How a Relaxation of the Strictness Definition Can Benefit MDD Approaches With Meta Model Hierarchies

Ralf GITZEL
Business Information Systems, University of Mannheim
68131 Mannheim, Baden-Württemberg, Germany

and

Matthias MERZ
IT-Center, University of Mannheim
68131 Mannheim, Baden-Württemberg, Germany

ABSTRACT

Meta modeling is an essential aspect of many MDD approaches. However, there are many different views on how such aspects as instantiation semantics, choice of model elements, or causal connections of layers should be handled. One common school of thought proposes the concept of strict meta modeling which provides more structure to instantiation and avoids "illogical" scenarios such as meta classes instantiating themselves. In this paper, we propose a relaxation to the strictness definition and describe the advantages we see in this for the automatic generation of code. As a proof of concept we present a small implementation in Java/XSLT which converts a simple relaxed model to Java code with JDO persistence.

Keywords: Model-Driven Development, Meta Modeling, Strictness, MDD, MOF, XMI, Code Generation, JDO

1. INTRODUCTION

Meta modeling is a mature and established concept in the world of software engineering (see [3] or [13] for a detailed discussion). It is also an essential aspect of many MDD approaches such as the one proposed by Atkinson and Kühne [5] or by Nordstorm [11] and is the empowering factor behind the often-cited goal of turning the UML into a "family of languages" [9].

A new and interesting way to look at meta modeling is to distinguish between ontological and linguistic meta modeling axes as described by Atkinson and Kühne [7]. Figure 1 illustrates this idea. The linguistic meta model (L1) has a single instance layer (L0). The classes L1 contains have names such as *Model Layer*, *Meta Class*, and *Class*. There is also a special association called *InstanceOf*. At this point, it is important to note that there is a difference between *Instance_of* and *InstanceOf* in the figure. InstanceOf is the arbitrary name of an association while Instance_of shows the relationship between the meta model elements in L1 and the model elements in L0 which are their instances. The elements of the meta model allow the creation of model hierarchies in the ontological direction (see O3, O2, and O1 in the figure). An example of an instantiation is marked bold in the figure. Using the modeling language defined by L1, an L0 instance of the L1 class called *Meta Class* can be connected with an instance of the class *Class* via a link which is an instance of *InstanceOf.*

When ignoring the box L1 in Figure 1 and turning L0 90° clockwise, a new (ontological) meta model hierarchy becomes plainly visible. There is not much mental transfer involved in now interpreting the InstanceOf associations connecting the elements of O2 with those of O1 as a process of instantiation. With the separation of ontological and linguistic aspects, it becomes easier to build (ontological) meta model hierarchies that are far larger than the typical four layer architecture (as found in CDIF [8] or MOF [12]), for

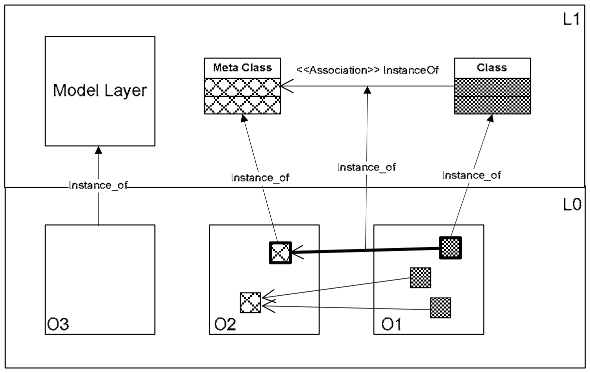

Figure 1 - Core Principle of Non-Linear Meta Modeling

example to separate different areas of concern. However, larger hierarchies also introduce new problems and opportunities. One aspect that requires a new critical evaluation is the concept of strictness.

The concept of strict meta modeling is commonly accepted, because it provides structure to instantiation and avoids "illogical" scenarios such as meta classes instantiating themselves. In this paper, we propose a relaxation to the strictness definition and describe the inherent advantages for the automatic generation of code (see section 3). As a proof of concept we present a small implementation in Java/XSLT in section 4 which converts a simple relaxed model to Java code with JDO persistence. We conclude our paper with a short review of related work and a summary of the advantages of a strictness relaxation.

2. STRICTNESS

Strictness is a concept that provides the structure of the model layers in a meta model hierarchy. It is often taken for granted (probably without actually knowing the term), particularly when using architectures such as MOF where it is an integral aspect. Atkinson and Kühne define strictness for meta modeling in the following way.

> In an n-level modeling architecture, M_0, M_1, ..., M_{n-1}, every element of an M_m-level model must be an *instance-of* exactly one element of an M_{m+1}-level model, for all $0 \leq m < n-1$, and any relationship other than the *instance-of* relationship between two elements X and Y implies that level(X) = level(Y) [6].

This definition states that model elements should generally only have relationships within their own layer and not between layers, with the exception of the instance-of relationship, which represents the connection between the different layers. Another restriction introduced by this definition is that an element in a model layer can only instantiate elements of its immediate parent layer. While the limitation on relationships and its implications are common sense in our opinion (whereas Atkinson expresses a different view in [6]), the statements made with regard to instantiation are less intuitive and require some thought.

It should be mentioned that, while the definition above includes a layer M0, M0 is usually interpreted as the user data layer and therefore the following examples all start at M1 or higher. However, this really is inconsequential for the discussion.

Figure 2 shows some examples where the strictness definition is violated with regard to instantiation limitations. The instantiation labeled Case 1 is part of a situation were a class is an instance of itself. While at a first glance, the definition seems to forbid such a design, it is important to realize that it actually allows self-instantiation – but only at the topmost level n, which is useful for recursive definitions of the highest layer as used in MOF [11]. On all other layers and therefore in the example, self-instantiation is not allowed. Case 2 is an instantiation in the wrong direction, which makes little sense and is therefore also forbidden by the definition.

3. RELAXED DEFINITION

The instantiation in Figure 2 called Case 3 on the other hand requires some careful attention. It performs the instantiation of a model element which is in a higher layer but not in the layer directly above the instance level. Again, the definition forbids this, even though at the first glance, there is little reason why this should be so. To analyze the matter further, the following variant definition, a *relaxation* of the original strictness definition as provided above, is proposed:

> In an n-level modeling architecture, M_0, M_1, ..., M_{n-1}, every element of an M_m-level model must be an *instance-of* exactly one element of an *Mo-level model*, for all $0 \leq m < n-1$, $m < o \leq n-1$, and any relationship other than the *instance-of* relationship between two elements X and Y implies that level(X) = level(Y).

A similar relaxation, although with a slightly different motivation, is proposed by Álvarez et al. The need for

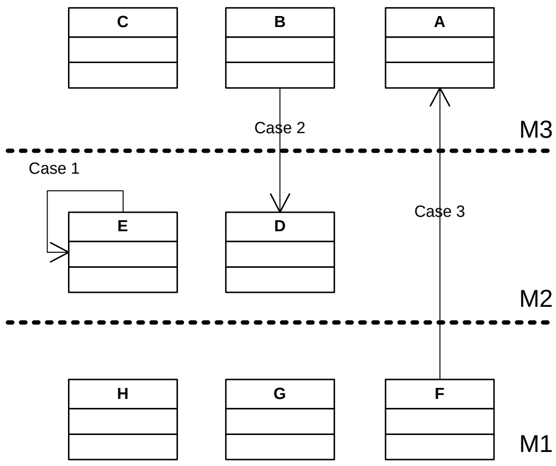

Figure 2 - Violations of Strictness

the relaxation arises from the use of nested layers – effectively, a layer M_n is a linguistic model for layers M_{n-1} and M_{n-2} [1]. If one subscribes to the view that ontological and linguistic meta modeling are completely orthogonal, no violation of strictness takes place. Thus, the arguments given in their paper are different from the ones found in this section and actually address a different problem. Albin gives an example for a meta model hierarchy in his book, where a relaxed hierarchy is implied (see [1], chapter 11, as well as the example below). However, the text never explicitly mentions strictness and is generally vague with regard to the meta modeling semantics used.

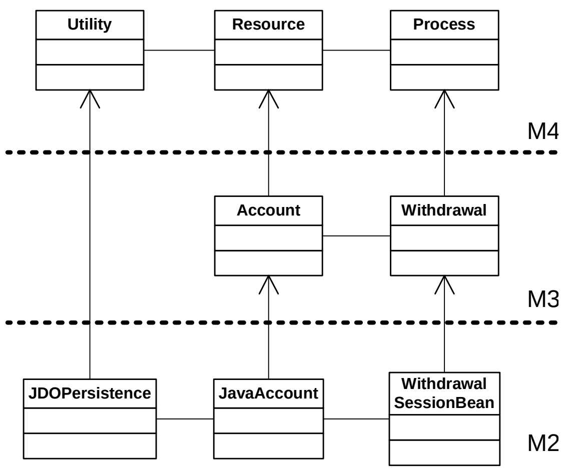

Figure 3 - A Scenario With Relaxed Strictness

Especially when using a model hierarchy for code generation, the ability to instantiate a model element on any of the lower layers can be quite useful, particularly to avoid *replication of concepts* as defined by Atkinson and Kühne [4]. For example, Figure 3 shows a meta model hierarchy with 3 layers. The top layer (called ontology layer) describes a business ontology as used by Albin ([1], chapter 11) which is based on the Convergent Architecture architectural model proposed by Hubert [10]. There is a meta class for processes and a meta class for resources, which can be used to model domain specific meta models such as the banking domain described by the layer labeled M3 in the figure. There is also a class called Utility in the M4 layer which describes all kinds of elements which support the processes and resources. The lowest layer in this scenario is a technical model. Its classes describe the technologies used to realize the domain concepts, e.g. the JavaAccount class denotes that the concept of an account should be implemented in Java (not necessarily as a single class though).

A JDO persistence mechanism (as seen in layer M2 of the figure) is a utility in the sense of the ontology layer but the Utility meta class is defined in the M4 layer. One solution is to replicate Utility in the M3 layer (e.g. by having a meta class called Utility which instantiates M4's Utility class). However, this leads to a replication of concepts. Another alternative is to define a class Persistence in the M3 layer (see Figure 4). At first, this seems to be a good solution (solving the replication issue) but is not really consistent with the interpretation of M3 as a domain specific layer.

The example above leads to two conclusions. First of all, there are scenarios where a relaxed strictness definition can be useful. Second, while there is often a technical way to avoid the need for the relaxation (i.e. by introducing a class in the middle layer(s)), this can lead to problems with regard to the semantics of the model layers, especially in the case of domain specific modeling.

Still within the context of the example above, one could argue that the need for a relaxed strictness

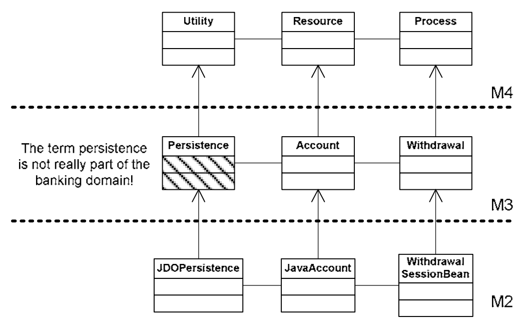

Figure 4 - Introducing a Persistence Meta Class in M2

definition is the result of poor modeling, i.e. that a more suitable hierarchy would not require such drastic changes to the basic concepts of meta modeling. However, in a meta model hierarchy, especially one with domain specific layers, it is sometimes not possible to access all layers for changes, for example, the M3 layer in Figure 3 serves as a basis for all its M2 layers and changes to this meta model negate many of the advantages of a meta model hierarchy.

There are, however, those who argue strongly in favor of strictness. Atkinson, for example, claims that without strictness, the meta model hierarchy would eventually collapse into a single super layer containing all model information with the layers becoming little more than packages [3], especially when associations other than instance-of cross layer boundaries [6]. However, the proposed relaxation does not affect associations other than instance-of and is therefore not affected by this problem.

Another argument in favor of strictness given by Atkinson and Kühne is that without it, it is impossible to understand a model completely within the context of its meta model without the influence of higher layers [6]. While the limitation to a single higher layer is very helpful in the context of linguistic meta models, it is not needed for ontological hierarchies, in fact, such a requirement goes contrary to the whole idea of dividing the model information over several layers.

To sum it up, there are valid arguments for strictness and some form of it should be included in most meta model hierarchies. However, there are also good arguments for a slight relaxation in the context of ontological meta modeling. The question remaining now is if there are also benefits for the code generation process.

4. CODE GENERATION SAMPLE IMPLEMENTATION

In the previous sections, we have described a possible relaxation of the strictness definition and made the claim that it will be beneficial to code generation. As a proof of concept, we have implemented a small prototypical code generation script, which reads in an XML file containing the model information described in Figure 6. Please note that the example model is deliberately vague (e.g. many attributes which might be useful configuration parameters have been omitted) due to space limitations. Still, it should serve as an indication that more complex models can be created with the same relaxation premise.

Our example centers around an Ant script, which is

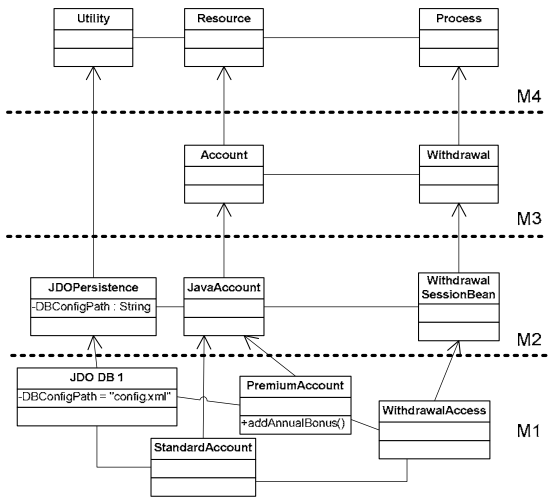

Figure 6 - A Code Generation Example

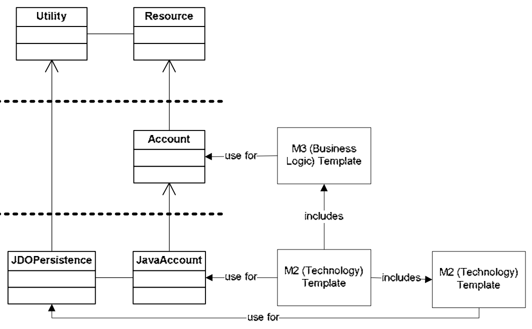

Figure 5 - Templates for Code Generation

used to generate, compile, and deploy the examples. It is augmented by a series of templates which cover both the business logic (M3) and the technological mapping (M2). The templates are written in XSLT and are segmented in such a way as to allow their assembly from elements derived from the different layers. This concept will be explained further after the introduction of an example. Our example templates generate Java classes which contain xdoclet 1.2 tags for the persistence specification. We have tested the generated code with the JBoss 3.2.3 application server, and the JDO-implementation IntelliBo 3.61. The database used was a MySQL-Database 4.0.15 with InnoDB extension. The scenario in Figure 6 describes only one of the test cases we have generated code for. Other examples are cases with non-persistent accounts, several withdrawal beans, and multiple database sources. (The latter case involves some technical problems regarding JDO but these are beyond the scope of this paper.) What all test cases have in common are the model layers M2 to M4, which effectively define a domain specific modeling language, in this case a language for (somewhat simplistic) bank account access software based on J2EE with JDO persistence.

Figure 5 shows the relationship between the models and the templates in detail. Each transitive instance of M4 Resource on the M2 level has a template for code generation. This template "includes" code generation information from the M3 layer, in the case of JavaAccount, this is the business logic template for Account. The M2 template for JavaAccount also includes a template for each relationship it is involved in which provides rules for generating the code in JavaAccount's corresponding class due to the relationship. While this will be a reference in most cases, the situation depicted in Figure 5 is different. JavaAccount has an association with JDOPersistence which is not to be interpreted as a class or component but rather stands for a concept which can be applied to resources. Therefore, the template fragment for

JDOPresistence will not generate a class but rather influence the code generation of all transitive resource instances which are connected to it.
Based on this code generation approach, it would be relatively easy to change the M2 layer with little or no changes to the upper layers, except in the case of major paradigm changes such as switching to an altogether new programming language.
In our opinion, the test cases where necessary for several reasons. First, we have given an example, that the information provided by the model is sufficient for code generation purposes, despite the reduction in model elements due to the relaxation. Also, we have shown that a code generator can handle the "gap" in the M3 layer without problems. While these facts might not be surprising, it is important to note that without such a practical implementation, the examples given in section 3 are of dubious nature at best as it is often subtle details which look fine in a model hierarchy concept but imply major complications in the code generation process.

5. RELATED WORK

Atkinson and Kühne have done some substantial work on meta modeling and have come up with the original definition of strictness (see for example [2], [3], [4], and [6]). Álvarez et al. have proposed a relaxation of the strictness definition similar to ours but for different reasons, in a limited fashion, and only applied to a non-linear meta model hierarchy [2]. Albin has an example with a relaxed meta model hierarchy but never explicitly refers to strictness as a concept [1]. We are not aware of any extensive study with a practical example similar to this paper.

6. CONCLUSION AND FUTURE WORK

In this paper we have proposed a relaxed definition for strictness in the context of meta model hierarchies. We have given arguments for and against such a relaxation and have come to the conclusion that its application within the context of ontological meta modeling is beneficial to automated code generation. Not only does it improve the quality of the input models. Also, given the relationship between layers and templates (see Figure 5), replication of concepts would lead to many unnecessary templates. In the figure, the Utility instantiation skips a layer and therefore for all domains (i.e. M3 layers), only one template is needed. If however, each M3 layer had its own "M2 Utility", each M1 instance would logically need its own template, as two meta classes (one of M2 Utility A and one of M2 Utility B) cannot share one instance. Writing one template for several M1 classes is possible but offers some technical problems, if only the listing of all applicable classes.
We have chosen the meta model hierarchy in sections 3 and 4, because it was proposed by other authors and thus acquits us from any accusations that the example model has been constructed to fit the idea of relaxation rather than the other way round. However, during the code generation process, it became obvious that other hierarchies are probably better suited for code generation, as the introduction of technology on the lowest layer leads to a lot of replication in the templates, i.e. each concept realized as a java session bean would need the same basic structure but the concepts share no common meta class which defines the shared technological properties.
Other examples where relaxed strictness can come in handy is for concepts which occur on all layers of a model such as instances of primitive data types (e.g. String or Integer).
This discussion of strictness came up in the context of a linguistic meta model, we are currently working on. In the future, we plan to present a fully fledged linguistic meta model for ontological meta modeling which uses a relaxed definition of strictness.

7. REFERENCES

[1] Albin, S. (2003): The Art of Software Architecture: Design Methods and Techniques, John Wiley & Sons, March 2003, online version

[2] Álvarez, J., Evans, A., and Sammut, P. (2001): Mapping between Levels in the Metamodel Architecture. In: Gogolla, M. and Kobryn, C. (Editors): UML 2001, Lecture Notes in Computer Science 2185, Springer Verlag, New York, pp. 34-46, 2001

[3] Atkinson, C. (1997): Meta-Modeling for Distributed Object Environments, In: Proceedings of the 1st International Enterprise Distributed Object Computing Conference (EDOC'97), IEEE Computer Society 1997, pp. 90-101

[4] Atkinson, C. and Kühne, T. (2001): The Essence of Multilevel Metamodeling. In: Gogolla, M. and Kobryn, C. (Editors): UML 2001, Lecture Notes in Computer Science 2185, Springer Verlag, New York, pp. 19-33

[5] Atkinson, C. and Kühne, T. (2002): The Role of Metamodeling in MDA. International Workshop in Software Model Engineering (in conjunction with UML '02), Dresden, Germany, October 2002. URL: http://www.metamodel.com/wisme-2002/

[6] Atkinson, C. and Kühne, T. (2002): Rearchitecting the UML Infrastructure. In: ACM Transactions on Modeling and Computer Simulation, Vol. 12, No. 4, October 2002, pp. 290-321

[7] Atkinson, C. and Kühne, T. (2003): Model-Driven Development: A Metamodeling Foundation. In: IEEE Software, September/October 2003 (Vol. 20, No. 5), IEEE, pp. 36-41

[8] CDIF Technical Committee: „CDIF – CASE Data Interchange Format, Extract of Interim Standard", EIA/IS-107, Electronic Industries Association, Januray 1994

[9] Duddy, K. (2002): UML2 Must Enable a Family of Languages. In: Communications of the ACM, November 2002, Vol. 45, No. 11, pg. 73-75

[10] Hubert, R. (2002): Convergent Architecture – Building Model-Driven J2EE Systems with UML, John Wiley & Sons, 2002

[11] Nordstrom, G., Sztipanovits, J., Karsai, G., and Ledeczi, A. (1999): Metamodeling – Rapid Design and Evolution of Domain-Specific Modeling Environments, In: Proceedings of the 6th Symposium on Engineering of Computer-Based Systems (ECBS '99), IEEE Computer Society, 1999, pp. 68-74

[12] OMG: Meta Object Facility (MOF) Specification, Version 1.4. OMG, April 2002, http://www.omg.org/cgi-bin/doc?formal/2002-04-03

[13] Völter, M.: Metamodellierung, http://www.voelter.de/services/mdsd.html