Navigating the MetaMuddle

Arnor Solberg, SINTEF/University of Oslo, Norway
Robert France, Raghu Reddy, Colorado State University, USA

Abstract

Developers of model transformations and other model-driven development (MDD) mechanisms that manipulate models often work at the metamodel level. The complexity of the UML metamodel can hinder the development of UML-based MDD technologies. In this paper, we identify some of the current barriers to understanding, using and evolving the UML 2.0 metamodel and present ideas on how to simplify these tasks using query based views and aspect oriented techniques.

Keywords: *Model driven development, metamodels, aspects, separation of concerns, UML.*

1 Introduction

Model driven development (MDD) aims to reduce complexity in software development through modularization and abstraction. Accomplishing abstraction entails developing support for modeling concepts at different levels of abstraction and transforming abstract models to more concrete descriptions of software. Accomplishing modularization entails developing mechanisms for separating concerns in software descriptions.

The model driven architecture (MDA) [1][2] initiative of the OMG proposes a concrete architecture for MDD. The core elements of this architecture are the Unified Modeling Language (UML) [3][4], Meta Object Facility (MOF) [5] and Common Warehouse Model (CWM) [6] metamodels. The MDA core is meant to facilitate tasks such as system modeling, model transformation, separation of concerns and model composition.

Developers of MDA technologies that manipulate UML models often work at the metamodel level. For example, model composition procedures in aspect-oriented modeling (AOM) [7] and model transformations are often expressed in metamodel terms. Furthermore, MDA technologies sometimes extend metamodels and define new language constructs to better support modeling activities in particular domains or product families. Developers of MDA technologies must therefore have a good understanding of the metamodels they are working with.

The complexity of the current UML 2.0 metamodel can make understanding, using, extending, and evolving the metamodel difficult. In this paper we identify some of the problems that contribute to the difficulty of developing MDA technologies that are based on the UML metamodel. Specifically, we summarize our experience with using the UML 2.0 metamodel in model transformation and composition development tasks. We then present our ideas on how aspect-oriented modeling techniques and view extraction mechanisms can be used to manage the complexity of understanding, using and extending metamodels.

2 Background

In this section we provide some background on the model transformation and aspect oriented modeling work that led to the identification of problems related to the use of the UML metamodel.

2.1 Model Transformation

A model-transformation can be viewed as a transformation between two model spaces defined by their respective metamodels. A source-model to target-model transformation specification describes how elements in the source-model space should appear in the target-model space by relating metamodel elements in the source and target metamodels. A metamodel describing transformation specification constructs can also be defined to

support the development of transformation specifications. The relationships among transformation concepts are illustrated in Figure 1.

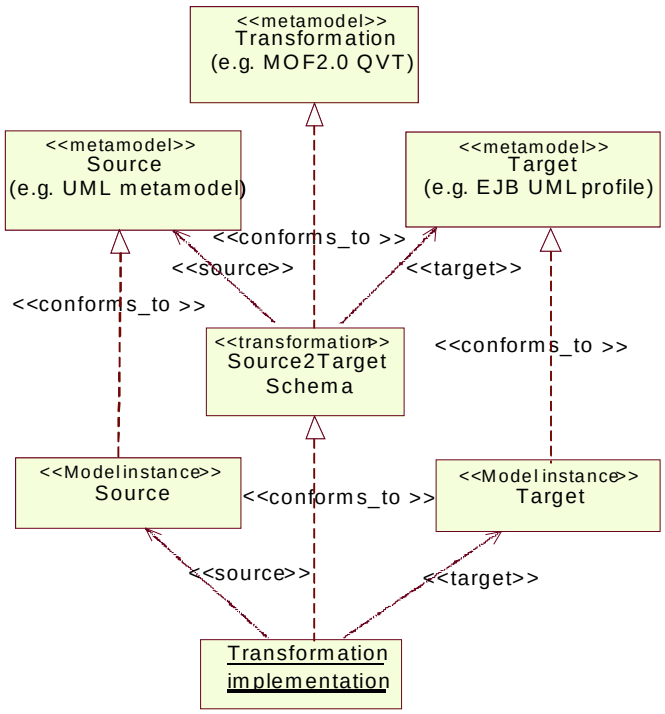

Figure 1: Conceptual transformation model

The source model instance conforms to the source metamodel (for instance the UML metamodel). The *Transformation implementation* object transforms a source model instance to the corresponding target model instance. The target model instance conforms to the target metamodel (for instance the UML profile for CORBA [8]). The specific transformation implementation for a specific source and target model is generated using the *Source2Target Schema*. This schema is the transformation specification, which maps source metamodel concepts to target metamodel concepts. The *Source2Target Schema* must conform to a transformation metamodel (e.g., the MOF 2.0 QVT specification [9]). This view of model transformations requires one to use metamodels when defining and supporting transformations.

2.2 Aspect Oriented Modeling

The Aspect-Oriented Modeling and Development Framework (AOMDF) that we are developing at Colorado State University is based on an aspect oriented modeling approach in which a design is expressed in terms of the following artifacts [10][11]:

- A *primary model* that describes core application functionality.
- A set of *generic aspect models*, where each model is a generic description of a crosscutting feature.
- A set of bindings that determines where in the primary model the aspect models are to be incorporated.
- A set of composition directives that influences how aspect models are composed with the primary model.

Before an aspect model can be composed with a primary model, the aspect model must be instantiated. An instantiation is obtained by binding elements in the aspect model to elements in the application domain. The result is called a *context-specific aspect model*. Context-specific aspect models and the primary model are composed to obtain an integrated design view [7]. Composition of models in the AOM approach is defined in

terms of a metamodel that extends the UML metamodel with operations to support merging of model elements [12].

3 Problems using the UML Metamodel

UML 2.0 is a large and complex language (i.e., it consists of a large number of conceptual units that are related in a variety of ways) and thus it is not surprising that its metamodel is complex. It will be extremely difficult to evolve the UML 2.0 metamodel to reflect changes in the UML standard using only manual techniques. How can one be sure that required changes are incorporated consistently across the metamodel? How can one determine the impact that a change will have on other metamodel elements? In particular, how can one ensure that the changes do not result in a metamodel that defines inconsistent or nonsensical language constructs?

In practice, only a small subset of the diagram types provided by the UML is used. Furthermore, not all of the concepts available in a diagram type are used for modeling an application or a product family. For this reason, one seldom needs to have full knowledge of the UML metamodel to specify model transformations or model composition – one can create transformation and composition mechanisms that use only a subset of the concepts in the UML metamodel. Unlikely as it seems, this raises a problem: The task of identifying and extracting the required subset of concepts from the UML metamodel must currently be performed manually. In other words, one needs to manually navigate through the meta-"muddle" in order to identify and extract the needed concepts.

As an illustration of the above problem, consider the task of extracting from the UML metamodel a simple view of UML 2 interactions models that focuses only on the concepts that are reflected in simple sequence diagram (i.e., those without fragments). Using the UML specification document as baseline for extraction is one possibility; another baseline may be the UML 2 superstructure metamodel that is available at the OMG's site as a Rational Rose file[1]. An examination of the Interactions section in the UML specification [3], reveals that the information required in the simple view is not available in one place in the metamodel.

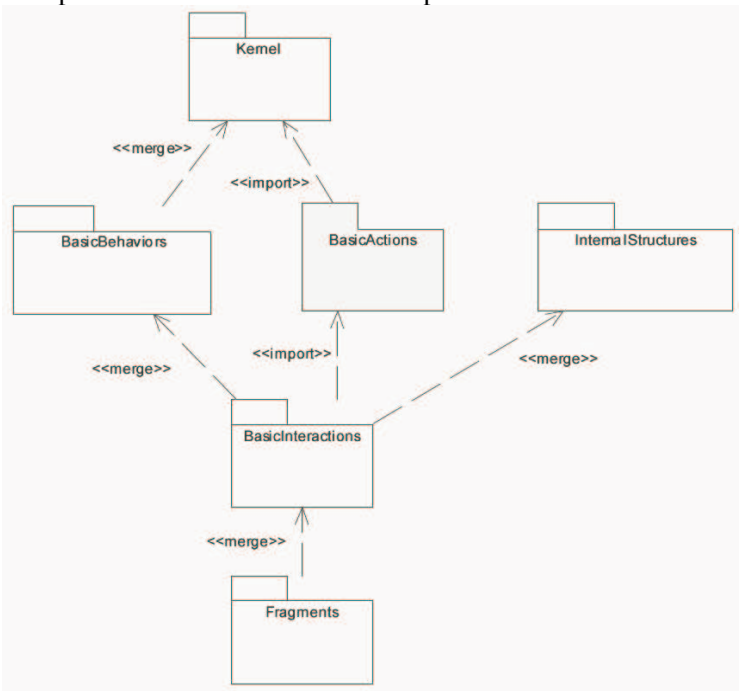

Figure 2: Dependencies of the BasicInteractions package (from the UML 2.0 specification)

[1] The Rational Rose model has very little documentation of the metamodel concepts though.

The dependency diagram for the BasicInteractions package is shown in Figure 2. Descriptions of the concepts contained in these different packages are found elsewhere in the 1000+ page metamodel specification. The specification does not present a convenient overview of the main concepts of interactions and their relationships. For example, one would expect that the relationship between message ends and lifelines would be easy to identify. Surprisingly, deriving this simple relationship requires one to navigate through a number of associations that are spread across the metamodel fragments and packages. The overview diagram of Lifeline is shown in Figure 3 and the overview diagram of Message is shown in Figure 4. Note that there are no direct references to message ends in the Lifelines fragment, and no direct references to lifelines in the Message fragment.

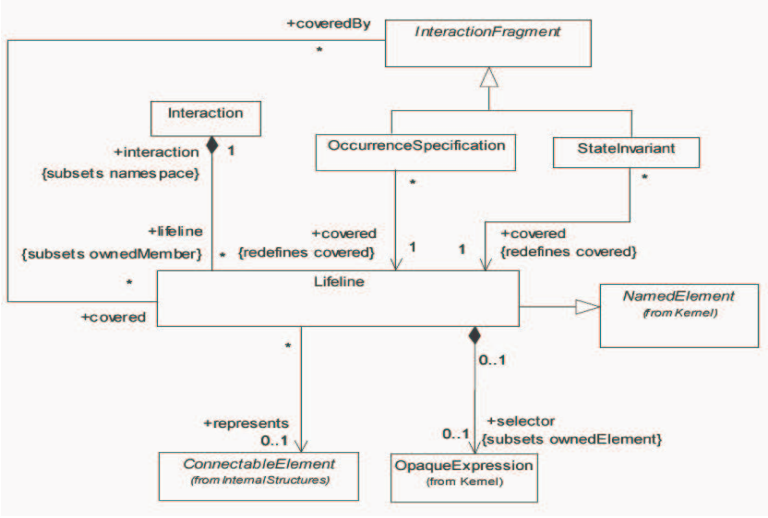

Figure 3: Lifelines (from the UML 2.0 specification)

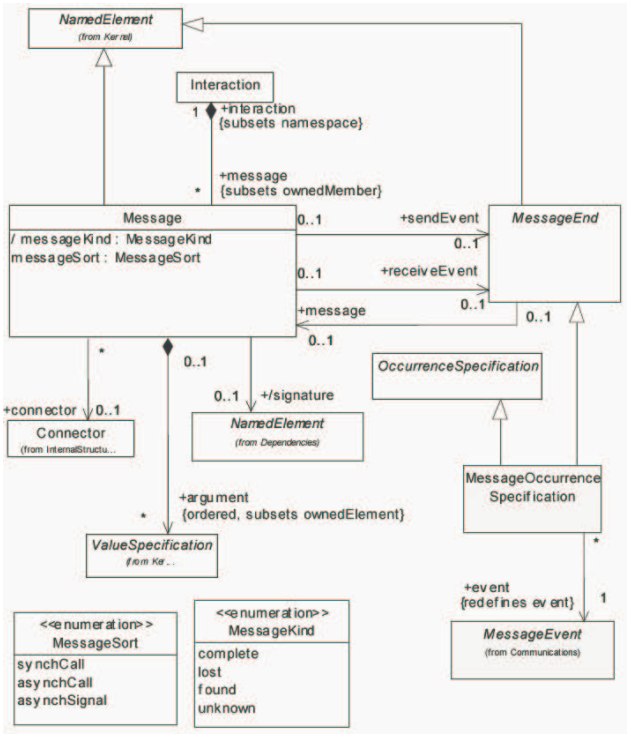

Figure 4: Messages (from the UML 2.0 specification)

Deriving the relationships between Lifeline and Message is time consuming and tedious. A snapshot of the work involved in finding the simple Interaction metamodel is shown in Figure 5.

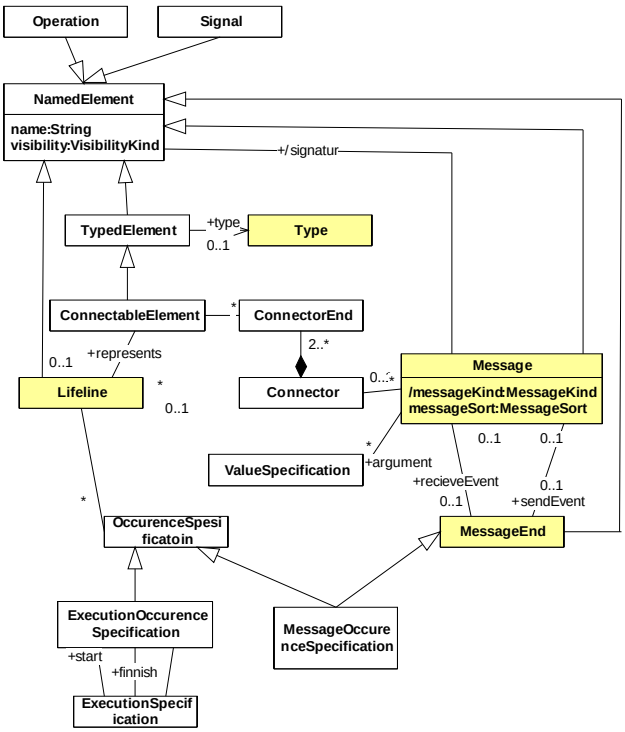

Figure 5: Finding the *simple* UML interactions metamodel.

The Interaction concepts of interest are shown in yellow (grey for non-color printouts). The other concepts are those that were navigated through to derive the required relationships. In some cases finding the derived properties requires one to move through inheritance hierarchies to find the needed concepts. For example, Lifeline inherits from Named Element, but none of the properties are included in diagrams or in the textual concept description part of the Interaction section. NamedElement again inherits properties from Element, for example, the /ownedElement association.

4 Supporting Effective use of the UML Metamodel

The example in the previous section gives some insight into why the task of defining transformations and composition procedures using the UML metamodel is currently tedious and error-prone. One way of alleviating the problems is to provide metamodel views of all diagram types provided by UML that describe only the concepts and relationships that appear in the diagrams. For example, Figure 6 shows a metamodel view that describes simple class diagrams. Similar diagrams for sequence, statemachine, activity and other UML diagrams can be derived. We have found that manual derivation of these views is tedious.

A more flexible and useful approach is to provide tools that allow developers to query the metamodel and to extract specified views from the metamodel. Query/Extraction tools should be capable of extracting simple derived relationships between concepts and more complex views that consist of derived relationships among many concepts. Metamodel users can use such tools to better understand the UML metamodel, and to obtain views that can be used in the specification of patterns, transformations and composition procedures. Users that need to extend or evolve the UML metamodel can use such tools to help determine the impact of changes (e.g., a

query that returns a view consisting of all classes directly or indirectly related to a concept in the metamodel can provide useful information) and to check that changes are consistently made across the metamodel.

The development of such tools is not beyond current technology. Current UML model development tools have some support for manipulating the UML metamodel that can be extended with query and extraction capabilities. Metamodeling tools, such as those developed by Xactium (see [13]) and Adaptive Software (see [14]) and the megamodelling tools advocated by Jean Bezivin [15] [16] have some of these capabilities and it is expected that these capabilities will improve to the point that they are easy to use and integrate with transformation and composition technologies.

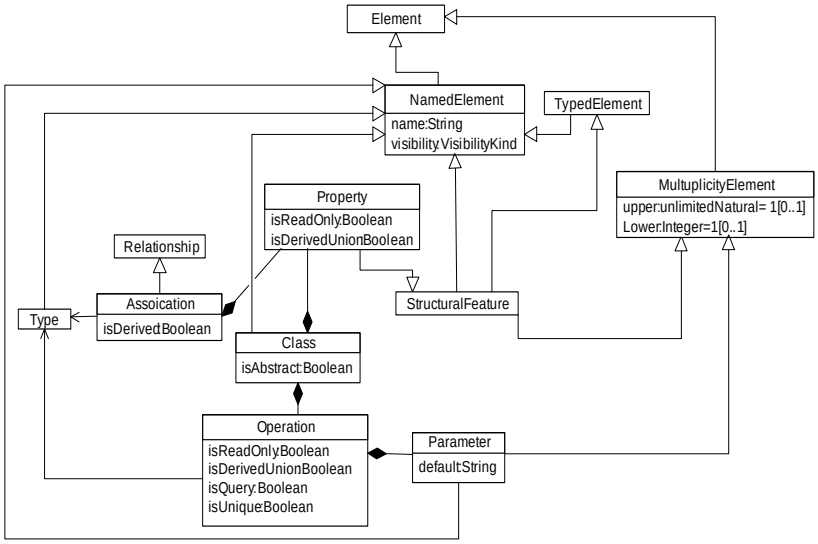

Figure 6: Simple metamodel view for class diagram

Another useful tool that can ease the task of using the UML metamodel is one that takes a UML model and produces a metamodel view that describes its structure. Such a tool can be used to support compliance checking of models manipulated by transformations and composition procedures. For example, the tool can be used to check that source, target and transformation models conform to their respective metamodels.

As pointed out earlier, evolving the complex UML metamodel will be a challenge. It is expected that the UML will change and thus the maintainers of the standard will be faced with the tasks of assessing the impact of suggested changes, making changes consistently across the metamodel, and verifying the consistency and soundness of the changed metamodel. Performing these tasks manually will be tedious and error-prone. To better manage the evolution of the UML metamodel we propose the use of aspect-oriented modeling (AOM) techniques. We suggest organizing the UML metamodel into modeling views called aspects. For example, one can define the following aspects:

- Aspects presenting views for each of the diagram types in the UML that contain only the concepts that are visualized in diagrams (e.g., abstract concepts such as NamedElement will not appear in these views).
- Aspects presenting views of abstract concepts reflecting language and UML-specific concerns such as name space management, element typing, connectivity of elements, and execution semantics.

Each aspect (view) presents an uncluttered view of how a concern is treated in the UML metamodel. Composition mechanisms are needed to compose the views in order to fully understand the relationships among non-orthogonal views. Aspect treatment of the metamodel allows one to make changes in a single aspect, or create a new aspect, and then compose it with other aspects to determine the impact that change has on the metamodel. Use of a composition mechanism will also help ensure that changes are consistently made across the

metamodel. We have developed a composition mechanism for UML class diagrams that can be used for this purpose [17].

5 Conclusion

The intent of this paper is not simply to critique the UML metamodel, but also to propose some approaches that can help manage the complexity of using and evolving the metamodel. Some of the problems we identified may be the target of ongoing research that we are not aware of – in which case, this paper serves to reiterate the need for such research. The approaches we propose are implementable using current technology. We plan to apply our AOM approach to the UML metamodel to determine whether the anticipated benefits will be realized.

References

[1] OMG MDA™ Guide v1.0.1, Object Management Group, http://www.omg.org/docs/omg/03-06-01.pdf.

[2] Soley, R.M, Frankel, D.S.,Mukerji, J.,Castain, E.H., Model Driven Architecture - The Architecture Of Choice For A Changing World, OMG 2001. http://www.omg.org/mda.

[3] The Object Management Group. Unified Modeling Language: Superstructure. Version 2.0, OMG document ptc/04-10-02, 2004.

[4] The Object Management Group. Unified Modeling Language: Infrastructure. Version 2.0 Final Adopted Specification. OMG document ptc/03-09-15, 2003.

[5] Meta-Object Facility (MOF™), OMG adopted specification ptc/03-10-04. Version 2.0, www.omg.org

[6] Common Warehouse Metamodel™, v1.1, OMG Document:formal/2003-02-03.

[7] R. B. France, I. Ray, G. Georg, and S. Ghosh. An aspect-oriented approach to design modeling. IEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, 151(4), August, 2004.

[8] UML™ Profile for CORBA™ version 1.0, April 2002, formal/02-04-01.

[9] Revised submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10), QVT-Merge Group 1.8, OMG document ad/2004-10-04.

[10] Devon Simmonds, Arnor Solberg, Raghu Reddy, Robert France, Sudipto Ghosh "An Aspect Oriented Model Driven Framework", Accepted to Ninth IEEE "The Enterprise Computing Conference" (EDOC 2005), Enschede, Netherlands, 19-23 September, 2005.

[11] Arnor Solberg, Devon Simmonds, Raghu Reddy, Sudipto Ghosh, Robert France, "Using Aspect Oriented Technologies to Support Separation of Concerns in Model Driven Development", Accepted in the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005), Edinburgh, Scotland, July, 2005.

[12] G. Straw, S. Ghosh, R. B. France, J. M. Bieman, G. Georg, E. Song, and N. McEachen. "Directives for Composing Aspect Models", Technical report, Colorado State University, 2005.

[13] Language Driven Development and XMF-Mosaic, Xactium Limited, Whitepaper, 2005 www.xactium.com.

[14] Adaptive Inc, http://www.adaptive.com/.

[15] Bézivin, J, Jouault, F, and Valduriez, P : On the Need for Megamodels. In: Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications.

[16] The ATL homepage, http://www.sciences.univ-nantes.fr/lina/atl/contrib/bezivin.

[17] Raghu Reddy, Robert France, Sudipto Ghosh, Franck Fleurey, Benoit Baudry, “Model Composition – A Signature-Based Approach”, Submitted to the Aspect Oriented Modeling workshop held in conjuction with MODELS/UML 2005.