Problems when Introducing Aspect-Oriented Constructs in Models of Functional Requirements and Possible Solutions to these Problems

Silvio Meier and Martin Glinz
Department of Informatics, University of Zurich, Switzerland
{ smeier | glinz }@ifi.unizh.ch

***Abstract*— The new paradigm of aspect-oriented programming demands new modularization techniques in the early phases of software development, i.e. requirements analysis and architectural design. Especially during the requirements analysis phase, there is a potential for conflicts between the introduced aspect-oriented paradigm and traditionally postulated qualities that must be fulfilled by a requirements specification. In this paper, we analyse the problems which may occur during the requirements phase when using aspect-oriented constructs. We also propose solutions for solving or mitigating the identified problems.**

I. Introduction

The introduction of aspect-oriented constructs in programming languages has been one of the major advances in modularizing software in the last few years. The usage of the aspect-oriented paradigm in programs improves the following qualities, as discussed in [1] and [2]:

- The modules[1] of a software are better *modularized*.
- The better modularization leads to a better *separation of concerns* and a clearer *responsibility* in the system, and therefore the artifacts are better *maintainable* and *reusable*.
- The time-to-market is reduced by a better modularized and simpler design, resulting in a reduction of costs.

Aspect-oriented decomposition of software creates the need for an early identification, separation and description technique for aspect-oriented artifacts, especially during the early phases of software engineering (requirements analysis and architectural phases). If aspect-oriented constructs are applied in the early phases of software engineering, the improvement of the above-mentioned qualities can also be applied to the artifacts at this stage. At the same time, there will be other improvements, such as a better traceability of crosscutting concerns in software requirements specifications (SRS). Summarized, we can say that several advantages result from the application of the aspect-oriented paradigm in SRSs.

In the case of software requirements artifacts, i.e. parts of a SRS, there are several qualities [3] (Unambiguous, Complete, Correct, ...) that are needed or desirable for a good specification. Especially the qualities *Understandable* and *Verifiable*/*Validatable* are strongly influenced by the introduction of aspect-oriented constructs in software requirements specifications.

In the rest of the paper, we will discuss the conflicts and problems arising from the introduction of aspect-oriented constructs in models of functional requirements[2]. In Section II, we will discuss the problems and conflicts that occur, while a possible solution for the problems is proposed in Section III. In Section IV, we conclude the paper and indicate the direction of future work of the topic.

II. Problems with Aspect-Oriented Constructs in Requirements Models

Aspect-oriented artifacts introduce a new modularization dimension for crosscutting concerns in SRSs. These artifacts are locally better understandable as compared to conventional artifacts. The main reason for this is a better separation of concerns, i.e. a local decoupling of software artifacts. However, the additional modularization dimension introduces also a new form of complexity. When trying to understand aspect-oriented requirements models in a global way (how the elements of the model interplay), we face more complexity than with conventional ones. As an illustration, take the merge operation in UML 2.0 [4], which works quite similar to the weave operation in aspect-oriented models and is extensively used for modularizing the UML metamodel. This modularization eases the comprehensibility of local metamodel packages. However, manually merging packages in order to obtain a global understanding is a nightmare. Even when merging is done automatically, the result can be rather difficult to understand. This is mainly due to the redundancy introduced by merging. Aspect-oriented artifacts behave in a similar way. Even with automatic weaving, the weaved product can still be rather complex and, hence, difficult to validate/verify. This leads us to our first problem:

Problem 1: When validating/verifying aspect-oriented models, one faces different levels of complexity, depending on the level of detail of the methods employed. Methods with a local focus are easier to apply to an aspect-oriented modularization than to traditional models. On the other hand, globally understanding weaved artifacts becomes more difficult, thus making validation/verification on this level harder.

Approaches dealing with SRSs need the possibility to describe different views on the system. However, the current approaches to aspect-oriented specification concentrate on particular views and neglect the other ones. For example, Araújo, Whittle and Kim [5] model behavior separately from structure, concentrating on the behavior view. They do not explain how the structure is modeled and how behavior and structure models interplay. Neither do they treat crosscutting concerns in other views. Suzuki and Yamamoto [6] use stereotypes for modeling structural crosscutting concerns in UML class diagrams. However, they do not describe how the crosscutting behavior is modeled and integrated into the class structure. We state this as our second problem:

Problem 2: Today's approaches neglect multiple views of crosscutting concerns. They rather concentrate on one view or few of them. However, for the use in requirements models, an approach supporting all possible kinds of views on a crosscutting concern is needed.

Most of the current approaches use a modeling language such as UML [4] which consists of loosely coupled sub-languages. These languages suffer from many problems concerning consistency, redundancy and view integration [7]. As a consequence, a user has to invest considerable intellectual effort for mentally integrating the different sub-models into one coherent model. When weaving aspects into an SRS based on such models, the effort required for understanding and interpreting this SRS may easily exceed the capabilities of an average user. Hence, we identify our third problem:

Problem 3: The understandability of aspect-oriented artifacts becomes worse when the language used for embedding the aspect-oriented constructs consists of loosely coupled sub-languages. The required effort may exceed the capabilities of an average user.

[1]Modules are either components of the core concern or aspectual constructs of the crosscutting concerns.

[2]However, the insights can also be applied in general to SRS artifacts.

III. Solutions to the Problems

Problem 1 (varying complexity for models on different levels of detail) can be handled by introducing different means for the validation/verification of SRSs on either a global or a local level. For a global validation/verification of the model, we suggest to do first a weaving[3] and then to use simulation techniques. With simulation, one can very easily validate/verify the model of a system. With the help of semi-formal simulation techniques and the use of stub simulation [8], one can even simulate partially finished models. It is also possible to simulate larger parts of the model (while it is hard to verify/validate large models by static means). For small portions of the system model, on the other hand, it is more convenient to perform validation/verification by static methods such as peer reviews or discussing the model parts with the customer.

Solution 1: Introduce different means for the local and the global level when validating/verifying models containing aspect-oriented artifacts. We suggest to use static methods like peer reviews for verifying/validating small parts (artifacts) in a local context of an SRS, while we recommend dynamic techniques such as simulation on the global level of the SRS.

Problem 2 is mainly caused by the focus of today's approaches. There is no reason why an aspect-oriented modeling language should not support more than one view. This can be handled by using a language that comprises all the views needed for describing aspect-oriented requirements models and by creating aspectual elements that are represented in all views.

Solution 2: Provide all views needed for the modeling of aspect-oriented constructs, i.e. a view for the internal behavior, the external behavior, the static structure (including associations between objects/classes) and the context elements of a requirements specification.

In particular, a modeling language for a software requirements specification should be based on an integrated approach, i.e. it should not consist of a set of loosely coupled modeling languages, such as UML [4]. Integrated models improve the understanding of the interplay between multiple views and concerns. Especially when dealing with requirements models, the models should be easily understandable as they will be used for the communication with the customer. Therefore, we propose to use an integrated modeling language as the basis for the integration of aspect-oriented language constructs.

Solution 3: The usage of an integrated modeling language reduces the intellectual effort for interpreting aspect-oriented models and eases communication.

IV. Conclusions

In this paper, we have identified three problems that occur when introducing aspect-oriented constructs in models of functional requirements and have sketched solutions for these problems:

- For handling different levels of complexity during validation/verification, different test means are appropriate. For a global focus, it is the best doing first a weaving and then a simulation. For a local focus, it is more efficient to use a static method like a peer review.
- The modeling language supporting aspect-oriented constructs should provide different views of the aspect-oriented artifacts, i.e. the static structure (including relationships), the internal behavior, the external behavior and a context description.
- Modeling languages which are extended by aspect-oriented constructs should consist of an integrated set of sub languages.

We have developed a modeling language in our research group called ADORA [7] which is currently being extended by aspect-oriented language constructs. ADORA integrates solutions 2 and 3: it is a language with tightly coupled sub-languages and consists of all mentioned views also for aspect-oriented artifacts. We are also working on the problem of simulating aspect-oriented models for implementing solution 1. An example of a conventional and an aspect-oriented model in the language ADORA can be found in Fig. 1.

[3]The weaving of crosscutting constructs increases also the complexity of the model, but when doing so, a model can be simulated.

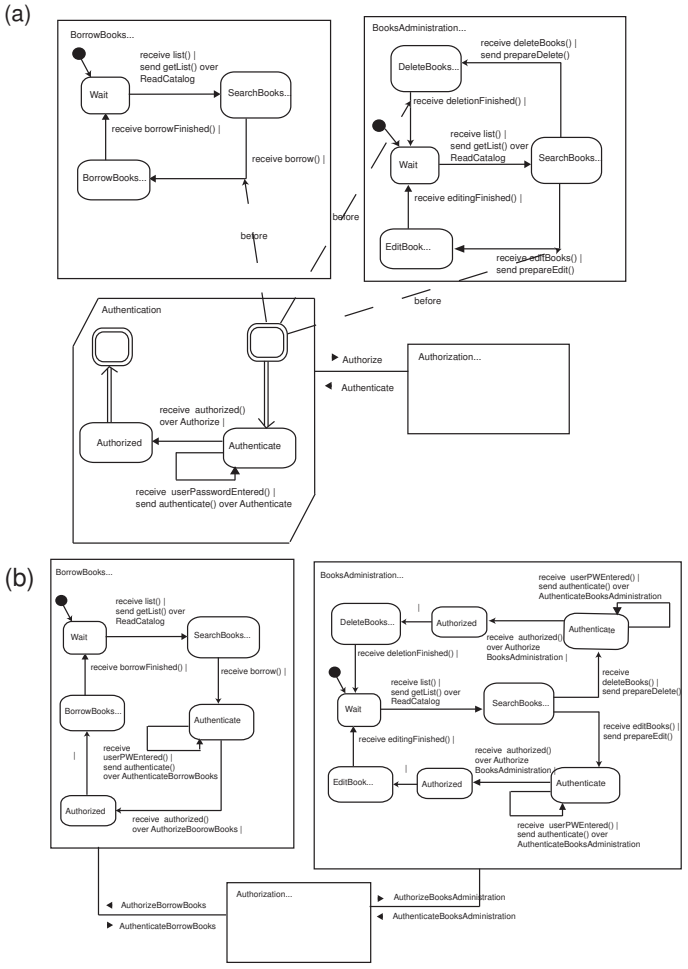

Fig. 1. An example of an aspect-oriented (a) vs. conventional model (b) in the language ADORA

References

[1] R. Laddad, *AspectJ in Action, Practical Aspect-Oriented Programming*. New York: Manning Publications Company, 2003.

[2] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and G. Griswold, William, "Getting Started with AspectJ," *ACM Communications*, vol. 44, no. 10, pp. 59–65, 2001.

[3] The Institute of Electrical and Electronics Engineers, *IEEE recommended practice for software requirements specifications. IEEE Std 830-1998.* IEEE Computer Society Press, 1998.

[4] Object Management Group, *UML 2.0 Superstructure Specification. OMG document ptc/03-08-02*. http://www.omg.org/cgi-bin/doc?ptc/2003-08-02, 2003.

[5] J. Araújo, J. Whittle, and D.-K. Kim, "Modeling and Composing Scenario-Based Requirements with Aspects," in *12th IEEE Requirements Engineering Conference*, 2004, pp. 58–59.

[6] J. Suzuki and Y. Yamamoto, "Extending UML with Aspects: Aspect Support in the Design Phase," in *Proceedings of the 3rd Aspect-Oriented Programming Workshop at the 13th Euorpean Conference on Object-oriented Programming (ECOOP 1999).* Springer, 1999, pp. 299–300.

[7] M. Glinz, S. Berner, and S. Joos, "Object-oriented modeling with ADORA," *Information Systems*, vol. 27, no. 6, pp. 425–444, 2002.

[8] C. Seybold, S. Meier, and M. Glinz, "Evolution of Requriements Models by Simulation," in *7th International Workshop on Principles of Software Evolution IWPSE 2004*, 2004, pp. 43–48.