Model-Driven Service Engineering with SoaML

Brian Elvesæter, Cyril Carrez, Parastoo Mohagheghi, Arne-Jørgen Berre, Svein G. Johnsen and Arnor Solberg

Abstract This chapter presents a model-driven service engineering (MDSE) methodology that uses OMG MDA specifications such as BMM, BPMN and SoaML to identify and specify services within a service-oriented architecture. The methodology takes advantage of business modelling practices and provides a guide to service modelling with SoaML. The presentation is case-driven and illuminated using the telecommunication example. The chapter focuses in particular on the use of the SoaML modelling language as a means for expressing service specifications that are aligned with business models and can be realized in different platform technologies.

Brian Elvesæter
SINTEF ICT, P. O. Box 124 Blindern, N-0314 Oslo, Norway, e-mail: brian.elvesater@sintef.no

Cyril Carrez
SINTEF ICT, P. O. Box 124 Blindern, N-0314 Oslo, Norway, e-mail: cyril.carrez@sintef.no

Parastoo Mohagheghi
SINTEF ICT, P. O. Box 124 Blindern, N-0314 Oslo, Norway, e-mail: parastoo.mohagheghi@sintef.no

Arne-Jørgen Berre
SINTEF ICT, P. O. Box 124 Blindern, N-0314 Oslo, Norway, e-mail: arne.j.berre@sintef.no

Svein G. Johnsen
SINTEF ICT, P. O. Box 124 Blindern, N-0314 Oslo, Norway, e-mail: svein.g.johnsen@sintef.no

Arnor Solberg
SINTEF ICT, P. O. Box 124 Blindern, N-0314 Oslo, Norway, e-mail: arnor.solberg@sintef.no

1 Introduction and Overview

1.1 Background and Motivation

There are two major trends in modern service engineering. Firstly, service-oriented architecture (SOA) has emerged as a direct consequence of specific business and technology drivers that have materialized over the past decade. Trends such as the outsourcing of non-core business operations, the importance of business process re-engineering and the need for system integration have been key influences in pushing SOA as an important architectural approach to information technology (IT) today. Secondly, modelling has become an integral part of software engineering approaches. Business process models are ideally used to describe how work is done within an enterprise, while various technical models describe the IT systems. There are several approaches to model-driven engineering (MDE) such as the OMG Model Driven Architecture (MDA)[1] [12] and efforts around domain-specific languages which have recently gained much popularity.

In MDA, business models are described as computational independent models (CIMs). For the IT models, MDA separates platform-independent models (PIMs) from platform-specific models (PSMs) in order to abstract the implementation technologies. It is also common to model a system from different views or perspectives such as its structure and behaviour. In addition to models, MDA includes the mechanism of transformations to provide mappings between representations of the system on different abstraction levels. Due to the powerful concepts of abstraction and refinement, MDA is being increasingly applied in various domains and for different types of applications, thus making it also an attractive solution for implementing the new wave of applications based on service-oriented architectures. In order to do so, new methodologies and languages with concepts required for modelling of services are required.

SOA has been promoted for some years without a specific language that supports modelling services. In order to meet this requirement the Service oriented architecture Modelling Language (SoaML) [16] was specified. The goals of SoaML are to support the activities of service modelling and design and to fit into an overall model-driven development approach. The SoaML profile defines extensions to UML 2 [17] to support the range of modelling requirements for service-oriented architectures, including the specification of systems of services, the specification of individual service interfaces, and the specification of service implementations. This is done in such a way as to support the automatic generation of derived artefacts following an MDA based approach.

The role of the SoaML specification is to specify a language, i.e., a metamodel and a UML profile, for the design of services within a service-oriented architecture. However, developing services requires a language and a process. It is not the role of the specification to define a methodology, but rather to provide a foundation for model-driven service engineering (MDSE) based on the MDA approach

[1] OMG Model Driven Architecture (MDA), http://www.omg.org/mda/

that can be adopted in different software development processes. The aim of this chapter is to provide a methodology for the SoaML language. This methodology was developed as part of the 7th Framework Programme research project SHAPE (ICT-2007-216408)[2].

1.2 Solution Idea

The business requirements of real-world applications require a flexible development approach to service-based IT landscapes that enables the businesses to exploit the benefits of the SOA paradigm in an efficient manner. We assume that there should be a systematic approach to service development which we refer to as service engineering. Service engineering should ideally cover all phases of a service life-cycle: from collecting requirements to the stages of development or identifying services, composition, operation, monitoring and evolution. A challenge is therefore to identify activities in software engineering related to the concept of services and SOA.

Generally speaking, a methodology describes a regular and systematic way of how to accomplish something. The term methodology here denotes a set or collection of methods and related artefacts needed to support the model-driven engineering of SOA-based systems. Our view of methodology aligns with [5] who defines methodology as a ”body of methods, meant to support all software development phases” and with [7] who defines methodology in the context of model-based systems engineering as the ”collection of related processes, methods, and tools used to support the discipline of systems engineering in a model-based or model-driven context”.

To address this need we present a model-driven service engineering (MDSE) methodology based on the OMG MDA approach [12]. The methodology guides solution architects in how to specify services that are aligned with the business process models. The focus of the MDSE methodology presented here is on the analysis and design of service-oriented architectures. Rather than providing a comprehensive methodology for supporting the engineering process for SOA systems, the aim is to define SOA modelling guidelines that can be included in existing model-driven methodologies. The MDSE methodology takes advantage of the SOA concepts defined in SoaML which allows to:

1. **Identify services** and the requirements they are intended to fulfil, and the anticipated dependencies between them.
2. **Specify services**, including the functional capabilities they provide, what capabilities consumers are expected to provide, the protocols or rules for using them, and the service information exchanged between consumers and providers.
3. **Define service consumers and providers**, what requisition and services they consume and provide, how they are connected and how the service functional

[2] Semantically-enabled Heterogeneous Service Architecture and Platforms Engineering (SHAPE), http://www.shape-project.eu/

capabilities are used by consumers and implemented by providers in a manner consistent with both the service specification protocols and requirements.
4. **Define policies** for using and providing services.
5. **Define architectural classification schemes** having aspects to support a broad range of architectural, organizational and physical partitioning schemes and constraints.

1.3 Outline of the Approach

Our MDSE methodology aims to integrate with existing business modelling practices within an enterprise, allowing building upon and extending existing modelling practices rather than replacing them. The methodology assumes that modern business modelling practices take advantage of business modelling tools that adopts the OMG MDA specifications Business Motivation Model (BMM) [13] and Business Process Modeling Notation (BPMN) [14]. From these models we will drive the specification of services as a set of SoaML model artefacts.

The MDSE methodology provides guidelines for how to use SoaML to define and specify a service-oriented architecture from both a business and an IT perspective. The methodology prescribes building a set of model artefacts following the iterative and incremental process paradigm. Figure 1 depicts the overall process and identifies the set of model artefacts to specify. The figure shows the set of work products prescribed by the methodology and the overall workflow. The icons indicate the associated BMM, BPMN or SoaML diagram(s) for each work product and the arrows show the most common path through the set of work products within an iteration.

Starting from the upper left we have the *Business Architecture Model (BAM)* which includes the business goals, business processes, capabilities, services architectures, and service contracts and choreographies. The *System Architecture Model (SAM)* specifies the service interfaces, interfaces and message types, service choreographies and software components. The *model-to-model (M2M) transformation* consists of transformation rules and procedural guidelines to support a semi-automated mapping from BAM to SAM.

The *Platform-Specific Model (PSM)* contains the design and implementation artefacts of the specified service-oriented architecture in the chosen technology platforms, e.g. cloud, Web Services, Java Enterprise Edition (JEE), multi-agent systems (MAS), peer-2-peer (P2P), grid and Semantic Web Services (SWS). We consider PSM-level modelling guidelines out of scope for the presentation of the methodology in this chapter and focus on the business architecture and system architecture modelling. PSM-level extensions to the methodology would involve defining further modelling guidelines and *model-to-text (M2T) transformation* rules for the technology platforms.

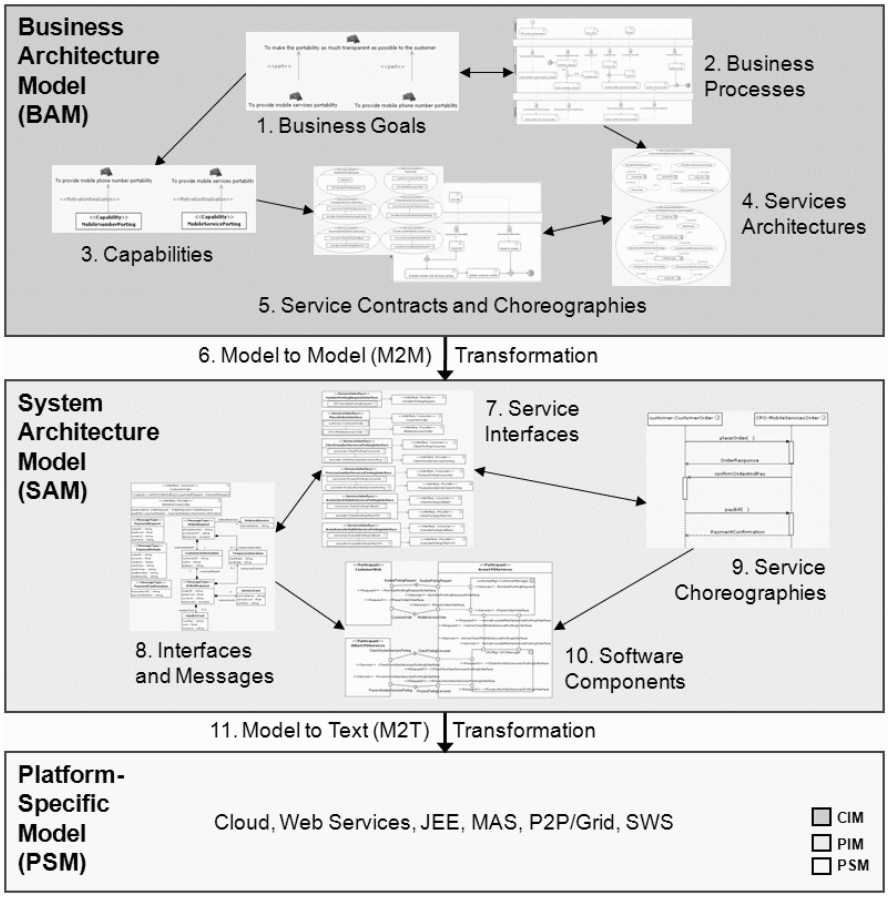

Fig. 1 The overall model-driven process

We use and extend the Eclipse Process Framework (EPF)[3] for implementing the methodology. EPF is a process framework that allows to define methodology and process content that can be customized and integrated with other engineering methodologies. The methodology presented here is dependent on modelling tools that support the OMG specifications BMM, BPMN and SoaML. In the example presented in this chapter we have used the UML modelling tool Modelio[4].

[3] Eclipse Process Framework (EPF), http://www.eclipse.org/epf/

[4] Modeliosoft, http://www.modeliosoft.com/

1.4 Structure of the Chapter

The remainder of this chapter is organized as follows: In Section 2 we introduce the telecommunication scenario that we use to illustrate our model-based service engineering methodology. Section 3 presents the business architecture modelling guidelines to describe the business perspective of an SOA, and Section 4 presents the system architecture modelling guidelines to describe the IT perspective of an SOA. Section 5 contains a discussion of our results in comparison with other related efforts on service modelling. Finally, Section 6 summarizes and concludes the chapter.

2 Illustrative Scenario

We have developed a methodology that takes advantage of existing business modelling practices and provides a guide for specifying services using SoaML concepts. The methodology is illuminated using the telecommunication scenario as introduced for the book. The scenario is about mobile phone services portability and has the following business goals:

- To provide mobile phone number portability.
- To provide mobile services portability.
- To make the portability as much transparent as possible to the customer.

In the following sections we provide guidelines for how to specify the services and their contracts, starting from the business processes descriptions. Specifically, we focus on the first goal addressing phone number portability. The starting point is a BPMN process description involving a customer and a number of cell phone operators (CPOs). The scenario assumes a cooperation between different CPOs as well as internal cooperation between the different departments of a CPO.

SoaML allows to specify service-oriented architectures at two levels of granularity. The *community-level* architecture is a public ”top-level” view of how independent participants collaborate without any single controlling entity or process. The *participant-level* architecture is an internal view that specifies how parts of a specific participant (e.g. departments within an organization) work together to provide the services of the owning participant. In the telecommunication scenario we will show how to apply SoaML to specify a *community-level* architecture and how this is refined into a *participant-level* architecture and further mapped to a software architecture for a specific CPO named *AcmeCPO*.

The BPMN process diagram shown in Figure 2 describes the request and assignment of mobile services portability. When a *Customer* requires a telephone number portability, the *AcmeCPO* has to provide not only the porting of the number, but also the porting of the services enabled on it, when possible. After checking the portability of the number the CPO executes the porting. At the end of the process the number is activated and bound to the new CPO.

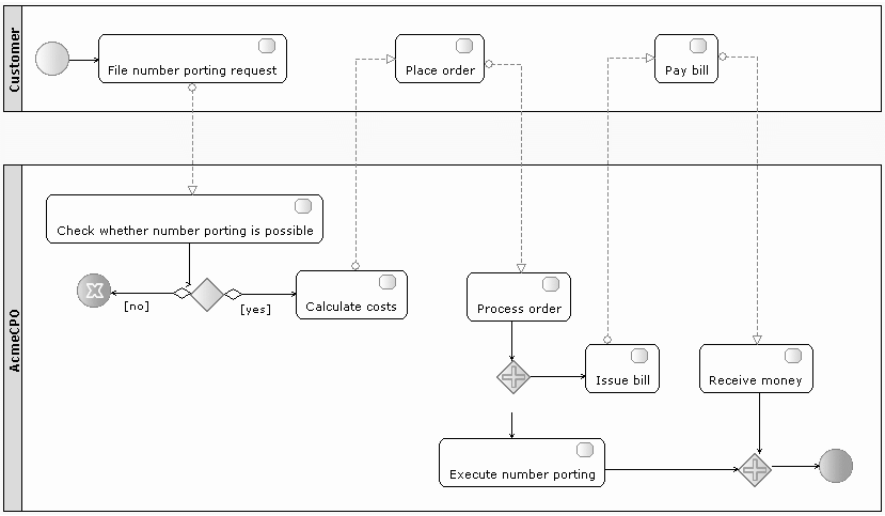

Fig. 2 Request and assignment of mobile phone services portability (BPMN diagram)

3 Business Architecture Modelling

This part of the methodology covers selected areas of CIM-level modelling resulting in a *Business Architecture Model (BAM)* that describes the business perspective of a service-oriented architecture. The BAM is used to express the business operations and environment which the service-oriented architecture is to support. The BAM includes business goals (Section 3.1), business processes with associated organisation roles and information elements (Section 3.2), and capabilities (Section 3.3) that are relevant for capturing business requirements and identify services within a service-oriented architecture. The BAM further describes the services architecture (Section 3.4) of the business community and the service contracts (Section 3.5) between the business entities participating in the community. Figure 3 depicts an activity diagram that shows the modelling tasks involved in the specification of the Business Architecture Model.

3.1 Business Goals

Business Motivation Model (BMM) [13] is a business-level modelling technique that supports the modelling of the business goals and objectives, the means and policies to achieve them, and the influencing factors that drive and control the work involved. This is typically used to define the overall business strategy in early phases of an engineering project.

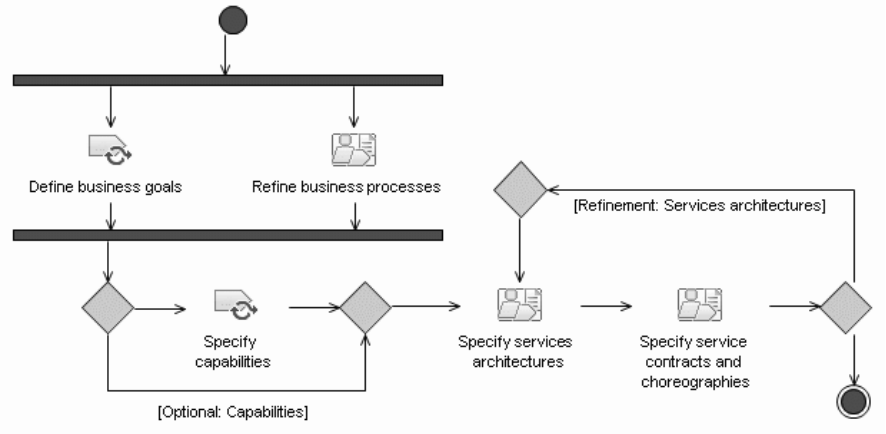

Fig. 3 Business architecture modelling activities (EPF activity diagram)

The purpose of the BMM is to agree with the business stakeholders the business goals that will be met by implementing the service-oriented architecture, so that a set of required high level business processes can be identified for further analysis. BMM provides a scheme or structure for developing, communicating, and managing business plans in an organized manner. In particular, the BMM supports the following:

- It identifies factors that motivate the establishment of business plans.
- It identifies and defines the elements of business plans.
- It indicates how all these factors and elements inter-relate.

Among these elements, business policies and business rules provide governance for and guidance to the business. Once produced and agreed, the BMM serves as a reference that ensures that a full assessment may be made of all the business implications of any proposed changes to the service-oriented architecture.

3.1.1 Modelling of Business Goals

Business goals are discovered by a process of workshops and interviews involving relevant stakeholders. The BMM describes a loose hierarchy of goals of the business within the particular area of concern, from the goals of a business stakeholder in developing a product to the business goals met by the product or its users.

The goals are created as classes containing motivation-related information. The name is expressed in a natural language and has properties such as scope, quantitative/qualitative value, etc. Relationships between goals such as ”part of”, ”positive influence”, ”negative influence”, ”guarantee” and ”measure” can be specified. Figure 4 shows the goals specified for the telecommunication example.

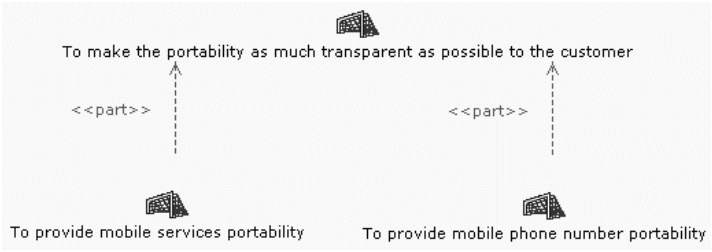

Fig. 4 Business goals diagram (BMM diagram)

3.2 Business Processes

Business Process Modeling Notation (BPMN) [14], is designed to communicate a wide variety of information on business processes to a wide variety of audiences, providing a standard notation that is readily understandable by all business stakeholders. Business process descriptions bring real business vision and constitute an excellent formalization and analysis tool when constructing systems. In the context of a development project they are used in business-oriented activities related to requirements, specifications and analysis.

Business processes may be at a number of levels of detail, from a high level description of the business processes down to task flows which comprise a set of detailed specifications for the business services to be realized in the service-oriented architecture. BPMN is designed to cover many types of modelling and allows the creation of end-to-end business processes. It allows the specification of private processes (both non-executable and executable), public processes, choreographies and collaborations.

Typically, BPMN is used to define business processes on the CIM level. The definitions are then mapped to more technical models on the PIM and PSM level. Our methodology does not address the full scope of business modelling at the CIM level, but rather assumes that some kind of business process models or descriptions already exist that have been developed using existing BPMN guidelines. However, typically these models must be further refined and mapped to SoaML concepts for describing the business perspective of an SOA.

3.2.1 Modelling of Business Processes

The BAM should contain refined descriptions of the business processes which are relevant to the service-oriented architecture to be defined. These are the business processes that will enable the goals to be met and include the roles which collaborate through services that are to be specified and developed.

The first step is to identify the relevant business processes for the service-oriented architecture, following these guidelines:

- Identify public and collaborative business processes that involve interactions and potential usage of software services between different business organizations. These processes are candidates for public *community-level services architectures* in SoaML.
- Identify private business processes for the business entities under your ownership control that are involved in the services architecture under consideration. These processes are candidates for internal *participant-level services architectures* in SoaML.
- The business goals resulting from the modelling steps outlined in Section 3.1.1 can be used to scope the selection of business processes.

Each of the selected business processes identified will be a candidate for a SoaML services architecture (see Section 3.4). Concerning our illustrative scenario, the public business process will encompass all the actors of the system (i.e. the customer and the different CPOs, as it will be exemplified hereafter); the private business process will encompass the *AcmeCPO*.

Once the business processes have been identified, their specification can begin. Each business process will be specified in a business process diagram and refined with respect to participants, their tasks, and information flow between the participants.

Participants represent the business units or organization roles that are involved in the execution of a process. Participants are specified using the BPMN constructs: *pools*, *participants* and *lanes*. Pools represent business organizations and lanes represent internal business units within an organization.

Each participant will perform different tasks and exchange information with other participants. The next step is to focus on the tasks which describe the interaction points between the business entities. These interaction points will be associated with service contracts in SoaML (see Section 3.5). Each task will possibly create, manipulate and use some information items. The following BPMN constructs are used for information modelling: *data objects*, *data inputs and outputs*, *data associations* and *messages*.

Figure 5 shows the refined business process model for the request and assignment of mobile phone services originally depicted in Figure 2. Three different participants take part in this process: *Customer*, *AcmeCPO* and *OtherCPO*. The interaction points between the participant tasks have been further revised by modelling data objects that are typed as specific message types, e.g. *PhoneNumberPortabilityRequest*. The tasks in the *OtherCPO* participant are considered private, thus from the perspective of *AcmeCPO* we only model the public information exchange.

Since the focus of our modelling refinement is on *AcmeCPO*, we can also specify an internal process that details the *AcmeCPO* pool. As can be seen in Figure 6 we have introduced two new roles represented by the lanes *CustomerManager* and *CPOManager* and assigned the tasks specified in the pool to these two roles according to whether the task is oriented towards the customer or another CPO. The figure only shows a partial left view of the BPMN diagram to illustrate the refinements with regards to the lanes.

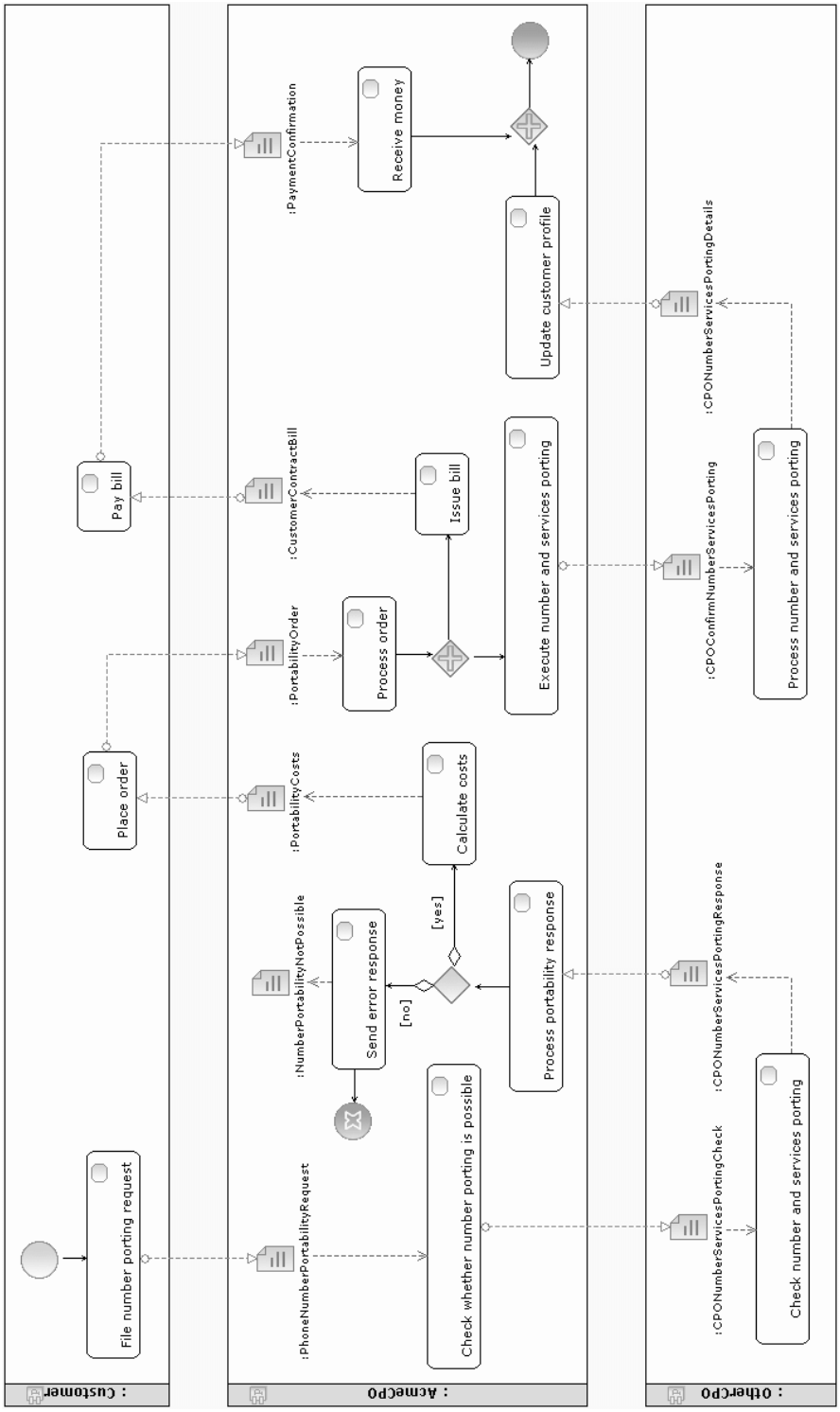

Fig. 5 Public community process for request and assignment of portability (BPMN diagram)

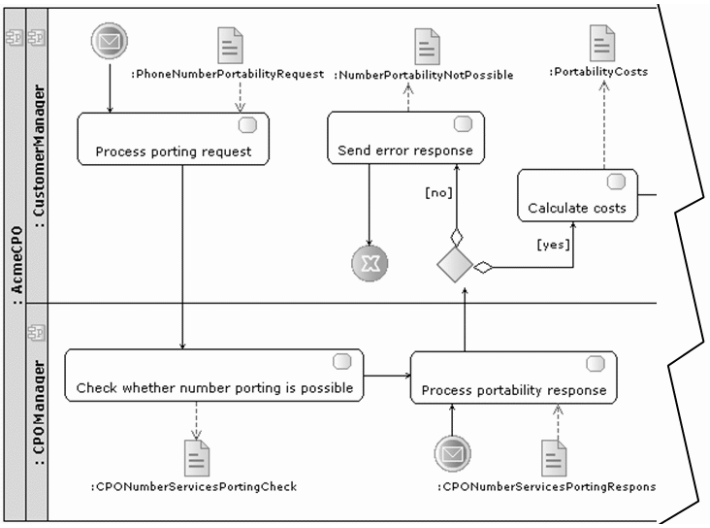

Fig. 6 Private *AcmeCPO* process for request and assignment of portability (BPMN diagram)

3.3 Capabilities

Capabilities identify or specify a cohesive set of functions or resources that a service provided by one or more participants might offer. Capabilities can be used by themselves or in conjunction with participants to represent general functionality or abilities that a participant must have. Capabilities are used to identify needed services, and to organize them into catalogues in order to communicate the needs and capabilities of a service area, whether that be business or technology focused, prior to allocating those services to particular participants. Capabilities can have usage dependencies on other capabilities to show how these capabilities are related. Capabilities can also be organized into architectural layers to support separation of concern within the resulting service architecture.

3.3.1 Modelling of Capabilities

The specification of capabilities is optional in our methodology. Capabilities represent high-level services with abstract operations. For large business architectures it may be a good idea to start with capability modelling to help identifying needed services, whereas for more technical architectures it might be easier to focus on the IT services directly. Once you have defined the capabilities, these can later be used to identify candidate services.

In order to identify the capabilities in the first place there exists different techniques:

- Goal-service modelling, which identifies capabilities needed to realize business requirements such as strategies and goals.
- Domain decomposition, which uses activities in business processes and other descriptions of business functions to identify needed capabilities.
- Existing asset analysis, which mines capabilities from existing applications.

Figure 7 shows a simple example of two capabilitites that were derived from the business goals (Figure 4).

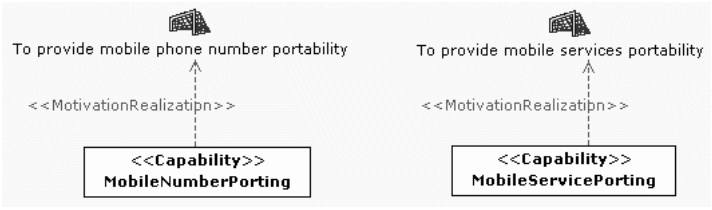

Fig. 7 Capabilities (UML class diagram)

3.4 Services Architectures

A *services architecture* is a high level description of how participants work together for a purpose by providing and using services expressed as *service contracts*. The services architecture defines the requirements for the types of *participants* and service realizations that fulfill specific roles. A *role* defines the basic function (or set of functions) that an entity may perform in a particular context. In contrast, a participant specifies the type of a party that fills the role in the context of a specific services architecture. Both service contracts and participants can be reused when composing different services in other services architectures.

3.4.1 Modelling of Services Architectures

Services architectures are modelled as UML collaborations with the stereotype ≪ServicesArchitecture≫. A services architecture has components at two levels of granularity: The *community services architecture* is a ”top level” view of how independent participants work together for some purpose. The services architecture of a community does not assume or require any single controlling entity or process. The public process described in Figure 5 maps to a community-level services architecture for the telecommunication scenario.

A participant may also have a *participant services architecture*, which specifies how parts of that participant (e.g., departments within an organization) work together to provide the services of the owning participant. Participants that realize

this specification must adhere to the architecture it specifies. The internal process described in Figure 6 maps to a participant-level architecture for *AcmeCPO*.

Participants are modelled as UML classes stereotyped ≪Participant≫. Participants are identified from pools, participants and lanes specified in the BPMN processes (see Section 3.2.1). Figure 8 shows the participants identified from the two BPMN process diagrams: the three pools in Figure 5 map to the three participants *Customer*, *AcmeCPO* and *OtherCPO* for the community-level services architecture. The two lanes in the Figure 6 map to the two participants *CustomerManager* and *CPOManager* which are internal roles in the *AcmeCPO* participant-level services architecture.

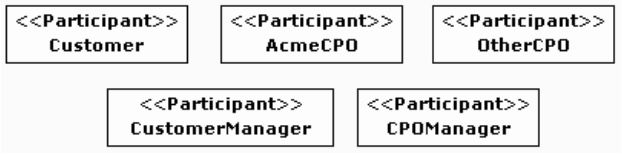

Fig. 8 Participants (UML class diagram)

The next step is to identify the possible interactions between the different participants. The interactions are represented as service contracts. A service contract is a UML collaboration with the stereotype ≪ServiceContract≫. In this step, only an empty UML collaboration is specified. The detailing of the service contracts will be further elaborated in Section 3.5.1.

Once all service contracts and participants have been identified, the service designer can use them to build the service architecture. Roles in the UML collaboration are typed by the identified participants, while UML collaboration uses are linked to the service contracts. The modeller has to bind the different roles to the appropriate collaboration uses, hence specifying how participants will interact.

Concerning the scenario, Figure 9 shows the two resulting services architectures. The *CommunityMobilePhoneServicesPortability* specifies the community-level architecture with its three participants that are connected together by four collaboration uses which are linked to the service contracts *NumberPortingRequest*, *PlaceOrder*, *CheckNumberServicesPorting* and *ProcessNumberServicesPorting*. The *AcmeCPOMobilePhoneServicesPortability* shows how *AcmeCPO* is organized internally to provide services. Two additional participants *CustomerManager* and *CPOManager* have been specified, connected by the service contracts *AcmeCheckMobileServicesPorting* and *AcmeExecuteMobileServicesPorting*. These two participants represent internal roles that are connected to the existing, external roles and the corresponding service contracts specified in the community-level services architecture.

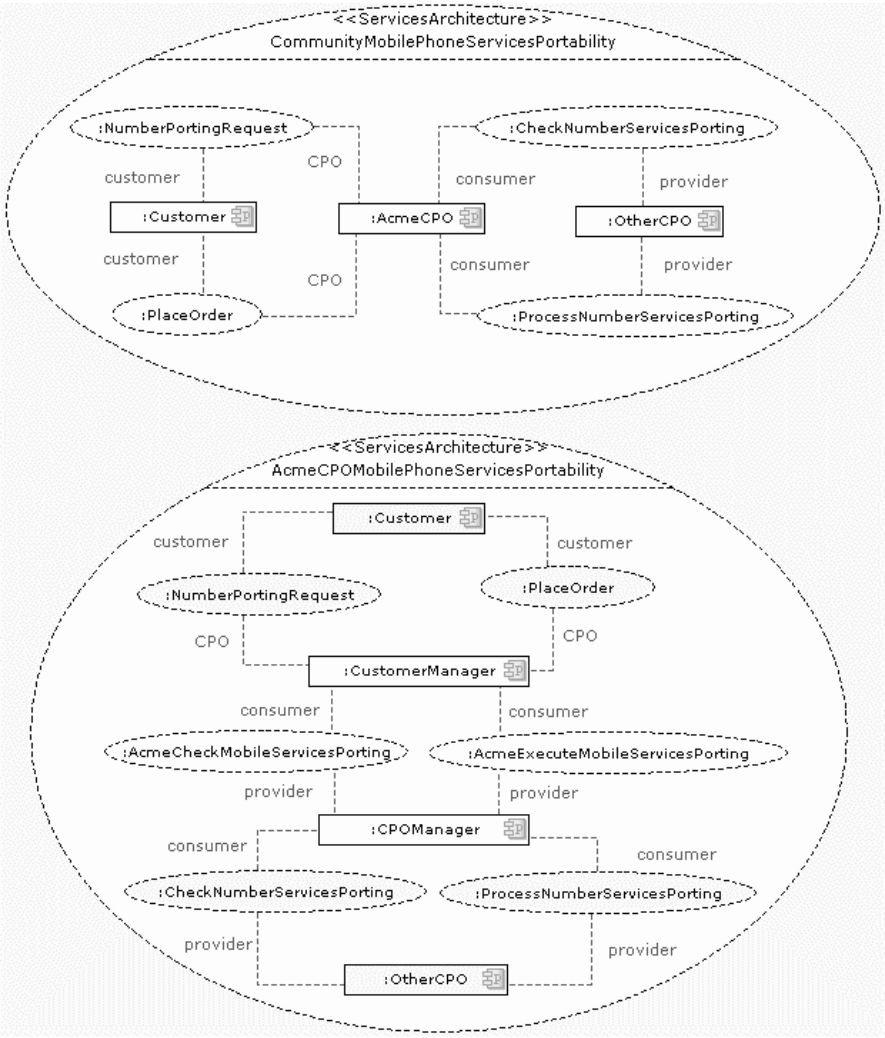

Fig. 9 Services architecture (UML collaboration diagram)

3.5 Service Contracts and Choreographies

SoaML allows different approaches to specify services. In our methodology, we have chosen to combine two different approaches, as we see one fits more at the business level (service contract approach) and the other at the IT level (service interface approach). In the business architecture modelling we suggest to use service contracts that are further refined to service interfaces in the system architecture modelling.

A *service contract approach* defines service specifications (the service contract) that define the roles each participant plays in the service (such as provider and consumer) and the interfaces they implement to play that role in that service. The service contract represents an agreement between the involved participants for how the service is to be provided and consumed. This agreement includes the interfaces, choreography and any other terms and conditions. Service contracts are frequently part of one or more services architectures that define how a set of participants provide and use services for a particular business purpose or process.

3.5.1 Modelling of Service Contracts

The specification of service contracts can be seen as the refinement of a services architecture. Service contracts are specified as UML collaborations stereotyped ≪ServiceContract≫.

The first step is to analyse the BPMN diagrams to identify service contracts. This is a design-choice as there is no single construct in the BPMN that resembles a service contract. However, a certain pattern of objects can reveal service contracts, for instance when two single tasks follow one after another across a pool or lane and are connected with a sequence flow and associated with a data object.

Four services contracts were identified from the community process (Figure 5). Two between the *Customer* and *AcmeCPO*: requesting a file number porting and place the order (and pay the bill). Two other service contracts were identified between *AcmeCPO* and *OtherCPO*: checking and actually processing the number porting. Another two service contracts were identified between the *CustomerManager* and the *CPOManager* in the analysis of the internal process (Figure 6). The six identified service contracts are shown in Figure 10.

Once the service contracts are identified, one has to specify the consumer and provider roles. These roles are typed by corresponding UML interfaces stereotyped ≪Consumer≫ and ≪Provider≫ respectively. At this modelling step we only identify the names and possibly some high-level operations in the interfaces. These interfaces will be further elaborated and refined as part of the system architecture modelling. Figure 10 shows the six service contracts, with their interfaces. The first one, *NumberPortingRequest*, represents a simple service where any consumer can use the service without any contractual obligations. It has two roles: *customer* and *CPO*, but only the *CPO* role has an interface type, namely *NumberPortingRequest*. The five other service contracts specify that there are contractual obligations on both the consumer and provider side, which means that both roles must have an interface type. For example, the service contract *PlaceOrder* specifies the interface *CustomerOrder* for the *consumer* role and the interface *MobileServicesOrder* for the *CPO* role.

Each interface will define (business) operations. The arguments of those operations specify the information elements that are exchanged. These information elements can be derived from the business processes, where the information items map to a message type or a data entity in SoaML. Message types and data entities are

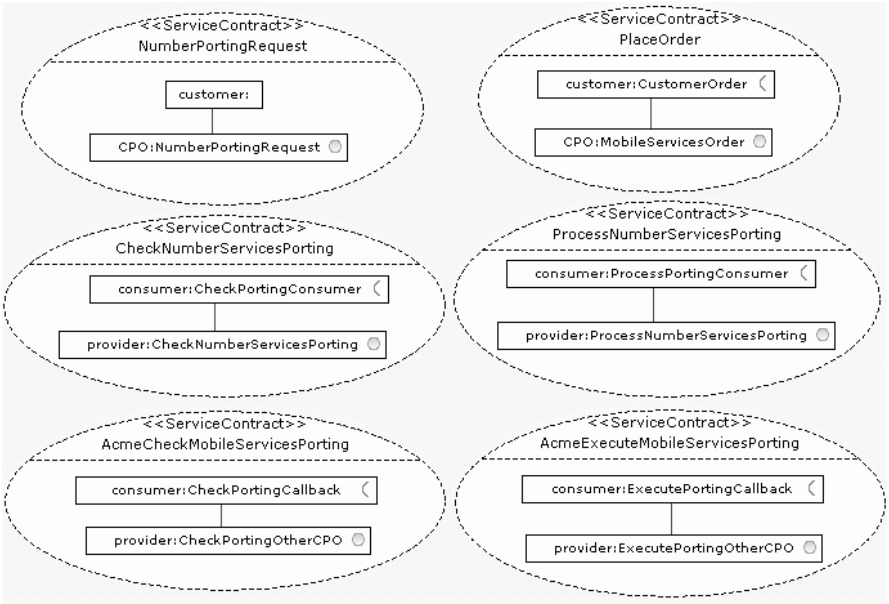

Fig. 10 Service contracts (UML collaboration diagram)

defined as stereotypes on a UML class. So a first step would be to create these information items as regular UML classes, and then refine them to either message types or data entities as part of the system architecture modelling (see Section 4.2.1). The information modelled here does not need to be complete. It may be sufficient to just link the class to a particular information standard or just describe the most important properties of the data objects. For instance Figure 11 shows a partial view of the data objects identified in the BPMN process (Figure 5). These data objects are represented as SoaML message types and have been linked with the BPMN data objects.

3.5.2 Modelling of Service Choreographies

A choreography is a specification of what is transmitted and when it is transmitted between participants to enact a service exchange. The choreography defines what happens between the provider and consumer participants without defining their internal processes - their internal processes do have to be compatible with their service contracts.

We recommend to model the behaviour of any complex service contract in order to get a better understanding of the interaction between the roles. A starting point for specifying the behaviour of the service contracts are the BPMN process diagrams. Thus, BPMN may be the preferred formalism to use for describing service choreographies at the business level. Figure 12 shows the part of the BPMN process

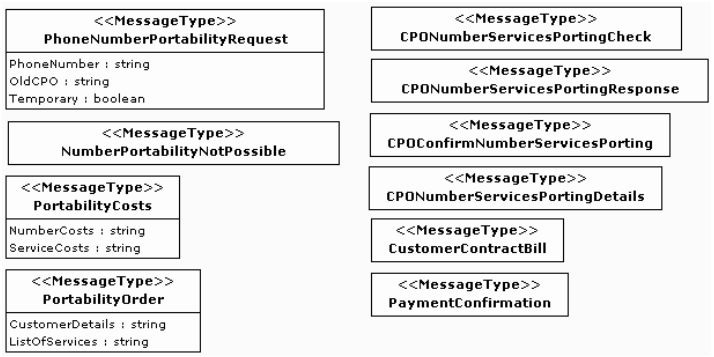

Fig. 11 Message types (UML class diagram)

related to the service contract *PlaceOrder*. Modelling the service choreographies also gives you an opportunity to revise and further refine the information exchange between the two parties. This model will later be refined at the IT level as part of the modelling of the corresponding service interface and its service choreography (see Section 4.3).

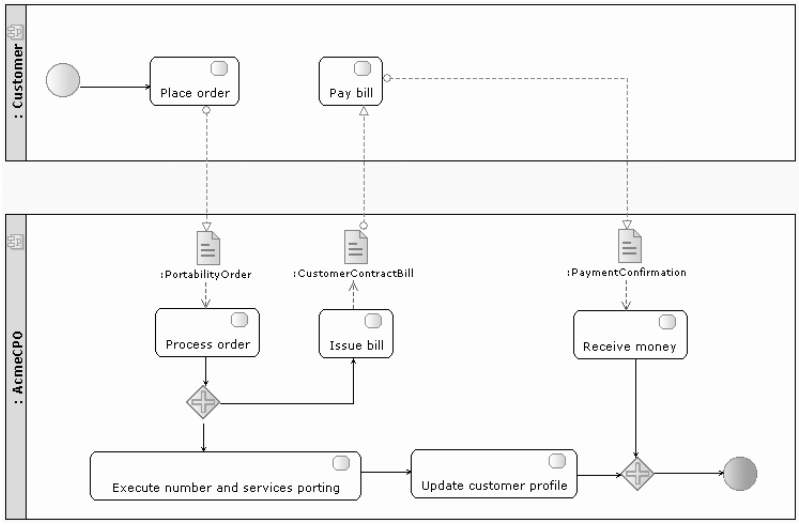

Fig. 12 Service choreography for the *PlaceOrder* service contract (BPMN diagram)

4 System Architecture Modelling

The *System Architecture Model (SAM)* describes the IT perspective of a service-oriented architecture. The SAM is a refinement of the BAM, and is used to express the overall architecture of the system at the PIM level. It partitions the system into components and defines the components in terms of what interfaces they provide, what interfaces they use, and how these interfaces should be used (protocol). Two aspects of component collaborations are described: the static model (structure) and dynamic model (behaviour). The structural model describes the components, their dependencies, and their interfaces; the dynamic model describes the component interactions and protocols.

The methodology first starts by refining the service contracts of the BAM by defining service interfaces (Section 4.1) with the associated interfaces and messages (Section 4.2) and service choreographies (Section 4.3) which define the protocols to use when accessing those interfaces. Software components are then specified and composed in a component architecture (Section 4.4). Figure 13 depicts an activity diagram that shows the modelling tasks involved in the specification of the System Architecture Model.

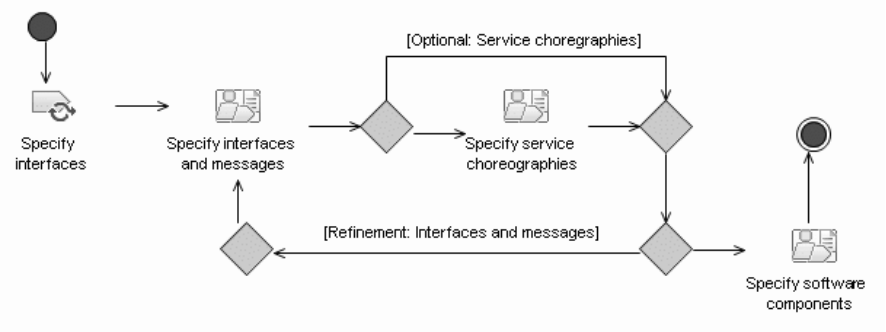

Fig. 13 System architecture modelling activities (EPF activity diagram)

4.1 Service Interfaces

The service contracts specified in the business architecture modelling (Section 3.5) are refined to *service interfaces* in the system architecture modelling. The service interfaces specify the interactions between the software components of the service-oriented architecture: service interfaces will serve as port types on software components (Section 4.4). SoaML allows for two approaches:

- **Simple interface based approach:** A *simple interface* specifies a uni-directional service, focusing on a one-way interaction provided by a participant on a port. The participant receives operations on this port and may provide results to the caller. A simple interface can be used with ”anonymous” callers where the participant makes no assumptions about the caller or the choreography of the service. The one-way service corresponds most directly to simpler remote procedure call (RPC) style Web services.
- **Service interface based approach:** A *service interface* specifies a bi-directional service, where ”callbacks” exist at the provider's side (in addition to the usual operations at the consumer's side), allowing an exchange from the consumer to the provider as a part of a conversation between the two parties. A service interface defines the interface and responsibilities of the provider and the consumer of a service, by including commands and information by which actions are initiated (Section 4.2), and by optionally including specific protocols (Section 4.3).

A service interface will type the service port of a software component, hence specifying that this component provides the service on that port (Section 4.4). SoaML also defines a *conjugate service interface*, where the consumer and provider are inverted from the associated service interface; the name of a conjugate service interface is the same as the associated service interface, with a tilde '~' in front. A conjugate service interface will type the request port of a software component consuming the service (Section 4.4).

4.1.1 Modelling of Service Interfaces

Service interfaces are modelled as UML classes stereotyped ≪ServiceInterface≫. The associated required and provided interfaces are modelled as UML interfaces with the stereotypes ≪Consumer≫ and ≪Provider≫ respectively; they are represented inside the service interface as parts with a connection. This is illustrated in Figure 14, where the ≪ServiceInterface≫ *PlaceOrderInterface* contains two parts, namely the ≪Provider≫ interface *CustomerOrder* and the ≪Consumer≫ interface *MobilServiceOrder*.

The required and provided interfaces are refinements of the interface types of the consumer and provider roles defined in the service contracts. However, in the service contracts these interfaces were primarily defined as business-level interfaces. These interfaces can be refined directly, or possibly mapped to a new set of interfaces that will be used to detail the IT-level artefacts. Figure 14 shows such a refinement, where each service contract maps to a corresponding service interface with a new set of consumer and provider interface types. These interfaces will be used to further detail the specification for the software components (see hereafter).

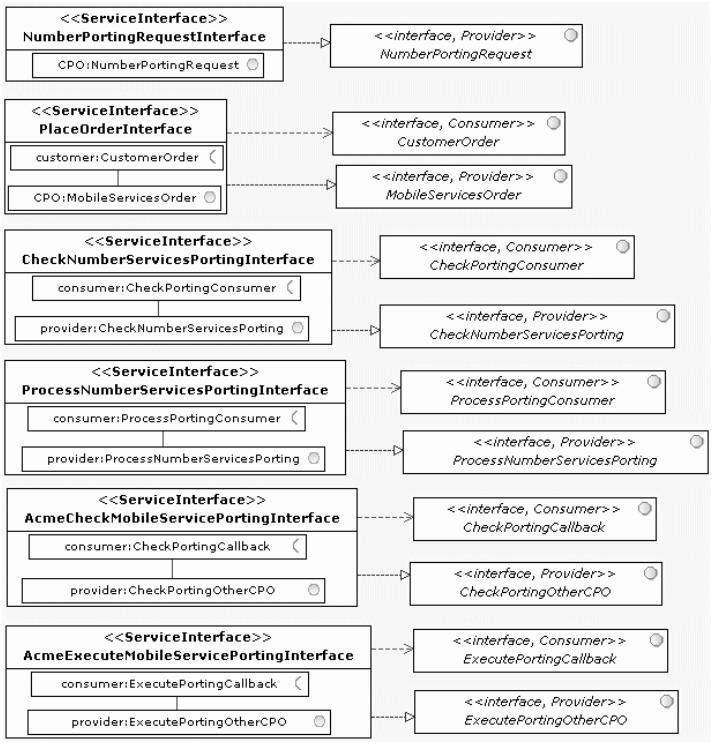

Fig. 14 Service interfaces (UML class diagram)

4.2 Interfaces and Messages

The provided and required interfaces of the previous section are refined with operations and callbacks, hence specifying how to interact with a component that provides or uses such interfaces. The operations are specified in the provided interface, while the callbacks are specified as part of the required interface, hence allowing a conversation between provider and consumer of a service.

Arguments of operations and callbacks can make use of messages, which specify the kind of data expected. There are several SOA interaction paradigms in common use, including document centric messaging, remote procedure calls (RPC), and publish-subscribe. The decision depends on cohesion and coupling, state management, distributed transactions, performance, granularity, synchronization, ease of development and maintenance, and best practices. SoaML supports document-centric messaging and RPC-style service data:

- **Document-centric messaging:** *Message types* specify the information exchanged between service consumers and providers. Message types represent ”pure data” that may be communicated between parties – it is then up to the parties, based on the SOA specification, to interpret this data and act accordingly. As ”pure data”

message types may not have dependencies on the environment, location or information system of either party. A message type is in the domain or service-specific content and does not include header or other implementation or protocol-specific information.
- **RPC-style service data:** *Service data* is data that is exchanged between service consumers and providers. The data types of parameters for service operations are typed by a data type, primitive type, or message type.

The choice of document-centric or RPC-style approach affects how to model the operation signatures (parameters and responses). The choice may differ from interface to interface within the same component architecture, depending on, e.g. technology platform choices and whether the services at stake are public external services or private internal services.

4.2.1 Modelling of Interfaces and Messages

As written in Section 4.1, provided and required interfaces are stereotyped with ≪Provider≫ and ≪Consumer≫. The provided interface will contain the operations of the service, while the required interface can have callbacks, which are specified as signals. For instance Figure 15 shows the ≪Provided≫ interface *MobileServicesOrder* with the operations *placeOrder* and *payBill*, and the ≪Consumer≫ interface with the callback *confirmOrderAndPay*; this callback asks for the consumer to actually pay the bill. Service choreographies (Section 4.3) specify how operations and callbacks are put together into a conversation between the two parties.

Message types are represented as UML classes stereotyped ≪MessageType≫. Service data are represented as UML classes with the stereotype ≪entity≫. The specification of message types and service data are closely linked to the specification of the operations and callbacks in the interfaces. For a document-centric approach you will typically only specify one input parameter and one response parameter that are typed as message types. Both message types and service data may have properties that can be either modelled as UML properties or associated UML classes. This is illustrated in Figure 15 where the two operations and the callback have been specified with their own separate message types as input parameters and responses. The message types *OrderRequest* and *OrderResponse* have both properties and associated classes that contain additional information to support the customer in browsing the cost of the services before selecting the ones to be ported and enabling whether a temporary or fixed porting should be enabled.

4.3 Service Choreographies

The behaviour of a service interface expresses the expected interaction between the consumers and providers of services. It is a refinement of the behaviour of service

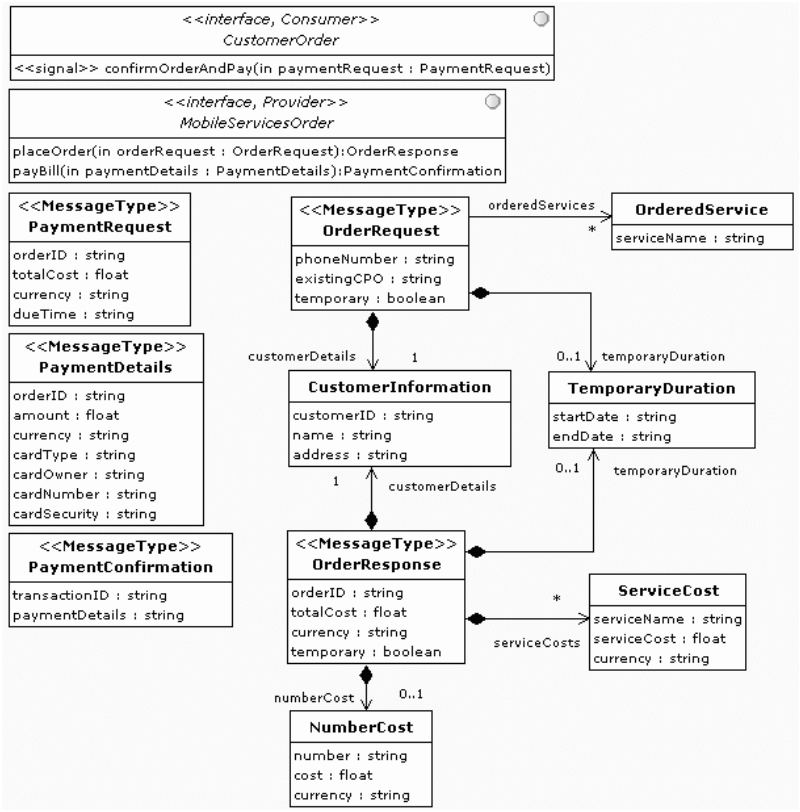

Fig. 15 Interfaces and message types (UML class diagram)

contracts (Section 3.5), and is mostly used in association with bi-directional services, where operations and callback need to be sequenced in a specific manner. The components taking part in the service at stake will then have to act according to the protocol specified by the service choreography.

4.3.1 Modelling Service Choreographies

Service choreographies can be specified as any UML behavior, the most common ones being activity, interaction or state machine. One has to specify the message sequence between the consumer and provider interfaces. The sequence should be linked to operations defined in the interfaces. The modelling of the service choreography is an iterative process that is linked to revising the interfaces and messages (Section 4.2), until a complete service choreography can be specified.

Figure 16 specifies the behaviour of the service interface *PlaceOrderInterface* using an interaction diagram. The service interface describes a bi-directional ser-

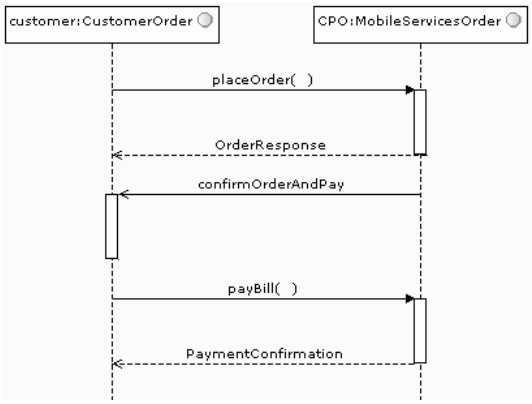

Fig. 16 Service choreography or service interface behaviour defined as a UML interaction (UML sequence diagram)

vice with a call-back at the client-side (see Figure 15). The *customer* invokes the operation *placeOrder* on the interface *MobileServicesOrder*. A message type *OrderResponse* is returned to the customer. Then the *CPO* invokes the callback *confirmOrderAndPay* on the interface *CustomerOrder* at the customer-side which triggers a signal forcing the customer to confirm and pay for the order and invoke the operation *payBill*.

4.4 Software Components

The component model focuses on specifying the involved software components that realizes the services architecture specified during the *business architecture modelling*, either for a community or a participant. Once the components are defined, a composite structure is used to show how implementations of these components form a composite service-oriented application. Service interfaces (Section 4.1) are used to type the ports of the components, which means that those components must abide by the specified provided/required interfaces and message types (Section 4.2) as well as the associated protocols (Section 4.3). While the services architecture of the *business architecture modelling* gathers all the pieces of the BAM together, the component model does the same for the *system architecture modelling*.

4.4.1 Modelling Software Components

Software components are modelled as components with composite structures using class diagrams in UML. Each participant in the services architecture of the BAM can be refined into SoaML participant in the SAM. SoaML participants of the SAM

represent software components that realize the service contracts specified for the business organizations (specified as SoaML participant) in the Business Architecture Model. For instance the three participants of the community services architecture of Figure 9 are refined and assembled in the component model of Figure 17 (i.e. components *CustomerWeb*, *AcmeCPOServices* and *OtherCPOServices*).

The next step is to connect the software components together, through their ports. Each port of a component is either a ≪Service≫ port or a ≪Request≫ port. The former will provide a service, and is typed by a service interface; the latter is a consumer of a service, and is typed by the conjugate service interface. A ≪Service≫ port will then have to implement the operations specified in the associated ≪Provider≫ interface, while the ≪Request≫ port will have to implement the callbacks specified in the associated ≪Consumer≫ interface. The ports can then be connected such that they have matching types. This is exemplified in Figure 17, where the three components are connected using request and service ports. For instance, the component *CustomerWeb* has a port which is typed by the conjugate service interface *˜PlaceOrderInterface*, which means that it has to implement the callback *confirmOrderAndPay* specified in the ≪Consumer≫ interface *CustomerOrder*. Similarly the component *AcmeCPOServices* has a port which is typed by the service interface *PlaceOrderInterface*, which means the component has to implement the operations defined in the ≪Provider≫ interface *MobileServicesOrder* (see Figure 14). As the two ports are typed by matching interfaces, they can be connected together.

Finally, the internals of the software components can be further refined. The two components *CustomerManager* and *CPOManager* inside the *AcmeCPOServices* corresponds to the participants defined for the participant-level services architecture of Figure 9. These two internal components are interconnected through the *AcmeCheckMobileServicesPortingInterface* and *AcmeExecuteMobileServicesPortingInterface* request (conjugate service interface) and provider (service interface) port types.

Note that not all software components can be specified fully in this manner. Indeed some components are provided or developed by third parties, which means that their internal structure may be unknown. However, such component must behave according to the service interfaces that type their ports, possibly including the corresponding service choreographies. This is why the system component *OtherCPOServices* of Figure 17 has no internal structure and is specified as a ”black-box”.

5 Related Work and Discussion

In this section, we discuss the work presented in this chapter and compare it with other efforts on service modelling. We first consider the model-driven methodologies in general and how they need extensions to cover service engineering activities. Then we focus on service engineering methodologies and how our service mod-

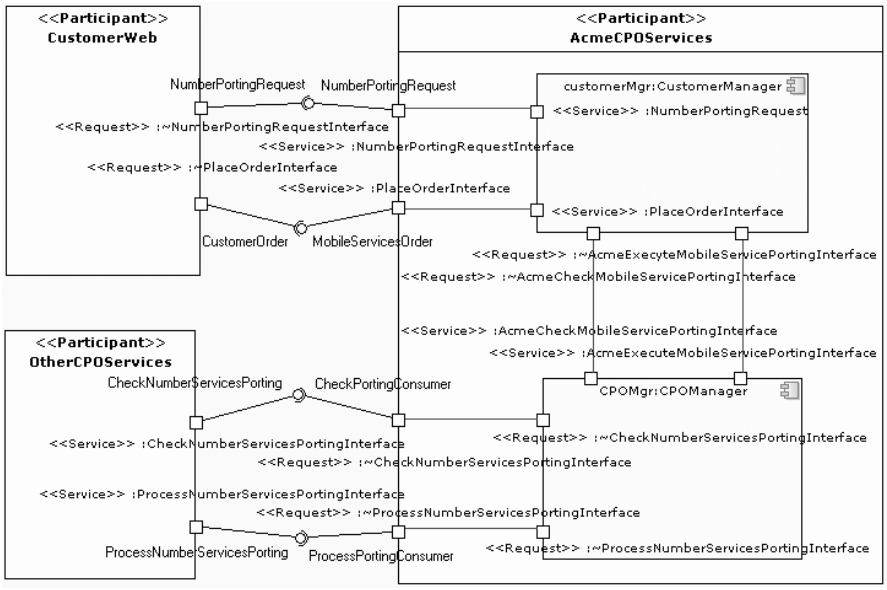

Fig. 17 Software components (UML class diagram)

elling approach can be integrated in these by using SoaML as a means for service specification.

5.1 Model-Driven Methodologies

There are many general purpose model-driven methodologies such as the Rational Unified Process (RUP) [11] and KobrA [3]. These methodologies have differences and similarities but they all emphasize on the systematic use of models as primary software artefacts throughout the software engineering life-cycle.

However, service engineering needs additional artefacts and activities which are not found in development paradigms such as object-oriented analysis and design (OOAD) where MDE is commonly applied. There are especially two characteristics of service engineering that should be addressed by a model-driven service engineering methodology:

- Core concepts of SOA such as service, services architecture and service contracts should be covered and there should be specific activities in the methodology that lead to the specification and modelling of these core concepts.
- Service orientation essentially means focusing on service functionality and services architecture, and how services fulfil business goals. This requires modelling services in a platform independent way. Implementation details on the other hand

are covered by transforming the PIMs to the selected platforms and should be hidden at the service modelling stage.

Thus existing general methodologies with focus on objects and components need higher level abstraction level including services which is the contribution of this work.

5.2 Service Engineering Methodologies

There are numerous methods for technologies related to service-based system engineering where [4] provides a comprehensive overview. SOAD (Service Oriented Analysis and Design) [18] developed by IBM is one of the first systematic approaches to service engineering. The idea was to combine object-oriented analysis and design, enterprise architecture and business process modelling in a hybrid approach in order to support SOA deployment. This idea was further developed in SOMA (Service Oriented Modeling and Architecture) [2], where services, flows and components realizing services are identified. The methodology describes a business-driven top-down approach, combined with and IT-based bottom-up approach (where existing services can be reused). Our approach is therefore close to the process of SOMA in starting from business models and aligning services to business goals. Another methodology that have influenced our work was developed in the FP6 EU project SIMS[5] [8] which defined a top-down approach for specifying mobile services, starting from a high level specification of the system and modelling composite collaboration representing the services and refining these to fine grained specification of components, interfaces and data.

Our contribution to service engineering methodologies is introducing SoaML as the modelling language to be used in the specification and modelling of services. SoaML standardization is in the final phase and it is already supported by several modelling tools. Thus SOA methodologies will benefit from using the language in their life cycle models and include activities proposed in this chapter in order to take advantage of the standard. Other phases of development such as deployment, monitoring and management are out of the scope of this work.

5.3 Discussion

Any project that adopts SOA should cover the phases of service identification and specification, in addition to the phases of composition, realization, monitoring and management of services. This paper proposes a methodology that starts from higher-level models such as goal models, requirement models or business process models down to the modelling of services and their realization by software components. We

[5] Semantic Interfaces for Mobile Services (SIMS), http://www.ist-sims.org/

have covered key concepts, activities, and models pertaining to a service engineering life-cycle method. The methodology takes advantage of SoaML as the modelling language and is therefore complementary to existing service engineering methods that are either silent about the language or take advantage of non-standard solutions for the purpose of service modelling. Another contribution of the work is proposing relations between business processes and goals and service constructs that may be basis for transformations or validation activities.

Many of the service-oriented methods analysed are mostly dedicated to a specific technology and define static procedures that can hardly be combined into a comprehensive methodology for integrated engineering frameworks. Because of the fact that engineering situations vary considerably from one application system development project to another, the traditional systems development methods are often not well suitable. Even though they claim to be universal and propose a large number of models and views for system analysis and specification, they cannot foresee all possible development situations. To overcome this, our approach follows the idea of Situational Method Engineering (see [10] and [6]) where reusable method chunks are assembled into customized engineering methods for particular application scenarios. Most of them use assembly techniques based on the reuse of existing method parts in the construction of new methods or in the enhancement of existing ones.

We have chosen the Eclipse Process Framework (EPF)[6] as the technical infrastructure for implementing our methodology. EPF is an open-source project for defining customizable software engineering processes, providing a specification framework for methods and processes along with editing and content management facilities. Recent approaches aim at providing a generic infrastructure for customizable software engineering methodologies. Most notably, OpenUP provides an open-source implementation of the Unified Process – a generic framework for iterative software engineering processes [9] – within EPF, and the IBM Rational Method Composer (RMC)[7] provides a commercial tool with IBM's own SoaML-based Service-Oriented Modeling and Architecture (SOMA) methodology [1]. Thus EPF provides the possibility to integrate the steps proposed in our methodology with other processes that cover other activities of software development.

6 Concluding Remarks and Future Work

In this chapter we have presented an overview of our MDSE methodology and in particular how the SoaML modelling language has been applied in a top-down manner to model a subset of the telecommunication use case. By following the methodology, which uses the SoaML language to represent both a business perspective and an IT perspective of SOA, better business and IT alignment can be achieved since the IT-level model can be viewed as a refinement of the business-level SOA model.

[6] Eclipse Process Framework (EPF), http://www.eclipse.org/epf

[7] Rational Method Composer (RMC), http://www-01.ibm.com/software/awdtools/rmc

Additionally, the SoaML model artefacts can be linked to business goals described in BMM to further help in the alignment process.

The MDSE methodology is currently in the finalization phase and is being revised according to user feedback and experience. Moreover, the methodology is being aligned with the latest changes in the SoaML specification which is also currently under finalization in the OMG. One aspect of the methodology that requires further work is to provide better guidelines for behavioural modelling. SoaML is quite open with regards to behavioural modelling, and explicitly states that any UML behaviour can be used. There is also a synchronization and integration to be done with the ongoing BPMN 2.0 specification [15], which introduces some service concepts that overlaps with the SoaML specification.

The MDSE methodology is implemented in the Eclipse Process Framework (EPF) to allow extensibility and open access. The core methodology presented focuses on the computational independent and platform independent abstractions levels according to the OMG MDA approach. The methodology has also been extended to support various platform technologies, in particular Web services, multi-agent platforms and Semantic Web Services. These correspond to PIMs for different architectural styles. Flexibility of the solution lies in being independent of various platforms in service design while it is extensible, generalization lies in standardizing the developed UML profile and metamodel, the business value lies in being adaptable to various platforms, while innovation lies in combining SOA with model-driven development and taking advantage of advances in both fields.

Acknowledgements The SoaML methodology presented here has mainly been developed in the 7th Framework Programme research project SHAPE (ICT-2007-216408). The overall aim of the project is to develop the foundations for the model-driven development of service-oriented system landscapes with support for the integration of other technologies in order to increase the effectiveness and quality of modern software and system engineering.

The authors acknowledge and thank collaboration with partners within the SHAPE project for stimulating input and feedback since the start-up in 2007.

References

1. Amsden, J.: Modeling with SoaML. Technical article, IBM (7 January 2010). Online: http://www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/index.html
2. Arsanjani, A.: Service-Oriented Modeling and Architecture - How to identify, specify and realize services for your SOA. Technical article, SOA and Web Services Center of Excellence, IBM, Software Group (9 November 2004). Online: http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/
3. Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wust, J., Zettel, J.: Component-based Product Line Engineering with UML. Addison Wesley (2002)
4. Bastida, L., Berre, A.J., Elvesæter, B., Hahn, C., Johnsen, S.G., Kamper, S., Kerrigan, M., Larrucea, X., Limyr, A., Muth, M., Nilsen, G., Roman, D., Rubina, J.M., Stollberg, M.: Model-driven Methodology and Architecture Specification. Deliverable D2.1, SHAPE Project (2009)

5. Blum, B.I.: A taxonomy of software development methods. Commununications of the ACM **37**(11), 82–94 (1994)
6. Brinkkemper S. Saeki, M., Harmsen, F.: Assembly Techniques for Method Engineering. In: 10th Conference on Advanced Information Systems Engineering, CAiSE'98, LNCS 1413, pp. 381–400. Springer (1998)
7. Estefan, J.A.: Survey of model-based systems engineering (MBSE) methodologies. Incose MBSE Focus Group (2007)
8. Floch, J., Carrez, C., Cieślak, P., Rój, M., Sanders, R.T., Shiaa, M.M.: A comprehensive engineering framework for guaranteeing component compatibility (2010). Journal of Systems and Software, to appear
9. Kroll, P., MacIsaac, B.: Agility and Discipline Made Easy: Practices from OpenUP and RUP. Addison-Wesley (2006)
10. Kumar, K., Welke, R.: Method Engineering: A Proposal for Situation-specific Methodology Construction. In: Cotterman, Senn (eds.) In Systems Analysis and Design : A Research Agenda, pp. 257–268. Wiley (1992)
11. Kurchten, P.: The Rational Unified Process: An Introduction. Addison Wesley (2003)
12. MDA Guide Version 1.0.1. Object Management Group, Document omg/03-06-01 (2003)
13. Business Motivation Model (BMM), Version 1.0. Object Management Group, Document formal/08-08-02 (2008). Online: http://www.omg.org/spec/BMM/
14. Business Process Model and Notation (BPMN), Version 1.2. Object Management Group, Document formal/2009-01-03 (2009). Online: http://www.omg.org/spec/BPMN/1.2/
15. Business Process Model and Notation (BPMN), Version 2.0 - Beta 1. Object Management Group, Document dtc/2009-08-14 (2009). Online: http://www.omg.org/spec/BPMN/2.0/
16. Service oriented architecture Modeling Language (SoaML), Version 1.0 - Beta 2. Object Management Group, Document ptc/2009-12-10 (2009). Online: http://www.omg.org/spec/SoaML/
17. Unified Modeling Language (UML), Infrastructure, Version 2.2. Object Management Group, Document formal/2009-02-04 (2009). Online: http://www.omg.org/spec/UML/2.2/
18. Zimmermann, O., Krogdahl, P., Gee, C.: Elements of Service-Oriented Analysis and Design - An interdisciplinary modeling approach for SOA projects. Technical article, IBM (2 June 2004). Online: http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/