Aspect-Oriented Framework Modeling

Andreas Rausch
Technische Universität München
Boltzmannstrasse 3
D-85748 Garching
+49 (89) 289 17362
rausch@in.tum.de

Bernhard Rumpe
Technische Universität München
Boltzmannstrasse 3
D-85748 Garching
+49 (89) 289 17376
rumpe@in.tum.de

Lucien Hoogendoorn
Technische Universität München
Boltzmannstrasse 3
D-85748 Garching
+49 (89) 289 17362
hoogendo@in.tum.de

ABSTRACT
Aspects and aspect-oriented programming have gained much attention in recent years, but the focus was limited to the late stages of software development, especially late design and programming. In this paper, we describe a model-based approach based on an industrial case study that uses aspects. The approach provides a merge of the implementation of a requirements model with the predefined and thus reusable aspect-implementation, which is given in form of a framework.

Categories and Subject Descriptors
D.2. Design.

General Terms
Design, Languages.

Keywords
Aspects, UML, Framework, Modelling, Reuse.

1. INTRODUCTION
Practical experience has shown that aspect-oriented programming is an interesting, although still rather new approach that helps to increase reuse through separation of concerns and decoupling, which in turn leads to cost reduction and at the same time to a higher quality of the result [EFB01].

Aspects are used to weave cross-cutting functionality usually provided by a framework together with the application code. Application developers successfully (re-)use the framework implementation, like for instance a tracing mechanism or persistence functionality, for their own application implementation. Therefore aspect-oriented programming techniques are used to ease the connection of a framework with application code using weaving techniques.

In the foreseeable future, many application as well as framework development projects will follow a model-based development approach. In these projects models of the application resp. framework are built, detailed mappings and tracing-relationships between different model abstraction levels are documented, transformational approaches for model refinement are applied, and models are used for code and test case generation as well as animation for feedback to the customer (cf. [HMR+98,RRH98,Rum03,Rum03b]). Consequently, a framework vendor will not only deliver the framework to his customers, but also the model that describes the framework.

As already mentioned, the aspect-oriented approach is up to now successfully used as a programming paradigm to ease the integration of application and framework on the implementation level. Consequently, one would expect an aspect-oriented modeling approach to integrate application and framework on the requirement and design level.

However, there is still no generally accepted, clear and precise concept available to integrate application and framework models. Dependencies between application and framework models are not explicitly defined or modeled at all. Because concrete dependencies between application and framework are not explicitly formulated and tracked during the development, developers have to go into the details of the concerned code parts and glue application and framework together. As frameworks are usual complex pieces of code this is an extremely sensitive and error-prone implementation step.

In addition, this leads to systems that are very brittle with respect to changes, especially in the framework, as there is no clear model or specification describing separately the application concerns, the framework concerns, and the connection concerns between application and framework.

To sum up, the problem we are investigating in this paper is the question of how to integrate the advantages of a model-based and an aspect-oriented development approach. The goal is to identify and extract aspects already in the requirements and the design model, to reuse independently developed frameworks as implementations of the identified aspects.

Therefore we will provide aspect-oriented framework models on the requirement and design level. Based on this, a technique to glue framework models together with the application specific model to an integrated system model is needed. We call this “model-weaving” similar to code-weaving as provided by aspect-oriented programming languages [CW02,HJPP02,Paw02,SHU02].

The rest of the paper is structured as follows: Section 2 describes – based on an industrial case study – the state of the art of gluing an application together with an framework using aspect-oriented programming techniques. Section 3 introduces aspect-oriented framework models including the framework’s free parameter that the application has to fill. Section 4 finally shows how an application model can be weaved together with an aspect-oriented framework model.

2. CONNECTING APPLICATIONS AND FRAMEWORKS WITH AOP

This section gives a brief description of our case study, which consists of a small application and a persistence framework. We show how the application and the persistence framework can be glued together using aspect-oriented programming techniques.

Our simplified persistence framework is shown in Figure 1. The central part of the framework is the *PersistenceService*. The persistence service provides a transaction mechanism. Any actions of an application using persistence service have to take place in the context of a transaction.

Transactions are opened by calling *beginTransaction()*, and closed by calling *commitTransaction()* or *abortTransaction()*. A *Transaction* manages an object store containing all the persistent application objects, which have to implement the interface *Persistable*.

If a *Persistable* object changes, a call to *fireObjectUpdated()* hast to ensure that any changes will be stored when the transaction commits. Similar methods for object creation and deletion exist but are not relevant in the scope of this paper.

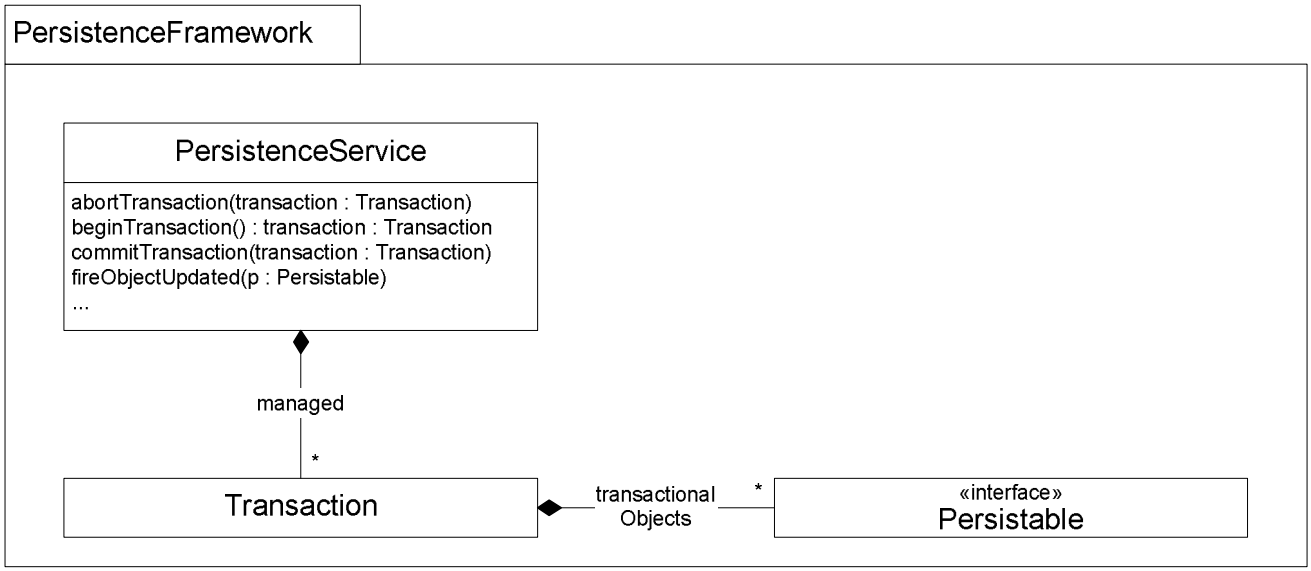

Figure 1. Simple persistence framework

Figure 2 shows our small sample application – a customer management system. The system can manage customers and their addresses. To store these objects persistently we will use the introduced persistence framework.

Therefore during development the model and the code have to be changed appropriately. All classes that have to be stored persistently have to implement the *Persistable* interface.

Methods that can cause changes to persistent data have to call *fireObjectUpdated()* at the end of the method execution. The persistent object that was changed is passed as a parameter.

Hence, the code of all persistent classes must be changed. A "implements Persistable" statement must be added and all "setter" methods, e.g. *setStreet()* and *setName()*, have to be changed by inserting the call of the method *fireObjectUpdated()*.

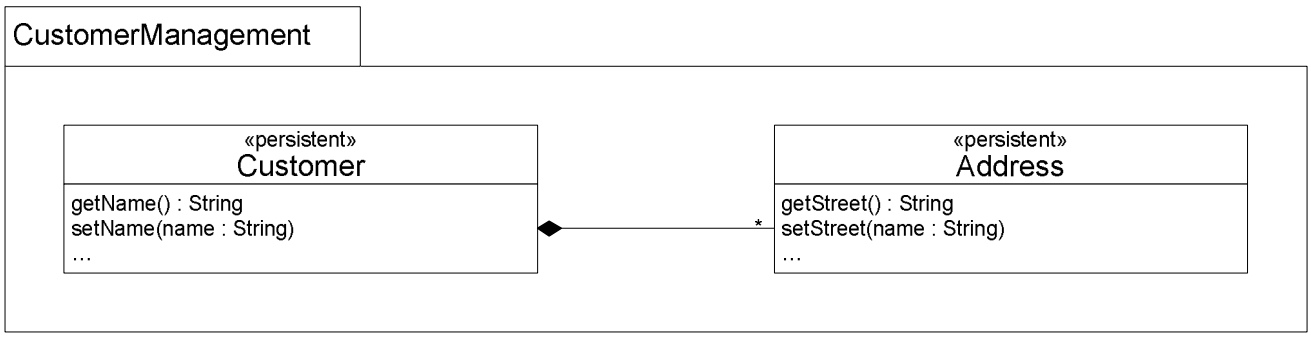

Figure 2. Simple application

Manually applying these code changes causes code tangling and scattering [KLM97]. For that reasons we may use AspectJ [KHH01] to keep these cross-cutting concerns in a separate aspect and to weave tool supported the aspect into the application code. We refer to [AJT03] for a detailed description of AspectJ.

Figure 3 shows the AspectJ implementation of the aspect that connects the application and the persistence framework. It contains to introductions (*declare parents…*) to add the supertype *Persistable* to the persistent application objects.

```
package CustomerManagement2PersistenceFramework;
import PersistenceFramework.*;
import CustomerManagement.*;

public aspect CustomerManagement2PersistenceFramework {

          declare parents: Customer implements Persistable;
          declare parents: Address implements Persistable;

          pointcut setter(Persistable p):
                    target(p) && call(void set*(..));

          after(Persistable p): setter(p)
          {
                    PersistenceService.fireObjectUpdated(p);
          }

          …
}
```

Figure 3. Connecting application and framework with AspectJ

3. ASPECT-ORIENTED FRAMEWORK MODEL

AspectJ enables us to connect the application and the framework at the code level, but we need a means of making aspects visible in the model to reason about the solution before implementing it.

Therefore an aspect-oriented framework model has to contain parts that represent the framework's "hot spots" [FPR01]. A hot spot serves as free parameter of a framework that the application has to fill. Hence, framework's hot spots have to be modeled as aspects which are not bound to a particular part of the application under construction. Figure 4 illustrates our approach. We have modeled an aspect as a separate entity which has a set of join points. For this purpose we introduced the stereotypes «*callJoinPoint*» and «*aspect*». A join point has parameters, which are shown in the attributes compartment of the join point box. These parameters define the context of the join point, similar to the way pointcut parameters expose context in AspectJ.

Advice is written in the operations compartment of the aspect box, and marked with the advice stereotype. A tag defines the kind of advice (*before*, *after* or *around*). The join point is passed as a parameter, so that the advice can access the join point context.

AspectJ introductions can also be modeled, using a class symbol with an «*introduction*» stereotype. The introduction entity serves as a kind of template for classes that are to be modified through aspect weaving. In our example, we introduce the *Persistable* interface to persistent classes (this was achieved with the *declare parents* construct in AspectJ). We use a generalization arrow marked with the «*implements*» stereotype for this purpose.

Based on this static model, we can describe the dynamic behavior of an advice. Figure 5 shows a sequence diagram for the update advice, which is executed after the *setCall* join point is reached. Here we can also see how the join point context is accessed by the advice. The advice makes use of the *persistentObject* parameter of the join point.

The pointcut *setter(…)* specifies all locations where the aspect connects to the application – all methods with the prefix "set". The advice *after(…)*is used to describe what happens when the join point is reached. In our case means this that the *fireObjectUpdated()* method is called.

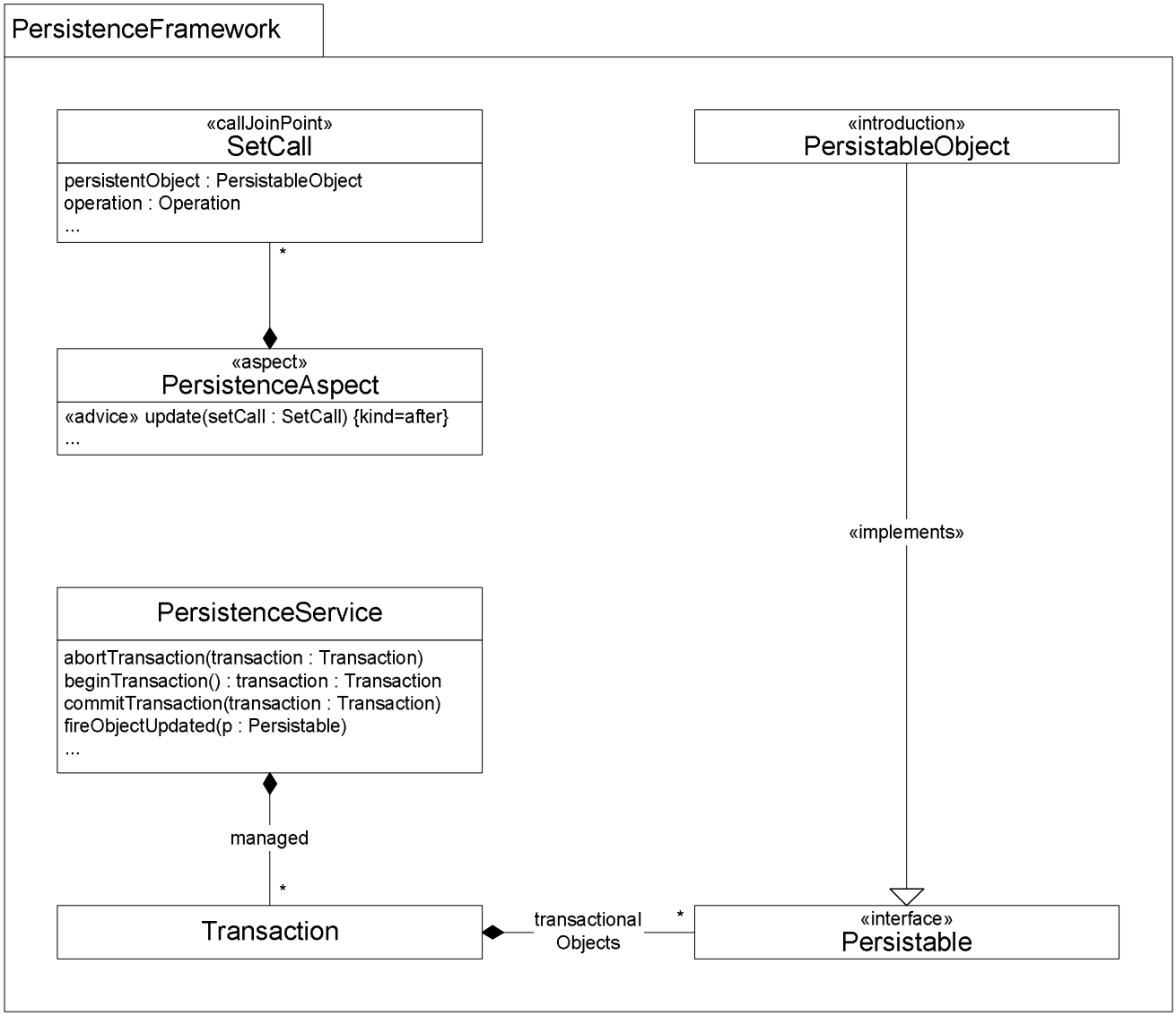

Figure 4. Aspect-oriented persistence framework

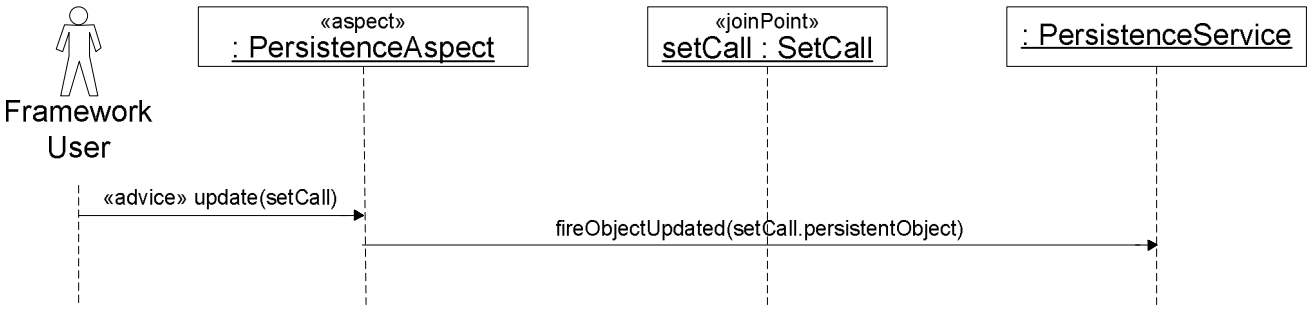

Figure 5. Persistence framework behavior modeling

4. WEAVING APPLICATION AND ASPECT MODELS

Based on the aspect-oriented framework model either by development on our own or by taking them from the shelf, we have to model the connection between our application and the framework.

To glue an aspect-oriented framework model with a given application model together, the concrete set of hot spots, like join points or introductions, have to be specified and the hot spots free parameters have to be bounded to elements of the application model.

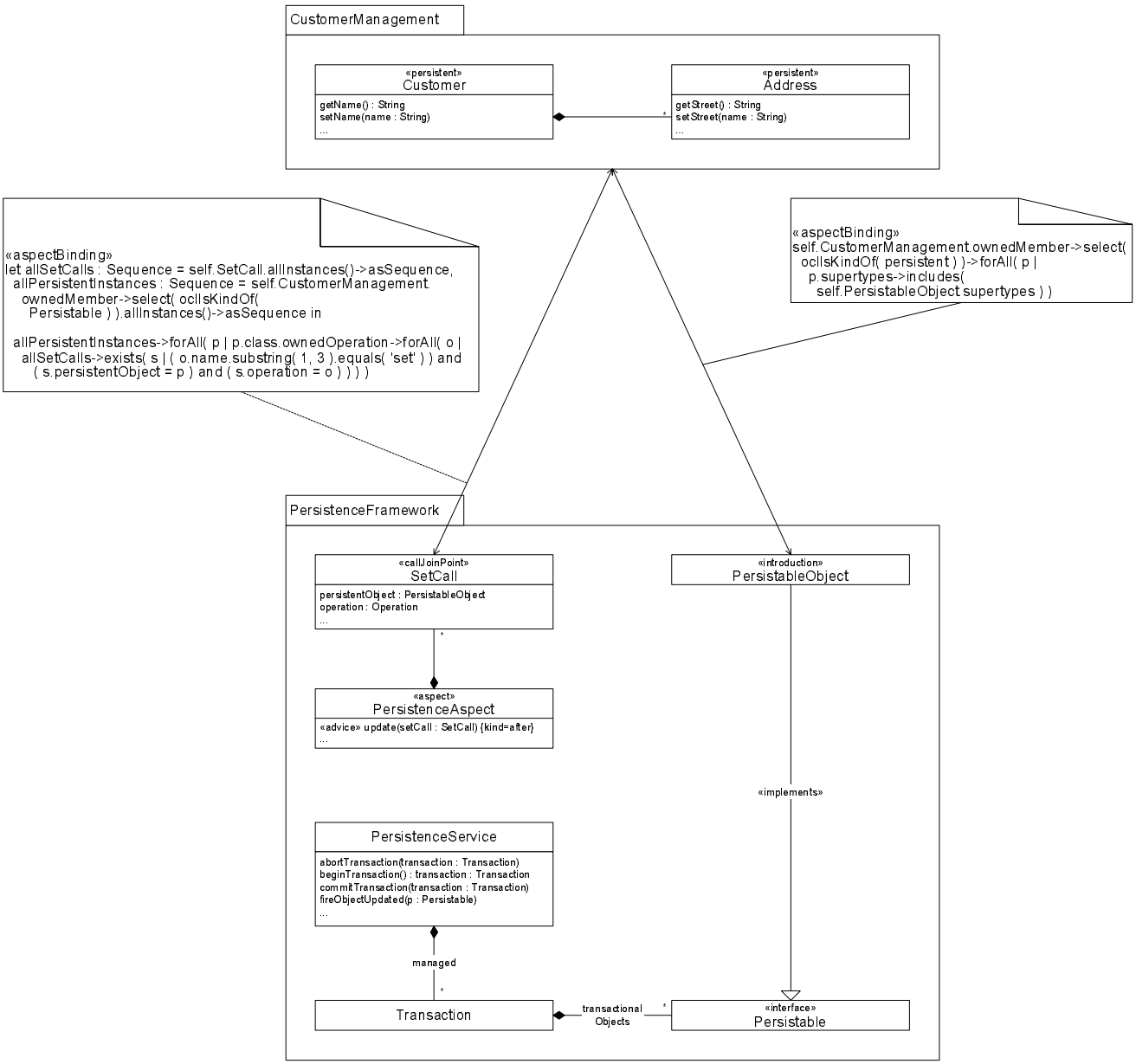

Figure 6. Weaving application and aspect-oriented framework models

For each hot spot class, like for instance SetCall and PersistentObject, at least one UML dependency between the hot spot and an UML element has to be modeled. To precisely describe the dependency, it is necessary to explicitly assess the meta-level of the models.

For that purpose, we use an OCL [OCL03] constraint on the meta-model that is labeled with the «*aspectBinding*» stereotype, as shown in Figure 6. The constraint is used to bind elements from the application package to hot spots in the framework.

The first OCL constraint concerning the introduction states that each persistent class in the application level has at least the same supertypes than hot spot framework class *PersistableObject*. Hence all persistent classes implement the class *Persistable* as required.

The second OCL constraint is a little more complex. For each operation of an instance of a persistent application class that starts with "set" exists an instance of the class *SetCall*. Whereas the attributes of this instance of the class *SetCall* refer to the corresponding set-operation and the called persistent application object.

5. FURTHER WORK

In this paper, we have presented some considerations and techniques, how to better integrate framework implementations in the early stages of software development. We have identified aspect-oriented framework models as the primary concept to allow this integration. It basically lifts the use of aspects from the implementation task to the analysis and design task and thus allows to incorporate aspects much earlier in the development process. This will allow the developers to increase their efficiency while at the same time to retain or even improve the quality of the results.

We have already identified three primary issues to further improve this approach:

- A specific language for the aspect binding has to developed to avoid large complex OCL statements
- An appropriate complete profile of the UML needs to be developed.
- Tool assistance for "model-weaving" is inevitable and must be consistent with AOP's code-weaving.

In the presented case study, we have shown how this approach can work. However, more experience with this approach is necessary to refine the presented concepts. Some interesting questions in this context are:

- What is the appropriate level of detail to model an aspect to be easily accessible and understandable for developers.
- Is it useful and feasible to provide several framework implementations for one aspect and provide an automated context sensitive selection process or an interactive selection assistant.
- Can we use the same technique for modeling "domain specific aspects" instead of technological aspects and thus extract certain aspects already during requirement elicitation.
- Can this technique be enhanced to develop own, domain specific aspects that allow to reuse cross-cutting functionality over similar applications. This is particularly interesting in large companies with many applications. E.g. calculation of interest rates can be regarded as such a domain specific aspect.
- Can models be tightly integrated with their implementations on a methodological level to ensure their consistency? This is important for aspects (regarded as a model plus a framework implementation) as well as for components and includes the question, how to ensure that the implementation matches the interface description.

In summary, we are confident, that aspect-oriented modeling will become an important technique in the portfolio of software engineers.

6. REFERENCES

[AJT03] The AspectJ Team. The AspectJ Programming Guide. Available at: http://www.eclipse.org/aspectj/. 2003.

[Coc02] Cockburn, A. Agile Software Development. Addison-Wesley, 2002.

[CW02] Clarke S., Walker R. J. Towards a Standard Design Language for AOSD. In: Proceedings of the 1st International Conference on Aspect-Oriented Software Development (AOSD), ACM, 2002.

[EFB01] Elrad T., Filman R. E., Bader A. (guest eds.). Aspect-Oriented Programming. In: Communications of the ACM 44, 10, 2001.

[Fow99] Fowler M. Refactoring. Addison-Wesley. 1999.

[FPR01] M. Fontoura, W. Pree, B. Rumpe. The UML Profile for Framework Architectures. Addison-Wesley. 2001.

[GHJV94] Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns, Addison-Wesley, 1994.

[HJPP02] Ho W.-M., Jézéquel J.-M., Pennaneac'h F., Plouzeau N. A Toolkit for Weaving Aspect Oriented UML Designs. In: Proceedings of the 1st International Conference on Aspect-Oriented Software Development (AOSD), ACM, 2002.

[HMR+98] Huber F, Molterer S., Rausch A., Schätz B., Sihling M., Slotosch O.. Tool supported Specification and Simulation of Distributed Systems. Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems, IEEE Computer Society. 1998.

[HRR98] Huber F., Rausch A., Rumpe B. Modeling Dynamic Component Interfaces. In: TOOLS 26, Technology of Object-Oriented Languages and Systems. M. Singh, B. Meyer, J. Gil, R. Mitchell (eds.). IEEE Computer Society. 1998.

[KHH01] Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J., Griswold W. G. An Overview of AspectJ. In: Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP), Springer Verlag, 2001.

[KLM97] Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C.,Loingtier J.-M., Irwin J. Aspect-Oriented Programming. In: Proceedings of ECOOP '97, Springer Verlag, 1997.

[KPR97] Klein C., Prehofer C., Rumpe B. Feature Specification and Refinement with State Transition Diagrams. In: Fourth IEEE Workshop on Feature Interactions in Telecommunications Networks and Distributed Systems. Ed.: P. Dini. IOS-Press. 1997.

[Kru00] Kruchten P. The Rational Unified Process. An Introduction, 2nd Ed. Addison-Wesley, 2000.

[OCL03] OMG. UML 2.0 OCL 2nd revised submission. OMG Document ad/03-01-07, Object Management Group, 2003.

[OMG01] OMG. Model Driven Architecture (MDA). Technical Report OMG Document ormsc/2001-07-01, Object Management Group, 2001.

[OMG03] OMG - Object Management Group. Unified Modeling Language Specification. V2.0. 2003.

[Paw02] Pawlak R. A Notation for Aspect-Oriented Distributed Software Design. CEDRIC Research Report, 2002. Available at : http://jac.aopsys.com/publications.html.

[PR03] Philipps J., Rumpe B.. Refactoring of Programs and Specifications. In: Practical foundations of business and system specifications. H.Kilov and K.Baclawski (Eds.), 281-297, Kluwer Academic Publishers, 2003.

[Rum02] Rumpe, B. Executable Modeling with UML. A Vision or a Nightmare? In: Issues & Trends of Information Technology Management in Contemporary Associations, Seattle. Idea Group Publishing, Hershey, London, pp. 697-701. 2002.

[Rum03] Rumpe, B. Agile Modelling with the UML. Habilitation Thesis. Munich University of Technology (in German). 2003.

[Rum03b] Rumpe, B. Agile Modeling in Lightweight Projects. In: Monterey 2002. Radical Innovations of Software and Systems Engineering in the Future. Workshop Proceedings. Venezia, Oct. 2002.

[Rum96] Rumpe B. Formale Methodik des Entwurfs verteilter objektorientierter Systeme. Utz Verlag Wissenschaft.

[SHU02] Stein D., Hanenberg S., Unland R. A UML-based Aspect-Oriented Design Notation for AspectJ. In: Proceedings of the 1st International Conference on Aspect-Oriented Software Development (AOSD), ACM, 2002.