A conceptual framework for service modelling[1]

Dick Quartel[a], Maarten W.A. Steen[b], Stanislav Pokraev[b] and Marten van Sinderen[a]
[a] *Centre for Telematics and Information Technology*
{D.A.C.Quartel, M.J.vanSinderen}@utwente.nl
[b] *Telematica Instituut*
{Maarten.Steen, Stanislav.Pokraev}@telin.nl

Abstract

This paper presents a conceptual framework for service modelling. This framework provides a conceptual basis for the modelling and reasoning about services, and the operations, such as composition and discovery, that 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 a single, common conceptual framework, thereby facilitating one to relate these models, e.g., to verify consistency or conformance. Therefore, a requirement on the framework is to 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.

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 [19], and its definition by OSI can be taken as a reference [10]. 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 design 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. The purpose of this framework is to bring clarity to the field of service-orientation by fixing terms and providing a conceptual basis for the modelling and reasoning about services and the operations, such as composition and discovery, that are performed on them at design and run-time.

The 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.

The structure of this paper is as follows. Section 2 analyses existing service definitions, derives and classifies general service properties. Sections 3 and 4

[1] 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.

present our conceptual framework for service modelling. Section 5 discusses the application of the framework. Section 6 relates our work to other research activities. And section 7 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 and a categorization into service aspects is proposed.

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 [22]. The IBM Services Research group defines a service as: "a provider/client interaction that creates and captures value" [9]. [17] 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 [2]. CBDI Forum also apply this interpretation to IT services: "a service is a type of capability described using WSDL" [18].

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.

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: "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. [19], 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, some 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 3.

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 and delivery 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.

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.

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 [16].

2.3. 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.

These 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. 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 [9], [21], [17]. 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 three successive abstraction levels: as a single interaction, as a choreography of interactions, and as an orchestration of smaller services.

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 order articles from some retailer, which are delivered to the user after he has paid for them. In addition, both the customer and retailer may define various conditions on the procurement process, concerning pricing, the delivery period and the ordering of payment and delivery.

3.1. 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 a (partial) 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 the systen's responsibilities in performing the interaction.

Figure 1 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. In this case, both the customer and retailer want to establish an order as the interaction result. The customer wants to order a notebook, whereas the retailer is willing to sell any article from its catalog. Furthermore, the customer wants the notebook to have certain properties, to be delivered in 5 days at a certain location, and has some maximum price in mind. The retailer has specified for each article a minimum price and delivery period. In addition, the retailer will only deliver in the region "Twente".

The procurement 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.

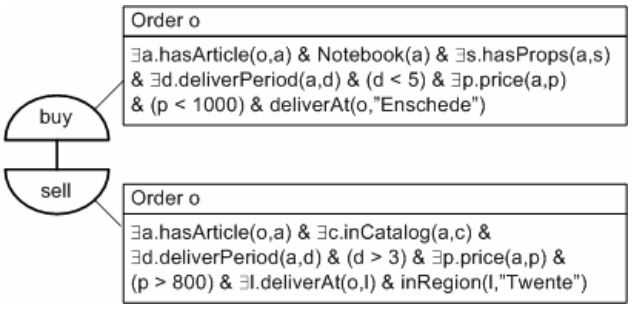

Figure 1. Procurement interaction

3.1.1. 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 2.4 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 Figure 1, 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 is the so-called *integrated perspective*, which defines the joint behaviour of the offered and requested service, abstracting from the distinction between a user and provider role. This more abstract definition of the service is called the *integrated service*. For this purpose, the action concept is introduced. An *action* represents an activity that is performed by a single system. Similar, to the interaction concept, an

action either occurs and establishes a result, or an action does not occur and establishes no (partial) result. Constraints can be attached to an action defining the possible results.

An action is used to represent an integrated interaction (service), by considering the systems that perform the user and provider roles as a single system. Furthermore, the constraints of the action are defined by the conjunction of the constraints defined by the (integrated) interaction contributions. Figure 2 depicts the integrated perspective of the procurement example in Figure 1.

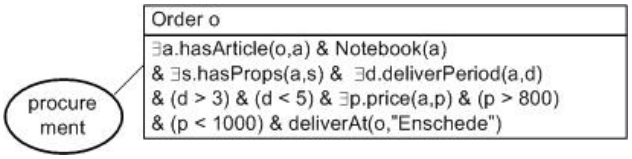

Figure 2. Procurement action

3.1.2. 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.

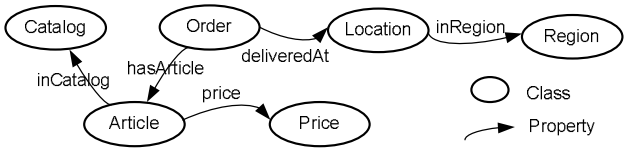

Figure 3. Procurement information model

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 Figure 1 has a single information attribute o of type Order.

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 Figure 1 specifies that the result can only be an order for a notebook that costs less than 1000 and can be delivered within 5 days. Such a result constraint can also be seen as the goal of the customer.

3.2. 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. Figure 4 depicts a possible refinement of the procurement service from Figure 1 into a number of interactions: select represents the selection of an article, checkout represents the establishment of the order comprising the selected article, pay represents the payment of the order (by credit card), and deliver represents the order delivery. In addition, the retailer offers the possibility to pay by bank transfer through interaction contribution pay2. The retailer allows this interaction only if some *precondition* is satisfied, i.e., the price of the selected article is larger than 500. Observe that contributions checkout, pay and pay2 refer to results established in the causally preceding contribution select, i.e., the article and the price of the article.

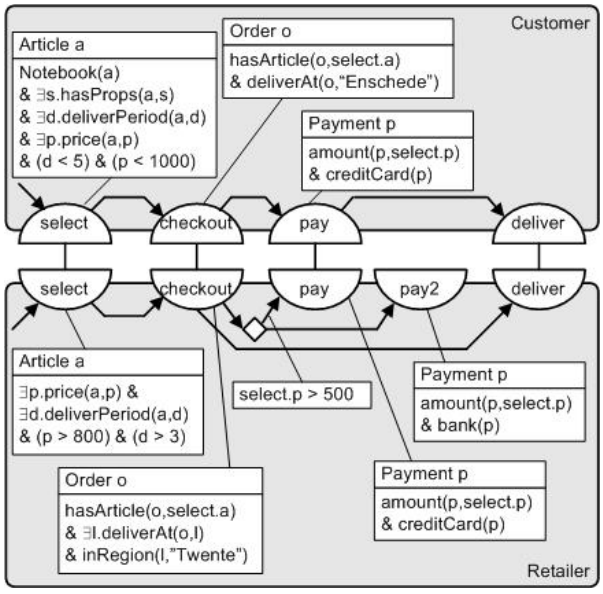

Figure 4. Procurement choreography

To represent multiple related activities, the *behaviour* concept is introduced, here graphically expressed as a rounded rectangle. A behaviour is associated with some system that performs the activities. Consequently, the activities that can be defined are actions and/or interaction contributions. For example, the behaviours in Figure 4

consist of interaction contributions only, whereas the behaviour in Figure 5 consists of actions only.

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

Similar to a service 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 as imposed by the contributions of the service provider. Analogously, the requested service defines these relationships and constraints from a user perspective. Behaviours customer and retailer in Figure 4 represent the requested and offered service (choreography), respectively. The example from Figure 1 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 customer wants to pay the order before it is delivered, whereas the retailer allows the payment and delivery to occur independently.

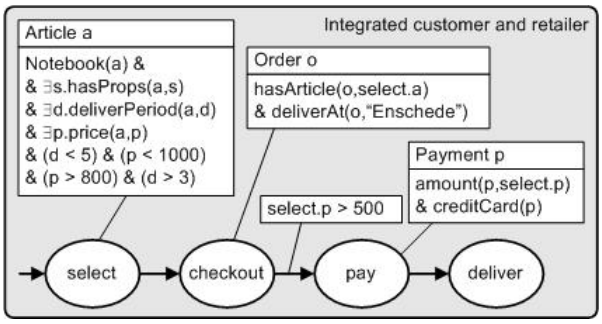

Figure 5. Procurement integrated choreography

Figure 5 depicts the choreography of Figure 4 from an integrated perspective, which defines the conjunction of the constraints and relationships as defined by the offered and requested service. The integrated service models the effective relationships between the interactions more concisely and clearly. For example, where the offered service allows interactions pay and deliver to occur independently, interaction deliver must depend on pay in the effective service behaviour due to the more strict conditions imposed by the requested service (user). Furthermore, interaction contribution pay2 is removed because it cannot participate in an interaction with the customer. Consequently, the integrated service facilitates one to analyse the service behaviour, e.g., if the interactions can occur in some order that is allowed by both the requested and the offered service. In this case, the integrated behaviour shows clearly that a procurement can only be completed if the price of the ordered article is larger than 500.

3.2.1. 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 action a are distinguished (see Figure 6): (i) enabling condition b represents that activity b must have occurred before a can occur; (ii) disabling condition ¬b represents that activity b must not have occurred before nor simultaneously with a to enable the occurrence of a; (iii) the start condition represents that activity a is enabled from the beginning of the behaviour and 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.

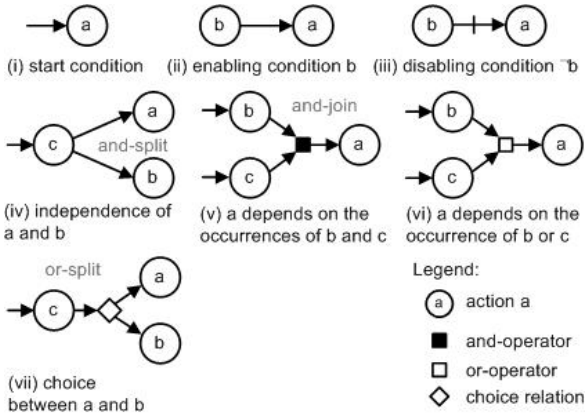

Figure 6. Causality conditions

3.2.2. 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 or defining new services that implement part of the required service functionality.

Figure 7 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 Figure 8 in section 3.3. In addition, interaction select is further refined by introducing a preparatory interaction catalog in which the user can request for a catalog 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.

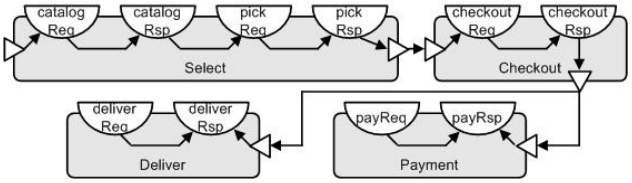

Figure 7. Structured choreography

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.

3.3. Service as orchestration

Besides the refinement of interactions, one may want to refine a service into a composition of smaller services in order to obtain an implementation of the service. Figure 8(i) depicts an example of the refinement of the offered procurement choreography from Figure 7 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.

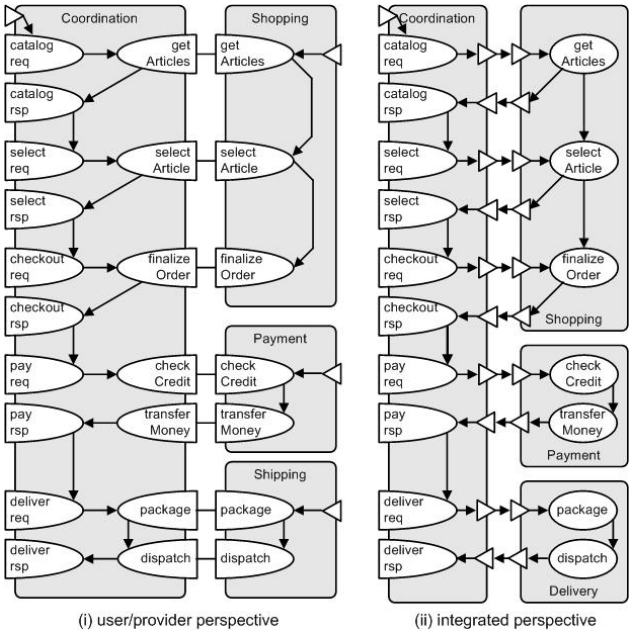

Figure 8. Procurement orchestration

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, 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.

Figure 8(ii) depicts a service model of the same example defined from an integrated perspective. This model defines the orchestration as a set of related integrated choreographies (in dashed gray). Each choreography represents one of the sub-services that is used to implement the procurement functionality from an integrated perspective. Consequently, the coordination service is not distinguished as a separate behaviour, but is

partly integrated into the choreographies and for the other part represented by the relationships between these choreographies.

An integrated model of an orchestration may define the composite service behaviour more concisely and clearly than from a user/provider perspective. Therefore, the model may be easier to analyse and allows for more freedom in the choice of offered services, in particular when the structuring of choreographies is ignored. In the latter case, the model does not suggest or impose any choice concerning which actions (functions) should be provided by which offered services.

3.3.1. Behaviour composition

In general, 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. Figure 4 and Figure 8(i) present examples of constraint-oriented composition.

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 7 presents an example of causality-oriented composition.

4. Conceptual model

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 2.4, except for the structural aspect and the quality aspect. The meta-models are represented using UML class diagrams.

4.1. Abstraction levels and roles

The meta-model in Figure 9, defines concepts that are used to denote distinct types of service models, varying in abstraction levels and roles being considered.

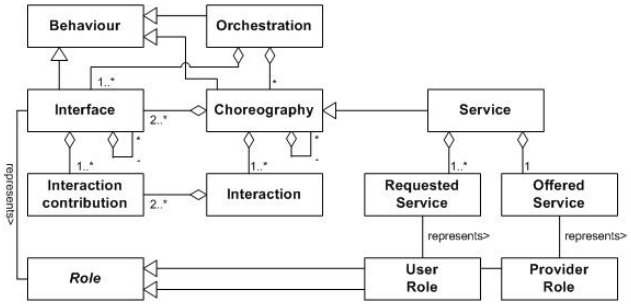

Figure 9. Service concepts

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.

4.2. Behavioural aspect

The meta-model in Figure 10 defines the concepts used to model the behavioural aspect.

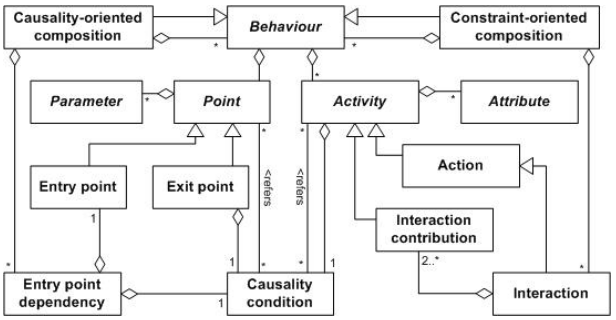

Figure 10. Behavioural concepts

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.

4.3. Information aspect

The meta-model in Figure 11 defines the concepts used to model the information aspect.

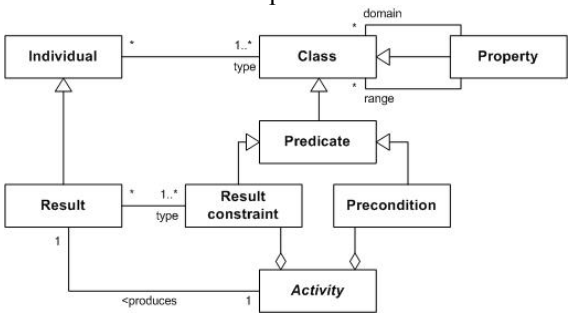

Figure 11. Information concepts

The information modelling concepts we use are borrowed from description logics [5]. 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 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.

4.4. Goal aspect

Service users and providers request and offer services, respectively, with a particular goal in mind. Analogous to [12] and other requirements engineering literature [24], 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 Figure 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 pay 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 admits any results. In this case only the provider goal defined by interaction contribution pay matches the user goal, since it allows a credit card payment.

5. Application of the concepts

The purpose of our basic conceptual framework is to serve as a common semantic meta-model to enable the use of different service modelling languages. In general, different languages are needed to support the creation of services. Distinct design and specification languages may be suitable for modelling distinct service aspects (see section 2). 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 [7], [8].

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 [1], 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., BPMN [4], ISDL [16] or UML. 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 defined as compositions or specializations of the basic concepts. In the latter case, such as when using domain specific languages, 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 extent the common semantic meta-model [15].

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. In addition, a method has been developed to verify the interoperability (or composability) of two or more services that is based on our basic design concepts [14]. Current work focuses on providing methods to (semi-)automatically construct mediators that resolve identified interoperability problems.

Concerning service discovery, [11] 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, [11] proposes 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.

Considering the concepts that have been introduced to model services, we believe that the resulting service models contain the information that is required to support concept- and process-based retrieval approaches. 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.

6. 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) [20], which provides a conceptual model for understanding Web services and related concepts. 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. [6] 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) [13] represents one of the first attempts at formalizing the semantics of Web services using ontology technology. 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 [23], [3]) and related projects are now picking up a lot of momentum. 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 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 web service to goal mediator. Goal to goal and web service to web service mediators correspond to refinement steps in our approach, where one interaction is decomposed into a choreography. 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.

7. 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, 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 proposed 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.

Our forthcoming work will focus on the validation of our framework in practice. This implies that we want to investigate mappings onto existing languages and exploit and/or extend the tools developed for these languages. Furthermore, we want to provide a 'grounding' of our conceptual framework 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 runtime. For example, the adaptive composition of services through ontology-based mediation is part of our ongoing work.

References

[1] A-MUSE project, http://a-muse.freeband.nl, 2006.

[2] Baida, Z. et al. A shared service terminology for online service provisioning. In: Proc. of the 6th Int. Conference on Electronic Commerce, vol. 60, pp. 1-10, ACM Press, New York, 2004.

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

[4] Business Process Modeling Notation (BPMN) Information, http://www.bpmn.org, 2006.

[5] Calvanese, D., et al. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003. ISBN 0521781760.

[6] Colombo, M., et al. Speaking a Common Language: A Conceptual Model for Describing Service-Oriented Systems. In: *Proc. of the 3rd Int. Conference on Service-Oriented Computing* (ICSOC 2005), pp. 48-60, December 2005, Amsterdam, The Netherlands.

[7] Dijkman, R.M., et al. A Rigorous Approach to Relate Enterprise and Computational Viewpoints. In: *Proc. of the 8th IEEE Enterprise Distributed Object Computing* (EDOC) Conference, Monterey, USA, pp. 187-200, 2004.

[8] Dijkman, R.M. and Dumas M. Service-oriented Design: a Multi-Viewpoint Approach. In: *Int. Journal of Cooperative Information Systems* (IJCIS), Special Issue on Service Oriented Modeling 13(4), pp. 337-368, 2004.

[9] IBM service definition, http://www.research.ibm.com/ssme/services.shtml, 2006.

[10] ISO. Information technology – Open Systems Interconnection – Basic Reference model – Conventions for the definition of OSI Services. ISO/IEC DIS 10731.

[11] Klein, M. and Bernstein, A. Toward High-Precision Service Retrieval. In: *IEEE Internet Computing*, Vol. 8, No. 1, pp. 30-36, Jan/Feb 2004.

[12] Lamsweerde, A. van. Goal-Oriented Requirements Engineering: A Guided Tour. In: *Proc. RE'01*, 5th Int. Symposium on Requirements Engineering, Toronto, August 2001, pp. 249-263.

[13] Martin, D., et al. OWL-S: Semantic Markup for Web Services W3C Member Submission 22 November 2004, http://www.w3.org/Submission/OWL-S.

[14] Pokraev, S., et al. Semantic Service Modeling: Enabling System Interoperability. Presented at I-ESA'06 and accepted for publication.

[15] Quartel, D., et al. Extending profiles with stereotypes for composite concepts. In: *Model Driven Engineering Languages and Systems*, Proc. of the 8th Int.l Conference, MoDELS 2005, Montego Bay, Jamaica, October 2005.

[16] Quartel, D., et al. Methodological support for service-oriented design with ISDL. In: *Proc. of the 2nd Int. Conf. on Service Oriented Computing*, NY, USA, 2004.

[17] Quartel, D, et al. On the role of basic design concepts in behaviour structuring. *Computer Networks and ISDN Systems*, 29:413–436, 1997.

[18] Sprott, D. and Wilkes, L. Understanding Service-Oriented Architecture. In: *CBDI Journal*, CBDI Forum, January 2004.

[19] Vissers, C.A., et al. The importance of the service concept in the design of data communication protocols. In: *Protocol Specification, Testing and Verification*, V, pp. 3-17. North-Holland, Amsterdam, 1986.

[20] W3C's Web Services Architecture. http://www.w3.org/TR/ws-arch/

[21] Wieringa, R.J. *Design Methods for Reactive Systems: Yourdon, Statemate, and the UML*. Morgan Kaufmann, 2003.

[22] Wikipedia. http://en.wikipedia.org. 2005.

[23] WSMO. http://www.wsmo.org. 2006

[24] Yu, E. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: *Proc.s of the 3rd IEEE Int. Symp. on Requirements Engineering* (RE'97) Jan. 1997, Washington D.C., USA. pp. 226-235.