MODELING OF SERVICE ORIENTED ARCHITECTURE
From Business Process to Service Realisation

Marek Rychlý and Petr Weiss
Faculty of Information Technology, Brno University of Technology, Czech Republic

Keywords: Service-Oriented Architecture, Business Process Model, Service Specification, Composite Services.

Abstract: This paper deals with modeling of Service-Oriented Architecture (SOA). SOA is an architectural style for analysis, design, maintaining and integration of enterprise applications that are based on services. Services are autonomous platform-independent entities that enable access to one or more capabilities, which are accessible by provided interfaces. The goal of SOA is to align business and IT architectures. Hence, a new designed service has to meet business requirements that are traditionally specified by a business process diagram. The approach, presented in this paper, helps to bridge the semantic gap between business requirements and IT architecture by using a method for transformation of business processes diagrams into services diagrams. In particular, the method deals with process realisation based on services and it describes choreographing of services towards fulfilling business goals.

1 INTRODUCTION

Service-Oriented Architecture is an architectural style for aligning business and IT architectures. It is a complex solution for analyses, design, maintaining and integration of enterprise applications that are based on services. Services are autonomous platform-independent entities that enable access to one or more capabilities, which are accessible by provided interfaces. A new designed service has to meet business requirements that are traditionally specified by a *Business Process Diagram* (BPD).

This paper deals with modeling of business process realisation. For this purpose, a method based on service modeling is introduced. The aim of the method is to transform business processes modeled in the Business Process Modeling Notation (Object Management Group, 2006a) into UML service diagrams. Those diagrams show how to choreograph services to fulfil business goals. Furthermore, the transformation method is designed considering fundamental SOA principles (Erl, 2005) such as loose coupling, service independence, stateless and reusability. This approach leads to easy extensibility of the proposed method, e.g. by using business services templates (Constantinides and Roussos, 2005), and allows linking the services to underlaying component-based systems with support of formal specification (Rychlý, 2007).

The remainder of this paper is organised as follows. In Section 2, the exemplary business process is introduced in more detail. The Section 3 describes service specification according to the mentioned business process. Section 4 focuses on some issues related to modeling of composite services, such as passing of data between services and holding states of services that are participating in the choreography of a composite service. Section 5 finishes transformation of the exemplary business process into a service that provides requested business functionality. Section 6 reviews main approaches that are relevant to our subject and discuss advantages and disadvantages of our approach compared with the reviewed approaches. To conclude, Section 7 summarises the presented approach, current results and outline the future work.

2 BUSINESS PROCESS MODEL

It is evident that an input for the method of transformation of BPD into service diagrams is a business process (BP). Figure 1 shows an exemplary "Purchase Order" *business process model* (BPM). This

Rychlý M. and Weiss P. (2008).
MODELING OF SERVICE ORIENTED ARCHITECTURE - From Business Process to Service Realisation.
In *Proceedings of the Third International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 140-146
DOI: 10.5220/0001765101400146

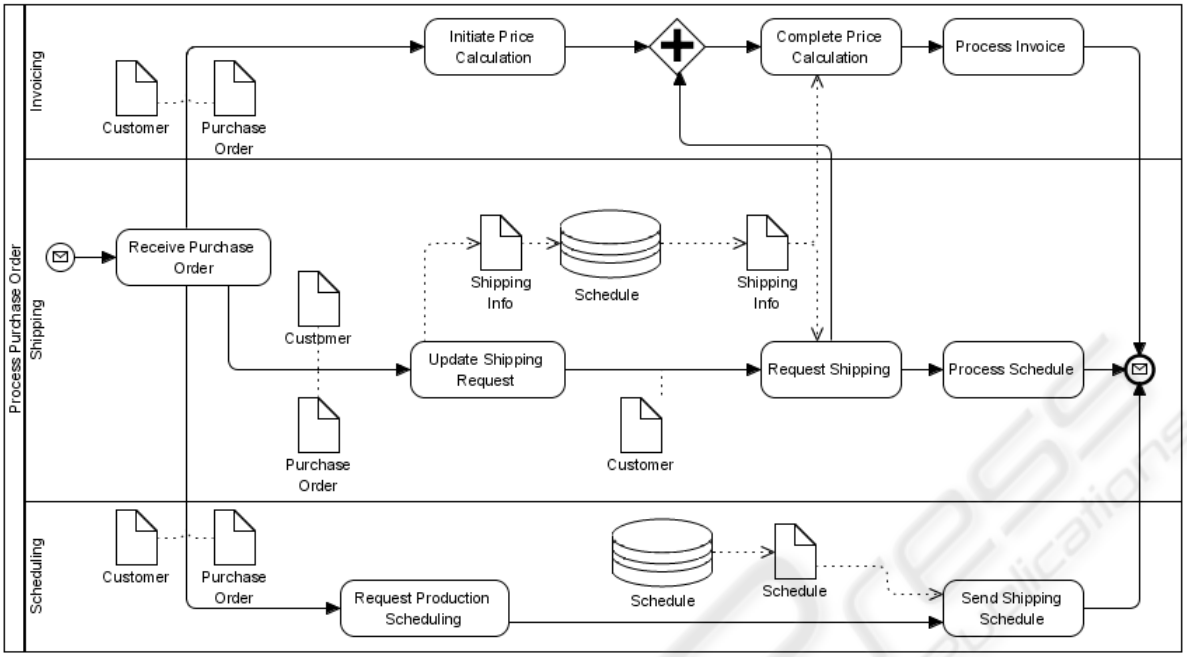

Figure 1: Business process model of the "Purchase Order" process.

example is adopted from (Object Management Group, 2006b).

As we can see the Figure 1, there are three roles that are responsible for realisation of the "Purchase Order" process: "Invoicing", "Shipping" and "Scheduling". Processing starts by receiving a purchase order message. Afterwards, the "Invoicing" role calculates an initial price. This price is not yet complete, because the total price depends on where the products are produces and the amount of the shipping cost. In parallel, the "Shipper" role determines when the products will be available and from what locations. After the shipping information is known, the complete price can be evaluated. At the same time, the process requests shipping schedule from the "Scheduler" role. Finally, when the complete price, shipping info and shipping schedule are available, the invoice can be completed and sent to the customer.

The next section describes the transformation of a BP diagram into service diagrams (BPD2SD transformation) in details.

3 MODEL TRANSFORMATION

According to (Arsanjani, 2004), the initial activity in the development of a new SOA-based system is the *service identification*. It consists of a combination of top-down, bottom-up, and middle-out techniques of domain decomposition of legacy systems, existing asset analysis, and goal-service modeling. The result of the service identification is a set of candidate services. More details about service identification can be found in (Inaganti and Behara, 2007). In the context of service oriented design, the service identification is a prerequisite for the BP2SD transformation. Since this paper presents basics of the transformation and uses a motivating example, the service identification was omitted.

The BPD2SD transformation consists of two basic steps. The first step is to identify which tasks from the BPD represent service invocations and therefore will be modeled as services in service diagrams. This decision is closely associated with the service identification and takes into account such aspects as which service providers provide which services, Quality of Service (QoS) requirements, security issues, etc. (Arsanjani, 2004) Such an analysis is beyond the scope of this paper. Here, we will assume that following tasks (from Figure 1) were identified as service invocations: "Initiate Price Calculation", "Complete Price Calculation", "Request Shipping", "Request Production Scheduling" and "Send Shipping Schedule". These tasks will be modeled as business services in the next step.

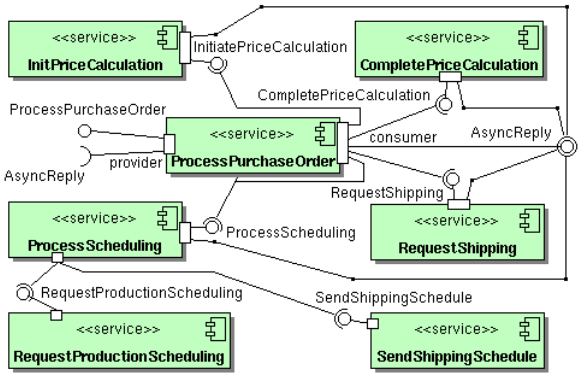

Figure 2: An overview of identified services, their interfaces and connections.

The second step, the transformation process itself, is based on a technique, which is introduced in (Amsden, 2005). The technique integrates business process modeling and object modeling by providing a *Business Services Model* (BSM) that is a mediator between business requirements and an implementation. This paper proposes an extension of the above mentioned technique and focuses more on service realisation.

The Figure 2 shows an overview of identified services derived from the BPM (see Figure 1). There are five primitive services: InitPriceCalculation, CompletePriceCalculation, RequestShipping, RequestProductionScheduling and SendShippingSchedule and two composite services: ProcessScheduling and ProcessPurchaseOrder. *Primitive services* are derived according to service invocation tasks, and are responsible for providing functional capabilities defined by the task. The convention used in this paper is to name a primitive service the same as the related task. A *composite service* is an access point to choreography of other primitive or composite services.

In this case, the ProcessScheduling service represents the business process itself and choreographs the rest of above mentioned services. The ProcessPurchaseOrder demonstrates principle of composite services by choreographing RequestProductionScheduling and SendShippingSchedule. Behaviour of these services is described later in the text.

Each service in Figure 2 is modeled as a *stereotype service*, which extends the UML class component (Object Management Group, 2005). This concept is introduced in (Weiss and Zendulka, 2007). Every service interacts with its environment via *interfaces*. During an interaction, service can act two different roles: *service provider* or *service consumer*. These two roles are distinguished in the service model by means of *a port*. The *provider port* of a service implements interfaces that specify functional capabilities provided to possible consumers of the service, while the *consumer port* requires interfaces of defined services to consume their functionality.

Each service provides at least one interface at the consumer port (see Figure 2). Such an interface involves *service operations*, which realise the functional capability of the service. The parameters of the service operations describe the format of an incoming message. Details of all services (denoted by stereotype service) including their interfaces (stereotype interface) and relationships are shown at Figure 3. The relationships between services and interfaces are labelled by stereotype use for interfaces of required services and by an implementation relation for provided interfaces.

A service can be invoked in two modes: *synchronously* and *asynchronously*. The mode, which is used for a particular interaction, depends on the format of incoming message that is sent to the service (i.e. "a request"). For example, if we want to use the InitPriceCalculation service in synchronous mode, the message will contain only basic service data (customerInfo and purchaseOrder). For the asynchronous mode, it is necessary to add the message identification information (replyToURL and requestID) that is used to identify a destination service for a reply to the asynchronous request.

4 COMPOSITE SERVICES

In our approach, we will prefer *a flat model* to a hierarchical model of the service-oriented architecture. It means that a composite service does not enclose its "internal" services participating in the choreography and does not delegate its interfaces to them. It only represents a controller of these services, i.e. the composite service communicates with its neighbouring services at the same level of hierarchy in the "producer-consumer" relationship (see service ProcessScheduling in our example, which can be viewed as a composite service consisting of services RequestProductionScheduling and SendShippingSchedule). The flat model provides *better reusability* of services, because the context of a service is defined only by its implemented (i.e. provided) and required interfaces, not by its position in the hierarchy. However, there are some problems we must cope with, especially the issue of passing of data between services that are participating in the choreography of a composite service.

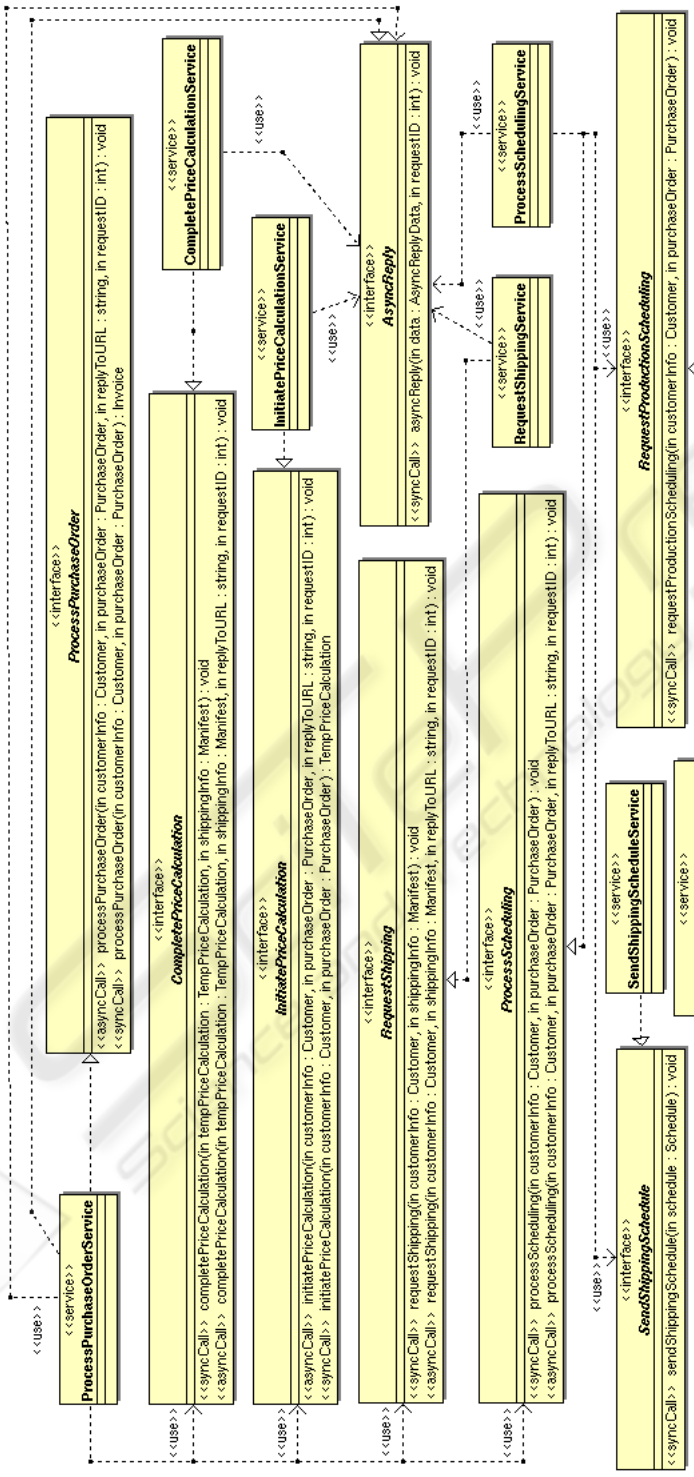

Figure 3: Specification of interfaces and implementations of all services and their relations.

The other characteristic feature of our approach is the restriction on services' interfaces to *provide only one functionality*. Regardless, it is possible to implement service with more interfaces (i.e. with more functional capabilities), but the interfaces must provide independent functionality. It means, services are not allowed to hold any state information between two independent incoming requests, in spite of which interface the service was invoked on. Although "stateless services" provide *better reusability* we must cope with the issue of holding states of services that are participating in the choreography in relation with the state of the composite service (e.g. passing of states between services InitiatePriceCalculation and CompletePriceCalculation in our example).

The solution of both issues, the issue of passing of data between services and the issue of holding states of services that are participating in the choreography, can be demonstrated on behaviour of the services ProcessPurchaseOrder, InitiatePriceCalculation and CompletePriceCalculation in our example (see Figure 3). At first the ProcessPurchaseOrder service requests the service InitiatePriceCalculation. Consequently, the service InitiatePriceCalculation prepares a price calculation (the initial calculation) and stores it for later use, i.e. the service changes a state of the price calculation process (see Figure 1) implemented by the composite service ProcessPurchaseOrder. This state must be transformed into an initial state of the service CompletePriceCalculation, which is requested subsequently by the service ProcessPurchaseOrder and which completes price calculation based on the initial calculation. The service ProcessPurchaseOrder forwards returning data (encoded state) of the first service to the other one. The merit of such design is independence of requested services, although they share data and the state.

5 THE SERVICE ProcessPurchaseOrder

In previous sections, we have described specifications of individual services that are participating in the business process (see Figure 1). More precisely, the services InitiatePriceCalculation, CompletePriceCalculation, RequestShipping and ProcessScheduling were described. This section deals with composing those services into a single service ProcessPurchaseOrder representing the business process in its entirety. *Specification* of the ProcessPurchaseOrder service has to include description of *its structure* (i.e. an architecture of the service) and *its internal behaviour* (i.e. collaboration of involved services and the "orchestration" of the ProcessPurchaseOrder service).

The *architecture* of the ProcessPurchaseOrder service is described in Figure 3. The service provides ProcessPurchaseOrder interface, which allows synchronous and asynchronous calls of the service, and auxiliary AsyncReply interface for accepting replies to asynchronous calls from required services. These required services are represented by their interfaces InitiatePriceCalculation, CompletePriceCalculation, RequestShipping and ProcessScheduling. Connection of the services is noticeable also in Figure 2.

The *behaviour* of the ProcessPurchaseOrder service is described in Figure 4. After receiving a request from a Consumer, the service asynchronously calls services InitiatePriceCalculation, RequestShipping and ProcessScheduling. When both InitiatePriceCalculation and RequestShipping notify (via the AsyncReply interface, see Figure 2) they have finished processing the requests, the CompletePriceCalculation service is called asynchronously with the result received from InitiatePriceCalculation.

While the CompletePriceCalculation is running, ProcessPurchaseOrder computes the "Process Schedule" internal process in parallel (see Figure 5). After CompletePriceCalculation notifies it had finished the computation, the "Process Invoice" internal process is being computed parallel to the "Process Schedule" process. As soon as both internal processes "Process Schedule" and "Process Invoice" are done and ProcessScheduling service notifies it is finished, ProcessPurchaseOrder finishes its processing by sending a result (the invoice) back to the consumer. The result is returned instantly if the service has been called synchronously or it is returned via the AsyncReply consumer's providing interface to the address, which has been included in the asynchronous call.

6 DISCUSSION AND RELATED WORK

The work presented in this paper has been influenced by several different proposals. First of all, we should mention UML profiles for SOA. (Amir and Zeid, 2004) introduces a simple UML profile that is intended to use for modeling of web services. (Johnston, 2005) provides a well-defined profile for modeling of service-oriented solutions. (Ortiz and Hernández, 2006) shows how services and their

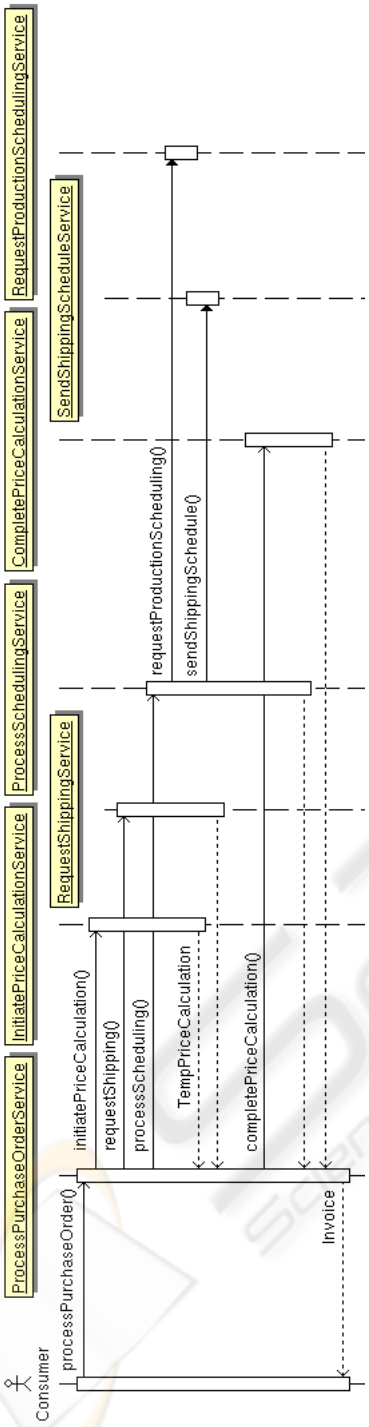

Figure 4: Behaviour of ProcessPurchaseOrder service as a sequence of service calls.

extra-functional properties can be modeled by using UML.

The ideas presented in those approaches are a good starting point to model both the structural and

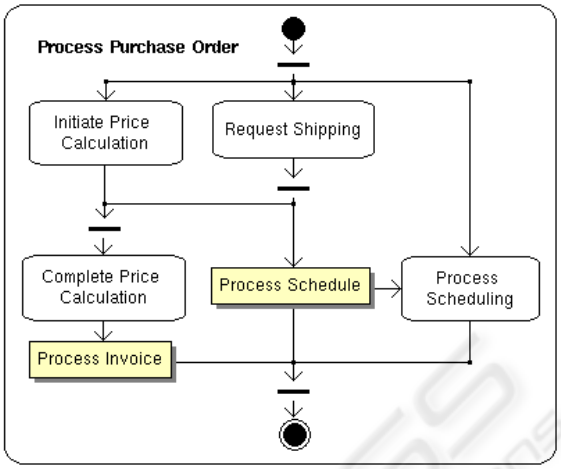

Figure 5: Behaviour of service ProcessPurchaseOrder in the context of used services and internal processes.

the behavioural properties of services by means of UML. However, modeling of services needs to take into account some additional aspects. Primarily, we have to know the connection between business requirements and functional capabilities of modeled services. (Murzek and Kramler, 2007) addresses this as a problem of transformation between several BPM languages. Particularly, *ADONIS Standard Modeling Language*, *Business Process Modeling Notation* (BPMN), *Event-driven Process Chains* and *UML 2.0 Activity Diagrams* are mentioned.

The relationship between BPMN and UML is also introduced in (Amsden, 2005). The author describes a high-level mapping of a BPM to UML 2.0 BSM that represents service specification between business clients and information technology implementers. For the purpose of service modeling, the BSM is too general. Hence, our approach follows the one presented in (Amsden, 2005) while considering SOA principles. It focuses especially on modeling of process realisation and service choreography.

(Object Management Group, 2006b) proposes a similar concept to our approach. However, that concept contains a number of incorrectness that solves our approach. First, one of the fundamental SOA principles, the stateless of services, is ignored. Services are designed in such a way that they store data affecting their functionality between two single incoming requests. Our approach solves this problem by using a composite service (the principle of using of composite services is described in Section 4).

Next, only synchronous communication is supposed in (Object Management Group, 2006b). We suppose both the synchronous and asynchronous invoking of a service, in our approach. In which way

is the service invoked depends on the format of the incoming request (see Section 3).

And finally, in our approach, service's functional capabilities can be easily extend by adding a new provided interface to the consumer port without affecting the previous service specification.

7 CONCLUSIONS AND FUTURE WORK

In this paper we outlined a method for modeling of service oriented systems. The method defines how a business process should be transformed into services and how these services should collaborate to fulfil business goals. Furthermore, we have proposed how to model fundamental SOA principles by means of UML, such as service stateless and reusability. Some of those features are novel and have not been integrated into the existing methods of modeling SOA.

The presented research is a part of a greater project, which deals with modeling and formal specification of SOA and underlaying component-based systems. The approach, which is presented in this paper, is aimed at the modeling of the top layer of several layers. Those layers spread from business-oriented abstraction (represented by a system's business process diagram) to implementation of individual primitive components at the bottom level.

Future work is mainly related to integration of the presented approach with formal component models. The integration allows formal verification of a whole modeled system (e.g. tracing of changes in a business process model to the changes in components' structure and behaviour related to known security issues). Ongoing research includes also automation of business process transformation into services by using business service patterns.

ACKNOWLEDGEMENTS

This research has been supported by the Research Plan No. MSM 0021630528 "Security-Oriented Research in Information Technology".

REFERENCES

Amir, R. and Zeid, A. (2004). A UML profile for service oriented architectures. In Vlissides, J. M. and Schmidt, D. C., editors, *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 192–193. ACM.

Amsden, J. (2005). Business services modeling: Integrating WebSphere business modeler and rational software modeler. IBM developerWorks.

Arsanjani, A. (2004). Service-oriented modeling and architecture: How to identify, specify, and realize services for your SOA. IBM developerWorks.

Constantinides, C. and Roussos, G. (2005). *Service-Oriented Software System Engineering: Challenges and Practices*, chapter Service Patterns for Enterprise Information Systems, pages 201–225. IGI Global, Hershey, PA, USA.

Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Inaganti, S. and Behara, G. K. (2007). Service identification: BPM and SOA handshake. *BP Trends*.

Johnston, S. (2005). UML 2.0 profile for software services. IBM developerWorks.

Murzek, M. and Kramler, G. (2007). Business process model transformation issues. In *Proceedings of the 9th International Conference on Enterprise Information System*.

Object Management Group (2005). UML superstructure specification, version 2.0. OMG Document formal/05-07-04, The Object Management Group. Also available as ISO/IEC 19501:2005 standard.

Object Management Group (2006a). Business process modeling notation (BPMN) specification. OMG Final Adopted Specification dtc/06-02-01, The Object Management Group.

Object Management Group (2006b). UML profile and metamodel for services (UPMS), request for proposal. OMG Document soa/2006-09-09, The Object Management Group, 140 Kendrick Street, Building A Suite 300, Needham, MA 02494, USA.

Ortiz, G. and Hernández, J. (2006). Toward UML profiles for web services and their extra-functional properties. In *IEEE International Conference on Web Services (ICWS'06)*, pages 889–892. IEEE Computer Society.

Rychlý, M. (2007). Component model with support of mobile architectures. In *Information Systems and Formal Models*, pages 55–62. Faculty of Philosophy and Science in Opava, Silesian university in Opava.

Weiss, P. and Zendulka, J. (2007). Modeling of services and service collaboration in UML 2.0. In *Information Systems and Formal Models*, pages 29–36. Faculty of Philosophy and Science in Opava, Silesian University in Opava.