Systematic Design of Web Service Transactions

Benjamin A. Schmit and Schahram Dustdar

Vienna University of Technology, Information Systems Institute,
Distributed Systems Group, Vienna, Austria, Europe
{benjamin, dustdar}@infosys.tuwien.ac.at

Abstract. The development of composite Web services is still not as simple as the original vision indicated. Currently, the designer of a composite service needs to consider many different design aspects at once. In this paper, we propose a modeling methodology based on UML which separates between the four concerns of structure, transactions, workflow, and security, each of which can be modeled by different experts. We have developed a proof-of-concept tool that is able to extract information from the model and transform it into a computer-readable format.

1 Introduction

Web services have become more and more commonplace during the last few years. An idea that has been associated with them from the start is that of composition: Web services should be located at run-time and assembled semi-automatically to provide more complex services. This goal, however, still involves some unsolved research questions, among others in the field of distributed long-running transactions.

An important aspect of Web service composition is that the designer or maintainer of a composite service until now had to be an expert in several fields. We have identified a need for knowledge about Web service structure (which services are used by which composite services), transactional issues, security issues, and about the workflow of the composite service. We therefore propose to split composite Web service design into these four views. Four largely independent models can then be created by different experts, with connections between them only where it is necessary.

Minor updates to a composite service are also facilitated by our approach since only a subset of the design diagrams need to be changed. Software tools can further help the programmer by automating transformations from design diagrams to (preliminary) code. Therefore, we have based our methodlogy on the Unified Modeling Language (UML), which is already supported by most design tools.

The methodology has not yet been fully completed, but the models developed with it can already be used profitably. As a proof of concept, we have developed a transformation tool based on the widely used Eclipse platform which extracts transaction information from the model. The output conforms to the

WS-Coordination specification and could easily be incorporated into a Web service platform implementing this specification.

Section 2 introduces a composite Web service case study that will be used to illustrate our work. Section 3 presents the elements of our UML metamodel and applies this metamodel to the case study. Section 4 presents our modeling tool and shows how it can extract information from the model created from our case study. Section 5 presents related work, and Section 6 concludes the paper.

2 Case Study

In this section, we will introduce a case study which we will refer to throughout the paper. Instead of using the traditional composite Web service example of a travel reservation system, we refer to our case study first introduced in [1], which describes the production of a movie in terms of Web services. Modeling such a comprehensive example with Web services may involve unprecedented design decisions as well as unexpected outcomes. It should lead to a more realistic estimate of the benefits of our methodology.

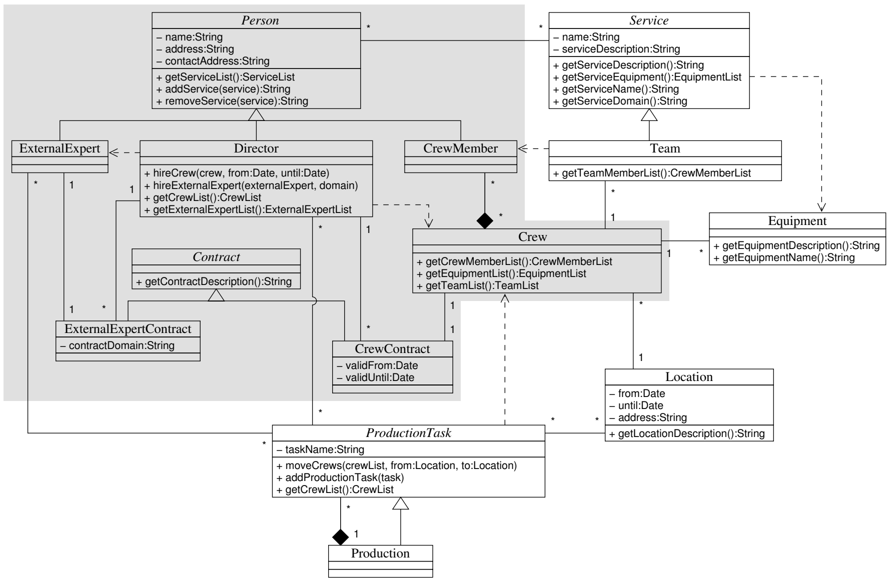

Person
– name:String
– address:String
– contactAddress:String
+ getServiceList():ServiceList
+ addService(service):String
+ removeService(service):String
Service
– name:String
– serviceDescription:String
+ getServiceDescription():String
+ getServiceEquipment():EquipmentList
+ getServiceName():String
+ getServiceDomain():String
ExternalExpert
Director
+ hireCrew(crew, from:Date, until:Date)
+ hireExternalExpert(externalExpert, domain)
+ getCrewList():CrewList
+ getExternalExpertList():ExternalExpertList
CrewMember
Team
+ getTeamMemberList():CrewMemberList
Equipment
+ getEquipmentDescription():String
+ getEquipmentName():String
Crew
+ getCrewMemberList():CrewMemberList
+ getEquipmentList():EquipmentList
+ getTeamList():TeamList
Contract
+ getContractDescription():String
ExternalExpertContract
– contractDomain:String
CrewContract
– validFrom:Date
– validUntil:Date
Location
– from:Date
– until:Date
– address:String
+ getLocationDescription():String
ProductionTask
– taskName:String
+ moveCrews(crewList, from:Location, to:Location)
+ addProductionTask(task)
+ getCrewList():CrewList
Production

Fig. 1. Film production case study

Figure 1 shows an overview on the case study. Because of the space limitation, we will concentrate on how the director of the movie hires film crews for the production of the film and external experts who assist him with their expertise (depicted in grey in Figure 1). Both processes need to be handled within a transaction scope.

In our example, experts and crews provide Web services which may be looked up via a Web services registry. A software architect in the director's office composes these services into a new service, which is then used by the director.

3 The UML Metamodel

We will now introduce a uniform methodology for Web services modeling based on the Unified Modeling Language (UML, [2]). An overview on this methodology has first been presented in [3].

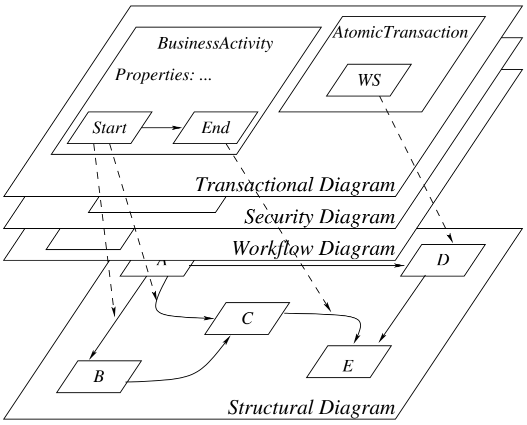


Fig. 2. A Design methodology for Web services

Figure 2 depicts the design idea. It is based on the paradigm of separation of concerns. The four concerns identified so far are structural, transactional, security, and workflow issues. (The order is different in the figure because we focus on the transactional diagram.) Each concern can be modeled by an independent expert, and the Object Constraint Language (OCL; part of the UML specification) is used to establish references between the diagrams.

As we have demonstrated in [3], using separate diagrams for separate design aspects makes the model easier to read, and different experts can work on the design simultaneously. The obvious drawback of this separation, the higher complexity of the methodology, is kept as small as possible by using OCL references between the different layers. We believe that workflow and transaction aspects belong to separate layers because this eases later corrections (e.g., adjusting transaction quality of service parameters). On the other hand, a part of the workflow diagram may be referenced e.g. as a compensation handler.

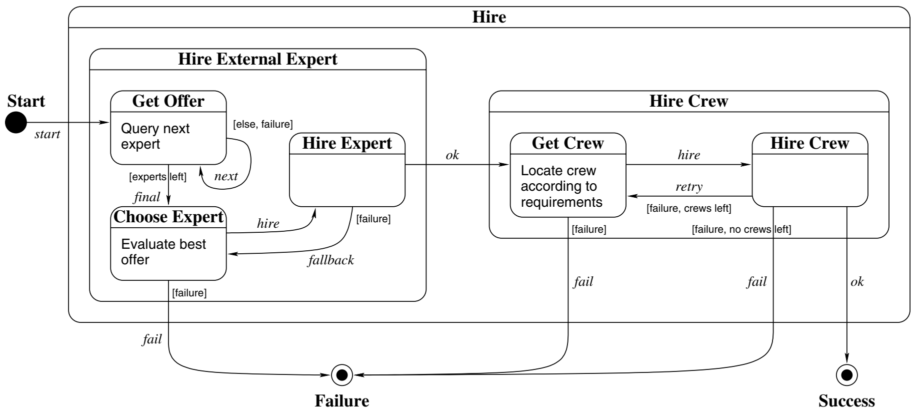


Fig. 3. Example structural diagram

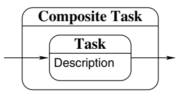
3.1 The Structural Diagram

For the structural diagram, we have chosen a UML statechart diagram. The Unified Modeling Language [2] has been chosen because it is widely used for modeling software and fits our purposes. We chose a statechart diagram (instead of adding a new diagram type that might more closely describe Web service structure) because existing tools already support this diagram type. Since we have not yet specified the workflow diagram, some workflow details are still included in the structural diagram.

The semantics that have been added to the diagram for Web services modeling (guards and threads maintain their existing semantics) are shown in Table 1.

Figure 3 shows a structural diagram of our example. Elements (transitions) from this diagram will be referenced in the transactional diagram.

Table 1. Added semantics for the structural diagram

●→	**Start element.** Processing of the composite Web service starts here.
→◉	**End element.** Processing of the composite Web service terminates here. Annotations may be either *Success* or *Failure*, which indicate whether the composite service terminates normally or abnormally at that point.
→	**Transition.** Indicates that another task of a Web service is handled next, or that the Web service is started (from the start element) or terminated (to the end element).
	**Task.** Composite tasks contain inner elements (tasks, transitions). Instead, non-composite tasks may contain a description (not intended to be processed).

3.2 The Transactional Diagram

The transactional view is formed by a UML class diagram. Again, we have chosen an existing UML diagram type so that existing UML tools do not need to be modified for processing the transactional diagram. We have used OCL references to identify the locations within the structural diagram where transactions are started, committed, or aborted. A UML profile describes the additional constraints for the transactional diagram.

In the diagram, a transaction is depicted as a UML class, i.e., as a box with three compartments. The first compartment contains the name of the transaction, a stereotype describing the transactional semantics, and tagged values that describe quality of service attributes. The second compartment names the participating Web services (the keyword *dynamic* indicates that the Web service is to be located at run-time, a process which is not covered by this paper). The third compartment holds the references to the start and end of the transaction, as well as invocations to other Web services (starting points for other transactions). Finally, the inheritance relationship is used to model subtransactions. Table 2 defines the keywords used within the UML profile for the transactional diagram.

Figure 4 shows an example of a transactional diagram. The three main transactions corresponding to the first two levels of states in Figure 3 are easily derived from the structural diagram. The start and termination transitions are indicated by OCL references, in the outermost transaction these are *Start.start*, *HireCrew.ok*, *ChooseExpert.fail*, *GetCrew.fail*, and *HireCrew.fail*. They

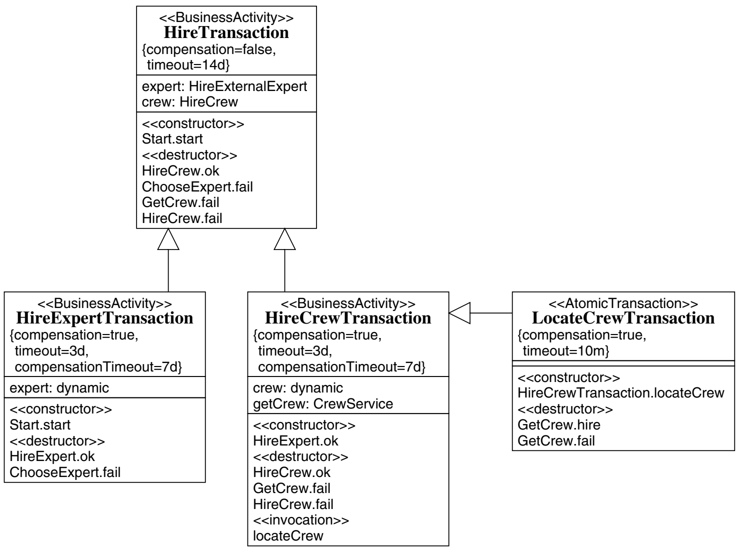


Fig. 4. Example transactional diagram

Table 2. Keywords in the transaction profile

Keyword	**UML Scope**	**Description**
Invocation	Class Stereotype	A Web service invocation running without a transactional scope, i.e. no transaction.
AtomicTransaction	Class Stereotype	An ACID transaction, as defined in [4].
BusinessActivity	Class Stereotype	A long-running, non-ACID transaction, as defined in [5].
compensation	Class Tagged Value	Flag that specifies whether the transaction as a whole can be compensated.
timeout	Class Tagged Value	Maximum time interval that the transaction can be active before it is rolled back.
compensationTimeout	Class Tagged Value	Maximum time interval measured from the start of a transaction that a committed transaction can be compensated.
dynamic	Attribute Type	Indicates that a Web service is to be bound at run-time.
constructor	Method Stereotype	Starting point for the transaction.
destructor	Method Stereotype	Termination point for the transaction.
invocation	Method Stereotype	Starting point for a subtransaction which is not depicted in the structural diagram.

correspond to the arrows entering and leaving the *Hire* state in the structural diagram.

Our example, however, also contains a fourth transaction that is used for finding film crews that may later be contacted to participate in the current production. Instead of referencing the structural diagram, the starting point of the *LocateCrewTransaction* lies within the *HireCrewTransaction*, i.e. within the transactional diagram itself. In the *HireCrewTransaction*, this starting point is described as a method stereotyped *invocation*.

3.3 Security and Workflow Issues

Since security aspects should be considered as early as possible, we propose the inclusion of security parameters (e.g., which Web service calls/transactions need to be encrypted or signed) in the design phase. We do not have developed a diagram for security yet. We intend to use OCL for references to entities in both the structural and the transactional design diagram, but this is still subject to future work.

The workflow diagram will offer a high-level view on the composite Web service. This design view will cover issues that cannot be addressed by the structural and transactional diagram alone, e.g. some of the challenges introduced in [3]. This diagram will reference elements of the structural and the transactional view.

4 Tool Support

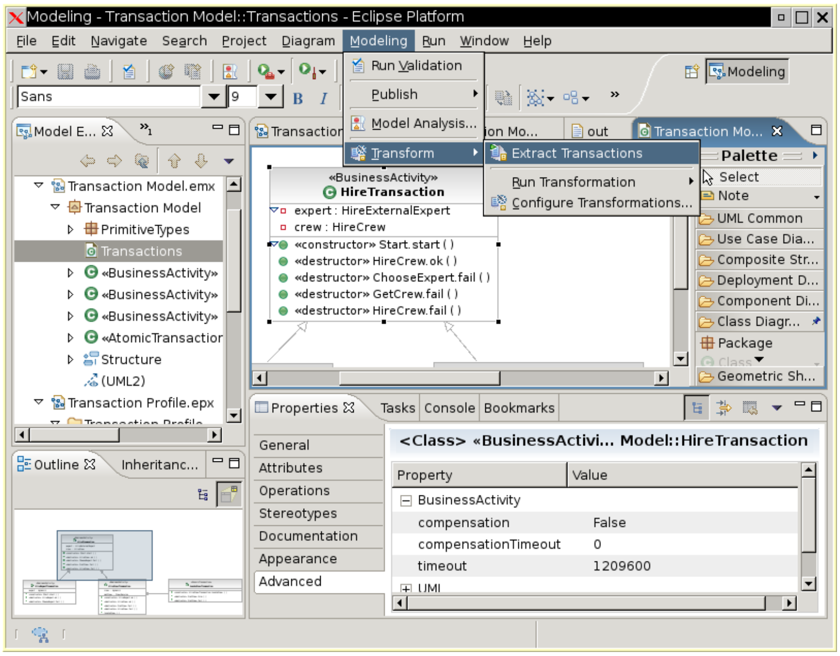
In order to show the usefulness of our design approach, we have implemented a proof-of-concept tool that works on the transactional design diagram. It has been built on the Eclipse platform [6] extended by the IBM Rational Software Architect tool suite [7].

4.1 Architecture

Eclipse is a highly modular integrated development environment for Java which also offers basic support for the Unified Modeling Language (UML, [8]) through the Eclipse Modeling Framework (EMF, [9]).

The IBM Rational Software Architect extends this platform, among other things, by adding a visual editor for creating and maintaining UML models. Transformations allow the developer to automatically transform the model into a code skeleton and synchronize changes in design and code. Transformations for creating Java, EJB, and C++ code are included.

We have written an extension to this platform which adds a new transformation method to the modeler. It extracts the transactions from a UML class diagram following the specification of our transactional diagram. The output currently consists of a coordination context for use in WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity [10, 4, 5], however, it can easily be adapted to confirm to other transaction specifications.



<Class> «BusinessActivi... Model::HireTransaction

Property	Value
⊟ BusinessActivity	
compensation	False
compensationTimeout	0
timeout	1209600
⊞ UML	

Fig. 5. The transformation plugin

4.2 Application to the Case Study

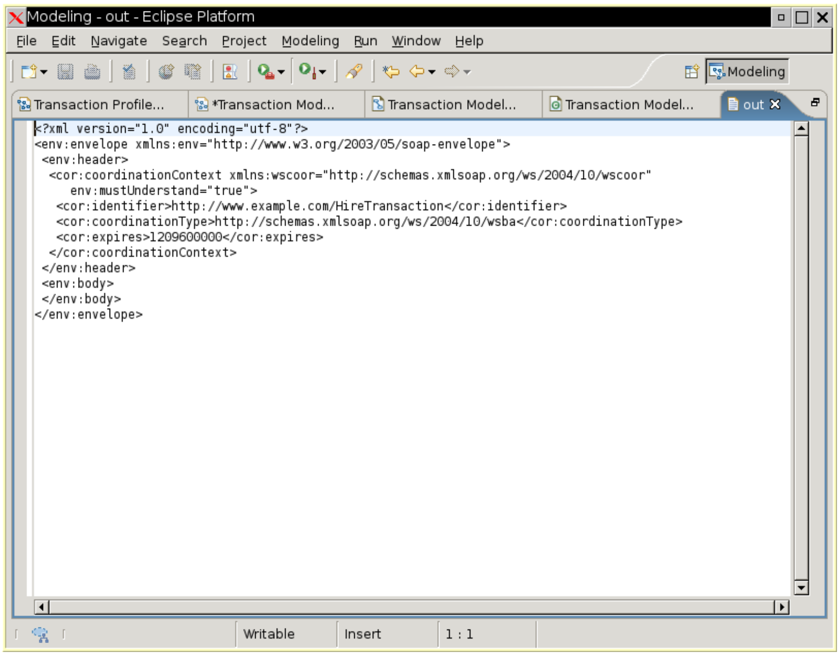
Figure 5 shows the transactional diagram on which the transformation is invoked in the so-called modeling perspective. The modeling perspective consists of four windows:

The model explorer (top left) shows a tree view of the elements within the project. In the example, we see the model with two diagrams and the *Transaction Profile* which defines the stereotypes used in the transactional diagram. The outline window (bottom left) shows a bird-eye view on the current diagram.

In the main window (top right), a part of the transactional design diagram can be seen. The class *HireTransaction* has been selected.

In the bottom right window, properties of the selected transaction can be seen. Within the EMF, they have been modeled as attributes to the *Business-Activity* stereotype of the *Transaction Profile* UML profile. This helps to keep the diagram lean (because such attributes are per default not shown in the main window), but still allows for a simple extraction by our plugin.

Figure 6 shows a simple output XML file the transformation plugin has generated from a part of our design. Right now, only the transactional model is considered by the prototype tool, but already its output could be used as a co-ordination context in a WS-BusinessActivity transaction. Changes in the design lead to changes in the context, taking some of the load away from the programmer of the composite Web service.



```
<?xml version="1.0" encoding="utf-8"?>
<env:envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:header>
  <cor:coordinationContext xmlns:wscoor="http://schemas.xmlsoap.org/ws/2004/10/wscoor"
     env:mustUnderstand="true">
   <cor:identifier>http://www.example.com/HireTransaction</cor:identifier>
   <cor:coordinationType>http://schemas.xmlsoap.org/ws/2004/10/wsba</cor:coordinationType>
   <cor:expires>1209600000</cor:expires>
  </cor:coordinationContext>
 </env:header>
 <env:body>
 </env:body>
</env:envelope>
```

Fig. 6. Output of the transformation plugin

4.3 Outlook

In the future, we plan to extend the tool so that it allows more comprehensive modeling and also considers the structural view. We aim to be able finally to extract a process description in BPEL [11] and/or a choreography description in WS-CDL [12] from our model, thereby greatly easing the task of Web service composition. Using the information contained in the design diagrams, these descriptions can be enriched by using various additional Web service related specifications.

When this work has been completed, we will explore the potential for further automation through the use of a security and a workflow diagram.

5 Related Work

Several independent (sets of) Web service transaction specifications have been released: WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity [10, 4, 5] have been used for the implementation of our tool. Possible alternatives would have been WS-CAF ([13], consisting of WS-Context, WS-Coordiation Framework, and WS-Transaction Management) or BTP [14].

Orriëns, Yang, and Papazoglou [15] divide the process of Web service composition into four phases: definition, scheduling, construction, and execution. The design should become more concrete at each step. UML is used as well, however, the model is founded on the design process and not on separation of concerns.

Dijkman and Dumas [16] also state the need for a multi-viewpoint design approach for composite Web services. Their paper discusses the views of interface behavior, provider behavior, choreography, and orchestration and uses Petri nets for the model itself. Distributed transactions are not mentioned.

Benatallah, Dumas, and Sheng [17] also use statechart diagrams to model composite Web services. Transactional behavior is mentioned as future work, but as yet there is no systematic approach for modeling this.

Karastoyanova and Buchmann [18] propose a template technique for Web services to ease service composition. Templates here are parts of a business process description that can be used for Web service composition. The concept may prove useful for transforming our model diagrams into business process specifications in the future.

Loecher [19] discusses properties of transactions in a distributed middleware setting. Though the author writes about Enterprise JavaBeans, some of the work can be applied to Web services as well.

Henkel, Zdravkovic, and Johannesson [20] mention the difference between technical and business requirements. Their paper proposes a layered architecture that allows to transform the business representation into a more technical representation. Several aspects of process design are described, among them also a transactional aspect.

Jablonski, Böhm, and Schulze [21] propose a separation of concern approach for workflow modeling called workflow aspects. They distinguish between a functional, a behavioral, an informational, an operational, and an organizational aspect. The book surveys workflow modeling and also mentions transaction and security issues.

Further information about Web service transaction specifications can be found in [22]. Database transactions are covered by [23], and additional information about advanced transaction models can be found in [24, 25].

6 Conclusion

In this paper, we have introduced a modeling methodology for composite Web services based on UML. The methodology is based on the concept of separation of concern, i.e., several experts can work on different aspects of the design concurrently. We have defined the structural and the transactional diagram and outlined our future work on the workflow and security diagrams.

We have shown the usefulness of our approach by implementing a transformation tool based on the Eclipse platform. This tool extracts transaction information from the model and transforms this information into a machine-readable XML document following the WS-Coordination specification.

The next steps in our research will be the development and formalization of the workflow and security diagrams. Also, we will expand our transformation tool towards a more comprehensive view on the model. Hereby, we hope to be able to automatically generate more complex descriptions, e.g., a BPEL or WS-CDL description. This automation will help designers to considerably simplify the development of composite Web services.

References

1. Schmit, B.A., Dustdar, S.: Towards transactional web services. In: Proceedings of the 1st IEEE International Workshop on Service-oriented Solutions for Cooperative Organizations (SoS4CO'05), 7th International IEEE Conference on E-Commerce Technology, Munich, Germany, IEEE (2005) To be published.
2. OMG: The unified modeling language, version 2.0. Specification (2004)
3. Schmit, B.A., Dustdar, S.: Model-driven development of web service transactions. In: Proceedings of the 2nd GI-Workshop XML for Business Process Management, 11. GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web, Karlsruhe, Germany, Gesellschaft für Informatik (2005) To be published.
4. BEA, IBM, Microsoft: Web services atomic transaction (WS-AtomicTransaction). Specification (2004)
5. BEA, IBM, Microsoft: Web services business activity framework (WS-BusinessActivity). Specification (2004)
6. Beck, K., Gamma, E.: Contributing to Eclipse. Principles, Patterns, and Plug-Ins. Addison-Wesley (2003)
7. Lau, C., Yu, C., Fung, J., Popescu, V., McKay, E., Flood, G., Mendel, G., Winchester, J., Walker, P., Deboer, T., Lu, Y.: An Introduction to IBM Rational Application Developer: A Guided Tour. IBM Press (2005) To be published.
8. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. 2nd edn. Addison-Wesley (2004)

9. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework. Addison-Wesley (2003)
10. BEA, IBM, Microsoft: Web services coordination (WS-Coordination). Specification (2004)
11. BEA, IBM, Microsoft, SAP, Siebel: Business process execution language for web services (BPEL4WS), version 1.1. Specification (2003) Adopted by OASIS as WS-BPEL.
12. Oracle, Commerce One, Novell, Choreology, W3C: Web services choreography description language version 1.0, W3C working draft 17 december 2004. Specification (2004)
13. Arjuna, Fujitsu, IONA, Oracle, Sun: Web services composite application framework (WS-CAF). Specification (2003)
14. OASIS: Business transaction protocol, version 1.1.0. Specification (2004)
15. Orriëns, B., Yang, J., Papazoglou, M.P.: Model driven service composition. In: Proceedings of the First International Conference on Service Oriented Computing. Volume 2910 of Lecture Notes in Computer Science., Springer-Verlag (2003) 75–90
16. Dijkman, R., Dumas, M.: Service-oriented design: A multi-viewpoint approach. International Journal of Cooperative Information Systems **13** (2004) 337–368
17. Benatallah, B., Dumas, M., Sheng, Q.Z.: Facilitating the rapid development and scalable orchestration of composite web services. Distributed and Parallel Databases **17** (2005) 5–37
18. Karastoyanova, D., Buchmann, A.: Automating the development of web service compositions using templates. In: Proceedings of the Workshop "Geschäftsprozessorientierte Architekturen" at Informatik 2004, Gesellschaft für Informatik (2004)
19. Loecher, S.: A common basis for analyzing transaction service configurations. In: Proceedings of the Software Engineering and Middleware Workshop 2004. Lecture Notes in Computer Science, Springer-Verlag (2004) To be published.
20. Henkel, M., Zdravkovic, J., Johannesson, P.: Service-based processes — design for business and technology. In: Proceedings of the Second International Conference on Service Oriented Computing. (2004) 21–29
21. Jablonski, S., Böhm, M., Schulze, W.: Workflow-Management: Entwicklung von Anwendungen und Systemen. Dpunkt Verlag (1997)
22. Papazoglou, M.P.: Web services and business transactions. World Wide Web **6** (2003) 49–91
23. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann (1993)
24. Elmagarmid, A.K., ed.: Database Transaction Models for Advanced Applications. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann (1992)
25. Procházka, M.: Advanced Transactions in Component-Based Software Architectures. PhD thesis, Charles University Prague, Faculty of Mathematics and Physics, Department of Software Engineering (2002)