

# Preserving the Separation of Aspects through all Abstraction Levels in Model-Driven Software Development

Marco Mosconi  
Technical University Berlin, Germany  
mosconi@cs.tu-berlin.de

## 1. Introduction

There are two fundamental mechanisms in software development that should help managing the increasing complexity of modern software systems: abstraction and modularization. Preserving abstractions and modularizations from early to latest possible development phases is crucial for communication, efficiency and consistency in the development process.

While Model-Driven Software Development (MDSD) mainly supports vertically aligned abstraction levels in terms of more or less platform dependent models, Aspect-Oriented Software Development (AOSD) offers new modularization features that allow the horizontal decomposition of a system on one abstraction level (mostly implementation) [1].

Additionally, the amount of manual input needed from a developer should decrease when going down abstraction levels to reduce the risk of late and therefore potentially expensive mistakes. In the context of MDSD this means generating as much as possible from abstract models in each refinement step, while in the case of AOSD there is clearly no manual intervention intended in the woven code.

## 2. Weaving in AOSD and MDSD

In MDSD, the production of software involves models, transformations and manually implemented code. Naturally there is a need for Separation of Concerns in each of these locations and at each abstraction level. Much of the refinement between abstraction levels is encoded in transformations that add platform details to the abstract input models. When comparing AOSD and MDSD, transformations are often seen as a solution to weave crosscutting concerns (the platform details) into the modeled application. I suggest to strictly separate this platform refinement from the crosscutting concerns that are usually encapsulated as aspects in AOSD.

In MDSD, the typical “crosscutting concerns” that should be handled by transformations are those that are inherent to the platform (the details left out in the PIM abstraction). Although they could also be seen as aspects, they are not really part of the specific application we are developing and we would not want to model them as part of our PIM. Instead, we use transformations directly or in combination with explicit platform description models to integrate the platform details into our application.

Application-specific concerns on the other hand, should not be handled (woven) by the same transformations, but be kept isolated as aspects, to allow also developers at the PSM or code-level to benefit from their separation.

Besides this distinction the weaving time is important: weaving in AOP usually happens at the very end of the construction process, so the developer never has to cope with the woven result. In contrast, model transformations appear at earlier stages, starting at the PIM level. In most MDSD approaches, the developer needs to complete the results of transformations manually. This is why I believe that a “weaving” of aspects at the model level is a bad thing, because we lose the separation of aspects in a critical phase of software development (integration and refinement of generated models and code). Instead, model transformations

should only add platform details and map abstract application concerns to concrete aspects, keeping their modularization as long as possible.

### 3. Preserving Separation of Aspects in MDS

The proposed approach is mainly driven by two goals: a) preserving the separation of application concerns/aspects in a model-driven refinement process until the final application is completed and b) keeping single transformations as simple as possible.

To accomplish this, we need a rich set of modeling constructs on the PIM level that are sufficient to define crosscutting concerns of different flavours while being independent of a concrete AO platform. Most existing AO modeling approaches deal with high-level aspects showing only structure and pointcuts or focus on the graphical representation of a concrete AOP platform. Moreover, generic AO concepts like AspectJ are mostly not sufficient to model (complex) aspects, at least at the PIM-Level. Here we need more domain-specific and course-grained model elements such as roles, collaborations, subjects or viewpoints that can be mapped to corresponding features of existing AO platforms.

When trying to maintain separation of aspects between transformation steps, there are several issues to address. Some of them are:

- Suitable notations for abstract aspect concepts like pointcuts, roles, etc.
- Relationships and dependencies between different aspects
- Transformations from abstract aspect concepts to concrete AO platforms
- Selection of suitable AO platforms
- Identification of refinements that cannot be automated
- Referencing joinpoints in more platform-specific levels (abstract pointcuts)
- Consequences of manual refinements (consistency)

### 4. Object Teams as a supporting concept and platform

In [3] a promising approach using subject-oriented design and composition patterns is presented to consistently preserve separation of concerns down to the PSM level. They mention two approaches to finally generate code from the PSM: a straight-forward mapping of subjects and composition patterns to Hyper/J and the “flattenig” to a standard OO-model which can be mapped to any OO language.

Our first goal is to define an alternative mapping of the modeled concerns to the aspect-oriented platform Object Teams [2], which offers suitable AO concepts and is able to preserve that separation even until runtime. We believe that a feature- and concept-rich AO concept like Object Teams is suitable to maintain separation of concerns/viewpoints throughout the different abstraction levels in an MDA-approach while keeping the transformations between them rather simple. In a next step, we will work on an extension of the abstract aspect modeling concepts to support roles and collaborations. To show their applicability, we will provide mappings to Object Teams and at least one other AO platform.

### 5. References

- [1] D. Wampler. *The Role of Aspect-Oriented Programming in OMG's Model-Driven Architecture*. Aspect Programming, Inc., 2003
- [2] Object Teams Homepage. <http://www.objectteams.org>
- [3] P. A. A. Barbosa, C. F. G. Contreras, J. M.M. Rodriguez. *MDA and Separation of Aspects: An approach based on multiple views and Subject Oriented Design*. AOM WS at AOSD, 2005