

Aspects, Models and Model Transformations

Raul Silaghi

Software Engineering Laboratory
Swiss Federal Institute of Technology in Lausanne
CH-1015 Lausanne EPFL, Switzerland
rsilaghi@acm.org

Our interest in the “Models and Aspects” workshop is twofold:

Extending Model Transformation Languages with AOP Support

Besides the obvious importance of PIMs and PSMs in MDA, *model transformations* are undoubtedly the key technology in the realization of the MDA vision [1]. Among other usages, model transformations are the ones responsible for refining PIMs into PSMs (or abstracting away from PSMs to PIMs) and mapping PSMs to concrete middleware-based implementations, providing thus an elegant approach to adapt PIMs to the peculiarities of the new middleware infrastructures that do not cease to appear.

A clear requirement in OMG’s RFP [2] was (and still is) that model transformations should be able to act on *any kind of model of any kind of metamodel*. Since model transformations are at the same time models compliant with the metamodel of the transformation language, model transformations should be able to *transform* other model transformations independently of their metamodels. As a consequence, all currently existing model transformation languages (to our knowledge) implement such a “reflective” behavior. However, the “reflective” use of model transformations is not trivial, since it requires transformation developers to be familiar with the metamodel of the transformation language itself.

In order to overcome this frustrating impediment for the INRIA Model Transformation Language (MTL) [3], we presented in [4] an MTL *weaver* that modifies MTL transformations according to some *weaving behavior* that is specified as a special kind of MTL transformations, called *MTL-aspects*. Inspired from the Aspect-Oriented Programming (AOP) [5] world in general, and from AspectJ [6][7] in particular, the syntax defining the weaving behavior in MTL-aspects is a small AOP-like extension to the concrete syntax of the MTL language itself. In this way, relying on a few high-level AOP-like but MTL-based constructs for defining the weaving behavior, MTL transformation developers should not have any problems using this MTL extension straightforwardly in order to define their “reflective” model transformations.

Even though our research is carried out on the INRIA MTL transformation language, most of the concepts presented in [4] are MTL independent and could easily be applied to the future QVT specification language by providing higher level constructs for specifying the weaving behavior. For example, we can very well imagine having MTL-aspects written at the QVT specification level, and then automatically refine them for the MTL language when applying them in the context of MTL-based projects. Although the constructs introduced in [4] are very suitable for imperative model transfor-

mation languages (e.g., “before method return” or “after call”), we believe that similar counterparts may be identified in declarative model transformation languages as well (e.g., “after rule match”), and thus a common ground could be found at the QVT specification level.

Generating Aspects

Even though it has been around for almost four years already, MDA still remains a vision in many different aspects. In order to proliferate this vision and make it a reality, and at the same time to facilitate the development of middleware-mediated distributed applications, there is an imperative need for tool support. Unfortunately, so far there is very little in terms of concrete tools that actually support MDA beyond traditional UML modeling and skeleton-class generation. In order to fill this gap and provide developers with integrated tool support that allows them to incrementally refine their design models along middleware-specific concern-dimensions at different stages in the development life cycle of distributed enterprise applications, we designed the *Parallax* framework [8][9].

Implemented as an Eclipse plug-in, Parallax enables developers to look at the system under development from different perspectives (or viewpoints) by providing an extensible system of views. Based on aspect-oriented support and through a well-defined system of plug-ins, Parallax enables developers to incorporate middleware-specific concerns in their designs at different MDA-levels of abstraction, and to view their enhanced designs through a prism of middleware platforms and see how middleware concerns are actually implemented at the code level.

In the current state, aspects are only used to enhance the core of Parallax in order to be able to store and query concern-specific information, and to enhance the code generators with middleware-specific code generation capabilities. As the language imposed by Eclipse plug-ins is Java, all aspects are written in AspectJ, and thus we use a Java aspect-plugin to enhance, for instance, the C++ code generator (which itself is written in Java as well) with CORBA code generation capabilities. However, the final code that is generated does not support the separation of concerns any more. All concerns have their code tangled as it was generated by the enhanced code generator. The developer needs the Parallax’s highlighting support in order to see clearly what concerns are addressed where. In order to overcome this problem, we are very much interested in the possibility of *generating aspects* as separate units, outside of the code itself, and to encapsulate in these generated aspects the middleware-specific crosscutting concerns that we are addressing. For example, we would generate the C++ code that addresses the pure functionality of the system to be implemented, and in addition, we would generate separate aspects for each middleware-specific concern that the final system has to incorporate. Moreover, please notice that such aspects would have to be generated for aspect-oriented extensions of the targeted programming language, i.e., for an aspect-oriented extension to C++ [10] in the case of the previous example. Assistance could be further provided in order to weave the aspects in the pure functional code. With this approach, the developer would get several separate building blocks (code and aspects) that implement the final system in a pure concern-oriented fashion.

References

- [1] Sendall, S.; Kozaczynski, W.: *Model Transformation – the Heart and Soul of Model-Driven Software Development*. IEEE Software, **20**(5), Special Issue on Model-Driven Development, 2003, pp. 42 – 45. An extended version is available as Technical Report, EPFL-IC-LGL N° IC/2003/52, July 2003.
- [2] Object Management Group, Inc.: *MOF 2.0 Query/Views/Transformations RFP*. Document ad/02-04-10, April 2002.
- [3] French National Institute for Research in Computer Science and Control (INRIA): *Model Transformation Language (MTL)*. <http://modelware.inria.fr/>, May 2005.
- [4] Silaghi, R.; Fondement, F.; Strohmeier, A.: “Weaving” *MTL Model Transformations*. Proceedings of the 2nd International Workshop on Model Driven Architecture: Foundations and Applications, MDFAFA, Linköping University, Sweden, June 10-11, 2004. LNCS Vol. **3599**, Springer-Verlag, 2005 (to appear). An extended version is available as Technical Report IC/2004/50, Swiss Federal Institute of Technology in Lausanne, Switzerland, May 2004.
- [5] Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C. V.; Loingtier, J.-M.; Irwin, J.: *Aspect-Oriented Programming*. Proceedings of the 11th European Conference on Object-Oriented Programming, ECOOP, Jyväskylä, Finland, June 9-13, 1997. LNCS Vol. **1241**, Springer-Verlag, 1997, pp. 220 – 242.
- [6] Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J.; Griswold, W. G.: *An Overview of AspectJ*. Proceedings of the 15th European Conference on Object-Oriented Programming, ECOOP, Budapest, Hungary, June 18-22, 2001. LNCS Vol. **2072**, Springer-Verlag, 2001, pp. 327 – 353.
- [7] Eclipse Project: *AspectJ*. <http://www.eclipse.org/aspectj/>, May 2005.
- [8] Silaghi, R.; Strohmeier, A.: *Parallax – An Aspect-Enabled Framework for Plugin-Based MDA Refinements Towards Middleware*. Book Chapter in “Model-Driven Software Development”, Volume II of “Research and Practice in Software Engineering”. Beydeda S., Book M., and Gruhn V. (Eds.), Springer-Verlag, 2005 (to appear). Also available as Technical Report IC/2005/021, Swiss Federal Institute of Technology in Lausanne, Switzerland, May 2005.
- [9] Software Engineering Laboratory at the Swiss Federal Institute of Technology in Lausanne: *The Parallax Project*. <http://parallax-lgl.epfl.ch/>, January 2005.
- [10] AspectC++, <http://www.aspectc.org/>, May 2005.