Generic Concern-Oriented Model Transformations Meet AOP

Raul Silaghi and Alfred Strohmeier

Software Engineering Laboratory
Swiss Federal Institute of Technology in Lausanne
CH-1015 Lausanne EPFL, Switzerland

E-mail: {Raul.Silaghi, Alfred.Strohmeier}@epfl.ch

Abstract. Separation of concerns allows developers to manage large distributed systems by tackling one problem at a time. Model transformations refine models along one concern-dimension. Aspects encapsulate implementation details that cut across the boundaries of several components. In this position paper, after a short introduction to these emerging technologies, we explain how generic concern-oriented model transformations can meet aspect-oriented programming in order to complete the life-cycle of software application development in a pure MDA-compliant way based on separation of concerns. At the end, we present some requirements that tool vendors should provide if they decide to support such an approach.

1 Introduction

Integration and interoperability have always been at the heart of software engineering for anything else than non-trivial software products and systems. Philosophically speaking, thinking is modeling [1], and thus, agreed upon or compatible models are the key to "inter-think", i.e., to facilitate communication among humans, among computers, and among humans and computers. Approaching this line of thought from the other side, we can say that making things interoperable requires thinking. Since thinking is modeling, we infer that we cannot make things interoperable without models. The Model Driven Architecture is the next evolutionary step in that direction, trying to raise the level of interoperability from a mainly syntactic interface level to a more expressive behavioral level (realized through models).

Model Driven Architecture (MDA) is a new approach to software architecture that provides a standardized way to IT-based information system specification by clearly separating the "what" and the "how", or as said in [2] "... that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform". Both specifications are expressed as models: *Platform Independent Models (PIMs)*, which specify the structure and functions of a system while abstracting away technical details, and *Platform Specific Models (PSMs)*, which are derived from PIMs and specify how the functionality is to be realized on a selected platform. Moving from one model to another is achieved by applying model transformations, and finally, code generators are used to map the most specialized PSM to a specific technology platform, such as CORBA, J2EE, .NET, or Web Services. More details and documents on MDA can be found in [3].

Separation of concerns is an approach to decomposing software into smaller, more manageable and comprehensible parts, each of which deals with, and encapsulates, a particular area of interest, called a *concern*. Basically, a concern can be viewed as anything that is of importance to the application, be it infrastructure, code, requirements, design artifacts, etc. Separation of concerns provides support for overcoming the "tyranny of the dominant decomposition" [4], from which many modern artifact notations suffer. For example, object-oriented approaches provide mechanisms to encapsulate certain kinds of concerns, such as data and functions. However, they do not provide mechanisms that would allow one to encapsulate cross-cutting concerns, such as distribution or security, in an effective way.

Aspect-Oriented Programming (AOP) [5] has been proposed as a technique for improving separation of concerns in software. This approach makes it possible to separately specify various kinds of concerns and localize them into separate units of encapsulation, called *aspects*. One can deal with both the concerns and the modules that encapsulate them at different levels of abstraction, not only at the code level. AspectJ [6] is a general-purpose aspect-oriented extension to Java that introduces concepts like *join points* and *pointcuts* to describe the structure of the cross-cutting concerns, and *advices* to specify the desired behavior to be performed throughout the identified structure.

Middleware is an essential element in large distributed systems like those that support enterprise applications that require the interoperation of multiple components. Since middleware itself is software, it is subject to concerns like software in general. In [7], several dimensions along which concerns about middleware can be separated were identified and grouped into four main categories. One of those categories was Middleware Services, as middleware addresses specific concerns of a system, for example, communication, distribution, concurrency, security, or transactions. Multiple attempts have been tried out to somehow *aspectize* away these concerns from the rest of the distributed application. Some of them have failed, some others have succeeded up to a certain degree. However, as clearly described in [8], the big difficulty stems from the fact that, although the mechanisms used to implement these middleware-specific concerns are physically separated from the "functional" part of an application, they still remain semantically coupled. Thus, without having any idea about the application (semantics), it becomes impossible to apply, for example, a *general transactional aspect* to a previously non-transactional code and to obtain the desired functionality, i.e., transactional behavior.

In the rest of this position paper we will argue that generic model transformations could be the way to introduce application specific information at the PIM and PSM levels, and that the same information could be used to specialize aspects that will be applied at the implementation level. A specialized model transformation would refine a model (PIM or PSM) along a single concern-dimension and would have associated a specialized aspect that would implement the concern at the code level.

2 Putting the Pieces Together

MDA identifies four types of model-to-model transformations (mappings) within the software development life-cycle [2]: *PIM-to-PIM transformations* relate to platform-

independent model refinement and are applied when PIMs are enhanced, filtered, or specialized; *PIM-to-PSM transformations* are used to project a PIM to the selected execution infrastructure; *PSM-to-PSM transformations* relate to platform-dependent model refinement; *PSM-to-PIM transformations* abstract models of existing implementations into platform-independent models.

Taking variability into account, developers should be given the possibility to specify a PIM and a PSM that are specific to a selected IT application, or IT "domain" if we refer to the new Domain-Driven Development track at OOPSLA 2003 [9]. Models specific to an application cannot be specified by applying a series of predefined transformations. For this reason, we suggest that model-to-model transformations should be *generic*, and a set of parameters should be used to specialize the transformation and express the properties that are specific to a given application.

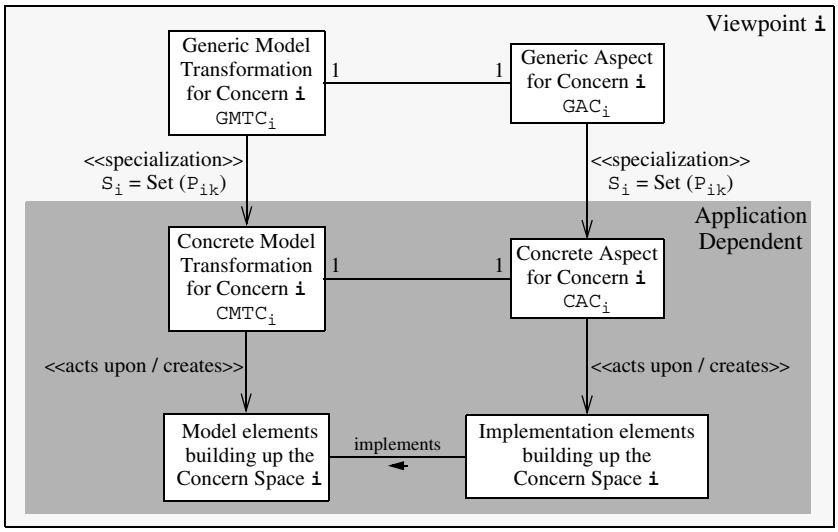

Fig. 1. Generic Concern-Oriented Model Transformation Meets AOP

We imagine that at each refinement step in the MDA approach a different concern should be handled by regarding the current model from the viewpoint corresponding to that concern. Fig. 1 presents what should be seen from a certain viewpoint `i`, at a certain level of abstraction, along one concern-dimension. As shown on the left-hand side, we would like to apply the generic model transformation $GMTC_i$ in order to refine the current model along the dimension associated with concern `i`, which can be seen from viewpoint `i`. However, beforehand, the generic model transformation needs to be specialized according to the particularities of the current application, particularities that we describe here by a set of parameters P_{ik}, where `i` indicates the concern under consideration, and `k` is used as an index to show that there might be several parameters that need to be set along a certain concern-dimension. Once the specialized/concrete model transformation for concern `i` ($CMTC_i$) is obtained, it can be applied and the result would be a refinement of the model elements building up the concern space `i`, i.e., the model elements seen from viewpoint `i` as being involved in addressing the concern `i`.

On the right-hand side of Fig. 1, one can notice that each model transformation (generic or concrete) has associated an aspect (generic or concrete, respectively) that would implement the concern under consideration at code level. Moreover, we tend to believe that the set of parameters S_i, used to specialize the generic model transformation, could be used to specialize the corresponding generic aspect as well, thus overcoming the problem of semantic coupling described in [8]. The order in which specialized/concrete aspects will be applied at code level (their precedence) is dictated by the order in which the specialized/concrete model transformations were applied at model level.

During the previously described process, models for a specific application evolve in a series of refinement steps, each step consisting of selecting the appropriate generic transformation (corresponding to a specific concern-dimension), configuring it to derive a specialized transformation, and finally applying the specialized transformation. At the implementation level, rather than having one code generator which takes the most specialized PSM and generates platform specific code, we propose to have a code generator for the pure "functional" model of the application instead, and then have *aspect generators*, which generate concrete aspects from concrete model transformations, for the cross-cutting nature of several concerns that the application needs to incorporate.

For a better understanding of our position, let's consider an example. Suppose we need to implement a system, and that besides the functional requirements of that system, certain middleware specific concerns need to be addressed as well, such as distribution, transactions, and security. Let's name these concerns C_1, C_2, and C_3. At a certain point in the MDA life-cycle, we will need to address these concerns in the context of our specific application. As previously presented, we will have three generic model transformation T_1, T_2, and T_3 (see Fig. 2), which will all be specialized with specific parameters according to the concern being handled and to the particularities of the application. For each specialized/concrete model transformation, e.g., $T_1<p_{11}, p_{12}, \ldots>$, a specialized/concrete aspect, e.g., $A_1<p_{11}, p_{12}, \ldots>$, will be automatically generated and will implement the concern under consideration at code level.

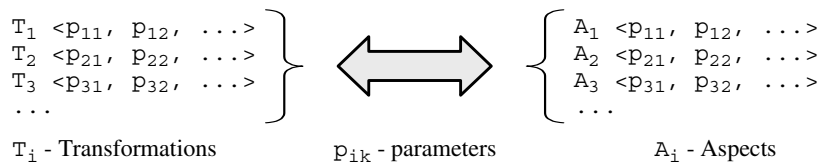

Fig. 2. Concrete Example of the Association Between Model Transformations and Aspects

Each generic transformation may define a set of pre- and postconditions. A configuration of a generic transformation not only specializes the transformation, but also specializes these conditions. Specialized preconditions are used to check whether the initial state of the model allows the application of the specialized/concrete transformation; specialized postconditions are used to check the consistency and integrity of the obtained model. Both pre- and postconditions can be expressed in a dedicated constraint language appropriate for the models (in the case of the Unified Modeling Language [10], the Object Constraint Language [11] is the obvious choice).

We do not address in this position paper how the cross-cutting concerns are modeled using UML, or how several concerns can be composed. Interesting papers on these subjects can be found at [12] and [13], the first two workshops on "Aspect-Oriented Modeling with UML". We do not discuss either how the components that will finally be implemented should be "shipped". Should we ship only the last, most specialized model, together with the implementation, or should we ship all the intermediate models, together with the transformations and the set of parameters that specialize each transformation? How should a developer make reuse of the models, transformations, and aspects that were used to implement a component?

3 Required Infrastructure

The refinement process based on generic concern-oriented model transformations should be supported by a dedicated tool infrastructure with the following facilities:

- Concern-oriented wizards for configuring the generic model transformations along a concern-dimension.
- Support for generic model transformations as described in [14], together with support for testing pre- and postconditions associated with model transformations.
- Aspect generator plug-ins for specific technology platforms and programming languages (in addition to code generator plug-ins) in order to generate concrete aspects from concrete model transformations.
- Version management capabilities for the model repository. An Undo/Redo facility for model transformations would also be appreciated.
- Visual tools capable of demarcating model parts that have been added to the model through different specialized/concrete transformations by using different colors. An association list between these colors and the concerns that have already been covered would be helpful in order to see what concerns have introduced what elements. And, why not, a list of the remaining concerns would give the developer an idea of what further refinements s/he needs to perform.
- Support for importing/exporting models in XMI [15] format.
- Guidance in the refinement process. A workflow model could track the refinement of a PIM or PSM through transformations. The workflow model could define which generic transformations can be applied at a certain refinement step, and therefore could determine the allowed sequence of transformations.

4 Conclusions

In this position paper we explained how generic concern-oriented model transformations can meet aspect-oriented programming in order to complete the life-cycle of software application development in a pure MDA-compliant way based on separation of concerns. We also discussed some challenging issues related to a possible tool infrastructure that would support such an approach.

References

[1] Valéry, P.: *Cahiers*. Vol. Tome 1. Gallimard (collection “La Pléiade”), Paris, 1975.

[2] Miller, J.; Mukerji, J.: *Model Driven Architecture (MDA)*. Object Management Group, Draft Specification ormsc/2001-07-01, July 9, 2001.

[3] Object Management Group, Inc.: *Model Driven Architecture*. http://www.omg.org/mda.

[4] Tarr, P.; Ossher, H.; Harrison, W.; Sutton, S. M. Jr.: *N Degrees of Separation: Multi-Dimensional Separation of Concerns*. Proceedings of the International Conference of Software Engineering, Los Angeles, CA, USA, May 16-22, 1999. ACM 1999, pp. 107 ─ 119.

[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 European Conference on Object-Oriented Programming, ECOOP, Budapest, Hungary, June 18-22, 2001. LNCS Vol. **2072**, Springer-Verlag, 2001, pp. 327 ─ 353.

[7] Rouvellou, I.; Sutton, S. M. Jr.; Tai, S.: *Multidimensional Separation of Concerns in Middleware*. Workshop on Multidimensional Separation of Concerns in Software Engineering, at the International Conference on Software Engineering, Limerick, Ireland, June 4-11, 2000.
http://www.research.ibm.com/hyperspace/workshops/icse2000/.

[8] Kienzle, J.; Guerraoui, R.: *AOP: Does it Make Sense? The Case of Concurrency and Failures*. Proceedings of the 16th European Conference on Object-Oriented Programming, ECOOP, University of Málaga, Spain, June 10-14, 2002. LNCS Vol. **2374**, Springer-Verlag, 2002, pp. 37 ─ 61.

[9] OOPSLA 2003: *Domain-Driven Development* track.
http://www.oopsla.org/oopsla2003/call_3d.shtml.

[10] Object Management Group, Inc.: *Unified Modeling Language Specification*, v1.4, September 2001.

[11] Warmer, J.; Kleppe, A.: *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1998.

[12] First International Workshop on *Aspect-Oriented Modeling with UML*, in conjunction with the 1st International Conference on Aspect-Oriented Software Development, Enschede, The Netherlands, April 22-26, 2002.
http://lgl.epfl.ch/workshops/aosd-uml/index.html.

[13] Second International Workshop on *Aspect-Oriented Modeling with UML*, in conjunction with the 5th International Conference on the Unified Modeling Language - the Language and its Applications, Dresden, Germany, September 30 - October 4, 2002.
http://lgl.epfl.ch/workshops/uml2002/index.html.

[14] Kovse, J.: *Generic Model-to-Model Transformations in MDA: Why and How?*. Workshop on Generative Techniques in the context of Model-Driven Architecture, at the 17th Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA, Seattle, WA, USA, November 4-8, 2002.
http://www.softmetaware.com/oopsla2002/mda-workshop.html.

[15] Object Management Group, Inc.: *XML Metadata Interchange (XMI) Specification*, v1.2, January 2002.

[16] Elrad, T.; Aksits, M.; Kiczales, G.; Lieberherr, K.; Ossher, H.: *Discussing Aspects of AOP*. Communications of the ACM **44**(10), October 2001, pp. 33 – 38.

[17] Mili, H.; Mcheick, H.; Sadou, S.: *CorbaViews – Distributing Objects that Support Several Functional Aspects*. Journal of Object Technology, **1**(3), Special Issue: TOOLS USA 2002 Proceedings, pp. 207 – 229.

[18] Kovse, J.; Härder, T.: *Generic XMI-based UML Model Transformations*. Proceedings of the 8th International Conference on Object-Oriented Information Systems, OOIS, Montpellier, France, September 2-5, 2002. LNCS Vol. **2425**, Springer-Verlag, 2002, pp. 192 – 198.

[19] Czarnecki, K.; Eisenecker, U. W.: *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[20] Mellor, S.; Balcer, M. J.: *Executable UML: A Foundation for Model Driven Architecture*. Addison-Wesley, 2002.