The Role of Metamodeling in MDA

Colin Atkinson
colin.atkinson@ieee.org

Thomas Kühne
Darmstadt University of Technology
64283 Darmstadt, Germany
kuehne@informatik.tu-darmstadt.de

Accepted at International Workshop in Software Model Engineering (in conjunction with UML '02), Dresden, Germany, October 2002

ABSTRACT
There is general agreement that metamodeling will play a pivotal role in the realization of the MDA, but less consensus on what the precise role of metamodeling should be and what form it should take. In this paper we first analyze the underlying motivation for metamodeling within the context of the MDA and derive a concrete set of requirements that an MDA supporting infrastructure should satisfy. We then present a number of concepts, which we believe are best suited to providing technical solutions to the identified requirements. In particular, we discuss why the traditional "language definition" view is insufficient for an optimal MDA foundation.

1 INTRODUCTION

The basic underlying motivation for the MDA [1] is to improve productivity in software development—that is, to increase the return derived from a given amount of effort. It delivers this benefit—

1. by raising the level of abstraction at which primary software artifacts[1] are written.
2. by reducing the rate at which these artifacts become obsolete.

The first goal aims to improve the *short-term* productivity of developers. In the same way that high-level programming languages increased productivity in the 60's and 70's by obviating the need for programmers to write assembler or binary code directly, modeling languages can further increase productivity by making it unnecessary for developers to write programs—i.e., to specify implementations—at all. Instead, models will become the primary artifacts developed by humans, and code—at the level defined by today's programming languages—will be a derived artifact, generated by tools (i.e., model compilers). This will allow developers to focus on describing business logic and problem concepts in as concise and abstract a manner as possible, leaving tools to handle the routine implementation issues.

[1] Primary software artifacts are developed by human engineers and are used to mechanically generate derived artifacts from them.

Increasing *short-term* productivity is only one part of the overall picture, however. It is also important to improve *long-term* productivity by minimizing the sensitivity of primary software artifacts to change. We therefore believe that the second and, in fact, more important goal of the MDA should be to maximize the useful lifetime of key software development assets.

To this end, in the following sections we identify the end user benefits which we believe the MDA should be designed to deliver, and then distill these into a concrete set of requirements for an MDA infrastructure. After criticizing the current approaches for providing such an infrastructure, we discuss what enhancements are needed for the full promise of the MDA to be realized.

2 REQUIREMENTS FOR AN MDA INFRASTRUCTURE

To increase the useful lifetime of primary software artifacts it is necessary to reduce their sensitivity to the inevitable changes that affect a software system. It is impossible to avoid change, but by managing changes appropriately their impact on software development investments can be minimized. Four mains kinds of change in particular need to be carefully managed:

I) **Deployment Platforms**
In today's rapidly changing deployment platform market it seems that as soon as developers have mastered one new platform

technology another comes along to take its place[2]. By documenting core business logic in platform independent artifacts, however, it can be decoupled from the high rate of change in platform technology. The key to making this work is to automate (to the greatest extent possible) the process of obtaining platform specific software artifacts from platform independent ones through the application of *user definable mappings*.

II) **Development Platforms**
If software artifacts are dependent on the particular development environment and tools that created them, their lifetime will be limited by the lifetime of the development platforms. This implies that artifacts should be decoupled from development environments by requiring tools to support *high-levels of interoperability*.

III) **Requirements**
Effectively managing requirements changes has always been a big challenge in software engineering, but has recently been complicated by expecting deployed enterprise applications to be continuously available, even in the presence of significant changes to the business logic. It follows, at a technical level, that in addition to supporting the static addition of new types at development time, deployed systems are expected to support the *dynamic addition of new types at runtime.*

IV) **Personnel**
As long as certain vital development knowledge is only present in the minds of developers, such information will be lost in the all too frequent event of personnel fluctuations. It is therefore not only essential to embody all such key knowledge within the primary software artifacts but to also describe them as concisely and appropriately as possible, maximizing the ease with which they can be understood by all interested stakeholders. This requires that primary software artifacts can be expressed using a *concise and tailorable presentation*.

Clearly all these different forms of change can occur concurrently, so the techniques used to accommodate them must be compatible with one another. Historically, one of the most effective ways of limiting the sensitivity of software to requirements changes is to structure it according to the principles of object-orientation. Concepts such as encapsulation, polymorphism, and inheritance allow one part of a program to be extended without other parts being affected. However, in its traditional form, object-oriented programming is not well-suited to handle the dynamic addition of new types (III). To meet this requirement, "meta" concepts are needed as well. Metalevel description techniques, in general, are also useful for supporting the above mentioned *user-defined mappings* (I) and *interoperability* (II) requirements. All these requirements (I)-(III) are best dealt with by establishing a definition (or meta-) layer above the subject of description so that humans and tools can systematically access and manipulate it. Finally, as evidenced by the success of the UML, the technology that has the best record of supporting concise presentation of information is graphical modeling. Again, however, it must be enhanced with metalevel description techniques to fully support *user-tailorability* (IV).

Thus, the best strategy for realizing the main goals of the MDA vision is to leverage the synergy between object-orientation, metalevels and modeling—in short, to utilize **Object-Oriented Metamodeling**. Based on the above discussion we can identify the following concrete list of requirements that an MDA infrastructure should ideally support. It should define–

1. the concepts that are available for the creation of models and the rules governing their use.
2. the notation to be used in the depiction of models.
3. how the elements of a model relate to (i.e., represent) real world elements, including software artifacts.
4. concepts to facilitate dynamic user extensions to (1) and (2), and models created from them.
5. concepts to facilitate the interchange of (1) and (2), and models created from them, between tools.
6. concepts to facilitate user defined mappings from models to other artifacts (including code).

Having established a list of such abstract requirements we are now able to evaluate various approaches with respect to the extent that they address these requirements and the elegance they display in doing so.

3 METMODELING AS LANGUAGE DEFINTION

Since the MDA is a relatively new vision, the OMG community is still in the process of adapting its standards to fully support it. At the heart of this adaptation process is the impending update of the UML and MOF modeling and metadata standards. While most members of the OMG community

[2] This is the form of change which is usually associated with the MDA approach.

would agree that object-orientation, metalevels and graphical modeling all have a role to play in supporting the MDA, they do not all agree on the relative importance of their respective roles and how they should be integrated.

Historically, the UML and MOF standards have focused on requirements (1)-(3) identified above, by viewing the UML metamodel and the MOF as essentially having the role of defining languages. The UML metamodel is viewed as defining the language for creating models, and the MOF as defining the language for creating metamodels. While this is understandable, given that the first three requirements essentially ask for an abstract syntax (1), a concrete syntax (2), and a semantics definition (3), i.e., a traditional language definition, this approach does not scale up to satisfy *all* the needs of an MDA infrastructure as characterized by requirements (1)-(6). In particular, limiting the role of metalevels to simply defining languages has the following concrete shortcomings:

First, users need to be able to extend both the abstract syntax and concrete syntax dynamically (4). The provision of this feature is best dealt with by providing a metalevel above the user modeling level which is *not* a classical language definition level (e.g., the M_2 level in the four layer metamodeling hierarchy), but a domain metalevel containing user metatypes. This level defines the library of modeling elements available to modelers including concepts such as TreeSpecies [2], Agent (e.g., for active classes), Breed, etc. These concepts all share the property that their instances can be instantiated (i.e., they are metatypes) and that they are typically specific to a certain modeling domain.

Second, the traditional language definition approach does not naturally accommodate the type-instance duality of elements which manifests itself as soon as more than two levels are present. When something (e.g., BorderCollie) has been instantiated (here from Breed) and can itself be instantiated (here to e.g., "Lassie") it has both an instance facet and a type facet. The traditional approach must then longwindedly explain how certain features of an element (belonging to the type facet) have an effect on some features (belonging to the instance facet) of elements derived from it.

Third, the language definition metaphor on its own does not address the need to define mappings (6). The systematic definition of mappings from user models to other representations (e.g., other models, code, storage formats, etc.) requires a meta-meta-level defining the language used for defining modeling languages. Yet, for mappings to be available for *both* the modeling (M_1) and instance level (M_0) such a level would need to play two roles at the same time. It would need to—

1. be the definition level for M_2 concepts, and
2. somehow provide interfaces to elements at both levels M_1 and M_0.

This is the dilemma that the MOF is currently facing. On the one hand it is trying to act as the UML meta-metamodel and on the other hand it is supposed to standardize modeling repositories, potentially containing elements from all levels, including M_1 and M_0. These two roles cannot be reconciled with each other if one assumes the usual premise that each level exclusively defines just the level below it.

Fourth, traditional language definition does not address the need for interoperability (5). Again, an additional meta-metalevel is required. While multiple metalevels are certainly not incompatible with the traditional language definition approach, the resulting combination is not very economical in terms of concepts and techniques required and the individual level contents needed. Time and again each level has to establish basic concepts such as classification and instantiation giving rise to the *replication of concepts* problem [3].

In summary, while the language definition metaphor is the most established way of meeting MDA requirements (1)-(3), it does not scale up well to meet the remaining requirements (4)-(6). The basic problem is that exclusively using the "language definition" metaphor yields an extensive use of meta description techniques at the expense of object-oriented techniques and graphical modeling concerns.

Unfortunately, despite these problems, the currently prevailing view in the UML/MOF revision process is to strengthen the "language definition" emphasis of the UML metamodel and the MOF, and to further weaken the role of object-orientation and graphical modeling at these levels. Indeed, some proposals go so far as to suggest that the UML metamodel should essentially be just a graphical rendering of the traditional elements of language definition technology (i.e. abstract syntax, concrete syntax, semantic domain and semantic mapping) [4]. Although this approach does in a sense introduces "more model" at the meta level it does so by graphically rendering a lot of detail which would be more appropriately be expressed textually. In contrast, the real benefit of graphical modeling is to provide high levels of abstraction, allowing essential information to be presented in a concise manner. This implies that one does not obtain a language definition *model* by just graphically rendering a very detailed language definition.

We believe that overemphasizing the "language metaphor" in this way is heading in the wrong direction. The other key foundations of the MDA (graphical modeling and object-orientation) should be given a much greater role to play and should be integrated uniformly across all levels of the MDA infrastructure. The following section discusses some keys ideas which we believe are needed to achieve this.

4 TOWARDS A SYNERGISTIC MDA INFRASTRUCTURE

In order to address MDA requirements (4)-(6) we believe that the language metaphor needs to be augmented by the "library metaphor". The latter concept originates form the observation that object-oriented programming languages support two ways of creating and composing programs—by use of language mechanisms, and by use of the class library, which offers predefined types for users to use and extend. Instead of exclusively focusing on defining and using language mechanisms in the definition and application of the UML/MOF both approaches should be exploited.

Supporting the library metaphor (as describe above) becomes most natural once fundamentally different flavors of instantiation are distinguished [5]. We refer to these as *logical* instantiation versus *physical* instantiation [3]. A certain modeling concept (e.g., BorderCollie) is both a *logical* instance of a modeling library concept (e.g., Breed) and a *physical* instance of a modeling language element (e.g., Class). Although this distinction of instantiation forms is never made explicit in the OMG four layer metamodeling hierarchy, it must exist in some form for it to be sound. Consider the user level concept C as an instance-of the metamodel element Class and O as an instance-of Object. These instance-of relationships (going from M_1 to M_2) then represent inter-level (*physical*) instantiations while the (user modeled) relation between C and O represents an intra-level (*logical*) instantiation.

Once physical and logical instantiation have been explicitly distinguished, it is easy to handle the dynamic addition of types. The logical metalevel (L_2) enables the definition of new user level type properties that can be created without implying a change to the language definition (physical metalevel, P_1). The physical metalevel is thus put in a position to provide a fixed and immutable definition of the notion of classifiers and instantiation, thereby making it unnecessary for the logical metalevels to repeat this exercise time and again. Each individual logical metalevel can then make the natural assumption that all its elements possess the type-instance duality, i.e., are instances from the logical metalevel above and give rise to instances at the logical metalevel below. Explaining the effects of instantiation is then

1. only required once at the physical metalevel, and
2. very simple, when using the concept of *deep instantiation* [6].

With the deep instantiation approach, defining the effects of instantiation is much simpler since the mapping exercise (e.g., mapping from attributes to slots) can be replaced by a simple decrement operation on meta-attributes ("level" and "potency") [6]. However, the mechanism's main advantage from a user's perspective is the ability to specify element properties across more than one level of instantiation. Such a mechanism becomes necessary, as soon as more than two user levels (i.e., the classic type & instance levels) are present. Odell's powertype concept [2] is such a mechanism but is not as concise and scalable as the deep instantiation approach. Taking everything into account it hence becomes possible to elegantly meet requirements (1)-(4), most notably by the use of a metamodeling infrastructure with a two-dimensional approach to instantiation relationships [3].

With regard to the two remaining requirements—interoperability (5) and mappings (6)—it is useful to view them as two sides of the same coin. The format used for enabling tool interoperability could be regarded as the target of a mapping, or taking the opposite viewpoint, the various target formats could be regarded as yet another way of representing models, i.e., model formats. Moreover, even the notation (2) (i.e., the presentation of models) could be interpreted as yet another model representation.

Since the presentation of modeling elements is best defined at the next higher logical metalevel, this could also be the location for various user defined mappings. For both purposes it makes sense to predefine default (re-)presentation forms and mappings, and override these for special elements accordingly.

5 CONCLUSION

Although there is general agreement that object-orientation, graphical modeling and metalevel description techniques are all key ingredients of a comprehensive MDA infrastructure, there is no consensus about how they should be integrated to maximum effect. The currently prevailing view is to emphasize the metalevel dimension at the expense of graphical modeling and object-orientation, and to essentially view the design of the UML metamodel and MOF as language definition problems. While language definition techniques certainly have a role to play, we believe that focusing on this to the exclusions of the needs and capabilities of the others is suboptimal and cannot elegantly satisfy all the fundamental requirements for a comprehensive MDA infrastructure

To support this position, in this paper we first identified the fundamental forms of change (I)-(IV) that the MDA approach can help manage. In doing so, we postulated that the most valuable role of the MDA approach is to improve long-term productivity by maximize the useful lifetime of primary software artifacts. We then used these forms of change (I)-(IV) to derive a concrete set of requirements (1)-(6) that a comprehensive MDA infrastructure should support. We also identified the "library metaphor" as a powerful complement to the traditional "language metaphor" for organizing the MDA infrastructure. Finally, in the last section of the paper we described the key ingredients which we believe are needed to provide support for all MDA infrastructure requirements (1)-(6)—

- a two-dimensional metalevel hierarchy, arranging modeling elements in multiple logical metalevels, next to one physical metalevel.
- the explicit recognition of the instance-type duality of modeling elements.
- deep instantiation as a mechanism to control element properties across more than one level.
- unifying the requirements of model presentation, representation, and mappings as user-defined transformations from a model to another target (i.e., graphical notation, storage formats, other models, respectively).

We believe that MDA infrastructures based on these concepts, exhibiting a synergetic interplay of object-orientation, metalevels, and graphical modeling, will be cleaner and simpler, yet more flexible and powerful.

6 REFERENCES

1. R. Soley, Model Driven Architecture, White paper, November 27, 2001.
2. J. Odell, Power Types, *Journal of Object-Oriented Programming*, May 1994.
3. Colin Atkinson and Thomas Kühne, Rearchitecting the UML Infrastructure Submitted for the *ACM journal "Transactions on Modeling and Computer Simulation"*, 2002.
4. Andy Evans and Stuart Kent, Meta-modeling semantics of UML: the pUML approach, *Proceedings of the 2nd International Conference on the Unified Modeling Language*, editors Bernhard Rumpe and Robert B. France, 1999.
5. Jean Bézivin and Richard Lemesle, Ontology-Based Layered Semantics for Precise OA&D Modeling, *Proceedings of the ECOOP'97 Workshop on Precise Semantics for Object-Oriented Modeling Techniques*, editors Haim Kilov and Bernhard Rumpe, 1997.
6. Colin Atkinson and Thomas Kühne, The Essence of Multilevel Metamodeling *Proceedings of the 4th International Conference on the UML 2001*, Toronto, 2001.