Towards Transactional Web Services

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

When Web services are composed, transactions are needed to maintain the consistency of the distributed data in all but the most trivial cases. Today's Web service transaction specifications still leave some issues unresolved, e.g., the specification of quality of service aspects. Therefore, we propose a modeling methodology that forces the software architect to address such issues at an early stage of development. The methodology is divided into four layers of UML diagrams which reference each other: structure, transactions, security, and workflow. This separation of concerns can also be used to incorporate the knowledge of several experts into the design.

1. Introduction

In the last few years, Web services have developed from a novel and unproven concept to a well-defined and commonly used technology. A large number of Web services related specifications — some of them overlapping or defining similar concepts in another way — has been written. From the start, an important idea has been Web service composition, the construction of larger Web services out of simple ones. Without transactions, however, composite Web services would be restricted to most simple applications.

Traditionally, a transaction has been defined by the ACID properties (Atomicity, Consistency, Isolation, Durability; [18]). For most Web service use cases, this transactional model alone is no longer sufficient. In addition, long-running transactions are needed which violate some of the ACID properties, usually atomicity and isolation. Quality of service attributes, such as the possibility for compensation, can further differentiate between transaction types. Several competing specifications for Web services transactions exist, but so far, none of them is widely used. In this paper, we identify some of the challenges that Web services transactions face today.

Since transactions and the problems they may introduce are only one aspect in the creation of a composite Web service, we propose that the transactional model should form one part of a larger modeling methodology for composite Web services. We present a multi-layer modeling methodology based on the Unified Modeling Language (UML). On the bottom of our metamodel is the structural view, which shows the state transitions of the composite service. The next level is dedicated to transactions and forces the architect to explicitly address transactional issues at design time. In the future, we will also include security and workflow as separate design views.

Section 2 shows related work and discusses recent transaction specifications. Section 3 introduces a Web services example that we will use throughout the remainder of the paper. Section 4 identifies problems and open research questions that still prevent widespread use of Web services transactions. Section 5 presents a modeling approach for Web services which can help software architects around some commonly discoverd obstacles in Web services design at an early stage. Section 6 concludes the paper.

2. Related Work

In this section, we present related work. First, we describe approaches similar to the modeling methodology that we will introduce in Section 5. Afterwards, we introduce the competing Web service transaction specifications BTP, WS-AtomicAction and WS-BusinessActivity, and WS-CAF. We also discuss the process description language BPEL4WS in this context. More general related work can be found in [31].

Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05)
0-7695-2384-6/05 $20.00 © 2005 IEEE

2.1. Related Modeling Approaches

Our methodology, which is based on the concept of separation of concerns and on the Unified Modeling Language (UML, [27]), was first introduced in [31].

Orriëns, Yang, and Papazoglou [28] 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 [11] 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 [22] 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 [25] 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 [19] 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.

2.2. Transaction Specifications

Standard works on the topic of (database) transactions are [18, 17]. A more current survey on advanced transaction models can be found in the PhD thesis of Procházka [30]. Papazoglou [29] gives a survey on the topic of Web services and transactions. Because of the ongoing development in this field, however, the paper is already slightly outdated.

2.2.1. BTP

The Business Transaction Protocol (BTP, [26, 15]) has been released in 2002 as an OASIS specification, but has gone through corrections and adaptations ever since. The latest working draft (version 1.0.9.5) is from November 2004. The protocol separates between application elements (in our case Web services) and BTP elements, which are responsible for transaction coordination. BTP elements can be superiors (nodes that coordinate a transaction) and inferiors (nodes that participate in a transaction coordinated by another node). A BTP element can also implement both roles, which allows the creation of tree structures.

BTP knows two kinds of transactions: With atomic behavior, all elements contributing to a transaction must eventually reach the same conclusion about a transaction (confirm or cancel). Cohesive behavior allows some subelements to cancel while others confirm, which is useful in the case of different providers offering similar services. The behavior can be different for different nodes within a BTP transaction tree, which allows for the construction of complex transaction patterns.

BTP uses a two-phase approach where cancelling a transaction relies on compensation. Some optimizations to the protocol are mentioned, especially for the case of a degenerated tree. Timeouts allow nodes to reach an independent decision if they get no response, but this feature also introduces a potential for conflicts. In the case of a contradiction, manual intervention may be necessary. BTP has not been specified explicitly for Web services, but uses an XML syntax. The "carrier protocol" is undefined, but BTP can be embedded within a SOAP message.

2.2.2. WS-AtomicTransaction and WS-BusinessActivity

Originally named WS-Transaction, this specification has been split into two parts: WS-AtomicTransaction [5] covers ACID transactions while WS-BusinessActivity [6] defines long-running transactions. These specifications are based on WS-Coordination [7], a framework that allows the creation of a distributed context in a Web services environment. All specifications have been updated several times, but do not contain version numbers. An overview on these specifications is given in [14].

WS-AtomicTransaction defines three coordination protocols: Completion is used by a participant to initiates a commit or a rollback, Volatile 2PC is used for volatile, e.g. cache data, and Durable 2PC implements the standard 2-phase commit protocol.

A Business Activity is built out of a number of atomic transactions and relies on compensation for transaction rollback. Business activities may be used to form (possibly hierarchical) Scopes. The protocols provided are BusinessAgreementWithParticipantCompletion, which forces the

participant that initiated the transaction to specify when the transaction should be completed, and BusinessAgreementWithCoordinatorCompletion, in which completion is managed by a coordinator.

2.2.3. WS-CAF

The Web Services Composite Application Framework (WS-CAF, [1, 24]) has been proposed in July 2003. It consists of three parts: Web Services Context (WS-CTX, [2]), like WS-Coordination, describes sharing a common context between Web services. The Web Services Coordination Framework (WS-CF, [3]) focuses on information flow among a number of participating web services. Finally, Web Services Transaction Management (WS-TXM, [4]) defines the transactional models supported by WS-CAF. At the time of writing, no updates for WS-CAF have been released, but work on the specifications continues within an OASIS technical comitee.

WS-CAF supports three types of transactions: Traditional ACID transactions, long-running actions (which may be implemented without isolation), and business process transactions, which consist of one or several transactions of the other two types. To be included in a business process transaction, a transaction needs to define a compensator. If a sub-transaction fails, the compensators for all previously executed sub-transactions are run in reverse order.

2.2.4. BPEL4WS

The Business Process Execution Language for Web Services (BPEL4WS or short BPEL, [8, 32]) does not really fit here. It can be seen as a description language for business processes in which transactions are only one aspect. BPEL can be used in conjunction with WS-Transaction, which has been superceded by WS-AtomicTransaction and WS-BusinessActivity. It defines the high-level concepts sequence, switch, while, pick, flow, and scope to express the control flow of a business process.

Business Processes are divided into two groups: Abstract business protocols define only business process schemas that are not directly executable. Executable business processes, on the other hand, need to be defined completely, with their port types defined by an underlying WSDL description. In a suitable environment, they can be executed without further parameterization.

3. Case Study

In this paper, we will deviate from the usual Web services examples of supply chains and flight reservations and try to deduce how the production of a movie may be done in tomorrow's world of Web services — a rich example of possible future applications. Figure 1 shows the participating people as well as necessary resources in a UML class diagram.[1] We have simplified this case study somewhat, and we will reference parts of it throughout this paper.

Producing a movie is a complex task that requires the coordination of many film teams (camera teams, makeup teams, stunt teams, etc.). Most film teams need some equipment, and some depend on other teams. A film team provides a certain service, e.g. stunts. Together, the film teams form a film crew. A film crew is assigned to a location where filming is done, and to a production task. The task is managed by one or more directors, who also hire and assign the film crews. Directors may also hire external experts whenever expert knowledge is required in the film's production.

The following entities from Figure 1 provide Web services:

Director: A Web service in the director's office provides functionality for hiring film crews as well as external experts. The service also provides an overview on the persons currently assigned to the director's production task. The service implements the *Person* interface, which supports other directors in finding candidates for cooperative film projects.

ExternalExpert, CrewMember: Like the director's Web service, freelance experts's and out-of-work crew members's Web services both implement the *Person* interface. That way, the director can look for suitable experts and film crews can hire new members. The *ExternalExpertContract* is located on the expert's site. The presence of a contract tells a director that the expert is currently employed.

Team: The services provided by a film team are queried through the *Service* interface. Through this interface film crews find new teams to round off their services. The service provided by a team can be more than the sum of the services provided by its members: a single stuntman provides only simple stunts; a team of stuntmen can perform more complicated stunts by working together.

Crew: A crew is formed by several teams that, taken together, provide the services needed for a production task. The *CrewContract* is also located on the crew's site. (Contracts of individual crew members are not covered by the example.)

Location: The location Web service assists the director in choosing suitable locations for a production task.

[1] We acknowledge the work done by Martin Treiber on a preliminary version of this class diagram.

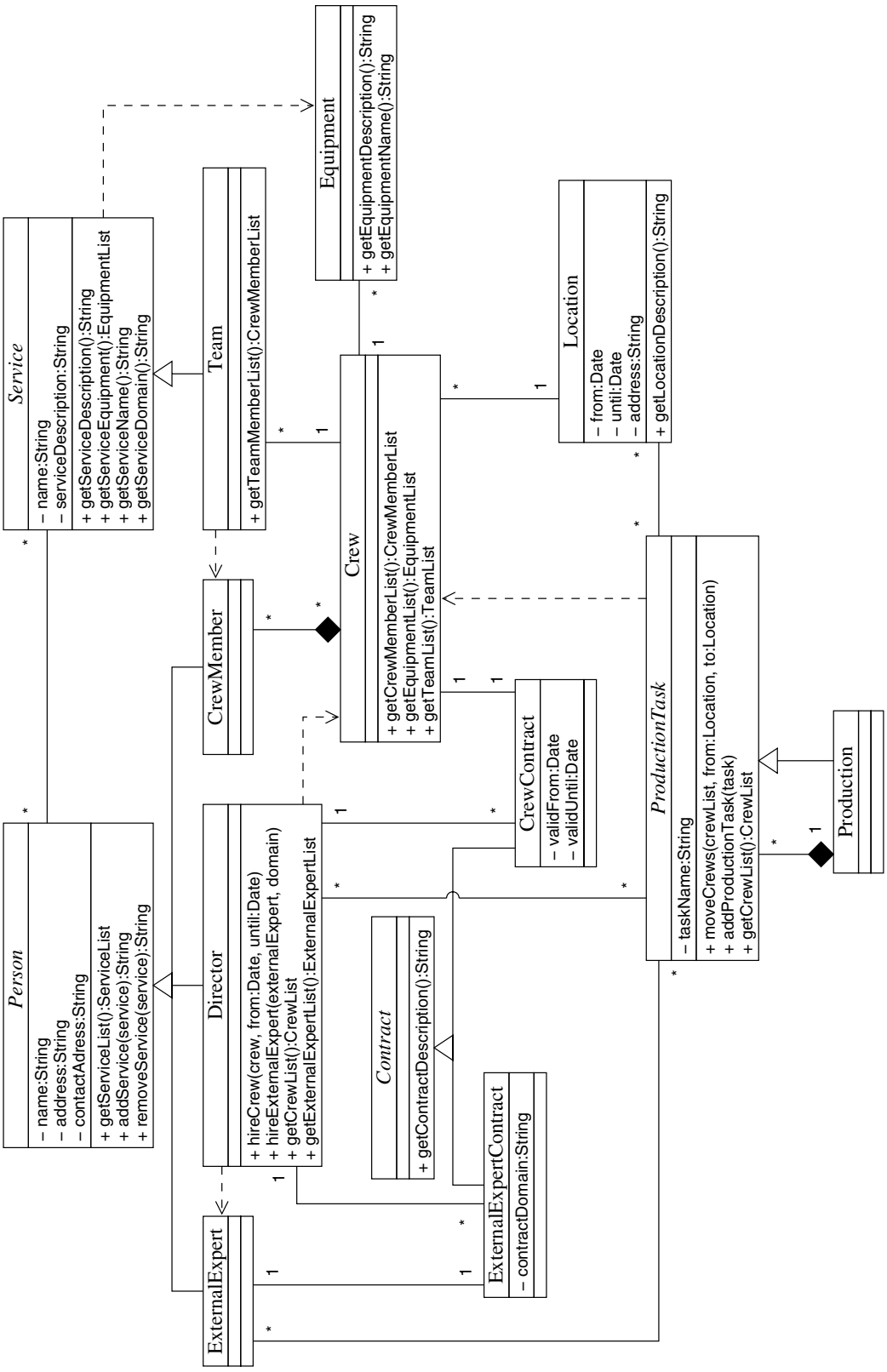

Figure 1. Film Production Case Study

Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05)
0-7695-2384-6/05 $20.00 © 2005 IEEE

Production, ProductionTask: A film production may contain several production tasks. A task's Web interface provides services for e.g. managing associated film crews.

Equipment: The equipment needed by a film team can be allocated through a Web service. This reduces the average equipment costs if tools can be used by several teams.

4. Challenges and Future Work

Now that we have described current Web services transaction specifications and introduced an example, we will identify some of the problems and open research questions on the use of transactions in Web services, especially in a distributed environment:

Different Transaction Types. The need for different types of transactions (ACID and long-running) has been addressed by all recent specifications. Unfortunately, the concrete implementation differs, so that it may be a difficult task to unify transactions written for different specifications. In a Web services environment, especially when using dynamic service discovery, it is unlikely that every participating service uses the same type of long-running transaction. In our example, this affects e.g., employment of new experts for a production task.

Quality of Service. The problem of different transaction types could at least partly be solved by introducing quality of service parameters. Parameters that qualify for inclusion in a QoS model could be the adherence to the ACID properties, a timeout after which the transaction should be aborted, whether transactions can be nested in a hierarchical structure, or whether only secure channels must be used to transmit data regarding the transaction. A combination of Web service transaction specifications with WS-Policy [9] may be used to express the QoS parameters.

Incompatible Message Format. Though XML has been accepted as the format for coordination messages between transaction participants, a uniform XML dialect for all transaction specifications has not yet been designed. (Semi-)automatic bridging between the various formats, which would allow interaction between transactional Web services using different specifications, can only work in a subset of all possible messages, because of the above mentioned difference in transaction types.

Compensation. Technically, all important transaction specifications support compensation. Since long-running transactions for Web services are usually performed without full atomicity, partly successful transactions need to be compensated if the transaction needs to be aborted. When only a database is affected, e.g., when reserving a location, compensation is relatively easy. When a transaction's results have been used by other transactions in the meantime (when the isolation property is weakened), compensation is tougher because many different Web services or other applications using the database may have been processing invalid data. This case is best handled by a well-reasoned design that considers compensation issues. We will further look into this question when we design the workflow layer of our modeling approach (see below). The hardest case, however, is compensation of real-world processes. Some of these processes, like cancelling the contract of a film crew, cannot be compensated without unwanted side effects, which in this case might be paying a cancellation fee. [12] discusses this problem in depth, and [23] proposes tentative holds as a way to ease the situation.

Scope Size. When transactions are organized hierarchically, with potentially many different Web services cooperating to reach a common outcome, the question of transaction scope size needs to be considered. Scope size here refers to the question whether we want to use a large number of small transactions or a smaller number of larger transactions. Smaller transactions may ease compensation, because they encapsulate fewer state changes in a business process. The use of larger transactions, on the other hand, may lead to a simpler Web service design and a smaller overhead in transaction processing. A yet unsolved problem is whether well-sized transaction scopes can be generated automatically, which is an interesting question when automatic service discovery is invoked — e.g., when an expert is sought for a film production task.

Service Location. When Web services for a composite service are chosen at run-time, a number of questions arise. We have just mentioned the problem of determining the right scope size. Also, the transaction type used by a dynamically located Web service needs to be compatible to the one the composite service intends to use, which overlaps with the quality of service question above. Finally, dynamic composition of transactional Web services will only be possible when all candidate services understand these problems and implement compatible solutions. In practice, this criterion can be satisfied best if all participants follow the same transaction specification, which is hard to enforce.

Security. Within the Web services protocol stack, security is covered by an own specification, WS-Security [20].

However, we believe that security cannot be an add-on feature, it has to be built in. If business critical operations, like the establishment of mutual contracts, are to be performed using Web services, minimum security requirements for transactions must be defined during the design phase. In Section 5, we will give an outlook on how security can be integrated into a modeling methodology.

Workflows. The BPEL specification can also be used to define workflows that may span several transactions. As for security considerations, we also believe that workflows need to be defined early in the design phase, as will also be outlined in Section 5.

5. A Web Services Modeling Approach

We will now present a methodology for composite Web services design using the Unified Modeling Language (UML, [27]). Then, we will explain how transactions can be modeled in this methodology. *Note: The order of the layers is not yet final, because we have not yet decided at which points references between the diagrams are necessary. Figure 4 shows a different order than discussed below because this order is better suited for presentation in this paper.*

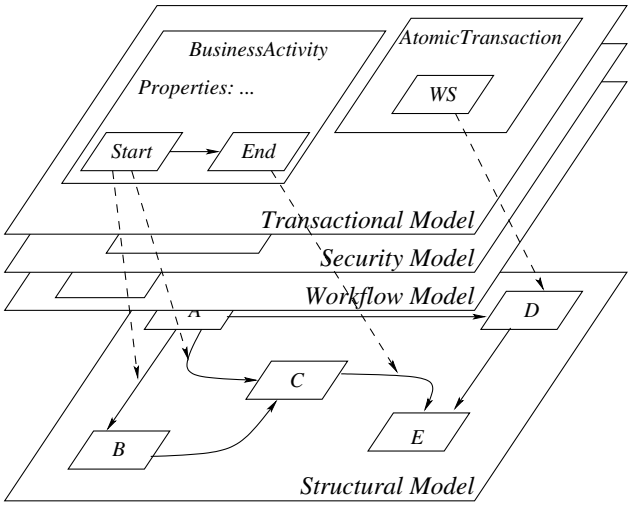

Figure 4. Layered Web Services Design

Our approach consists of a separation of concerns during the design phase. A more elaborate description can be found in [31]. As shown in Figure 4, we separate between structural, transactional, security, and workflow issues:

Structure. Our methodology starts with the design of the basic functionality of a composite Web service. We use a UML statechart diagram for designing service hierarchies and cooperations. Only functional aspects are defined in this diagram, other aspects have been moved into other views. The elements of the structural diagram can be referenced from higher-level diagrams using the Object Constraint Language (OCL; part of the UML specification). More information as well as an example can be found in [31].

Transactions. The second layer defines the transactions used in a composite Web service. We use a UML class diagram to represent this information. The semantics of this diagram type will be described below.

Security. We have already stated the importance of security considerations in Web services compositions. Therefore, 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 model 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.

Workflow. The workflow layer 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 layer alone, such as the problem of concurrent access to the same data we mentioned in Section 4. This layer will reference elements of the structural and the transactional view, and may reference some standard patterns typical for Web services. Currently, some of the information that should belong to the workflow layer is still in the structural layer, a clear separation between these layers is one of our next goals.

A modeling methodology can help the software architect to identify some commonly-made problems early in the development process. In our case, the methodology forces the architect to think about the four concerns that we are addressing independently, which also helps to introduce expert knowledge: The structural design diagram can be developed by a software architect, the transactional view by an expert on transactions, and so on. References in the higher-level diagrams help to keep the complexity of the different design issues at bay, since only a single aspect of a distributed Web application needs to be addressed at a time. A single diagram would contain too much information to be useful, which was another reason why a layered approach was chosen.

For our methodology, UML was chosen for three main reasons: It provides the descriptive power that allows us to express information about all views in the model, it includes the Object Constraint Language for references into the model diagrams, and it is widely accepted as a modeling language. In theory, any other modeling language that

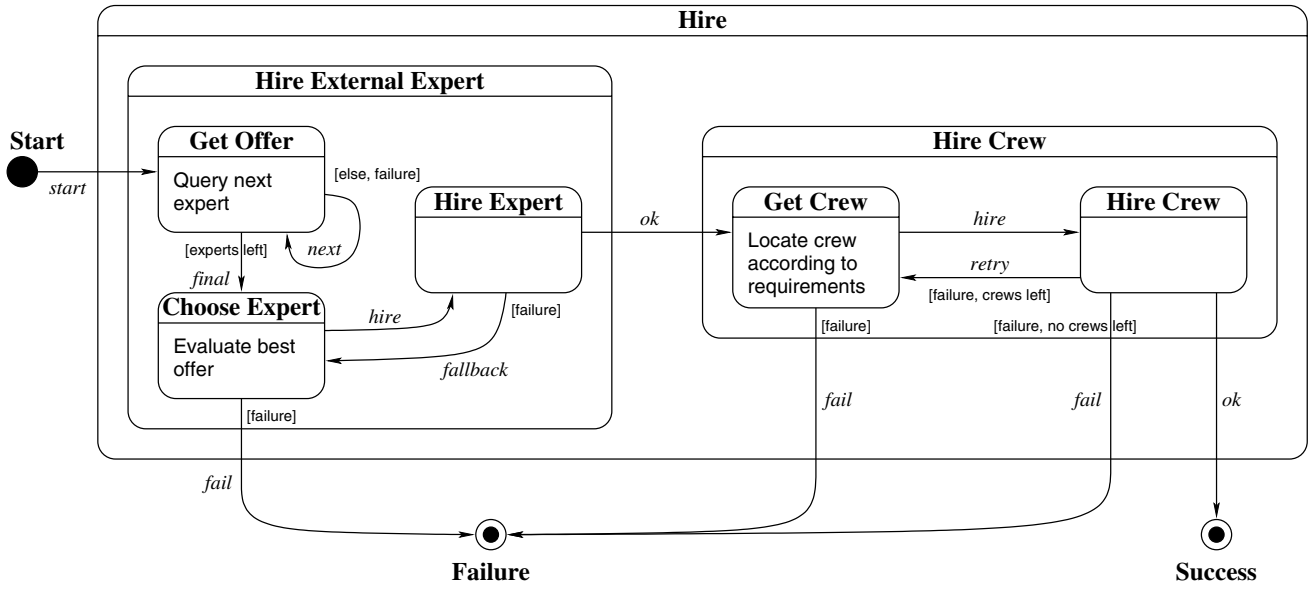

Figure 2. Example for the Structural View

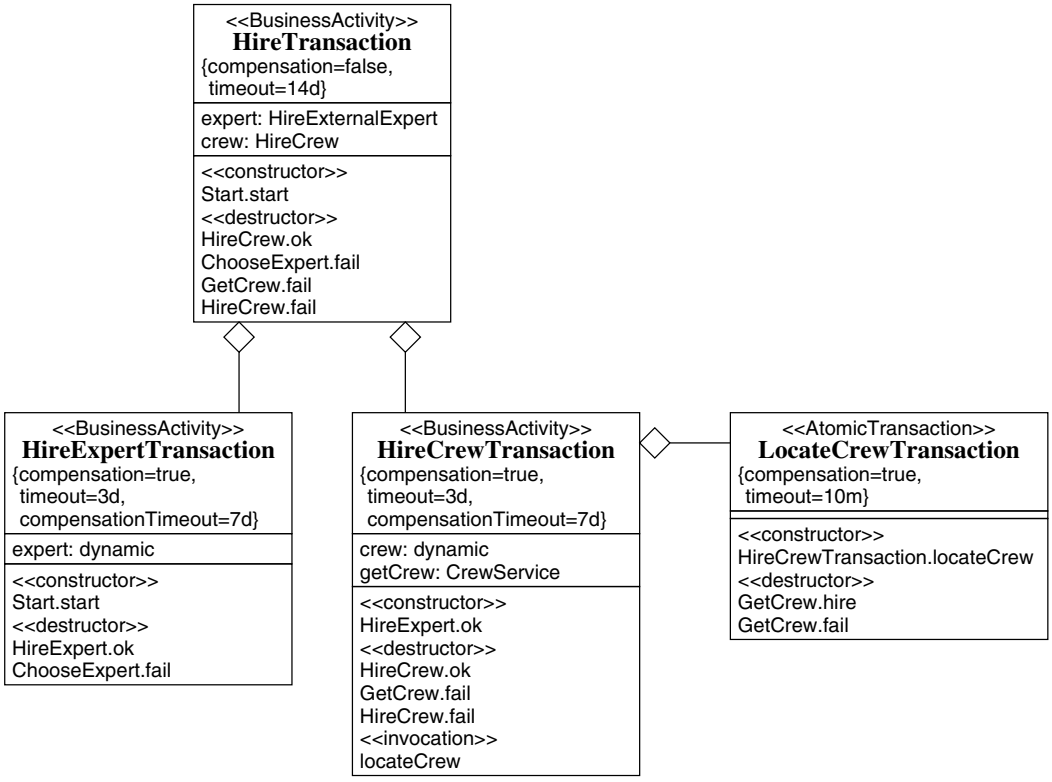

Figure 3. Example for the Transactional View

fulfills the first two criteria could be used as well, and different modeling languages can be used for different views.

5.1. The Transactional View

For this paper, we have modeled a structural and a transactional diagram for a part of our case study. In this example, the director is looking for crews and experts for a producation task.

Figure 2 shows the structural diagram for the selected example. The overall process consists of hiring both crews and experts. For the experts, we query all available Web services of known experts, using some Web service registry for the lookup. From the results, we choose the one that best fits the task and ask if he wants to join the project. For the crews, we have chosen a different approach to be able to illustrate other types of transactions as well: We use a locator service offered by an umbrella association of film crews to get an ordered list of fitting film crews, and again try to hire the best suited.

Figure 3 shows the transactional view of this example. Each transaction is modeled as a UML class. The main state from Figure 2 is easily recognizable as a top-level transaction, and the level below can also be mapped directly. However, the "LocateCrewTransaction" is not directly visible in the structural diagram, it has been introduced during the transaction design phase.

The distinction between ACID and long-running transactions is made by the use of stereotypes (here AtomicTransaction and BusinessActivity). Quality of Service aspects are introduced as tagged values. In the example, we have used the attributes "compensation" (a boolean value that states whether there is a compensating transaction) and "timeout" and "compensationTimeout", which specify the timespan during which a transaction or a compensating transaction can be active.

In our example, the overall transaction may last up to two weeks, and when it has finished, the outcome is definite. A crew or an expert must react to the offer within three days, and such a contract can be revoked for one week after it has been made. The transaction that locates a crew is atomic, and since in itself it does not affect the state of the process, compensation does not involve additional work and can, therefore, be done without a time limit.

The role of the attributes in a software class is fulfilled by the participating Web services, where the keyword "dynamic" is used when a Web service is bound only at runtime, either by querying a registry or by accessing another Web service. The constructors of a transaction are OCL references to the transitions in the structural diagram at which a transaction needs to be started. In the same way, the destructors refer to the transitions where a transaction should be committed or aborted. Finally, methods stereotyped "invocation" refer to constructors of subtransactions which cannot be mapped to transitions in the structural model.

6. Summary and Conclusion

In this paper, we have presented a modeling approach for Web services using UML. We have introduced the four layers of structure, transaction, security, and workflow, of which the first two already have been defined. For the structural view, we use a UML statechart diagram with which we model the basic aspects of a composite Web service. The transactional view consists of a UML class diagram which shows how the transactions cooperate, and which quality of service aspects they need to fulfill.

To show the applicability of our model, we have introduced an example scenario of distributed Web services. We have described a process in this scenario using our methodology and created a structural and a transactional design diagram which supplement each other. Related work as well as recent transaction specifications have been discussed.

We have also discussed a number of challenges that will need to be addressed before Web services transactions can become widespread. Among the most important are quality of service parameters, a uniform message format for transactions, resolving compensation issues, determining the right transaction scope size, and locating transaction-enabled Web services. The security and workflow layer of our model are our future work, and considered important parts of the final model. In order to efficiently use the final model, we plan to develop a tool based on the Eclipse platform [10, 13] that guides the developer through the design and transforms the model into basic description documents (e.g. using BPEL).

To conclude, we must state that current Web services transaction specifications do not yet address all the problems that we have identified. More research in the field of business transactions on the one hand and their application to Web services on the other hand is needed before they can be commonly used, especially if Web services are to be composed at least semi-autmatically. In this case, a majority of transactional Web services need to support mutually compatible standards.

As a systematic approach to Web service development, our modeling methodology — though not yet complete — can help to detect design weaknesses at an early stage in the development process. It forces architects to think about several design issues and supports a distributed design approach which is able to introduce the respective knowledge of several experts. The additional views of security and workflow issues will eventually make a unified approach to Web service design a reality.

References

[1] Arjuna, Fujitsu, IONA, Oracle, and Sun. Web services composite application framework (WS-CAF), version 1.0. Specification, July 2003.

[2] Arjuna, Fujitsu, IONA, Oracle, and Sun. Web services context (WS-Context), version 1.0. Specification, July 2003.

[3] Arjuna, Fujitsu, IONA, Oracle, and Sun. Web services coordination framework (WS-CF), version 1.0. Specification, July 2003.

[4] Arjuna, Fujitsu, IONA, Oracle, and Sun. Web services transaction management (WS-TXM), version 1.0. Specification, July 2003.

[5] BEA, IBM, and Microsoft. Web services atomic transactions (WS-AtomicTransaction). Specification, Nov. 2004.

[6] BEA, IBM, and Microsoft. Web services business activity framework (WS-BusinessActivity). Specification, Nov. 2004.

[7] BEA, IBM, and Microsoft. Web services coordination (WS-Coordination). Specification, Nov. 2004.

[8] BEA, IBM, Microsoft, SAP, and Siebel. Business process execution language for web services, version 1.1. Specification, May 2003.

[9] BEA, IBM, Microsoft, SAP, Sonic, and VeriSign. Web services policy framework (WS-Policy). Specification, Sept. 2004.

[10] K. Beck and E. Gamma. *Contributing to Eclipse. Principles, Patterns, and Plug-Ins*. Addison-Wesley, Oct. 2003.

[11] B. Benatallah, M. Dumas, and Q. Z. Sheng. Facilitating the rapid development and scalable orchestration of composite web services. *Distributed and Parallel Databases*, 17(1):5–37, Jan. 2005.

[12] D. Biswas. Compensation in the world of web services composition. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, Revised Selected Papers*, volume 3387 of *Lecture Notes in Computer Science*, pages 69–80. Springer-Verlag, July 2004.

[13] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose. *Eclipse Modeling Framework*. Addison-Wesley, Aug. 2003.

[14] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in web services. *Communications of the ACM*, 46(10):29–34, Oct. 2003.

[15] S. Dalal, S. Temel, M. Little, M. Potts, and J. Webber. Coordinating business transactions on the web. *IEEE Internet Computing*, 7(1):30–39, Jan. 2003.

[16] R. Dijkman and M. Dumas. Service-oriented design: A multi-viewpoint approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, Dec. 2004.

[17] A. K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, Apr. 1992.

[18] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 1993.

[19] M. Henkel, J. Zdravkovic, and P. Johannesson. Service-based processes — design for business and technology. In *Proceedings of the Second International Conference on Service Oriented Computing*, pages 21–29, Nov. 2004.

[20] IBM, Microsoft, and VeriSign. Web services security (WS-Security). Specification, Apr. 2002.

[21] S. Jablonski, M. Böhm, and W. Schulze, editors. *Workflow-Management: Entwicklung von Anwendungen und Systemen*. Dpunkt Verlag, July 1997.

[22] D. Karastoyanova and A. Buchmann. Automating the development of web service compositions using templates. In *Proceedings of the Workshop "Geschäftsprozessorientierte Architekturen" at Informatik 2004*. Gesellschaft für Informatik, Sept. 2004.

[23] B. Limthanmaphon and Y. 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, Jan. 2004.

[24] M. Little. An open standards approach to web services business transactions. In *Proceedings of the XMLOpen Conference 2004*, Sept. 2004.

[25] S. Loecher. 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, Sept. 2004. To be published.

[26] OASIS. Business transaction protocol, version 1.0. Specification, June 2002.

[27] OMG. OMG unified modeling language specification. Specification, Mar. 2003.

[28] B. Orriëns, J. Yang, and M. P. Papazoglou. Model driven service composition. In *Proceedings of the First International Conference on Service Oriented Computing*, volume 2910 of *Lecture Notes in Computer Science*, pages 75–90. Springer-Verlag, Dec. 2003.

[29] M. P. Papazoglou. Web services and business transactions. *World Wide Web*, 6(1):49–91, Mar. 2003.

[30] M. Procházka. *Advanced Transactions in Component-Based Software Architectures*. PhD thesis, Charles University, Faculty of Mathematics and Physics, Department of Software Engineering, Feb. 2002.

[31] B. A. Schmit and S. Dustdar. Model-driven development of web service transactions. In *Proceedings of the Second GI-Workshop XML for Business Process Management*, Mar. 2005. To be published.

[32] S. Tai, R. Khalaf, and T. Mikalsen. Composition of coordinated web services. In *Middleware 2004: ACM/IFIP/USENIX International Middleware Conference. Proceedings*, volume 3231 of *Lecture Notes in Computer Science*, pages 294–310. Springer-Verlag, Oct. 2004.