture all elementary and generic service properties that are relevant during the service development process. We capture these properties by analysing existing service definitions and from earlier experience. Furthermore, we want the same concepts to be applicable throughout the service development process when modelling and refining services at successive abstraction levels. The framework distinguishes three generic abstraction levels, and describes an approach to assess the conformance between the service models produced at these abstraction levels.

Keywords Service-orientation · Service concept · Service modelling · Service refinement · Interaction modelling

D. A. C. Quartel (✉) · M. J. van Sinderen
University of Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands
e-mail: d.a.c.quartel@ewi.utwente.nl

M. J. van Sinderen
e-mail: m.j.vansinderen@ewi.utwente.nl

M. W. A. Steen · S. Pokraev
Telematica Instituut,
P.O. Box 589, 7500 AN Enschede, The Netherlands

M. W. A. Steen
e-mail: Maarten.Steen@telin.nl

S. Pokraev
e-mail: Stanislav.Pokraev@telin.nl

1 Introduction

Service-orientation is currently considered as a promising paradigm to deal with the complexity of IT systems. Informally the service-oriented paradigm is characterized by the explicit identification and description of the externally observable properties of a system, e.g., an application or business process. Systems can then be linked, based on the description of their external properties. According to this paradigm, developers do in principle not need to have any knowledge about the internal functioning of the systems being linked.

We believe the service concept has been used implicitly and explicitly in preceding paradigms like object- and component-orientation, but not to its full potential. Furthermore, observing the many different interpretations of the term service that can be found, we conclude that a general definition and understanding of the service concept is still missing in the area of distributed computing. This in contrast to, e.g., the area of data communication systems, where the importance of this concept has been recognized since (Vissers and Logrippo 1986), and its definition by

OSI can be taken as a reference (ISO 1994). However, also in this area it has taken quite a while before the merit of and need for the service concept as a way to abstract from internal protocol details was recognized and fully understood.

The service concept should precisely define which system properties are modelled, and which are not. The selection of properties should be based on the intended use of this abstraction in structuring and developing IT systems, which is also denoted as service-oriented architecture. For example, one may want to develop new services by composing models of existing services, and using discovery and trading techniques at run-time to find actual implementations of these services. In order to support such a scenario, service models should represent, e.g., interaction properties to define orchestrations and choreographies of services, and more abstract properties such as goal or purpose to facilitate discovery and trading.

The aim of this paper is to present a conceptual framework for service modelling and refinement, called the COSMO (COnceptual Service MOdelling) framework. The purpose of this framework is to bring clarity to the field of service-orientation by fixing terms and providing concepts to model and reason about services, and to support operations, such as composition and discovery, which are performed on them at design and run-time.

The COSMO framework focuses on basic concepts to represent essential, elementary and generic service properties. In this way, the framework can be used as a basis for the development of more complex concepts that represent combinations of service properties by composing them from the basic concepts. Typically, such concepts are introduced to facilitate the modelling task by representing frequently occurring compositions of service properties.

In addition, we want the proposed concepts to be applicable throughout the service development process, in which services are modelled, and may be refined into more detailed models. This increases usability and facilitates one to analyse the refinement relation between service models produced at successive abstraction levels. The framework distinguishes three generic abstraction levels, and describes an approach to assess the refinement (also called conformance) relation between the service models produced at these abstraction levels.

The structure of this paper is as follows. Section 2 analyses existing service definitions and derives general service properties. Section 3 structures the COSMO framework into service aspects and abstraction levels. Sections 4 and 5 present our service modelling concepts. Section 6 describes an approach to assess the refinement relation between service models. Section 7 discusses the application of the framework. Section 8 relates our work to other research activities. And Section 9 presents our conclusions and future work.

2 Service modelling requirements

This section presents a number of regularly encountered interpretations of the service concept. From these interpretations generic service properties are derived.

2.1 Existing service definitions

The service concept is widely used in both computer and business science. However, the usage of the concept differs considerably in these areas and even in different "schools of thought" within these areas.

Service as interaction In economics and business science, a service is seen as the non-material equivalent of a good. Service provision has been defined as the economic activity that does not result in ownership, and this is what differentiates it from providing physical goods. It is claimed to be a process that creates benefits by facilitating a change in customers, a change in their physical possessions, or a change in their intangible assets (Wikipedia 2005). The IBM Services Research group defines a service as: "a provider/client interaction that creates and captures value"(IBM 2006). Quartel et al. (1997) also use this interpretation and define a service as the common behaviour of some system and its environment, which is defined in terms of common interactions, the results established in these interactions, and the causal dependencies between them.

Service as capability Often the service concept is connected to the system or entity providing it. Thus a service is the capability of a service provider to produce some intangible benefits to its environment (Baida et al. 2004). CBDI Forum also apply this interpretation to IT services: "a service is a type of capability described using WSDL" (Sprott 2004).

Service as operation In object-oriented and component-based design, each operation or method defined on an object or component is usually seen as a service of that object or component. A service is a part of the object's behaviour, which a client can invoke. In some OO languages, e.g., Java, Corba IDL, these operations can be bundled together in an interface specification. Thus an interface is a collection of service definitions. Confusingly, such a collection of operations is called a service in WSDL. However, the current state of practice in interface definition (or service definition if you like) is that only the signature of each operation is specified. The signature specifies the types of the inputs and outputs of an operation, but not its effect or the relationships between the different operations. The signatures of the addition and multiplication operations on two numbers, for example, will be equal, whereas the

effects of these operations are quite different. Some examples of works that go beyond this simplistic interface definition are from the Semantic Web community, e.g., OWL-S (Martin et al. 2004), and outside it, e.g., WS-Agreement (Andrieux et al. 2005).

Service as application Web services, but also services in general, are most commonly seen as applications (pieces of software) that can be accessed over the Web. The W3C, for example, uses the following definition (W3C 2004): "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network." However, they also make a distinction between the abstract concept of service and its concrete provider: "A Web service is an abstract notion that must be implemented by a concrete agent. The agent is the concrete piece of software or hardware that sends and receives messages, while the service is the resource characterized by the abstract set of functionality that is provided." In practice, this distinction is not made explicit very often.

Service as feature In the telecommunications domain the term service is usually used to refer to a feature that can be provided on top of the basic telephony service, such as call forwarding, call back when busy and calling line identification.

Service as observable behaviour In data communication, a service is traditionally defined as the observable, or external, behaviour of a system. Vissers and Logrippo (1986), for example, define a service as "the behaviour of the [service] provider as it can be observed by the users." In other words, the service of a system is the set of all possible interactions between the system and its environment and their ordering in time. Sometimes the external behaviour of a system is divided over more than one interface, where each interface is a part of the system boundary. In this case, a service is the behaviour of the system as it can be observed at a particular interface. If you take this to the extreme and make each interface as small as one operation, you get more or less the same interpretation of 'service as operation.'

2.2 General service properties

Based on the definitions in Section 2.1, the following general properties of services can be identified.

Involves interaction A service involves one or more interactions between a *service user* and some system that provides the service, also called *service provider*. These interactions can be described from three different perspectives: a user, provider and integrated perspective. From a user and a provider perspective, the participation of respectively the user and the provider of the service is defined, while abstracting from the participation of the other. This means that the *provider perspective* defines the external observable behaviour of the service provider, while the *user perspective* defines the external behaviour that is expected from the user. The *integrated perspective* defines the joint (integrated) behaviour of the user and provider, abstracting from the particular choice on how the user and provider participate and cooperate in performing the interactions. More on these perspectives will be said in Section 4.

The property that a service involves interaction can be found in all of the definitions in Section 2.1. The definitions of service as 'interaction,' 'capability' and 'observable behaviour' consider this interaction from both a user and a provider perspective. Furthermore, the integrated perspective can also be found in the definition of a service as 'interaction.' The other definitions mainly focus on the provider perspective.

Provides some value The execution of a service provides some value to the user and the provider. In case of IT services, this value may only involve 'intangible benefits,' such as the change in possession of goods and money. For services in general, the value may also involve 'tangible things,' such as the actual exchange of parcels using a parcel delivery service. In the latter example, the value of the service may comprise the intangible change of the ownership of the parcel, as well as the tangible exchange of the parcel itself.

The value of a service is established through the combination of the possible results established in the interactions between the service user and provider. Whether tangible or intangible, these interaction results are typically modelled using information types and values.

The property that a service provides some value (or benefit) is made explicit in the definitions of a service as 'interaction' and 'capability.' The other definitions also contain this property, but leave it implicit by refering, e.g., to the inputs and outputs of operations, the functionality of some application, a provided feature, or the behaviour (functionality) that can be observed.

Unit of (de)composition The service concept defines a unit of composition or decomposition. Business processes and supporting applications are composed from or decomposed into services, which define smaller business process or application pieces that may be reused when chosen properly. From a user/provider perspective, such a (de) composition has the form of a set of interacting services, where each service may act as a user, a provider or both. From an integrated perspective, a (de)composition is

described in terms of dependencies between services, e.g. temporal or causal relationships.

The property that a service forms a unit of (de) composition is inherent to the service-oriented paradigm, which fosters the development of services by composing other services. Each of the definitions in Section 2.1 supports this property.

Broad spectrum concept The service concept is meant to be applied at successive abstraction levels along a broad spectrum of the design process, i.e., from specification to implementation. Assuming the design goal is the development of the service provider, the service concept can be applied recursively using the external and internal perspective on a system (Quartel et al. 2004). This property is a consequence of the property that a service can be used as a unit of (de)composition.

3 Structure of the COSMO framework

The concepts presented in this paper are structured along two axes as depicted in Fig. 1. The horizontal axis distinguishes five service aspects, representing a classification of service properties that need to be modelled. This classification corresponds to aspects found in frameworks for enterprise architectures like GRAAL (van Eck et al. 2004) and ArchiMate (Jonkers et al. 2004). The vertical axis distinguishes three abstraction levels at which a service may need to be modelled. These abstraction levels result from the application of an external and internal system view for service-oriented design (Quartel et al. 2004). This paper focuses on concepts for modelling the behaviour, information and goal aspect of services at each of the identified abstraction levels.

3.1 Service aspects

We structure the properties of services that need to be modelled into the following service aspects:

Structure The structural aspect is concerned with modelling the interacting systems that provide or use services, and their interconnection structure. The interconnection structure comprises amongst others the ports or interfaces at which services are offered.

Behaviour The behavioural aspect is concerned with the activities that are performed by systems, as well as the relations among them. The behaviour of a service consists of the interactions between the service's provider and its users, as well as their causal dependencies or ordering in time. It defines the external behaviour of the service provider partly or completely, depending on whether one or multiple types of services are provided.

Information The information aspect is concerned with modelling the subject domain of systems, representing entities and phenomena in the real world that are known to the system. The value of a service is established through the exchange of information (messages) that has to be interpreted in terms of the subject domain model of the interacting systems.

Goal The goal aspect is concerned with modelling the goal or value of a service. A service provider offers a service that provides some value. Likewise, service users use the service with a particular goal in mind. It is important to make these motivations clear, such that it can be assessed if a service matches the user's goals.

Quality The quality aspect is concerned with modelling the non-functional characteristics of services, i.e., their qualities of service. These qualities often play an important role in the selection of services.

The abovementioned aspects represent partially overlapping, i.e., non-orthogonal views on a service. They overlap, because it is generally impossible to specify one aspect without referring to the other aspects. For example, to specify certain quality characteristics one must refer to the behaviour, and in order to describe the behaviour, it is usually necessary to refer to the information that is processed during the execution of that behaviour.

3.2 Abstraction levels

We distinguish the following generic abstraction levels at which a service can be modelled:

Single interaction At this level, a service is modelled as a single interaction between a service user and provider. The resulting service model defines the value (or goal) that is requested by the service user and the value (or capability) that is offered by the service provider. This model can be used, e.g., to specify or analyse that the goal of some user and the capability of some provider should match.

level of abstraction \ aspect	Structure	Behaviour	Information	Goal	Quality
Single interaction					
Choreography					
Orchestration					

Fig. 1 Structure of the COSMO framework

Choreography At this level, a service is modelled as multiple related interactions between a service user and provider. The resulting service model defines the external behaviour that is requested by the service user and that is offered by the service provider. This model can be used, e.g., to specify or analyse interoperability between the service user and provider.

Orchestration At this level, the service that is offered by some service provider is modelled as a composition of other services. Typically, the resulting service model defines the service provider as a coordinator (also called orchestrator), which interacts with other service providers and combines the values obtained in these interactions to offer some added value to its user. This model can be used, e.g., to specify or analyse a possible implementation of the offered service.

During the development process, a service can be modelled successively at the abstraction levels presented above, such that the choreography refines the model of the service as a single interaction, and the orchestration refines the choreography. Furthermore, these abstraction levels may be applied recursively, since the composed services in an orchestration may at first be modelled as a single interaction, and subsequently be refined into choreographies and orchestrations.

4 Formalising the service concept

We define a *service* as the establishment of some effect through the interaction between two or more systems. Usually one of the involved systems plays the role of service provider and the others play the role of service user. However, this distinction is not essential.

Our service definition closely resembles the ones found in IBM (2006), Wieringa (2003) and Quartel et al. (1997). We assume the effect has or creates some value for one or more of the involved systems and satisfies some goal or accomplishes some desired effect.

This section introduces basic service concepts to model the effect of the service, the interactions, and the involved user and provider roles. These concepts are introduced by considering a service at the three successive abstraction levels introduced in Section 3.2. In this paper, we focus on concepts and not on a notation to express them. Obviously, we need some notation to talk about the concepts, and therefore introduce one at our convenience, borrowing from existing languages. An analysis of the suitability of existing languages to express our concepts is beyond the scope of this paper.

To illustrate the presented concepts, we use an e-procurement example throughout this section. In this example, a customer can purchase an article from some retailer, which is delivered to the user after he has paid for it. In addition, both the customer and retailer may define various conditions on the procurement process, such as pricing, method of payment, location of delivery and the ordering of payment and delivery.

4.1 Service as interaction

At a high abstraction level a service can be modelled as a single interaction between two or more systems. This *interaction* represents an activity in which the involved systems produce some common result in cooperation. At this abstraction level, we are only interested in what result (s) can be produced, and not in how this is done. 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.

The interaction result represents the effect of the service. Each system may have different expectations on this effect, and therefore impose different constraints on the interaction result. This is modelled by defining an interaction as the composition of two (or more) interaction contributions, one for each involved system. An *interaction contribution* represents the participation of a system in the interaction, by defining the constraints this system has on the possible interaction result, and thereby its responsibilities in performing the interaction.

Figure 2 models the example procurement service as a single interaction between a customer and a retailer. Interaction contributions buy and sell represent the participation of the customer and retailer in this interaction, respectively. The associated text boxes define the constraints they each have on the interaction result, using a notation based on description logic (Baader et al. 2003). In this case, both the customer and retailer want to establish a purchase as the interaction result. The customer wants to order a notebook, whereas the retailer is willing to sell any article from its catalogue. Furthermore, the customer wants the notebook to have a maximum price, to be paid using

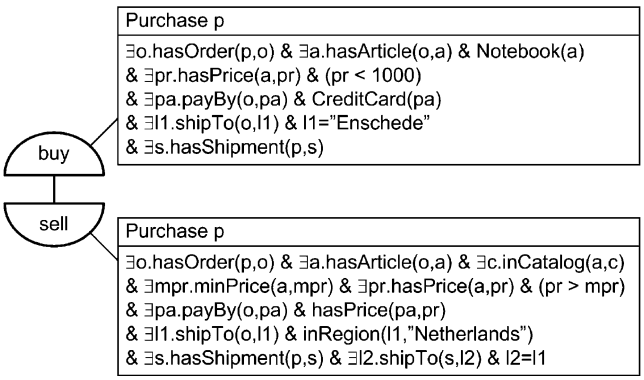

Fig. 2 Procurement interaction

credit card, and to be delivered in "Enschede." The retailer only sells articles from its catalogue, has specified for each article a minimum price, and will only deliver at addresses in the "Netherlands."

The purchase interaction can only occur if the constraints of both the customer and the retailer can be satisfied. In case multiple results are possible that satisfy the constraints, e.g., multiple notebooks may have the required properties, only a single result is established. Since the interaction concept abstracts from how to select the result, the result is assumed to be selected non-deterministically.

4.1.1 User and provider roles

We use the term system in its general meaning, representing, e.g., people, organizations, software applications or hardware systems. A system may be involved in multiple services, and may even act as a user of one service and a provider for another. Therefore, we cannot say that a system is either a service provider or a service user. Furthermore, the specific system that provides some service may not be known at design time or even at discovery time. For these reasons, we currently do not model the involved systems explicitly. Instead, we model the *role* of the system in a service, where we distinguish two roles: the *user role* and the *provider role*. Since we use behavioural concepts to model roles, the structural service aspect as described in Section 3.1 will not be considered in this paper. The structural aspect becomes important again when we want to create a deployment model.

In the example of Fig. 2, the interaction contributions represent the constraints of the customer and retailer roles, respectively. The model leaves open which of these is the provider and the user.

The user and provider roles define two complementary perspectives on a service, which we denote as the user and the provider perspective, respectively. The *user perspective* defines the participation of the user in the service, representing the expectations the user has on the effect, and thus on the service provider. This partial definition of the service is also called the *requested service*. The *provider perspective* defines the participation of the provider role, representing the expectations it has on the user. This partial definition of the service is also called the *offered service*.

A third perspective we use is the so-called *integrated perspective*, which defines the joint behaviour of the offered and requested service. The action concept is used to represent a joint activity (integrated interaction), by abstracting from the distinction between the user and provider roles. We do not consider this perspective further in this paper, but refer for an explanation to Quartel et al. (2006).

Besides the user and provider roles, one may think of additional (supporting) roles of systems that are involved in a service to facilitate the interaction between a user and the provider. For example, supporting systems may be involved to monitor QoS metrics (e.g., availability and response time) or to perform security related message transformations (e.g., encrypt/decrypt messages). The question is whether these roles should already be considered in an abstract service model, or can be abstracted from and considered later when refining the provider (and user) roles. In the latter case, the supporting functionality could for example be considered as separate services and modelled using the user and provider roles, or be considered part of the service platform that implements the service interactions. This question is subject of our future work.

4.1.2 Modelling activity results

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 them.

Figure 3 depicts part of a simple information model for the procurement example. This model does not include individuals and the valuations of their properties, which together we call the *state* of a system.

Activity results can be represented using aforementioned information modelling concepts. For this purpose, a so-called *information attribute* is associated with an activity. This attribute has a type and will be assigned a value when the activity occurs. The value is an individual that represents the activity result. The type is a class that represents the possible set of activity results. For example, interaction contribution buy in Fig. 2 has a single information attribute p of type Purchase.

In addition, a so-called *result constraint* can be defined on an information attribute to constrain its possible values. This result constraint is a predicate that states the properties that have to be satisfied by the individual that represents the activity result. For example, the result constraint of interaction contribution buy in Fig. 2 specifies that the result can only be a purchase that has an order for a notebook that costs less than 1,000, can be paid using credit card, and can be delivered in "Enschede." Such a result constraint can also be seen as the goal of the customer.

4.2 Service as choreography

In general, a service cannot be implemented as a single interaction and we have to refine the abstract interaction into a structure of multiple smaller more concrete interactions.

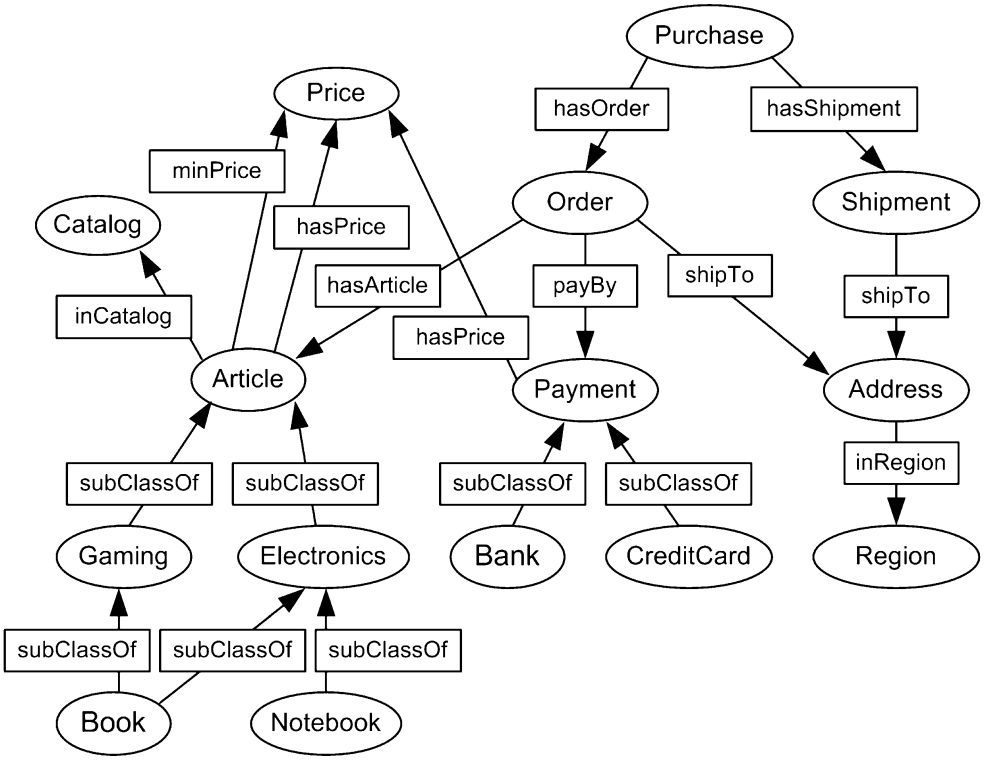

Fig. 3 Procurement information model

Figure 4 depicts a possible refinement of the procurement service from Fig. 2 into a number of interactions: select represents the selection of an article, checkout represents the establishment of the delivery address, pay represents the payment of the order, and deliver represents the order delivery. Interaction pay is actually defined as two alternative interactions: one involving interaction contributions pay and pay1 and the other involving interaction contributions pay and pay2.

The retailer offers the possibility to pay by bank transfer through interaction contribution pay1, or by credit card through interaction contribution pay2. In addition, the retailer allows credit card payment only if some *precondition* is satisfied, i.e., the price of the selected article must be larger than 500. Observe that also contributions pay1 and pay2 refer to results established in the causally preceding contribution select, i.e., the price of the article. Interaction contribution deliver refers to the address established in contribution checkout.

To represent multiple related activities, the *behaviour* concept is introduced, here graphically expressed as a rounded rectangle. 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. For example, the behaviours in Fig. 4 consist of interaction contributions, whereas a behaviour representing the integrated perspective would consist of actions.

The definition of a service as a set of related interactions is called a *choreography*. A choreography defines the external behaviour of the user and provider role, and abstracts from any internal activities.

Similar to a service that is defined by a single interaction, a choreography can be considered from a user, provider and integrated perspective. From a provider perspective, the offered service defines the relationships between and constraints on the interactions in terms of the contributions

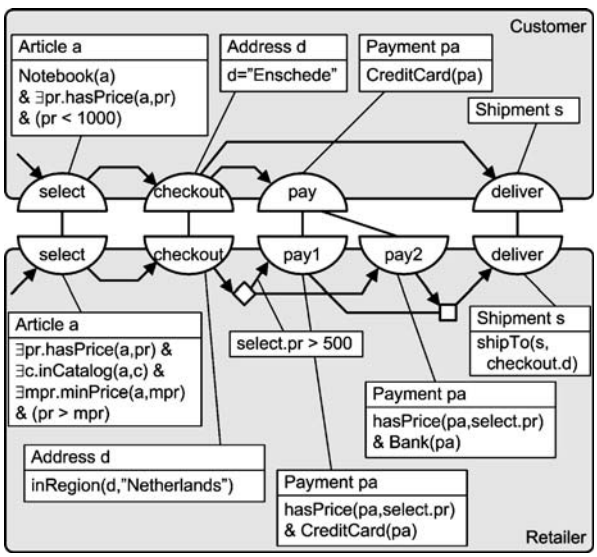

Fig. 4 Procurement choreography

of the service provider. Analogously, the requested service defines these relationships and constraints from a user perspective. For example, behaviours Customer and Retailer in Fig. 4 represent the requested and offered service (choreography), respectively, for the procurement service. The example from Fig. 2 already showed that the user and provider roles may define different result constraints. Figure 4 shows that the user and provider role may also define different relationships between the interactions. For example, the retailer wants the order to be paid before it is delivered, whereas the customer allows the payment and delivery to occur independently.

4.2.1 Modelling relations between 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 relates each activity to a 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 of some activity a are distinguished (see Fig. 5): (1) enabling condition b represents that activity b must have occurred before a can occur; (2) disabling condition ¬b represents that activity b must not have occurred before nor simultaneously with a to enable the occurrence of a; (3) the start condition represents that activity a is enabled from the beginning of the behaviour and is independent of any other activity. These basic conditions can be combined using the conjunction and disjunction operators to represent more complex conditions. For example, workflow operators such as and-join, and-split and or-split can be represented using a combination of enabling and disabling conditions.

4.2.2 Interfaces

A choreography can be structured into multiple smaller, related choreographies representing groupings of interactions. Typically, such a structuring is based on grouping interactions that have strong functional relationships, and separating interactions that have weaker relationships. The aim of this structuring is to increase clarity and comprehensibility of the service definition, to facilitate its mapping onto an implementation, and to separate required from optional functionality. For example, the identified groupings may represent suitable units of functionality for searching and selecting existing services or for defining new services that implement part of the required service functionality.

Figure 6 depicts an example of a structure of choreographies. In this example, each interaction from the procurement choreography is split into two sub-interactions, a Request followed by a Response, such that the result of the response conforms to the result of the original interaction. For example, payReq represents a request to perform a payment, and payRsp represents the response that informs about the outcome of the payment activity. This type of refinement is needed if one wants to implement the payment interaction using one or more other services; see also Fig. 7 in Section 4.3. In addition, interaction select is further refined by introducing a preparatory interaction catalog in which the user can request for a catalogue of articles, followed by an interaction pick in which an article is selected.

Sub-choreographies are defined as separate behaviours. To represent the causal dependencies between these behaviours, so-called entry and exit points (represented as triangles) are used. For brevity, only the offered choreography is shown and the result constraints have been omitted.

We use the term (requested and offered) *interface* as a synonym to (requested and offered) choreography. So, in contrast to current practice, interfaces should also define the relationships between interaction contributions (e.g., operations). Furthermore, a service definition comprising multiple interfaces should also define the relationships between (the interaction contributions from) these interfaces.

4.3 Service as orchestration

Besides the refinement of interactions, one may want to refine a service into a composition of smaller services in

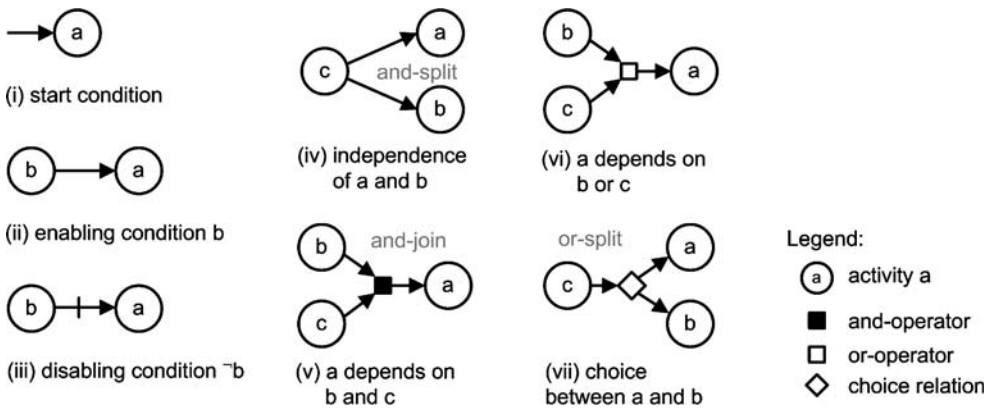

Fig. 5 Causality conditions

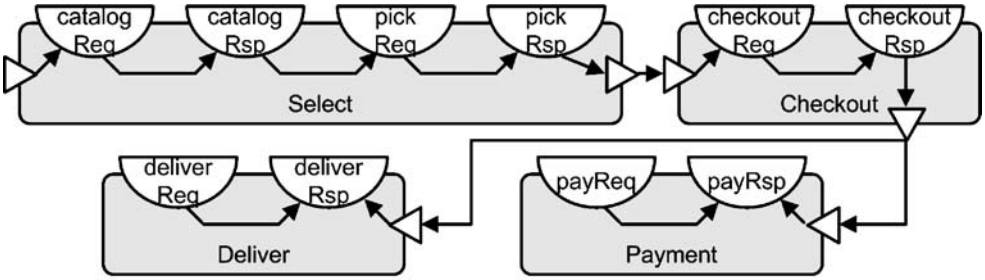

Fig. 6 Structured choreography

order to obtain an implementation of the service. Figure 7 depicts an example of the refinement of the offered procurement choreography from Fig. 6 into a number of services: a Shopping service that allows one to select and order articles, a Payment service that handles payments, a Shipping service that delivers articles, and a Coordination service that coordinates the use of aforementioned services to provide the procurement service.

The Shopping, Payment and Shipping services are all offered services. The Coordination service refines the offered procurement choreography by inserting requested services between the procurement interaction contributions. These requested services are used to implement parts of the procurement choreography. In principle, the Coordination service might implement part of the procurement functionality as well, e.g., order handling. However, in many cases it is considered good practice to provide such functionality by separate services, making the coordination service primarily responsible for coordinating and combining the results of the requested services. This coordination pattern helps to maintain loose coupling between the offered services.

The definition of a service as a composition of smaller services, including a coordination service, is called an *orchestration*. In the example above, the orchestration is defined as a composition of requested and offered services, i.e., each service is defined from a user and/or provider perspective. Observe that the procurement interactions have been refined into request and response interactions to model their implementation using other services. In contrast, the interactions of the sub-services don't need this refinement (yet), since the orchestration abstracts from their implementation.

4.3.1 Behaviour composition

A service orchestration is defined as a behaviour that is composed of sub-service behaviours. Containment of one behaviour by another (the composite), is represented by behaviour instantiation. A behaviour instantiation represents that some behaviour instance is created in the context of the behaviour that contains the instantiation. For brevity, in the examples of this paper, a behaviour and its instantiation have been represented as one. However, normally they should be represented separately.

Behaviours in a composite behaviour can be related using constraint-oriented composition and/or causality-oriented composition.

Constraint-oriented composition is used to define two or more interacting behaviours. This composition technique is based on the interaction concept, which decomposes an action into an interaction that consists of two or more interaction contributions. These contributions define the participation of different behaviours in the interaction, which may impose different constraints on the possible interaction results. This allows for an abstract style of service specification and design, i.e., in terms of constraints, thereby abstracting from how these constraints are satisfied by some implementation. Figures 4 and 7 present examples of constraint-oriented composition.

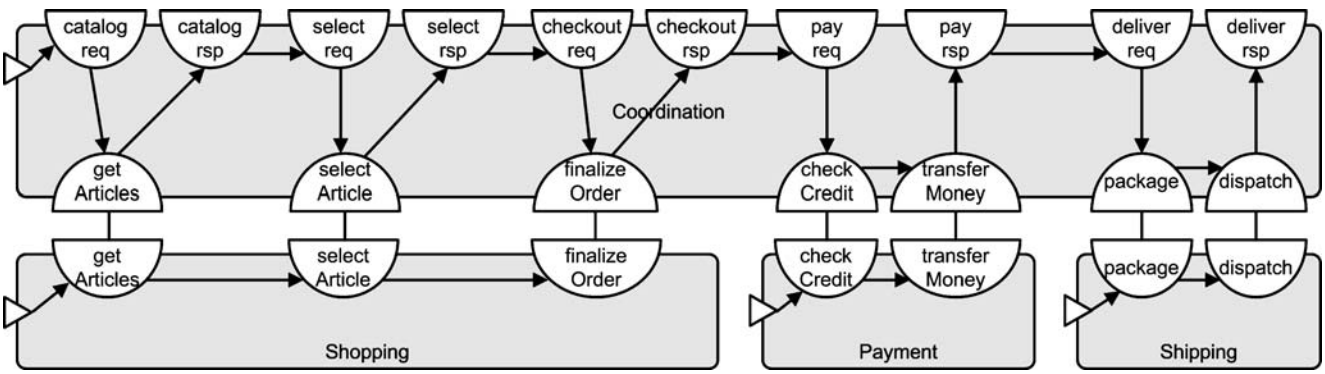

Fig. 7 Procurement orchestration

Causality-oriented composition is used to define causal dependencies between behaviours. This composition technique is based on the decomposition of a causality relation, such that an activity and its causality condition can be defined in separate behaviours. For this purpose, entry and exit points are used, which represent causality conditions entering and exiting a behaviour, respectively. Like a causality relation associates a causality condition to an activity, an entry point dependency associates a causality condition to an entry point. Entry and exit points are represented by triangles that point into or out of a behaviour, respectively. Like activities, points can have 'attributes', which are called parameters. Figure 6 presents an example of causality-oriented composition.

5 Conceptual models

This section gives an overview of the concepts that have been introduced in the preceding sections. For this purpose a number of meta-models are presented. The first meta-model concerns concepts related to the abstraction levels at which services can be defined and the possible roles involved. The subsequent meta-models concern the service aspects identified in Section 3.1, except for the structural aspect and the quality aspect. The meta-models are represented using UML class diagrams.

5.1 Abstraction levels and roles

The meta-model in Fig. 8, defines concepts that are used to denote distinct types of service models, varying in abstraction levels and roles being considered.

A service is defined as a choreography consisting of one or more interactions, which includes the possibility to define a service as a single interaction. Since each interaction can be further decomposed into a choreography, a choreography can be composed from sub-choreographies. Orthogonal to this composition, a choreography can be defined as a composition of multiple interfaces, which are defined as compositions of interaction contributions or sub-interfaces.

An interface represents the role of a system involved in the choreography. Two types of roles are distinguished: a user and a provider role. Interaction contributions from the same interface should all be associated with the same role. The terms requested and offered service are used to denote an interface representing the user and provider role, respectively.

An orchestration refines an offered service. An orchestration consists, on the one hand, of the interfaces of the offered service and, on the other hand, of a number of choreographies representing the offered service's usage of sub-services.

5.2 Behavioural aspect

The meta-model in Fig. 9 defines the concepts used to model the behavioural aspect.

The behavioural aspect of interactions, choreographies and orchestrations are modelled as behaviours that may be composed of other behaviours using constraint-oriented and/or causality-oriented composition. The concept of role is also mapped onto the behaviour concept.

5.3 Information aspect

The meta-model in Fig. 10 defines the concepts used to model the information aspect.

The information modelling concepts we use are borrowed from description logic (Baader et al. 2003). Individuals represent entities from the real world. Classes represent abstract types of entities from the real world. And properties represent relationships between entities.

Individuals are classified into classes. One individual can have multiple types, e.g. a Ferrari car can both be classified as a Vehicle and as a RedThing. Properties are also classes

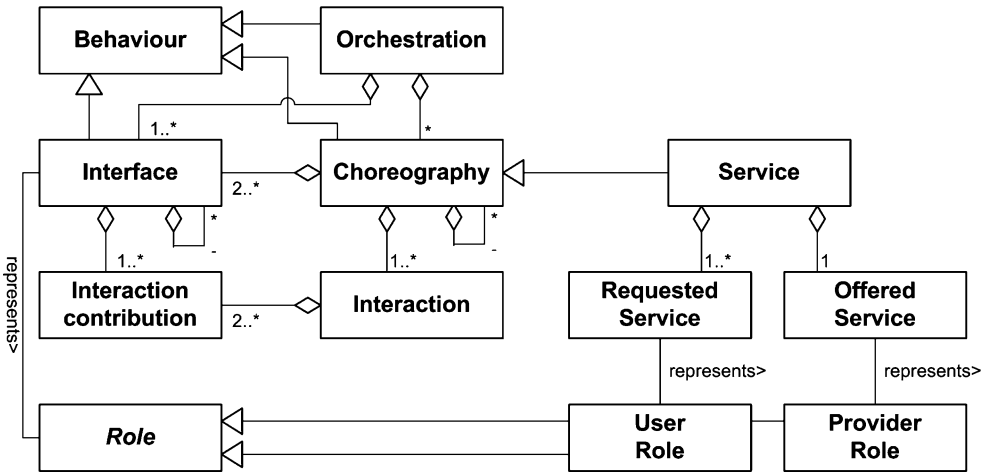

Fig. 8 Service concepts

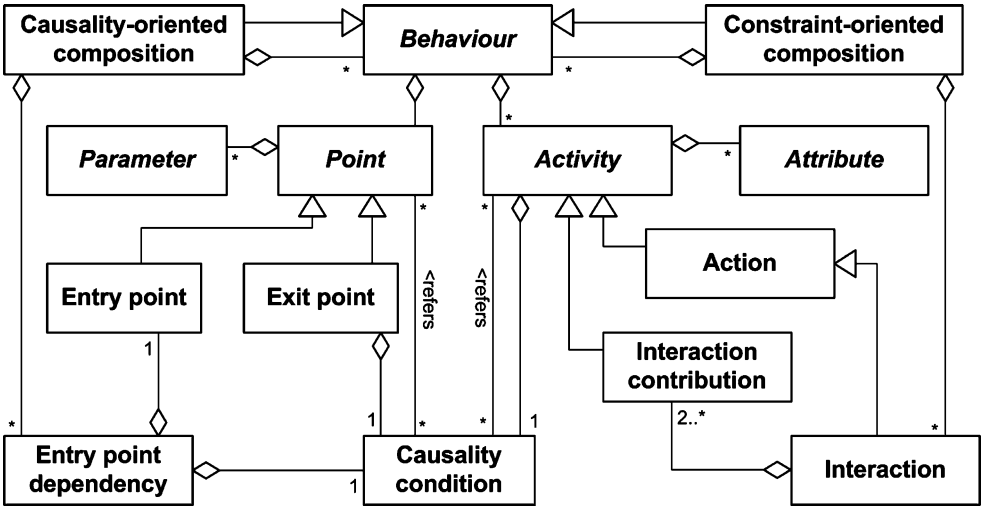

Fig. 9 Behavioural concepts

defining relations between one or more domain classes and one ore more range classes.

The information model is linked to the behavioural model in the following ways. Each activity has a precondition and a result constraint. The precondition is a predicate, which defines the class of "states of affair" in which the activity is enabled. When an activity occurs it produces a result, which satisfies a predicate representing the result constraint. In other words, the result is an individual belonging to the class of admitted results defined by the result constraint.

5.4 Goal aspect

Service users and providers request and offer services, respectively, with a particular goal in mind. Analogously to Lamsweerde (2001) and other requirements engineering literature (Yu 1997), we define a goal to be a specification of the properties that need to be ensured, i.e., a specification of the desired future state of affairs. As such a goal corresponds one-to-one to our concept of result constraint and since a service can always be abstracted to a single interaction between a user and a provider, we can represent their goals as result constraints on their respective interaction contributions. In addition, we use the term desired effect as a synonym for goal and result constraint, and the term effect is a synonym for result.

The definition of the user and provider goal should provide a high-level description of the service that facilitates the discovery of services. For this purpose, the abstraction level at which a service is modelled as a single interaction seems to be suitable. For example, interaction contribution pay of behaviour Customer in Fig. 4 defines the user goal to perform some payment using credit card. Based on the correspondence of result types, two provider goals are candidates to match the user goal, i.e., the result constraints defined by interaction contributions pay1 and pay2 from the Retailer. Whether a provider goal matches the user goal can be determined by checking if the conjunction of the predicates that represent these goals

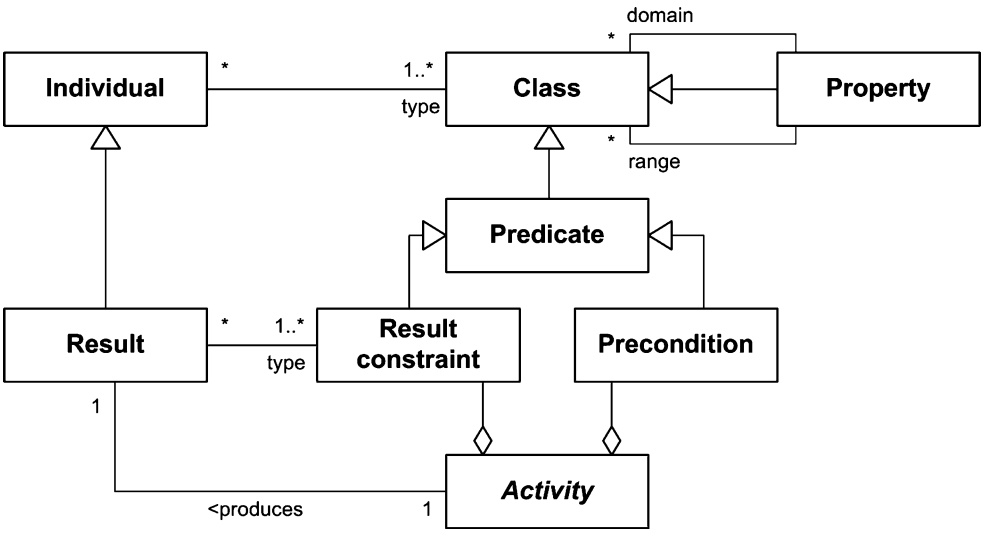

Fig. 10 Information concepts

admits any results. In this case only the provider goal defined by interaction contribution pay1 matches the user goal, since it allows a credit card payment.

6 Refinement

During the service lifecycle, multiple models of a service may be used at distinct abstraction levels. Section 3.2 introduced three generic abstraction levels. The associated service models are related through a so-called refinement relation. For example, the orchestration in Fig. 7 is a refinement of the offered choreography in Fig. 4, which is again a refinement of the single interaction contribution sell in Fig. 2.

In order to assess the consistency, or conformance, between service models at different abstraction levels one should be able to assess the correctness of the refinement relation between these models. This allows one, for example, to assess if an orchestration correctly implements some choreography, or to assess if a choreography correctly supports some goal that is represented as a single interaction (contribution). This section describes the elements of a method to assess the conformance between service models, illustrated by examples.

6.1 Conformance criteria

A service model is considered a refinement of another service model, if the former model defines additional properties of the service, while preserving the properties defined in the latter model. The opposite of refinement is abstraction, which constitutes the process of removing properties. Accordingly, we call the latter model a (more) abstract model and the former model a (more) concrete model. For example, the service model in Fig. 2 is more abstract than the model in Fig. 4, since the former model only defines what the goal (effect) of the service is, while the latter model adds details about how this goal is obtained.

We say that a concrete service model *conforms* to an abstract service model, if the concrete model is a correct refinement of the abstract model. For example, the concrete model in Fig. 4 conforms to the abstract model in Fig. 2 if it preserves the properties of the abstract model, such that both define the same goal.

Depending on the strictness of the requirement to preserve the abstract service properties during refinement, distinct refinement (conformance) relations can be identified. For example, one may require that all possible instances of the goal as specified in Fig. 2, i.e., all possible purchases, should be made possible by the choreography in Fig. 4. Alternatively one could relax this requirement by stating that the choreography may support a subset of these purchases. For convenience, we assume a strict interpretation of refinement in the rest of this section. However, the presented ideas also apply to less strict interpretations.

6.2 Assessment approach

Figure 11 illustrates our approach to assess the correctness of the refinement relation, i.e., the conformance, between an abstract and a concrete model of some service.

The approach consists of two steps. The first step determines the abstraction of the concrete service by abstracting from the service properties that were added in the refinement step. The second step compares this abstraction to the original abstract service by checking the equivalence between both abstract models. The refinement is considered correct if both models are equivalent. Otherwise the refinement is considered incorrect.

The idea underlying this approach is that the abstraction of some model is unique, assuming the properties one wants to abstract from are known. This implies that the obtained abstraction should be equivalent to the original abstract model after all properties have been removed that were added in subsequent refinements of this model.

Alternatively, one could follow an approach that constrains the refinement step, e.g., in terms of design guidelines. However, a model can in principle be refined in many different ways. This makes it difficult, if not impossible, to define constraints that can deal with all possible refinements. Instead, one may pre-define and use particular types of refinements based on specific design choices. This may limit however the creativity and design freedom of the service developer.

In order to perform the abstraction step in Fig. 11 we need so-called abstraction rules that define how to abstract from service properties. Below we consider the abstraction of behaviour and information properties.

6.3 Behaviour abstraction

A behaviour consists of one or more related activities, as defined in Section 4.2. An activity of an abstract behaviour is called an *abstract activity* and an activity of a concrete behaviour is called a *concrete activity*. We assume that the

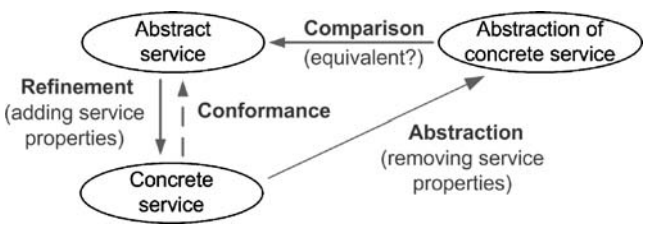

Fig. 11 Conformance assessment approach

occurrence of each abstract activity corresponds to the occurrence of one or more concrete activities. This assumption allows us to relate an abstract behaviour to a concrete behaviour in order to assess conformance. A concrete activity that corresponds to an abstract activity is called a *reference activity*, since it is considered a reference point in the concrete behaviour for assessing conformance. A concrete activity that is not a reference activity is called an *inserted activity*, since it is inserted during behaviour refinement.

Two elementary types of behaviour refinement are distinguished: activity refinement and causality refinement.

Activity refinement allows one to model in more detail a real-world activity that is represented by a single abstract activity. This activity is decomposed into a concrete activity structure, which consists of multiple related, more concrete (sub-)activities. The concrete activity structure makes its result available through the occurrence and associated attributes of one or more of its final activities, which are the reference activities that correspond to the original abstract activity. A concrete activity structure can make its result available through the occurrence of (1) a single final activity, (2) a conjunction of multiple, independent final activities, (3) a disjunction of multiple, alternative final activities, or (4) a combination of these options.

An example of a concrete activity structure having a disjunction of final activities is formed by interaction contributions pay1 and pay2 of behaviour Retailer in Fig. 4. We assume this activity structure refines a single abstract interaction contribution pay', as depicted in Fig. 12. The result of pay' is defined as the union of the possible results of pay1 and pay2, i.e., pay' represents all possible payments, which constitute bank payments or credit card payments having an amount larger than 500.

An example of a concrete activity structure having a conjunction of final activities is formed by interaction contributions pay and deliver of behaviour Customer in Fig. 4. We assume this activity structure refines a single abstract contribution purchase', as depicted in Fig. 13. The result of purchase' is defined as the intersection of the possible results of pay and deliver, i.e., the conjunction of

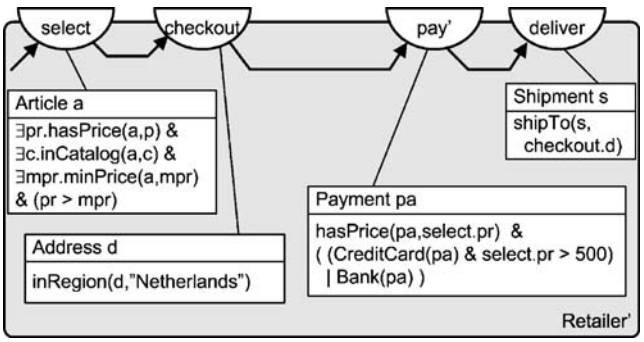

Fig. 12 Abstraction of pay1 and pay2 in behaviour Retailer in Fig. 4

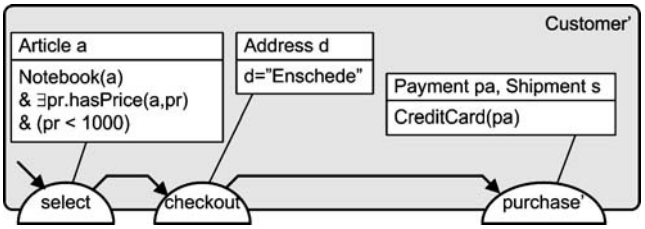

Fig. 13 Abstraction of pay and deliver in behaviour Customer in Fig. 4

their result constraints. This means that purchase' represents all possible purchases that have been paid and delivered.

An example of a concrete activity structure having a single final activity is formed by interaction contributions select, checkout and pay' of behaviour Retailer' in Fig. 12. We assume this activity structure refines a single abstract interaction contribution order', as depicted in Fig. 14. Activity order' represents the result of the ordering process as a whole. Instead, inserted activities select and checkout represent intermediate results in the ordering process. Reference activity pay' represents the result of the whole ordering process at a more concrete level, which constitutes the result of pay' itself, i.e., the payment, and the (relevant) intermediate results it can refer to because it depends on the occurrence of the inserted activities. Consequently, the result of abstract activity order' constitutes the intersection of the possible results allowed by select, checkout and pay'. Observe that the attribute reference checkout.d by interaction contribution deliver in Fig. 12 has to be replaced by reference order'.d.

Inserted activities, such as select and checkout, may be abstracted from one by one in arbitrary order. For example, one may consider activity order' again as an inserted activity and activity deliver as a reference activity, forming a concrete activity structure having a single final activity. Similarly to the previous example, this activity structure can be abstracted into a single abstract activity purchase', as depicted in Fig. 15. In this case, purchase' represents the result of the whole purchasing process.

Causality refinement allows one to model the relations between abstract activities in more detail through adding inserted activities. Abstract activities are not further detailed, and therefore correspond to a single reference activity.

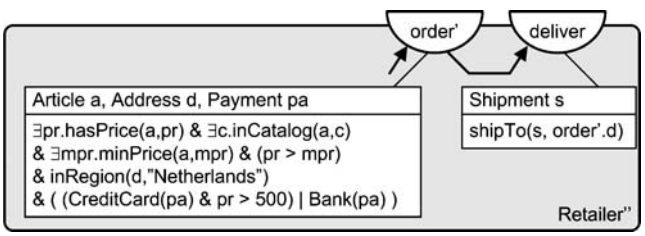

Fig. 14 Abstraction of select, checkout and pay' in behaviour Retailer' in Fig. 12

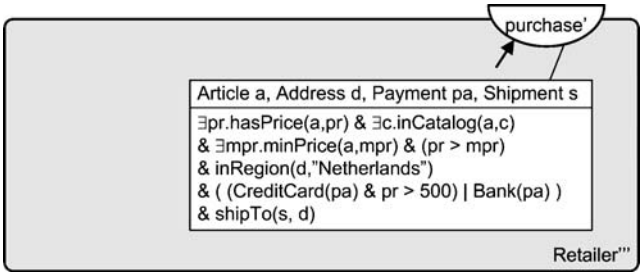

Fig. 15 Abstraction of order' and deliver in behaviour Retailer'' in Fig. 14

Causality refinement should obey the following conformance criteria: (1) an indirect relation between reference activities defined via an inserted activity in the concrete behaviour must be equivalent to the relation defined directly between the corresponding reference activities in the abstract behaviour; (2) similarly, an indirect relation between attributes must be equivalent to the direct relation.

For example, interaction contribution checkout in behaviour Customer' in Fig. 13 may be considered an inserted activity that is used to refine the relation between interaction contributions select and purchase'. In this case the refinement constitutes the establishment of the delivery address as an intermediate result. Figure 16 depicts the behaviour that results after abstracting from inserted activity checkout. In this case, the indirect relation between reference activities select and purchase' has to be replaced by a direct relation.

Abstraction rules The above examples illustrate some of the rules we use to obtain the abstraction of a concrete model to assess the conformance between the concrete model and an abstract model. A complete and precise set of abstraction rules for activity and causality refinement has been defined and presented in earlier work (Quartel et al. 2002).

6.4 Information abstraction

Information is used to model the result (effect) of some activity, as explained in Section 4.1.2. This strong association between the information and behaviour aspect suggests that behaviour refinement and information refinement are closely related. However, the types of behaviour refinement considered in Section 6.3 can be applied while the abstraction level at which activity results are represented remains unchanged. For example, consider abstract activity pay' in Fig. 12 of which the result is modelled as a collection of partial results. Using activity refinement this activity can be decomposed into a disjunction of alternative final activities pay1 and pay2, each establishing one of the partial results, such that the union of these partial results equals the original collection (see Fig. 4). As another example, consider the refinement of abstract activity order' in Fig. 14. This activity can be decomposed into inserted activities select and checkout, which establish as intermediate results the ordered article and the delivery address, and reference activity pay', which establishes the payment. In this case, the result of abstract activity order' is decomposed into partial results, which are distributed over the results of inserted activities select and checkout and reference activity pay', such that the result of order' equals the conjunction of the partial results. Furthermore, the result of order' reveals the same level of information detail as the partial results in the refinement.

Therefore, we conclude that independently of behaviour refinement, one may want to apply information refinement to model activity results in more detail. Two elementary types of information refinement are distinguished: specialization and aggregation.

Specialization allows one to refine an abstract activity result of abstract type TA into a concrete activity result of concrete type TC, such that TC is a sub-type of TA. In terms of the associated information model this means that class TC is defined as a sub-class of class TA.

For example, the information model in Fig. 3 defines an abstract type Payment and two more concrete types Bank and CreditCard payments. Therefore, a more abstract representation of the result of activity pay' in Fig. 12, would be to use type Payment to abstract from the kind of payments that can be made, as illustrated in Fig. 17a. In order to assess whether the concrete activity result of pay' conforms to the abstract activity result of pay'', one should first check whether a subclass relationship exists between the concrete and abstract result type. This is illustrated in Fig. 17b, where the possible concrete activity results are represented by a separate class aClass, and reasoning capabilities of the ontology language should help to establish a subsumption relationship between this class and the Payment class. As a second check, one may want to establish that all possible results of Payment are preserved by aClass, i.e., Payment is a subclass of aClass, in case one adopts a strict conformance requirement. In this example, the strict requirement does not hold, since credit card payments smaller than 500 are not allowed by pay'.

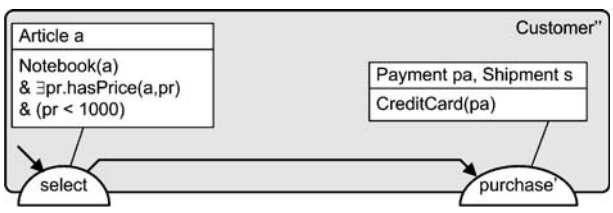

Fig. 16 Abstraction of checkout in behaviour Customer' in Fig. 13

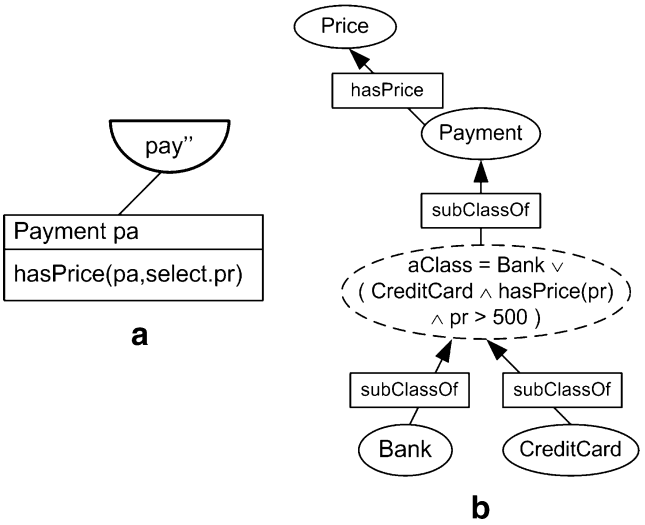

Fig. 17 Abstraction of special types of payment

Property aggregation allows one to refine an abstract activity result of abstract type TA into a collection of concrete activity results of concrete types TC_i, such that each TC_i represents a property of TA. In terms of the associated information model this means that class TC_i represents the range of a property relation (different form the sub-class relation) between TA and TC_i.

For example, the information model in Fig. 3 defines type Order as having the properties Article, Address and Payment. Therefore, a more abstract representation of the type of the result of activity order' in Fig. 14 would be to use type Order, to abstract from the constituents of an order, as illustrated in Fig. 18a. In order to assess whether the concrete activity result of order' conforms to the abstract activity result of order'', we define class aClass that represents all possible individuals having properties has Article, payBy and shipTo. Adopting the open world assumption, one cannot determine that this class is a subclass of Order, because one can not exclude that individuals exist, not being of type Order, that have forementioned properties. Instead, one can reason that Order is a subclass of aClass, meaning that all possible orders can be represented in the refinement. Therefore, to assess that Order is the correct abstraction of the concrete activity result, one needs to use knowledge of the refinement step (as explained in Section 6.2), i.e., that abstract type Order has been refined.

Observe that one may also decide to abstract from some of the properties. For example, Fig. 18c presents abstract activity order'' that abstracts from properties hasArticle and payBy, but not from property shipTo. This requires one to add ∃d. shipTo(o,d) to the result constraint. Alternatively, one could decide not to abstract from any of the properties, but just represent the abstract result type as being of type Order (instead of the conjunction of Article, Address, Payment).

Analogously to the previous example, one may define an abstraction of activity purchase' in behaviour Retailer''' in Fig. 15, such that its result is defined in terms of abstract type Purchase. This allows one to compare the obtained abstraction to the original definition in behaviour Retailer'' in Fig. 2, in order to establish conformance between the choreography of the retailer in Fig. 4 and the single interaction contribution in Fig. 2.

Abstraction rules The above examples illustrate some of the rules we use to obtain the abstraction of a concrete model involving information refinement. However, in contrast to the behaviour aspect, a complete and precise set of abstraction rules has not been defined yet and is subject of current research.

6.5 Example

As an example of the application of the approach depicted in Fig. 11 consider that one wants to assess if the offered choreography of the Retailer in Fig. 4 conforms to the offered service represented by the single interaction contribution sell in Fig. 2. In this case, conformance can be shown by the sequence of abstraction steps as represented by Figs. 12, 14, 15, 17 and 18, and finally renaming purchase' into sell. However, alternative steps and sequences of steps are possible. For example, the abstraction step represented by Fig. 17 could have been performed directly after the step illustrated by Fig. 12.

7 Application of the COSMO framework

The purpose of the COSMO framework is to serve as a common semantic meta-model to enable the use of different service modelling languages. Figure 19 depicts the role of this common semantic meta-model.

In general, different languages are needed to support the creation of services. Distinct design and specification lan-

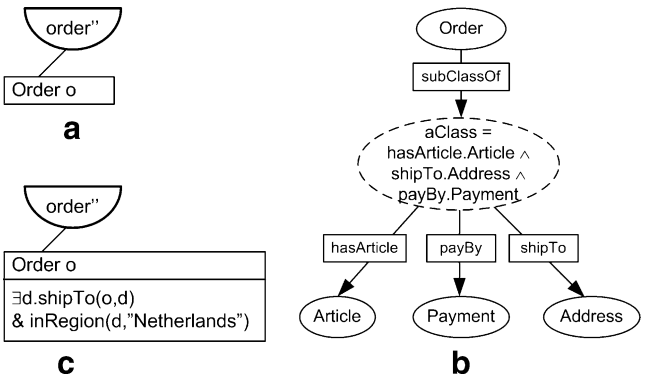

Fig. 18 Abstraction of order properties

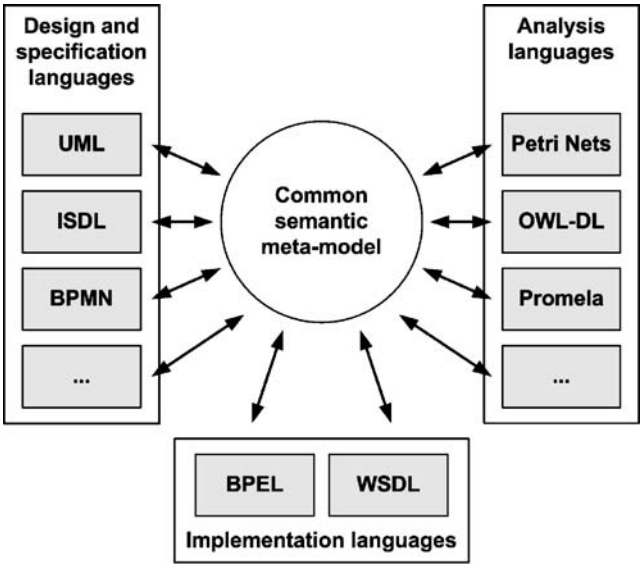

Fig. 19 Role of common semantic meta-model

guages may be suitable for modelling distinct service aspects (see Section 3.1). Distinct analysis languages may be used dependent on the type of analysis one wants to perform. And the choice of implementation languages largely depends on the specific service platform that is used.

The models produced by these languages have to be related, for example, to verify consistency or conformance. The approach we follow to facilitate the comparison of models, is by mapping them onto the concepts of the common semantic meta-model. In this way, techniques that are defined to verify consistency or conformance on instances of the common semantic meta-model, can be re-used to verify the relationships between models produced by different languages. An alternative would be to define such techniques for each pair of languages being used, which is likely to be less efficient and more complex (Dijkman et al. 2004; Dijkman and Dumas 2004).

An important requirement on the common semantic meta-model is to capture all elementary service properties that are relevant during the service creation process. Based on earlier work and experience, we propose the concepts in Section 4 as a starting point for developing a service creation environment in which multiple languages can be applied to support various design tasks consistently during the creation process. This environment is currently being built in the A-MUSE project (A-MUSE 2006), including the required mappings from the applied languages onto the common semantic meta-model. Application of the environment on various cases must demonstrate the applicability and completeness of the proposed basic service concepts. In particular, the service creation environment should support operations on services such as their specification, composition and discovery.

Service specification is typically performed using some existing modelling language, e.g., UML activity diagrams, ISDL (Quartel et al. 2004) or BPMN (BPMN 2006). The hypothesis is that our basic design concepts are sufficiently generic and elementary, such that the concepts from these languages used for service modelling can be mapped onto compositions or specializations of the basic concepts. For ISDL this mapping is straightforward because of the strong resemblance between the basic concepts and the concepts underlying ISDL. Also for UML activity diagrams a mapping can be defined, based on earlier work reported in (Quartel et al. 2005a). However, to validate the hypothesis we aim at defining more mappings for common specification languages (such as BPMN 2006) and domain specific languages (such as the ones used in the A-MUSE project). In the latter case, i.e., when using languages that model domain specific aspects, some information may get lost in the mapping onto the common semantic meta-model. In case such information is essential for the particular design at hand, stereotyping can be used to extend the common semantic meta-model (Quartel et al. 2005b).

Service composition is supported by identifying two generic and elementary techniques to compose services, i.e., constraint- and causality-oriented composition. Also here the hypothesis is that structuring or composition constructs from existing languages can be mapped onto those techniques. An example of a validation of this hypothesis is provided by a method that has been developed to verify the interoperability (or composability) of two or more services that is based on our basic concepts, in particular the concept of constraint-oriented composition (Pokraev et al. 2006). This method applies a mapping from the basic concepts on Petri Nets, using work from (Van Sinderen et al. 1995), to perform reachability and deadlock analyses. Current work focuses on providing methods to (semi-)automatically construct mediators that resolve identified interoperability problems (Pokraev et al. 2007). This work uses two language mappings: a mapping between our information concepts and OWL, and a mapping between our behaviour concepts and BPEL/WSDL, which is described in (Dirgahayu 2005).

Concerning service discovery, (Klein and Bernstein 2004) describes four existing approaches towards service retrieval (in order of increasing recall and precision): keyword-based, property table-based, concept-based and deductive retrieval. A limitation of the keyword and property-table based approaches are their low recall and precision. The deductive approach seems to be too difficult to apply in practical situations and has a high computational complexity. A disadvantage of concept-based approaches is the effort needed to build an ontology, and that a single ontology will probably not suffice, requiring multiple ontologies to be related. In order to deal with these limitations, Klein and

Bernstein (2004) propose a so-called process-based approach that captures information about the service behaviour. This approach is claimed to obtain better results than the concept-based approach.

Using the concepts from our framework, we claim that service models can be produced that support concept- and process-based retrieval approaches. This claim is based on the observation that the kind of information that is required as input to these approaches can be modelled using our concepts. The goal concept seems particularly suitable to be used as input to concept-based approaches, since interaction results are represented in terms of concepts from the subject domain of the involved systems. Instead, choreographies and orchestrations seem particularly suitable for the process-based approach.

8 Related work

There are some ongoing related efforts in creating a conceptual framework for service modelling. A prominent example is the W3C's Web Services Architecture (WSA), which provides a conceptual model for understanding Web services and related concepts (W3C 2004). Although we cannot cover as broad a range of concepts as the WSA in the scope of this paper, we claim that our service concept is more general than the Web services considered by W3C. Our service concept could be realized as Web services, but also as economic exchanges or human–computer interaction. Another extension of our work with respect to the WSA is that we provide (behavioural) semantics to the service concept. For semantics the WSA refers to OWL-S (see below).

Colombo et al. (2005) propose a conceptual model that describes actors, activities and entities involved in a service-oriented scenario and the relationships between them. This work is similar to and partially extends the W3C's Web Services Architecture. However, they also omitted the specification of the semantics of the concepts described. As it stands their conceptual model is a glossary of terms without an indication of how the various concepts could be expressed in a concrete modelling language.

OWL-S (since its first incarnation as DAML-S) represents one of the first attempts at formalizing the semantics of Web services using ontology technology (Martin et al. 2004). In OWL-S a service is formally described by a Service Model. The Service Model shows the possible steps required to execute a service. It describes a service in terms of its *inputs, preconditions, outputs, effects*, and, where appropriate, its component sub-processes. It also describes the control flow in terms of the service's *state*, including initial activation, execution and completion. Our work differs in two ways from OWL-S. Firstly, we use a declarative, causality-based behaviour specification formalism, which allows for constraint-oriented composition of service specifications. The process ontology of OWL-S enables only an imperative, and therefore prescriptive, specification style. Secondly, OWL-S takes only the provider's perspective into consideration, whereas we treat both participants in a service interaction equally. Therefore, we cannot identify inputs or outputs, but only specify preconditions and results. Our approach is more abstract than OWL-S. We abstract from messages being exchanged and talk of values being established or passed. And we generally do not make explicit in our models which is the initiating party of a service interaction. Such (implementation) details can be added at subsequent refinement steps, but that is beyond the scope of this paper.

The Web Service Modeling Ontology (WSMO 2006) and related projects are now picking up a lot of momentum (WSMO 2006; Bruijn et al. 2005). WSMO is a formal ontology for describing several aspects of Semantic Web Services. It consists of four main components—*Ontologies, Goals, Web Services* and *Mediators. Ontologies* provide terminology and formal semantics of information that is used by the other components. A *goal* is a specification of the objectives of a service user. A *web service* is a specification of the functionality of the service provider. *Mediators* are used as connectors between ontologies, goals and web services.

Both goals and web services are described in terms of *non-functional properties*, used *ontologies*, desired *capabilities* and *interfaces*. A *capability* specifies *what* a service does. It describes the *preconditions* (i.e., state of the lexical domain before the service execution), *assumptions* (state of the environment before the service execution), *postconditions* (state of the lexical domain after the service execution) and *effects* (state of the environment after the service execution). An *interface* specifies *how* the functionality of the service can be achieved. It describes the *choreography* and *orchestration* of a service. A *choreography* describes the interactions between the service requestor and the service provider. An *orchestration* describes how the service makes use of other services to achieve its capability.

There are a lot of similarities between our work and WSMO. WSMO ontologies correspond to our information models. WSMO capabilities correspond to our interaction contributions. That is, the preconditions and assumptions correspond to the causality constraints of the service requestor and service provider respectively. Postconditions and effects correspond to our result constraints. WSMO choreographies and interfaces both correspond to our interfaces. WSMO orchestrations correspond to our orchestrations, i.e., the decomposition of the provider's behaviour into smaller services that realize the external behaviour.

Mediators are largely implicit in our work. For example, an interaction in our framework can represent a WSMO goal to goal mediator. Goal to web-service and web service to web service mediators correspond to refinement steps in our approach, where one interaction is decomposed into a choreography. We believe mediators are currently used in WSMO merely as a syntactic construct, whereas our framework allows one to reason about their semantics, e.g., the matching of goals in terms of interaction constraints, and the relationship between goals and web-services in terms of conformance relations. How to support semantic mappings between used ontologies will be dealt with in a forthcoming paper.

All in all our conceptual framework is more parsimonious than WSMO, i.e., it has less concepts but with comparable expressive power. Furthermore, we feel that the behavioural semantics of WSMO choreographies and orchestrations are rather weakly specified.

9 Conclusions

Although service-orientation is widely recognized as a promising approach to deal with the complexity of IT systems, its central concept—that of service—has so far not been used to its full potential due to the lack of a comprehensive conceptual framework.

Based on an analysis of commonly found interpretations of the service concept, we identified and classified general (meta-) properties that should be addressed by services, which can be classified into: structural, behavioural, information, goal and quality properties. Using the simple example of a procurement service, we introduced and illustrated basic concepts that support the properties identified and underlie the service concept. Moreover, these basic concepts helped us to explain, relate and in fact formalize important notions, such as requested service, offered service, choreography and orchestration.

All this work finally led to our proposed conceptual framework for service modelling, named COSMO, which has been summarized with three related class diagrams to capture roles and service types, behavioural aspects, and information aspects, respectively. Our main conclusions with respect to the framework are:

- The framework is constructed from a small number of basic concepts, which are based in practice, as argued above, and at the same time provide a powerful conceptual basis for modelling;
- The framework is language-independent, but at the same time the basic concepts of the framework can be related to many of the popular languages used in the context of service design, analysis and implementation. This opens the possibility to use the framework as a common semantical basis for comparing models produced with different languages;
- The framework is domain-independent, i.e., no assumptions are made with respect to the type of systems for which services should be modelled. We expect that our framework has a wide spectrum of application, e.g., can be used to model services at a business, application and component level, thus beyond the usual domain of web services;
- The framework is particularly strong in the area of behavioural modelling, when compared to other approaches. In addition, the information aspect of our framework can profit from (absorb) ongoing developments in the area of ontologies;
- The framework supports the modelling of services at different abstraction levels. Three generic abstraction levels are identified, including the assessment of the refinement (conformance) relation between models defined at successive abstraction levels.

Our forthcoming work will focus on the validation of the COSMO framework in practice. This implies that we want to further investigate mappings onto existing languages and exploit and/or extend the tools developed for these languages. Furthermore, we want to implement a 'grounding' of our conceptual framework by developing transformations onto existing technology, in particular web service standards. Practical cases to be considered will comprise the support and implementation of operations on services, especially those that are applied at run-time. An example of such a case is the adaptive composition of services through ontology-based mediation.

Acknowledgements This work is part of the Freeband A-MUSE project (http://a-muse.freeband.nl), which is sponsored by the Dutch government under contract BSIK 03025.

References

A-MUSE (2006). A-MUSE project, http://a-muse.freeband.nl.

Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., et al. (2005). Web Services Agreement Specification (WS-Agreement) Version 2005/09. http://www.gridforum.org/Public_Comment_Docs/Documents/Oct-2005/WS-AgreementSpecificationDraft050920.pdf.

Baader, F., Calvanese, D., McGuinnes, D., Nardi, D., & Patel-Schneider, P. (2003). *The description logic handbook: Theory, implementation and applications*. Cambridge: Cambridge University Press. ISBN 0521781760.

Baida, Z., Gordijn, J., & Omelayenko, B. (2004). A shared service terminology for online service provisioning. In *Proceedings of the 6th Int. Conference on Electronic Commerce, vol. 60* (pp. 1–10).

BPMN (2006). Business Process Modeling Notation (BPMN) information, http://www.bpmn.org.

Bruijn, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Keller, U., et al. (2005). Web Service Modeling Ontology (WSMO), W3C member submission 3 June 2005, 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. In *Proceedings of the 3rd International Conference on Service-oriented Computing (ICSOC)* (pp. 48–60).

Dijkman, R., & Dumas, M. (2004). Service-oriented design: A multi-viewpoint approach. *International Journal of Cooperative Information Systems (IJCIS), Special Issue on Service Oriented Modeling, 13*(4), 337–368.

Dijkman, R., Quartel, D., Ferreira Pires, L., & van Sinderen, M. (2004). A rigorous approach to relate enterprise and computational viewpoints. In *Proceedings of the 8th IEEE Enterprise Distributed Object Computing (EDOC) Conference, Monterey, USA* (pp. 187–200).

Dirgahayu, T. (2005). *Model-driven engineering of web service compositions: A transformation from ISDL to BPEL*. MSc thesis, University of Twente, Enschede, The Netherlands.

IBM (2006). IBM service definition, http://www.research.ibm.com/ssme/services.shtml.

ISO (1994). Information technology—Open Systems Interconnection —Basic Reference model—Conventions for the definition of OSI Services. ISO/IEC DIS 10731.

Jonkers, H., Lankhorst, M., van Buuren, R., Hoppenbrouwers, S., Bonsangue, M., & van der Torre, L. (2004). Concepts for modelling enterprise architectures. *International Journal of Cooperative Information Systems, 13*(3), 257–287.

Klein, M., & Bernstein, A. (2004). Toward high-precision service retrieval. *IEEE Internet Computing, 8*(1), 30–36.

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

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott D., McIlraith, S., et al. (2004). OWL-S: Semantic markup for web services W3C Member Submission 22 November 2004, http://www.w3.org/Submission/OWL-S.

Pokraev, S., Quartel, D., Steen, M., & Reichert, M. (2006). Semantic service modeling: Enabling system interoperability. In *Proceedings of the International Conference on Interoperability for Enterprise Software and Applications (I-ESA'06)* (pp. 221–231).

Pokraev, S., Quartel, D., Steen, M., Wombacher, A., & Reichert, M. (2007). Business level service-oriented enterprise application integration. In *Proceedings of I-ESA'07*.

Quartel, D., Dijkman, R., & van Sinderen, M. (2004). Methodological support for service-oriented design with ISDL. In *Proceedings of the 2nd International* Conference on Service Oriented Computing (pp. 1–10).

Quartel, D., Dijkman, R., & van Sinderen, M. (2005a). An approach to relate business and application services using ISDL. In *Proceedings of the 9th IEEE Enterprise Distributed Object Computing (EDOC) Conference* (pp. 157–168).

Quartel, D., Dijkman, R., & van Sinderen, M. (2005b). Extending profiles with stereotypes for composite concepts. In *The 8th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS)* (pp. 232–247).

Quartel, D., Ferreira Pires, L., & van Sinderen, M. (2002). On architectural support for behaviour refinement in distributed systems design. *Journal of Integrated Design and Process Science, 6*(1), 1–30.

Quartel, D., Ferreira Pires, L., van Sinderen, M., Franken, H., & Vissers, C. (1997). On the role of basic design concepts in behaviour structuring. *Computer Networks and ISDN Systems, 29*, 413–436.

Quartel, D., Steen, M., Pokraev, S., & van Sinderen, M. (2006). A conceptual framework for service modelling. In *Proceedings Tenth IEEE International EDOC Enterprise Computing Conference* (pp. 319–330).

Sprott, D., & Wilkes L. (2004). Understanding service-oriented architecture. In *CBDI Journal*, CBDI Forum, January.

van Eck, P., Blanken, H., & Wieringa, R. (2004). Project GRAAL: Towards operational architecture alignment. *International Journal of Cooperative Information* Systems, 13(3), 235–255.

Van Sinderen, M., Ferreira Pires, L., Vissers, C., & Katoen, J. (1995). A design model for open distributed processing systems. *Computer Networks and ISDN Systems, 27*, 1263–1285.

Vissers, C., & Logrippo, L. (1986). The importance of the service concept in the design of data communication protocols. *Protocol Specification, Testing and Verification, V*, 3–17.

W3C (2004). Web services architecture W3C Working Group Note 11 February 2004. http://www.w3.org/TR/ws-arch/.

Wieringa, R. (2003). Design methods for reactive systems: Yourdon, statemate, and the UML. San Mateo, CA: Morgan Kaufmann.

Wikipedia (2005). http://en.wikipedia.org.

WSMO (2006). http://www.wsmo.org.

Yud, E. (1997). Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on* Requirements Engineering (RE'97) (pp. 226–235).

Dick A. C. Quartel is an Assistant at the University of Twente, The Netherlands. His research interests include design methods and service architectures for networked systems, business process and service modelling, and semantic service interoperability. He is currently a workpackage leader in the Dutch Freeband A-MUSE project (BSIK 03025) on service design and semantic interoperability. He received his Master's and PhD degree in Computer Science from the University of Twente, The Netherlands. You can reach him at d.ac.quartel@utwente.nl.

Maarten W. A. Steen is a scientific researcher at Telematica Instituut since 1999. He has worked both in knowledge-building research projects as in applied projects with industry. His focus of research has always been on methods and techniques for conceptual modelling in the domain of enterprise systems development and service-oriented architecture. An area which is starting to become known as Model-Driven Engineering (MDE). He has lectured on these topics at industry seminars and at universities. Before joining Telematica Instituut, Maarten Steen worked at the University of Kent at Canterbury on the application of formal methods in the area of Open Distributed Processing. More specifically, he worked on techniques for partial specification, such as consistency checking and composition, and on enterprise modelling and policy specification. Maarten Steen holds a PhD in Computer Science from the University of Kent at Canterbury, UK, and an MSc.(Eng.) in Computer Science from the University of Twente, The Netherlands. You can reach him at Maarten.Steen@telin.nl.

Stanislav Pokraev is a member of Scientific Staff at Telematica Instituut, The Netherlands since 2001 and PhD candidate in the Computer Science department of the University of Twente, The Netherlands since 2003. Previously he was employed as scientific researcher by KPN Research, The Netherlands. Stanislav holds a MSc (Eng) degree from the Technical University of Sofia, Bulgaria. His main research interests are formal information modeling and service-

oriented business integration. You can reach him at Stanislav.Pokraev@telin.nl.

Marten J. van Sinderen is an associate professor at the University of Twente, The Netherlands, and manager of A-Services Internet, one of the strategic research orientations of the Centre for Telematics and Information Technology, the ICT research institute of the University of Twente. His research interests include design methods and architectures for networked systems, and service platforms for supporting context-aware mobile applications. He currently leads the Dutch Freeband A-MUSE project (BSIK 03025) on service design and semantic interoperability. He received his Master's degree in electrical engineering and his Ph.D. degree in computer science from the University of Twente, The Netherlands. You can reach him at m.j.vansinderen@utwente.nl.