Benjamin A. Schmit, Schahram Dustdar

Model-driven Development of Web Service Transactions

Composite Web service design using model-driven approaches has been in use for several years now, but the modelling of transactional properties is still uncommon and has not yet been subject to much research. For a distributed system of autonomous components like Web services, especially when they are used for implementing business processes, transactional guarantees can be of vital importance. In this paper, we propose a model-driven approach which introduces a separate design layer dedicated to transactions. We show that our systematic modelling approach is able to introduce transactions in the design without increasing the complexity of the basic UML diagram. Our approach can also be reused to specify other properties of Web services such as security requirements or workflows in additional layers.

1 Introduction

Web services have slowly become more and more commonplace over the last years. Languages like BPEL [BIM+03] have facilitated the composition of several simple Web services into larger composite services. As Web service compositions grow, the complexity of designing and maintaining business processes increases with them. Tools for methodological design like UML [Omg03] have been available for years, and they have also been applied to business process design [KHK+03, OrYP03, BeDS05].

An important property of business processes are transactions. It must be possible to guarantee that a business process can have only pre-defined, consistent outcomes (e.g. success or complete failure, but never a partial result). Transactions can be divided into at least two types that are relevant for business process modelling [Papa03]: ACID transactions (which have been used in databases for decades) and long-running transactions which violate some of the traditional ACID properties. These two main types can be further augmented with quality of service attributes.

Several specifications exist which augment the basic Web service standards with transactions (e.g. [BeIM04b, Oasi04, AFI+03]). The specifications use XML to express transactional semantics. Programmers can combine them with BPEL in order to implement business processes which depend on the availability of transactions.

Implementing transactions directly e.g. according to the WS-BusinessActivity specification is error-prone. It is also directly opposed to the model-driven architecture, whose goal is to minimize the amount of hand-written code by formalizing the design step. On the other hand, including transactional properties as annotations to the existing design diagrams might easily make them unreadable, subverting the gains of the model-driven approach.

In this paper, we propose the use of two layers of design diagrams. The structural layer can be created with existing model-driven methodologies, and the transactional layer uses a UML class diagram to model the transactions. These layers are merged by OCL references from the transactional to the structural view. This approach allows us to easily manage the added complexity and also helps the architects when design changes are necessary.

In Section 2, we present a case study which we will refer to throughout the paper. Section 3 extracts transactional requirements from the case study and identifies general challenges with transactions in Web services. As a response to these challenges, Section 4 introduces our modelling methodology. Structure and transactions of the case study are modeled in two separate diagrams, and the merge points are identified. Section 5 discusses related work. Finally, Section 6 sums up the main points of the paper and reaches the conclusion. It also gives an outlook on future work.

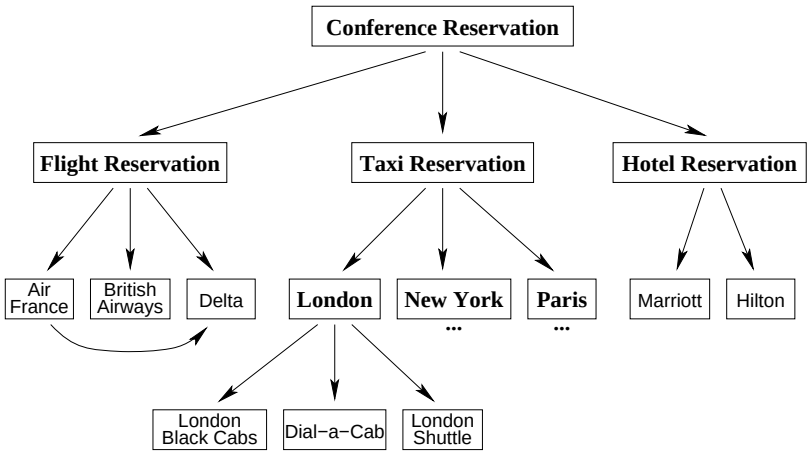

Figure 1: Case Study: Conference Reservation System

2 Case Study

We will motivate the approach presented in our paper with a case study. Our example is an extension of similar case studies found in various papers on Web service composition. The assumptions in this case study contain a few flaws which may not yet be apparent, but will be revealed during the transaction design phase.

Figure 1 shows how the Web services in our example work together. Web services in bold font are composite Web services; they require other Web services in order to operate correctly. The Web services depicted in normal font are typically provided as a company's gateway to the outside world. Each of them is managed independently. A UDDI registry may be used to locate services implementing a given interface, e.g. airlines, but this feature is not yet included in our case study.

The task of organizing a trip to a conference consists, among other things, of booking a flight to the conference location, booking a hotel, and organizing the trip between the airport and the hotel by booking a taxi (for the example, we ignore the possibility of a taxi stand in front of the airport). The booking services have kindly been provided by umbrella organizations.

The flight reservation service queries the Web services of some airlines for the availability of a flight with a given set of restrictions (airports, number of stops, price). Some airlines access Web services of associated airlines for completing the request (e.g. most Air France flights within the USA are operated by Delta).

The (fictive) Global Association of Taxi Drivers operates a Web service that offers a single access point for the major cities' taxi associations. As an example, the London Taxi Driver's Association's Web service may query several local taxi companies — other local associations will likely do the same.

Finally, the hotel reservation service provides uniform access to several hotel chains. Since most major chains operate globally, a localized service level (as in the taxi reservation example) is not implemented here.

3 Requirements and Challenges

In this section, we will identify some transactional requirements that can be derived from the case study. We will also identify some general challenges for Web service transactions. Not all of the problems indicated here have been addressed in this paper, some are subject to future work. This list can serve as a guideline for designers of composite Web services.

3.1 Transactional Requirements

In order to implement the collaborative Web services of the case study, the transaction subsystem (in fact subsystems, since it is unlikely that each company uses the same transaction software) needs to fulfill a number of requirements:

Long-running Transactions: It is generally accepted (see e.g. [Papa03]) that ACID transactions are unsuitable for most business processes. Traditional database transactions typically have a short duration, and therefore database tables

affected by a transaction can be locked while they are running.

On the other hand, the transactions needed for our case study involve the cooperation of a large number of Web services. Those services do not even belong to the same company and may be distributed globally. In such a setting, network connections may fail, subtransactions may need to be compensated, alternative options may need to be considered, and even human intervention may be necessary. Locking a database table for the entire time the long-running transaction is active is therefore no longer practical.

The solution proposed by current Web service transaction specifications like [BeIM04b, Oasi04, AFI+03] consists of weakening the atomicity and isolation properties so that several concurrent long-running transactions may access the same underlying database tables. They are typically called business transactions, and can consist of a composition of several ACID transactions.

Alternative Process Paths: In some situations, alternative paths within a business process may lead to equally acceptable results. When we want to book a taxi from Heathrow airport to a downtown London hotel, the goal to have a taxi ready when we leave the airport (flight delays are not considered here) is more important than the price difference of a Pound between the available taxi companies.

For the flight selection, on the other hand, the selection of the transaction that will eventually be committed will usually depend on (preliminary) results returned by the involved Web services. Air France, for example, does not offer direct flights from Vienna to London. Booking a non-stop flight with British Airways removes the inconvenience of switching planes in Paris as well as the possibility of missing the second flight because of a delay in the first one, and the single flight ticket may be cheaper than two of them. However, if for some reason we can't reserve a British Airways flight, it would still be good to use Air France's Web service as a fallback. All of this is known in advance and can be specified explicitly in the business process.

For the hotel, we have no opinion in advance. We will ask all available hotel chains and commit the transaction with the lowest price at the specified level of service.

Phased Transactions: As explained in [PaCh03, LiZh04], business transactions could greatly benefit from a multi-phase model. In this model, a first pre-transaction phase should establish tentative holds on the resources that will be accessed in the transaction. In our example, the price of a flight can be queried before the main transaction phase. If the price should later change or the flight become unavailable, the airline Web service will notify its client that the tentative hold has been removed, and the pre-transaction phase needs to be repeated. This procedure reduces the number of (main) transactions needed in a complex business process and therefore increases the chance of a successful commit.

After the main-transaction phase (which executes the agreement protocol), a post-transaction phase can be used to exchange materials related to the transaction, e.g. an electronically signed contract or further details such as when the passengers should be at the airport and how much baggage they can take with them. These details can be exchanged after the transaction has committed because they are not important to the transaction's outcome, and removing them from the transaction's body further reduces the size of the transaction, which in turn reduces the chances for transaction rollbacks.

Quality of Service: Another issue that needs to be considered is quality of (transaction) service. We have already discussed the difference between ACID and long-running transactions, but even these two models can be further subdivided.

Examples of quality of service aspects are whether the transactions can be organized hierarchically, whether a transaction is local to a single Web service or can be extended for operation in a composite Web service, whether a transaction is aborted after some time of inactivity, or whether data regarding the transaction is transmitted via secure channels only. These aspects need to be considered when a composite Web service is designed.

3.2 Challenges

Because the requirements for Web service transactions differ from those for conventional ACID transactions, some of the solutions developed for database transactions cannot be reused and new concepts have to be introduced. We have identified a number of challenges that need to be addressed:

Transaction Model: For a single Web service, a traditional database transaction may in some cases be sufficient. However, as soon as Web services are composed to form a larger composite service, non-ACID transactions are needed so that resources do not have to be locked for long periods of time [LiZh04]. A Web service that uses ACID transactions per default should be able to distinguish between a simple request to its ports and a composite request by another Web service.

Compensation: With a non-ACID transactional style, implementing compensation becomes a necessity. Many Web services do not provide ports for compensation actions (such as returning the ticket to the airline with a full refund). If a participant in a composite Web service transaction decides that the transaction needs to be rolled back, it must be possible to undo all preliminary results.

Timeouts: The challenge of compensation directly leads to the question of timeouts. A company needs to be able to specify a maximum time that a transaction can be running. It would be bad for business if customers could prolong their transactions and roll back (or compensate) at any time. Airlines, for example, usually charge different cancellation fees depending on the time remaining until the flight.

Transaction Hierarchies: When Web services are assembled to form a composite service, it may be helpful to use hierarchical transactions to reflect the structure of the composite Web service. Within a workflow, hierarchical transactions are also useful because subtransactions can then be exchanged if they fail. In our example, if the subtransaction involving a given London taxi company fails, we want to create a second subtransaction with another taxi company. In this case, it is enough if a single subtransaction commits.

Enforcing Transaction Semantics: Where transaction hierarchies are used, it may happen that lower-level Web services do not support the transactional properties required by the higher-level composite services that access them. A transaction needs to be able to query the properties of subtransactions and report a failure if its features are insufficient.

Scope of Transactions: In the case of hierarchical transactions, we have to decide whether we want to use a small number of larger transactions or a large number of relatively small transactions, i.e. whether the scope of a single transaction should be large or small. Smaller transactions should reduce the work needed for a potential compensation in most cases, but they introduce more overhead in transaction processing. A problem that has not yet been solved is whether well-sized transaction scopes can be generated automatically.

Registration: For some Web services, the question when all participants have entered a transaction can be hard to judge. A stock exchange Web service, for example, may involve an arbitrary number of interested parties who state their bids in a common transaction. When the transaction commits, the best bid is selected. However, it may always be possible that a better bid will arrive after the agreement protocol has been executed.

Dynamic invocation: When Web services are to be composed dynamically, i.e. at run-time instead of at build time, an additional difficulty is introduced. The Web service registry needs to be able to understand differences between transactional models so that it does not return services that do not fulfill the desired transactional guarantees.

Deadlocks: The distributed nature of Web services adds another difficulty to the problem of deadlock detection. Different programmers may independently implement a sequence of queries to the same Web services, which can interlock during execution. Detection of such distributed deadlocks is a complicated topic (see e.g. [Elma86]), especially since short timeouts are not an option for Web services. Again, a good design methodology can help to discover this problem.

Workflow Issues: In many cases, several equivalent transactions have to be started in order to compare the offers of different companies. Depending on the preliminary results (compulsive business offers), a single transaction is committed while the others are rolled back. Either the transaction subsystem or a workflow engine in the background must support this typical behavior and allow the specification of an objective function.

3.3 Design Issues

When Web services are built in an ad-hoc way, not all of the above requirements are usually addressed directly, and not all of the challenges are recognized by the developers. Even when a design phase precedes the implementation, Web service-specific challenges may be overlooked.

Therefore, we propose a uniform modelling methodology for Web service transactions based on UML [Omg03]. Our approach aims at enhancing

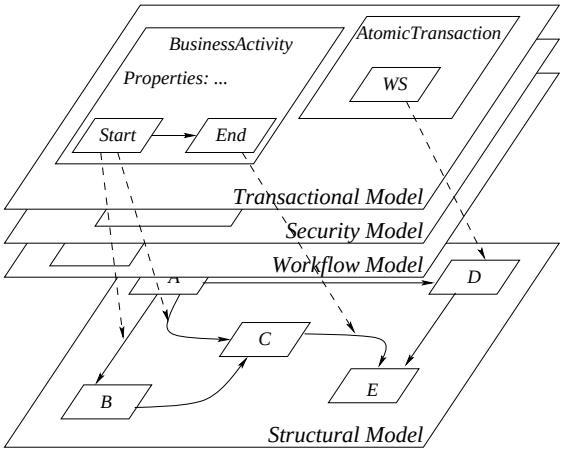

Figure 2: The Basic Idea of Our Modelling Methodology

existing UML diagrams with a transactional view. The methodology has been developed to support a design that considers the requirements and challenges of Web services that have been mentioned above.

4 Modelling Transactions

The basic idea behind our modelling approach is a layered design. At the bottom layer we use the (possibly already existing) UML description of the Web services. Various UML diagram types can be used for the representation of this basic architecture, as well as other languages such as BPMN [Bpmi04], UMM [Cefa01], or ISDL [QuDS04].

On top of these diagrams, other diagram layers can be placed. In this paper, we examine the representation of transactions, but for the future we plan to enhance our modelling methodology to include at least additional layers for security and workflow management. Figure 2 depicts the basic idea.

As the figure shows, the high-level transactional model references objects in the low-level structural model. These references are used for establishing transaction boundaries without adding additional complexity to the structural model diagram. The benefits of this approach will be illustrated towards the end of this section.

For the diagrams themselves, all modelling languages able to express the necessary functionality (composite Web service structure, transactions, security, or workflow) can be used. Different metamodels can be used for different layers as well. The only additional requirement is the availability of references to elements of the structural model.

4.1 Extracting Transactions from the Structural Model

In Figure 3, we have depicted a UML statechart diagram of our case study from Section 2. This diagram still contains a mixture of Web service structure and workflow issues, which will have to be divided into two separate layers in the future. Depending on their role in the collaboration, different participants will be interested in different subparts of this diagram, which lead to different transactional requirements as shown below:

The end user of the composite Web service only knows the states Start, Reservation (“Running”), Success, and Failure. The whole process should therefore either succeed or fail, and in case of failure any preliminary results should be deleted (atomicity guarantee). Compensation is not required.

The reservation service queries the flight reservation service, the taxi reservation service, and the hotel reservation service in sequence (for simplicity, we have chosen not to use concurrency in this example). Each of those services either fails or succeeds. In the case of a failure, results from earlier services need to be compensated to fulfill requirement **1**. The flight reservation, however, cannot be compensated — therefore, its transaction needs to be prolonged until the other transactions commit successfully.

The flight reservation service internally invokes the Web service of each airline in turn (again we

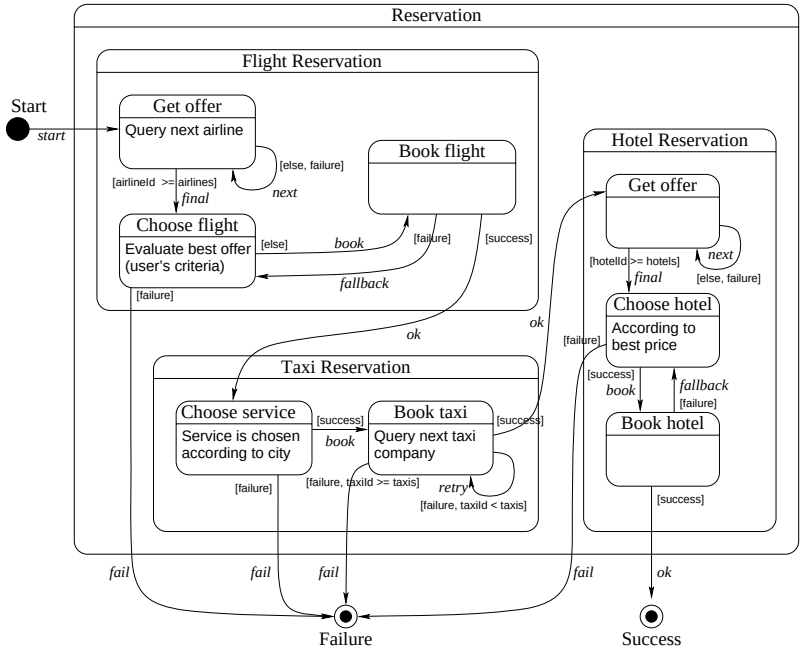

Figure 3: Structure Statechart Diagram

disregard concurrency issues). Then, it compares the offers to find the one which best suits the user's requirements. Finally, the flight is booked. As we have stated in requirement **2**, the airlines do not offer compensation. A transaction with an airline may run as long as 4 hours, then it is terminated by the airline's server. Therefore, we wait until the taxi and the hotel are booked until we confirm the transaction.

The taxi reservation service itself only invokes underlying Web services depending on the desired location, and therefore does not need to fulfill transactional guarantees. The local taxi reservation services, however, provide atomic services since the servers are geographically close together. Therefore, the local transaction requires the short timeout of 5 minutes. On the other hand, taxi reservations can be compensated within an hour after booking.

The hotel reservation works similar to the airline reservation, except that the hotel reservation can be canceled. However, according to **2**, compensation is not necessary at the higher level. (In a real-world example, we would, after this realization, rearrange the design of the subtransactions of the reservation service so that the flight transaction is invoked last.)

4.2 The Transactional Diagram

The structural model diagram already contains much information, and adding transactional semantics to the diagram would not improve readability. Therefore, we use a separate UML diagram to capture the transactional requirements identified above.

For the transactional model, we have used a UML class diagram. We did not introduce a new diagram type because the class diagram is expressive enough for our needs, and existing UML tools already support this diagram type. Each transaction is modeled as a class. Subtransactions that are invoked by higher-level transactions are depicted as subclasses. Finally, tagged values and stereotypes add the necessary transactional semantics.

For referencing elements from the structural model, the Object Constraint Language (OCL, [Omg03]) is used. It is defined as part of the UML specification and is therefore supported by many UML tools. However, UML can also work with other expression languages if necessary.

Figure 4 shows the transactional model diagram. We have used the terms "atomic transaction" and "business activity" from [BeIM04a, BeIM04b] to indicate ACID and long-running transactions. (The transaction specification used by a design diagram,

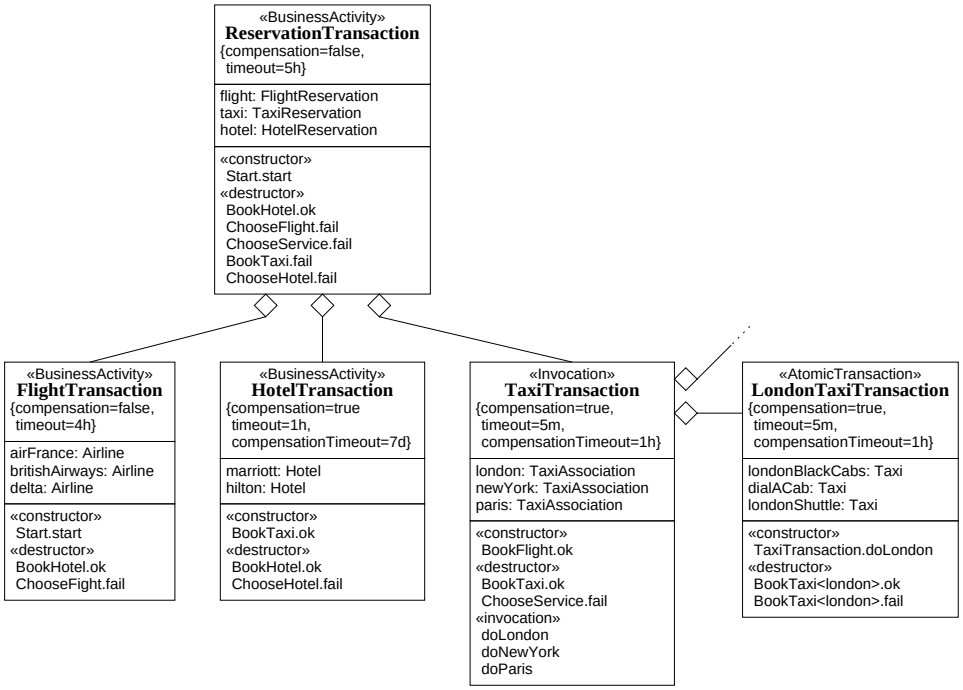

Figure 4: Transaction Class Diagram

including the agreement protocol executed, needs to be defined separately to complete the semantics of the model. In our case, this needs to be done for AtomicTransaction and BusinessActivity.) They are added to the transaction class as a stereotype. If no transaction needs to be used for a Web service, the stereotype `Invocation` is used.

The support for compensating a whole transaction is added to the class as the tagged boolean value `compensation`. Similarly, quality of service properties can be specified. In our diagram, the timeout for the transaction itself and the timeout for invoking a possible compensating transaction have been included.

The Web services that are coordinated by a transaction are displayed as attributes. The constructors of the transaction class indicate the transitions in the structural diagram at which the transaction must be started. Similarly, destructors show the termination (commit or rollback) of the transaction. Finally, methods described by the `invocation` stereotype can reference the constructors of subtransactions which cannot be mapped to a transition in the structural model.

For clarity, we have left the individual (non-composite) Web services out of the transactional model diagram — atomicity is assumed for all non-composite services that are not included in a transactional diagram. Excluding those services improves the readability of the diagram.

4.3 Merging the Diagrams

Figure 5 illustrates how the structural and the transactional model work together. Each constructor and destructor in the transactional diagram either maps to a transition in the structural diagram or to an invocation in the transactional diagram. An important point that the figure also demonstrates is that — as we have stated earlier — a single diagram for both structural and transactional view is almost unreadable.

5 Related Work

In this section, we discuss two main types of related work: Related modelling languages may have been used instead of UML in our paper. This would not have changed the underlying concept of separation of concerns. Related methodologies are alternative approaches, both based on UML and other modelling languages.

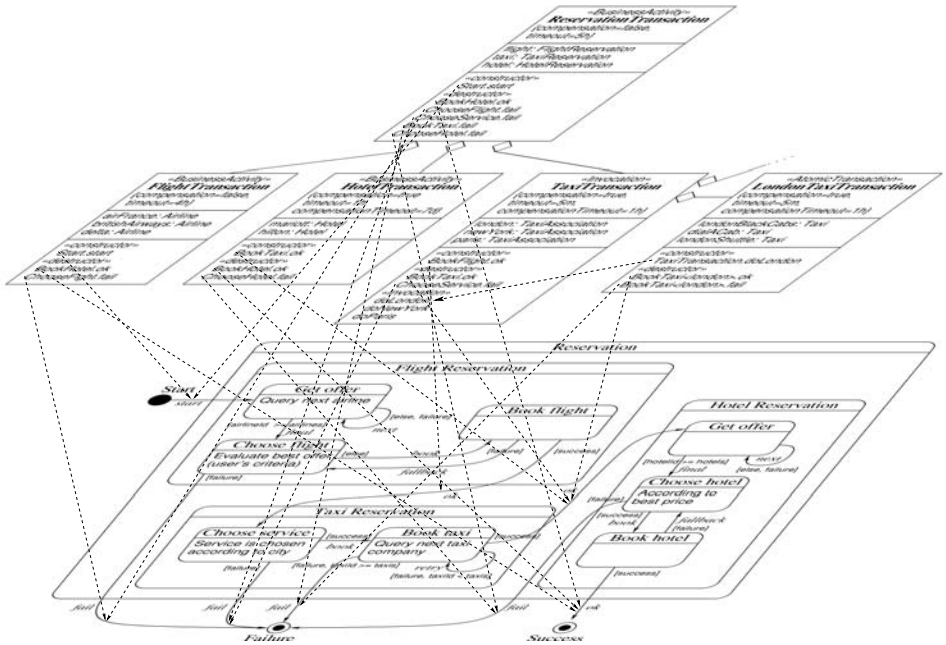

Figure 5: Merging Structural and Transactional Model

5.1 Related Modelling Languages

The *Business Process Modelling Notation (BPMN)* [Bpmi04] standard describes a notation for designing business processes. The claim of the document is to unify existing notations, and to ease design of executable business processes in BPEL4WS [BIM+03]. Similar to UML, the specification allows several diagram types. Some transactional properties (boundaries, compensation) are also supported by the specification. We use the broader UML specification for our approach because we want to add additional layers like security to our methodology in the future.

UN/CEFACT's Modelling Methodology (UMM) [Cefa01] is a UML profile for modelling business processes. Basically, it supports four hierarchically organized views: Business domains, requirements, transactions, and services. Using these views, a business process can be modeled top-down. However, graphical modelling of transactional properties is not mentioned.

The *Interaction System Design Language (ISDL)* [QuDS04] provides another graphical language for modelling Web services. We did not use the language in our paper since UML is more widely known and additionally supports referencing diagram elements with OCL.

5.2 Related Methodologies

[KHK+03] describes how models from the UML and ADF methodologies can be transformed into platform-specific models. From these models, descriptions in BPEL4WS [BIM+03] can be derived. However, transactions are only mentioned as a side aspect of modelling in the paper. [NoKo04] extends this approach by defining patterns for the rules.

[OrYP03] discusses Web service composition in several phases (definition, scheduling, construction, and execution). During these phases, the model should gradually become more concrete. The methodology is based on UML, OCL, and a set of composition rules. Transactions are not explicitly mentioned in these rules.

[DiDu04] states that a multi-viewpoint approach is needed for designing composite services. The paper identifies the viewpoints of interface behavior, provider behavior, choreography, and orchestration. Petri nets are used for the modelling approach. The paper does not discuss distributed transactions issues.

[BeDS05] also uses statechart diagrams to model composite Web services. The paper focuses on distributed composition. Transactions are shortly mentioned in future work, where it states that transactional semantics should be integrated into the model for a group of states in a statechart. However, no systematic approach is given yet.

5.3 Other Related Work

[KaBu04] proposes a template technique for Web services flows in order to ease service composition. These templates are parts of a business process description that can be used for composition. This concept may be useful for transforming our model diagrams into business process specifications in the future.

[HeVo03] defines a two-directional mapping between UML activity diagrams and BPEL process specifications as well as CSP process descriptions. These mappings can be used to find syntactic and semantic discrepancies in the description. The modelling process itself is not described. The paper does not explicitly mention transactions.

[Loec04] addresses transactional properties in a distributed middleware setting. The paper discusses Enterprise JavaBeans, but some of the work can be applied to Web services as well.

Comprehensive information about Web service transaction specifications can be found in [Papa03]. An overview on database transaction issues is given by [BrGS92, JaKe97].

6 Summary, Conclusion and Outlook

In this paper, we have demonstrated the need for a uniform design methodology for Web services. One layer of this methodology needs to be concerned with transactions. We have then identified requirements and challenges for Web service transactions for our case study and in general. Starting from these challenges, we have proposed the use of UML class diagrams as a transactional layer above a UML statechart diagram describing the service's structure.

While modelling the case study, we have identified some problems with our original assumptions, e.g. that the flight should be booked before the hotel and taxi is reserved. In a real-world example, discovering flawed assumptions would lead to a (possibly iterative) redesign. A major advantage of the model-driven approach is that conceptual flaws can be identified before implementation. The proposed introduction of new views can improve the detection of such flaws.

Throughout the paper, we have emphasized the necessity of separate views (so far, we have identified structure, transactions, security, and workflow). Figure 5 shows that it is infeasible to combine all these views into a single diagram, therefore references between the diagrams are necessary. Whether UML or another modelling language is used is of secondary importance — we have used UML because it is the de-facto standard for model-driven architectures.

An interesting result of our work is that most related papers do not discuss transactional properties of Web services. We think that these properties are an important ingredient for model-driven Web service design that must not be overlooked.

6.1 Future Work

This paper raises a number of questions that have not yet been answered and therefore it can only be the first part of a larger endeavor. Design requirements for the missing layers of security and workflow will need to be found, and the necessary semantics will have to be added to a UML diagram.

The transactional layer itself is also not yet complete. Some requirements have not yet been included in our model, other challenges still need more research before they can be supported by a modelling methodology. In the end, the model will have to be formalized, i.e. the set of stereotypes and tagged values that is used will have to be formally defined.

After this step, it should be possible to automatically derive transactions and transactional properties from the design diagrams. This automation can be used either to implement Web services that fulfill certain transactional guarantees, or to check whether existing services provide the transactional features needed for composition. When the metamodel is complete, it may well be possible to automate the transition from the UML diagrams to XML-based process descriptions.

References

[AFI+03] Arjuna, Fujitsu, IONA, Oracle, and Sun. Web Services Transaction Management, Version 1.0. Specification, 2003.

[BeDS05] Boualem Benatallah, Marlon Dumas, and Quan Z. Sheng. Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services. *Distributed and Parallel Databases*, 17(1):5–37, January 2005.

[BeIM04a] BEA, IBM, and Microsoft. Web Services Atomic Transaction. Specification, November 2004.

[BeIM04b] BEA, IBM, and Microsoft. Web Services Business Activity Framework. Specification, 2004.

[BIM+03] BEA, IBM, Microsoft, SAP, and Siebel. Business Process Execution Language for Web Services, Version 1.1. Specification, May 2003.

[Bpmi04] BPMI. Business Process Modeling Notation (BPMN), Version 1.0. Specification, May 2004.

[BrGS92] Yuri Breitbart, Hector Garcia-Molina, and Avi Silberschatz. Overview of Multidatabase Transaction Management. *VLDB Journal*, 1(2):181–239, June 1992.

[Cefa01] UN/CEFACT. UN/CEFACT's Modelling Methodology, Version 10. Specification, November 2001.

[DiDu04] Remco Dijkman and Marlon Dumas. Service-oriented Design: A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, December 2004.

[Elma86] Ahmed K. Elmagarmid. A Survey of Distributed Deadlock Detection Algorithms. *SIGMOD Record*, 15(3):37–45, September 1986.

[HeVo03] Reiko Heckel and Hendrik Voigt. Model-Based Development of Executable Business Processes for Web Services. In *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 559–584. Springer-Verlag, September 2003.

[JaKe97] Sushil Jajodia and Larry Kerschberg, editors. *Advanced Transaction Models and Architectures*. Kluwer Academic Publishers, June 1997.

[KaBu04] Dimka Karastoyanova and Alejandro Buchmann. Automating the Development of Web Service Compositions Using Templates. In *Proceedings of the Workshop "Geschäftsprozessorientierte Architekturen" at Informatik 2004*. Gesellschaft für Informatik, September 2004.

[KHK+03] Jana Koehler, Rainer Hauser, Shubir Kapoor, Fred Y. Wu, and Santhosh Kumaran. A Model-Driven Transformation Method. In *Proceedings of the Seventh International Conference on Enterprise Distributed Object Computing*, pages 186–197. IEEE, September 2003.

[LiZh04] Benchaphon Limthanmaphon and Yanchung Zhang. Web Service Composition Transaction Management. In *Proceedings of the Fifteenth Australian Database Conference*, volume 27 of *Conferences in Research and Practice in Information Technology*, pages 171–179. Australian Computer Society, January 2004.

[Loec04] Sten Loecher. A Common Conceptual Basis for Analyzing Transaction Service Configurations. In *Proceedings of the Software Engineering and Middleware Workshop 2004*, volume 3437 of *Lecture Notes in Computer Science*. pages 31–46. Springer-Verlag, September 2004.

[NoKo04] John Novatnack and Jana Koehler. Using Patterns in the Design of Inter-organizational Systems — An Experience Report. In *Proceedings of the OTM Workshops 2004*, volume 3292 of *Lecture Notes in Computer Science*. Springer-Verlag, October 2004.

[Oasi04] OASIS. Business Transaction Protocol, Version 1.0.9.1. Specification, 2004.

[Omg03] OMG. Unified Modeling Language Specification Version 1.5, March 2003.

[OrYP03] Bart Orriëns, Jian Yang, and Mike P. Papazoglou. Model Driven Service Composition. In *Proceedings of the First International Conference on Service Oriented Computing*, number 2910 in Lecture Notes in Computer Science, pages 75–90. Springer-Verlag, December 2003.

[PaCh03] Jonghun Park and Ki-Seok Choi. Design of an Efficient Tentative Hold Protocol for Automated Coordination of Multi-Business Transactions. In *Proceedings of the IEEE International Conference on E-Commerce*, pages 215–222, June 2003.

[Papa03] Michael P. Papazoglou. Web Services and Business Transactions. *World Wide Web: Internet and Web Information Systems*, 6(1):49–91, March 2003.

[QuDS04] Dick Quartel, Remco Dijkman, and Marten van Sinderen. Methodological Support for Service-oriented Design with ISDL. In *Proceedings of the Second International Conference on Service Oriented Computing*. ACM, November 2004.

Benjamin A. Schmit

PhD student at the
Vienna University of Technology
Information Systems Institute
Distributed Systems Group
Vienna, Austria, Europe
benjamin@infosys.tuwien.ac.at

Schahram Dustdar

Full professor at the
Vienna University of Technology
Information Systems Institute
Distributed Systems Group
Vienna, Austria, Europe
dustdar@infosys.tuwien.ac.at