

A Model Framework Weaving Approach Position Paper

Gerd Beneken¹, Frank Marschall¹, Andreas Rausch²

¹ Technische Universität München
Boltzmannstr. 3
85748 Garching

² Technische Universität Kaiserslautern
Gottlieb-Daimler-Straße
D-67653 Kaiserslautern

Current development approaches

Nowadays, in software development usually various models and description fragments are created. Some of these artifacts describe the core of the application, such as the data model or the user interaction model. Other artifacts describe cross-cutting concerns, such as security or the requirement: “every change of data has to be confirmed by the user, before it is written into the database”. During the development process these artifacts are combined, transformed, and finally implemented manually or even automatically. For instance a designer may combine a model of a dialog component specifying an action, that changes data, with a common description of a generic confirm dialog. The integrated dialog description may be afterwards implemented by a programmer. The manual combination of both artifacts and the transformation of the combined description into code are error-prone and hard to change.

Industry basically tests or actually uses two approaches for the combination today:

1. At the level of code and execution Aspect-Oriented Programming (AOP) is used [KLM+97]. An aspect defines a cross-cutting concern and the weaving instructions. A code weaver provides the actual *weaving at compile or runtime*.
2. At the level of design models and analysis models Model-Driven Software Development (MDS) is used [KWB03]. A model in MDS is a first class development artifact. Thus models are significantly more abstract than the implementation of code. However, these models cannot be executed. Hence, a abstract model is transformed into another, typically more detailed. A series of such transformations results in executable code. Thereby, a model transformer or a code generator reads some of the artifacts. The other artifacts and the combination rules are *implemented* in transformation rules or code generation templates (or somewhere else in the transformation / generation approach).

Problem statement

Both approaches have their advantages and drawbacks. AOP is at a low level of abstraction using the source code. But the aspects can be programmed explicitly and changed without affecting the other code. MDS works at an arbitrary level of abstraction. But MDS hides and mixes most of the concerns in transformation rules and code generation templates, which are usually hard to understand and to change, in particular if they mix more than one cross-cutting concern. Thus separations of concerns in MDS is currently not well supported. Several authors try to lift the ideas of AOP on the level of MDS and reap the benefits of these worlds: Using an arbitrary level of abstraction and handle cross-cutting concerns explicitly. One common approach is to provide a UML-Profile that has specific stereotypes to model AOP concepts [RRK+03]. This leads to an Aspect-Oriented Generative Modeling approach (AOGM).

As discussed in [Völ05] one main difference AOP and MDS is that MDS works by transforming static models. MDS has no relevance during execution time. Hence the models can be more abstract. Whereas, AOP contributes behavior to points in system execution. In many AOP approaches it is therefore possible, to consider dynamic aspects in the definition of pointcuts, like for instance the current call stack in AspectJ [AJ05].

Combining these approaches in AOGM leads to models that are on the same level of abstraction than the code. Hence, one still follows AOP and GP approach. Instead of using programming syntax you are forced to use UML syntax on a code level. This approach does not scale up, as UML is not the best choice in describing code. Therefore code may be the better option.

Model framework weaving approach

We propose a Model Framework Weaving (MFW) approach that consists of

- Model frameworks: A model framework is a model describing a specific aspect on an abstract level. A model framework defines hot-spots to integrate it with other models and model frameworks.
- Weaving Patterns: Weaving patterns describe how several source models can be weaved and transformed into one target model.
- Model Configuration: A model configuration consists out of a set of model frameworks and weaving patterns that finally generate executable source code.

The weaving of models and the model frameworks can take place at an arbitrary level of abstraction. One core model and several model frameworks are woven into one target model using the weaving patterns. We use UML to define the core model and the model frameworks and BOTL (*Bidirectional Object oriented Transformation Language*) to define the weaving patterns. A BOTL rule can be used either

- to enrich an existing model with information about additional aspects (this is conventional MSD) or
- to weave several source models and map them to one target model.

A comprehensive presentation of the BOTL model transformation language and its semantics can be found in [Mar05] and [BM03].

Weaving models using Hot Spots and Model Transformation

Model frameworks are used to represent aspects. They have hot-spots, which are the points, where the core model and the model framework are interwoven. The weaving patterns are defined using BOTL transformation rules, that combine the hot-spots with parts of the core model.

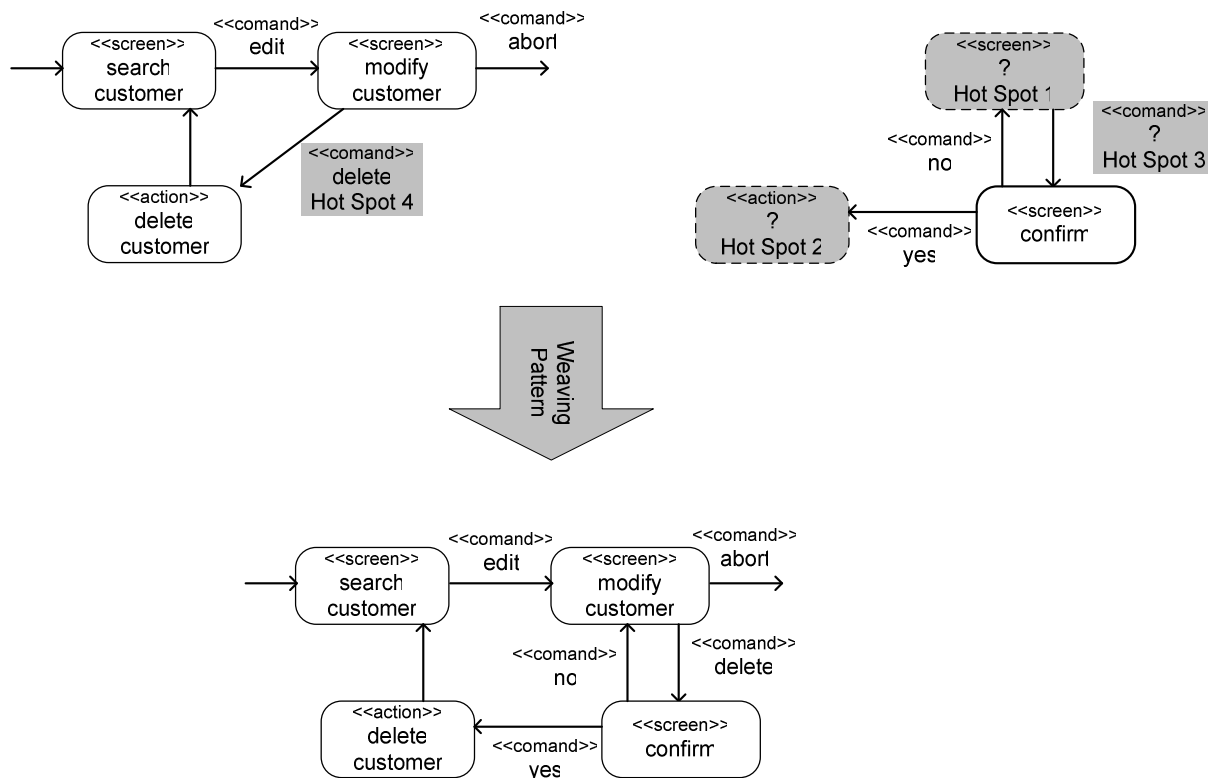


Figure 1: A graphical example of model weaving in an activity diagram

Figure 1 gives an example of model weaving. An activity diagram, that depicts a part of a page flow in a web application is interwoven with a model framework, that realizes a “confirm” aspect: Whenever something is deleted in the application a confirm screen shall be shown, before the deletion. The BOTL rule defines a search pattern in the core model as its left side. In the example it searches for a `<<screen>>` that is connected to an `<<action>>` by a `delete-<<command>>`. It maps the matches to parts of an existing or newly generated target model, which is defined at the right side of a BOTL rule. In our example the matches are mapped to the hot-spots of the “confirm” model framework which is duplicated for every single match.

The transformation rule is described in terms of instance patterns of the metamodels [BM03]. In the example the source and the target metamodel are both the UML metamodel. Thus the rule uses instances of the UML metaclasses like `Transition` or `ActionState` to express the desired transformation.

The example does not show the transformation rules in detail, it just sketches how the hot-spots of the model framework are filled with parts of the core model. The complete BOTL rules are complex, because they have to specify both the matching of the hot-spots and rules to copy the unchanged parts of the core model and the model framework.

The transformation rule in the example is a simple find and replace operation. It replaces one part of the core model by the model framework. It needs no additional context information. But, for instance, if the generic confirm dialog has to display which element is to be deleted, it needs additional information belonging to the left side of the transformation rules. In such cases either an other hot-spot has to be added to the model framework or the rule has consider the additional information.

Conclusion

We think, that the model driven development lacks of the explicit treatment of aspects. Because aspects can be defined using common modeling approaches, the weaving instructions need more attention. We propose to use a model transformation language, such as BOTL to define these instructions. Aspects can be modeled explicitly, either

- using just a transformation rule, that enriches an existing model (which is conventional MDSD) or
- using an own model framework together with transformation rules to weave two or more models.

The main advantage of AOP is, that an aspect can intercept the control flow of a program at a given point (join point). Behavior can be added to an existing program that way. Similar modifications of behavior can be modeled with model frameworks and BOTL. Behavior is modified by weaving sequence diagrams, activity diagrams or state diagrams. And the weaving patterns / transformation rules are defined using the metamodels of these diagrams and the join points are formulated with search patterns, which are the left sides of BOTL rules. The two advantages of the proposed approach are: first that the weaving and the modification of behavior can take place at an arbitrary level of abstraction, with almost arbitrary (meta)models and secondly that the cross-cutting concerns are modeled explicitly in model frameworks.

Further work

Currently we are investigating sets of transformation rules that specify typical aspects like e.g. logging or security at the level of UML models. Using the existing tool support for BOTL we already can transform UML models automatically according to these rules. However it turns out that such rules are hard to define and to read, if the metamodels are complex. Since the UML metamodel is definitely a very complex metamodel, we are currently working on a simplified graphical syntax to define rules for UML model transformations.

References

- [AJ05] The AspectJ Team. The AspectJ Programming Guide. Available at: <http://www.eclipse.org/aspectj>, 2005.
- [BM03] Peter Braun, Frank Marschall: „BOTL – The Bidirectional Object Oriented Transformation Language”, Technical Report TUM-I0307 at the Technische Universität München, 2003
- [KLM+97] Gregor Kiczales, John Lamping, Anurag Menhdhekar et al.: ”Aspect-Oriented Programming”, Proceedings European Conference on Object-Oriented Programming 1997, LNCS 1241, pp. 220-242, 1997
- [KWB03] Anneke Kleppe, Jos Warmer, Wim Bast: MDA Explained, Addison-Wesley, 2003.
- [Mar05] Frank Marschall: “Modelltransformationen als Mittel der modellbasierten Entwicklung von Software-Systemen“, Phd. Thesis, Technische Universität München, 2005
- [RRK+03] Andreas Rausch, Bernhard Rumpe, Cornel Klein, Lucien Hoogendoorn: Aspect-Oriented Framework Modeling, In the proceedings of the 4th AOSD Modeling with UML Workshop (UML Conference 2003), October 2003
- [Völ05] Markus Völter: Models and Aspects – Patterns for Handling Cross-Cutting Concerns in the context of MDSD. In the proceedings of EuroPLOP, 2005.