Service-Based Realization of Business Processes Driven by Control-Flow Patterns

Petr Weiss

Service-Based Realization of Business Processes Driven by Control-Flow Patterns

Petr Weiss

Department of Information Systems,
Faculty of Information Technology, Brno University of Technology,
Bozetechova 2, 612 66 Brno, Czech Republic,
weiss@fit.vutbr.cz

Abstract. This paper deals with Service-Oriented Analysis and Design (SOAD). SOAD comprises requirements elicitation, analysis of the requirements to identify services and business processes and finally implementation of the services and processes. It is a heterogeneous approach that combines a number of well-established practices. The key question in SOAD is how to integrate the practices to model a true service-oriented system with all its requirements. 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 process diagrams into service diagrams. The service diagrams describe how the process is realized by using services. In details, the method transforms diagrams denoted in Business Process Modeling Notation into a set of UML diagrams. The principle of the method is demonstrated in an example.

Keywords: Service-Oriented Architecture, Service-Oriented Analysis and Design, Business Process Modeling, Service Specification, Composite Services.

1 Introduction

Service-Oriented Architecture (SOA) is a complex solution for analysis, design, maintenance and integration of enterprise applications that are based on *services*. Services are autonomous platform-independent entities that provide one or more functional capabilities via their interfaces. In other words, SOA is an architectural style for aligning business and IT architectures. If a new service is considered it has to fulfill both business requirements and the fundamental SOA properties such as *loose coupling*, *service independence*, *stateless* and *reusability* [4]. It means that Service-Oriented Analysis and Design (SOAD) should focus not only on services but also on business requirements.

This paper introduces an approach for integration between Business Process Modeling and Service Modeling. The approach is based on a technique that transforms business process models into models of service orchestration by using control-flow patterns. Service orchestration represents realization of a business process. To be more specific, business process diagrams are denoted in Business Process Modeling Notation (BPMN) and service orchestrations are modeled by means of UML.

The reminder of this paper is organized as follows. Section 2 containes an overview of main approaches that are relevant to the subject. Section 3 introduces the concept of primitive and composite services and presents the idea of service specification. Next, in Section 4 the transformation rules are explained. They are illustrated with an examplary business process in Section 5. Finally, Section 6 discusses advantages and disadvantages of our approach compared with the approaches presented in Section 2. Conclusions and future work are contained in Section 7.

2 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. [1] introduced a simple UML profile that is intended to use for modeling of web services. [6] provided a well-defined profile for modeling of service-oriented solutions. In [11] it is showed how services and their extra-functional properties can be modeled by using UML. The ideas presented in these approaches are a good starting point to model both the structural and the behavioral properties of services by means of UML. However, Service Modeling needs to take into account some additional aspects. Primarily, we have to know the connection between business requirements and functional capabilities of modeled services. [13] introduced 21 patterns that describe the behavior of a business process. Next, [16] showed how these patterns can be modeled by means of UML and BPMN. The author discussed the readability and technical properties of each model. Furthermore, [17] analyzed using of Control-flow, Data and Resource patterns in BPMN, UML and BPEL. The relationship between BPMN and UML is also introduced in [2]. The BPMN is defined in [9]. The author describes a high-level mapping of a business process to UML 2.0 Business Service Model (BSM) that represents service specification between business clients and information technology implementers. Lastly, [7] addressed main problems of transformation between business process modeling languages. Particularly, ADONIS Standard Modeling Language, BPMN, Event-driven Process Chains and UML 2.0 Activity Diagrams were mentioned.

3 Service specification. Primitive and composite services

Within this paper, each service is modeled as a stereotype *service* that extends the UML class *Component* [8]. This concept is introduced in [15]. Every service interacts with its environment via interfaces. During an interaction, service can play 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 (stereotyped as `<<prov>>`) of a service implements interfaces that specify functional capabilities provided to possible consumers of the service, while the consumer port (stereotyped as `<<cons>>`) requires interfaces of defined services to consume their functionality.

In our approach, we prefer a *flat model* to a hierarchical model of the service-oriented architecture. For this purpose we introduce two types of services: *primitive services* and *composite services.* Primitive services are derived from service invocation tasks, and are responsible for providing functional capabilities defined by a task. A composite service is an access point to an orchestration of other primitive or composite services. It means that a composite service does not enclose its internal services participating in the orchestration and does not delegate its interfaces to them. It only represents a controller of those services, i.e. the composite service communicates with its neighboring services at the same level of hierarchy in the *producer-consumer relationship.* 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.

A specification of each service includes description of its *architecture* and internal *behavior*. The architecture is defined by interfaces that the service provides and requires. Each service provides at least one interface at the consumer port. Such an interface involves service operations that realize the functional capability of the service. Parameters of these operations describe format of an incoming message. The internal behavior describes which actions are executed when an operation of the service is invoked. In the case of a composite service, the behavior describes an orchestration that this service provides. It means which actions have to be performed in order to execute the corresponding orchestration.

4 The transformation

As mentioned above, SOAD provides techniques required for identification, specification and realization of services and service flows. It is obvious that 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. In the context of this paper, the service identification is a prerequisite for the transformation. Since this paper is mainly focused on the transformation, the details about the service identification are omitted here. More about service identification can be found in [5].

The transformation consists of two basic steps. The objective of the first step is to identify which tasks from the Business Process Diagram (BPD) represent service invocations and therefore will be transformed into services. This decision takes into account such aspects as which service providers provide which services, Quality of Service (QoS) requirements, security issues, etc. [3] and is closely related with the results of the service identification. Such an analysis is beyond the scope of this paper.

The second step, the transformation process itself, generates an orchestration of the identified services. The orchestration represents realization of the given business process. The transformation is based on transformation rules. Each rule defines how to transform a control-flow pattern (eventually basic BPMN element) into an UML service pattern. Each service pattern is described by its structure and by a communication protocol. Due to the limitation of space in this paper, only basic rules are introduced (see Table 1 to Table 3). Some rules include context information that is needed to understand the using of the rule. For this purpose, the actual transformation step is inserted into a red-line-bordered area. To distinguish a primitive service from a composite one, the convention used in this paper is to name a primitive service with a capital letter (e.g. serviceA) and a composite service with a number (e.g. service1). Besides, some BPD elements can by transformed in more than one way (see e.g. Table 2).

Table 1. Transformation rules for basic BPMN elements.

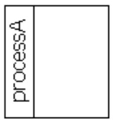
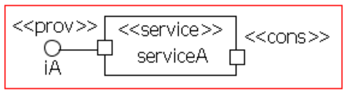
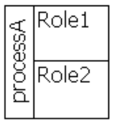
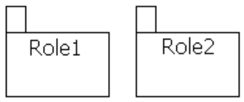
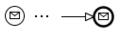
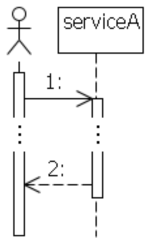
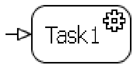
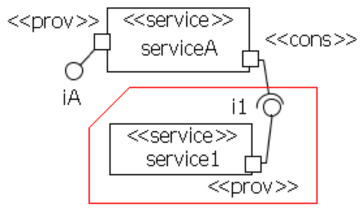
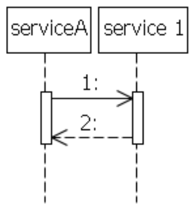
BP pattern	Service pattern	Communication protocol
		None
		None
	None	
	 <<prov>> <<service>> serviceA <<cons>> iA i1 <<service>> service1 <<prov>>	

Table 1 contains transformation rules for basic BPD elements. The first row of the table depicts a rule for a *pool*. A pool represents a business entity in the given process. Graphically, a pool is a container for the control flow between BPD elements. Each pool in a BPD is mapped to a corresponding composite service in a service diagram. The composite service is then responsible for the execution of that part of a business process that is modeled within the pool. Hereafter, we assume that all BPD elements belong to the `processA` pool, unless otherwise indicated.

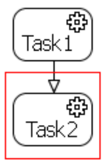
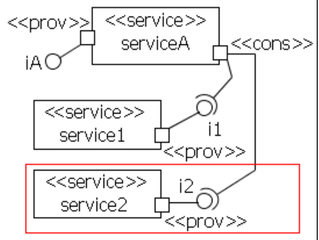
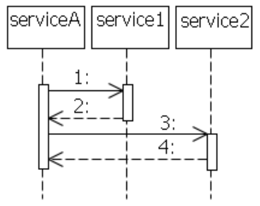
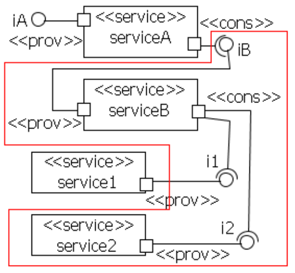
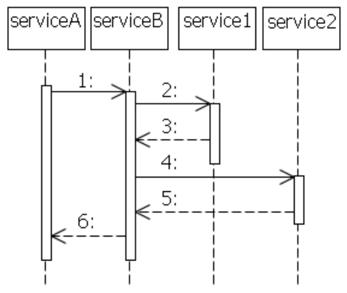
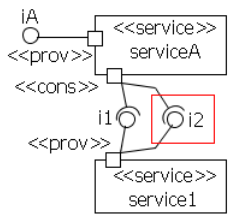
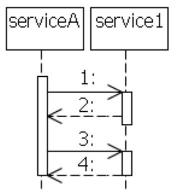
A pool can contain one or more *lanes*. Lanes model internal sub-partitions (usually called *roles*) within a business entity. These roles encapsulate a given part of the BP and ensure its execution. In the context of service diagrams, this relationship is modeled by means of *packages*. Next, a *message start event* is mapped to a *message* (implicitly asynchronous) and a *message end event* is mapped to a *replay*.

The last row in Table 1 contains a transformation rule for a single *service invocation task*. Generally, a task represents some unit of work performed in a process that is not further modeled. A service task is a task that provides some sort of service. Graphically, it is used to depict a service task with a small sprocket-symbol in the upper right corner to distinguish a service task from a task. A service task in a business process diagram is represented by a primitive service in a service diagram. As mentioned above, `Task1` (from Table 1) belongs to `processA`. It means that `Task1` is executed within the scope of `processA`. As a result, a new `service1` is created and connected to `serviceA`. This is made by means of an *assembly connector*. The connector defines that `service1` provides some capabilities that `serviceA` requires. In this case, it means that `service1` is controlled by `serviceA`. In the other words, `serviceA` is an access point to `service1`. The corresponding sequence diagram describes details of the communication between `serviceA` and `service1`. As we can see, the invocation of `Task1` is modeled by means of an incoming message to `service1`.

In Table 2, it is shown how to transform a sequence of two service invocation tasks. The transformation can be done in three different ways. The first one is to add a new primitive service to the existing service diagram and to specify the communication protocol in the corresponding sequence diagram (see Table 1, the first row). The alternative way is to create a new primitive service and a new composite service. These two services will be added to the service diagram as follows (see Table 2, the second row): `service2` is the primitive service that represents `Task2`, and `serviceB` is the composite service that acts as a gate to `service2` and `service1`. Hence, `service2` is connected to `serviceB` and `service1` has to be switched to `serviceB` if it was connected to another service before this transformation step. Interactions among these services are depicted in the sequence diagram. Lastly, the sequence of two service tasks can be transformed into a single primitive service. It is achieved through adding a new interface to the provider port of the existing primitive service and connecting this interface to the corresponding composite service. Hence, it follows that such a service provides functional capabilities that correspond to both of the tasks.

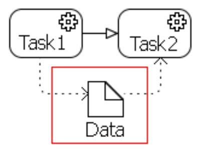
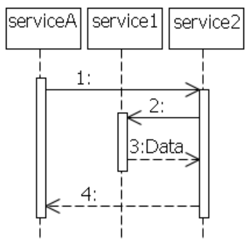
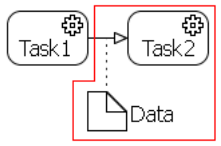
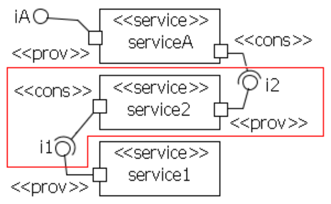
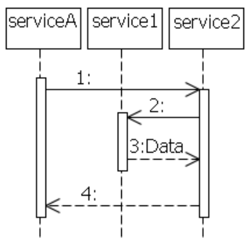
When the service is invoked, the capability is chosen according to the message that is sent to the service. See Table 2 (the third row) for details. Generally, the first rule is usually used when we need to create a new standalone service. The second rule is used when we need to control the primitive services for some reason, i.e. because of QoS or security issues. If we want to use an existing service with only some modifications, we can apply the third transformation rule.

Table 2. Transformation rules for the Sequence Pattern.

BP pattern	Service pattern	Communication protocol
		
		
		

Transformation rules for passing of data in a BPD are in Table 3. In this case, a *data object* is sent from one task to another, via a sequence flow. The first row describes the transformation of a general data passing. As we can see in the sequence diagram, the composite service is responsible for forwarding data from one primitive service to the another one. This rule can be used also for tasks that are not adjacent to each other in a BPD.

Table 3. Transformation rules for the Data-Passing Pattern.

BP pattern	Service pattern	Communication protocol
	None	 serviceA service1 service2 1: 2: 3:Data 4:
	 iA <<service>> serviceA <<prov>> <<cons>> i2 <<cons>> <<service>> service2 <<prov>> i1 <<service>> service1 <<prov>>	 serviceA service1 service2 1: 2: 3:Data 4:

The alternative rule (the second row) can be used only for two adjacent service tasks. In this case, `service1` just produces data that is needed to pass to `service2`. For this purpose, `service1` has to be switched directly to `service2` if it has been connected to another service before this transformation step. The sequence diagram then shows details of the communication among these services. At first, `serviceA` invokes `service2`. After that, `service2` calls `service1` and waits for the data. As soon as the data is passed, `service2` can complete its processing and send the result message back to `serviceA`.

5 Example

Figure 1 shows an exemplary "Purchase Order" business process model. This example is adopted from [10]. As we can see, there are three roles, which are responsible for realization 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 produced and the amount of the shipping cost. In parallel, the "Shipping" role determines when the products will be available and from what locations. At the same time, the process requests shipping schedule from the "Scheduling" role. After the shipping information is known, the complete price can be evaluated. Finally, when the complete price and shipping schedule are available, the invoice can be completed and sent to the customer.

In this example, we assume that following tasks (from Figure 1) were identified as service invocation tasks: "Initiate Price Calculation", "Complete Price Calculation", "Request Shipping" and "Request Production Scheduling". We can generate a service orchestration (see Figure 2) from this BPD by using the transformation rules mentioned in Section 4.

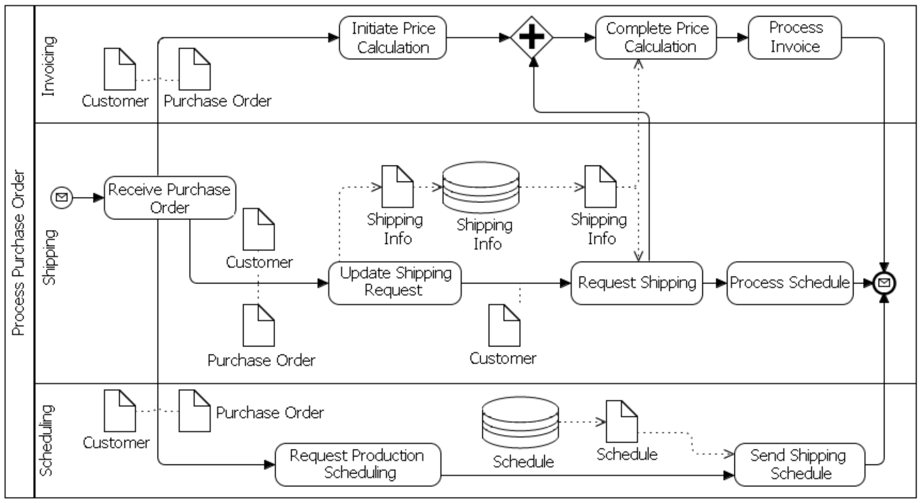


Fig. 1. BPMN model of the Purchase Order process.

The orchestration is composed from one composite and three primitive services. The composite service – the `PurchaseOrder` service – represents the business process itself. This service controles the rest of following primitive services. The `PriceCalculation` service provides two interfaces (i.e. two functional capabilities) `iInitialPrice` and `iTotalPrice` according to the "Initiate Price Calculation" and "Complete Price Calculation" tasks. Similarly, `ProductionScheduling` and `Shipping` provide functional capabilities specified by "Request Production Scheduling" and "Request Shipping", respectively. Figure 3 depicts the service topology, which belongs to the orchestration.

In Figure 2, we can also see that the `PurchaseOrder` service provides the `iAsyncReplay` interface at its consumer port. Each service that enables asynchronous communication has to implement this auxiliary interface. Consumed services use this interface to send replies back to the consumer.

As a composite service, the `PurchaseOrder` service is responsible to keep its subordinate services independent and stateless. This is done by controlling of passing of data between its subordinate services. The principle can be shown on the collaboration between `PurchaseOrder` and `PriceCalculation` (see Figure 4). In order to complete the invoice, the price has to be calculated. Consequently, the `PurchaseOrder` service requests the `PriceCalculation` service. `PriceCalculation` calculates the initial price and returns it to `PurchaseOrder`. `PriceCalculation` does not store the calculated price for later use, i.e. it does

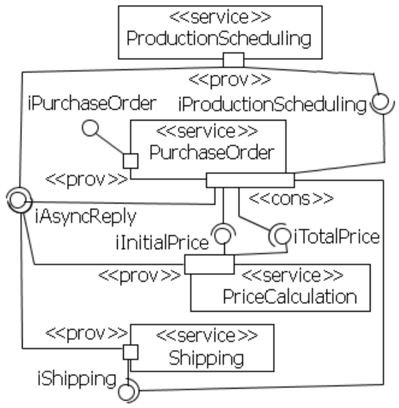


Fig. 2. The derived service orchestration.

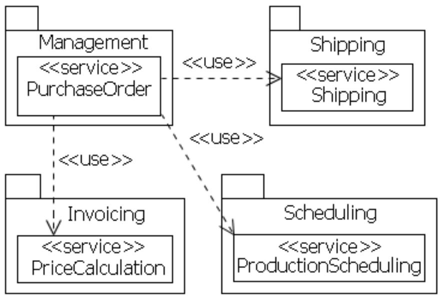


Fig. 3. The service topology.

not hold any state information. When the service is invoked again (in order to calculate the final price), its initial state has to be set by sending of the initial price.

Because this paper deals mainly with the issue of generating models of service orchestration from a BPD, it is payed less attention to specify single services. The details about messages, which are being passed between services, and details about architectures of every single one service mentioned in this example can be found in [12].

6 Discussion

The transformation technique presented in this paper is motivated by several approaches introduced in Section 2, especially by [16], [17] and [5]. Compared to these approaches, the technique proposed in this paper is primarily focused on realization of business processes rather than on modeling of business processes by means of UML. [2] provides a method for integrating Business Process Modeling and Object Modeling by treating each business process as a contract. The contract is a mediator between business clients and IT implementers. It specifies service providers playing roles in a collaboration fulfilling some business rules in order to achieve some business goals. Our technique follows this method by describing how to implement the business process by services in the context of SOA.

[10] proposes a similar concept to our approach. However, that concept does not provide any description of the relation between business process diagrams and UML models. What is more, it contains a number of incorrectness. The most significant one is that the stateless of services (the fundamental SOA principle) is

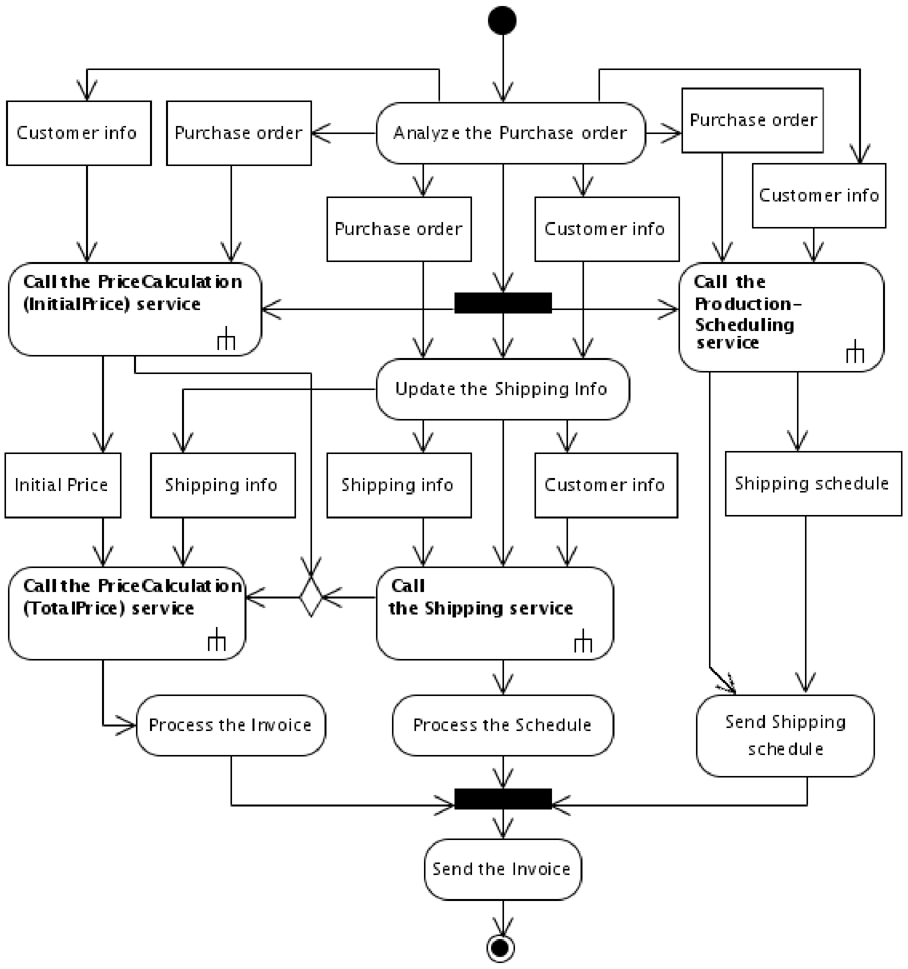


Fig. 4. The behavior of the `PurchaseOrder` service.

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 (see Section 3 and 5). Furthermore, in our approach, services' interfaces are restricted to provide only one functionality. Regardless, it is possible to implement services with more interfaces (i.e. with more functional capabilities), but the interfaces have to provide independent functionality. This allows to add, remove or alter any capability, which is provided by the service, without large modifications in the service specification. This provide better reusability and leads to easy extensibility of modeled services.

7 Conclusion

This paper concerns Service-Oriented Analysis and Design, especially integration between Business Process Modeling and Service Modeling. We outlined a transformation technique that defines how a business process should be transformed into an orchestration of services and how the services should collaborate to fulfill the business goals. The transformation is based on control-flow patterns and is designed in conformity with fundamental SOA principles such as loose coupling, service independence, stateless and reusability. Some of these features are novel and have not been integrated into existing service-oriented modeling methods.

The presented ideas are a part of a greater project, which deals with modeling and formal specification of SOA and underlying component-based systems. The approach, which is presented in this paper, is aimed at modeling of high-level-abstract layers of the SOA hierarchy. The future work will focus on formal description of the proposed transformation and on an integration with formal component models. This allows formal verification of a whole modeled system (e.g. tracing of changes in a business process model to the changes in components' structure). Furthermore, the formal description of the transformation allows validation of the generated service orchestration. Besides, the ongoing research includes design of additional transformation rules, especially rules for business patterns based on gateways and events, and applying the transformation in a real-world case study.

References

1. Amir, R., Zeid, A.: A UML profile for service oriented architectures. In: OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, pp. 192–193. ACM, Vancouver (2004)
2. Amsden, J.: Business services modeling: Integrating WebSphere Business Modeler and Rational Software Modeler, http://www.ibm.com/developerworks/rational/library/05/1227_amsden/
3. Arsanjani, A.: Service-oriented modeling and architecture: How to identify, specify, and realize services for your SOA, http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/
4. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River (2005)
5. Inaganti, S., Behara, G. K.: Service Identification: BPM and SOA Handshake. BPTrends (2007, March), http://www.bptrends.com/publicationfiles/THREE%2003-07-ART-BPMandSOAHandshake-InagantiBehara-Final.pdf
6. Johnston, S.: UML 2.0 Profile for Software Services, http://www.ibm.com/developerworks/rational/library/05/419_soa/
7. Murzek, M., Kramler, G.: Business Process Model Transformation Issues - The Top 7 Adversaries Encountered at Defining Model Transformations. In: Proceedings of the ninth international conference on enterprise information systems, pp. 144–151. INSTICC, Paphos (2007)
8. Object Management Group: UML Superstructure Specification, version 2.0, http://www.omg.org/cgi-bin/doc?formal/05-07-04

9. Object Management Group: Business Process Modeling Notation (BPMN) Specification, http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf
10. Object Management Group: UML Profile and Metamodel for Services (UPMS), Request For Proposal, http://www.omg.org/docs/soa/06-09-09.pdf
11. Ortiz, G., Hernandez, J.: Toward UML Profiles for Web Services and their Extra-Functional Properties. In: IEEE International Conference on Web Services (ICWS'06), pp. 889–892. IEEE Computer Society, Chicago (2006)
12. Rychly, M., Weiss, P.: Modeling Of Service Oriented Architecture: From Business Process To Service Realisation. In: ENASE 2008 Third International Conference on Evaluation of Novel Approaches to Software Engineering Proceedings, pp. 140–146. INSTICC, Funchal (2008)
13. van der Aalst, W. M. P., Barros, A. P., ter Hofstede, A. H. M., Kiepuszewski, B.: Advanced Workflow Patterns. In: Etzion, E., Scheuermann, P.: CoopIS. LNCS, vol. 1901, pp. 18–29. London: Springer-Verlag (2000)
14. Weiss, P.: Using UML 2.0 in Service-Oriented Analysis and Design. In: Proceedings of the 12th Conference and Competition STUDENT EEICT 2006 Volume 4, pp. 420–424. Faculty of Information Technology BUT, Brno (2006)
15. Weiss, P., Zendulka, J.: Modeling of Services and Service Collaboration in UML 2.0. In: Information Systems and Formal Models, pp. 29–36. Faculty of Philosophy and Science in Opava, Silesian University in Opava, Opava (2007)
16. White, S.: Process Modeling Notations and Workflow Patterns. BPTrends (February 2008), http://www.bptrends.com/deliver_file.cfm?fileType=publication&fileName=03%2D04%20WP%20Notations%20and%20Workflow%20Patterns%20%2D%20White%2Epdf
17. Wohed, P., van der Aalst, W. M. P., Dumas, M., ter Hofstede, A. H. M., Russel, N.: Pattern-based Analysis of BPMN - An extensive evaluation of the Control-flow, the Data and the Resource Perspectives (revised version), http://www.workflowpatterns.com/documentation/index.php