Model-Driven Development: A Metamodeling Foundation

Colin Atkinson
University of Mannheim
68161 Mannheim, Germany
atkinson@informatik.uni-mannheim.de

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

KEYWORDS
model driven development requirements, metamodeling, language definition, domain meta concepts

ABSTRACT
There is general agreement that metamodeling is an essential foundation for model driven development, but there is less consensus on the precise form it should take and role it should play. In this article we first analyze the underlying motivation for model-driven development and then derive a concrete set of requirements that a supporting infrastructure should satisfy. In particular, we discuss why the traditional "language definition" interpretation of metamodeling is not a sufficient foundation, and explain how it can be extended to unlock the full potential of model driven development.

1 INTRODUCTION

Ever since human beings started to program computers, software engineers have been working to raise the level of abstraction at which this activity takes place. The first FORTAN compiler was a major milestone in computer science since, for the first time, it allowed programmers to specify *what* the machine should do rather than *how* it should do it. The raising of programming abstraction levels has continued apace since then, and today's object-oriented languages enable programmers to tackle problems of a complexity undreamt of in the early days of programming.

Model driven development (MDD) [1, 2] can be regarded as a natural continuation of this trend. Instead of requiring developers to use a programming language spelling out *how* a system is implemented, it allows them to use models to specifying *what* system functionality is required and what architecture is to be used.

This move to yet higher-levels of specification holds the potential to drastically reduce the complexity of problems considered to be hard by today's standards. Issues like object allocation, method lookup, exception handling, etc. which had to be programmed manually in the past are nowadays handled automatically by compilers behind the scenes. The aim of MDD is to achieve the same degree of automation for issues which today are very complex when dealt with manually, such as system persistence, interoperability, distribution, etc.

Because of its potential, many MDD initiatives are underway and companies are already trying to deliver supporting technology. However, there is no universally accepted definition of precisely what MDD is and what support for MDD entails. In particular, many requirements for MDD support—where they have been identified—remain implicit. In this article we therefore first review the underlying motivations for MDD and then derive a concrete set of requirements that MDD infrastructures should support. We finally analyze the technical implications of these requirements and discuss some of the basic principles by which they can be supported.

2 REQUIREMENTS FOR MODEL DRIVEN DEVELOPMENT

The underlying motivation for model-driven development is to improve productivity—that is, to increase the return that a company derives from its software development effort. It delivers this benefit in two basic ways—

1. by improving the *short-term productivity* of developers—that is, by increasing the value of primary software artifacts in terms of how much functionality they deliver.

2. by improving the *long-term productivity* of developers—that is, by reducing the rate at which primary software artifacts become obsolete.

The level of *short-term productivity* depends on how much functionality can be derived from a primary software artifact. The more executable functionality that can be generated, the higher the productivity. To date, most tool support for MDD is centered around this form of productivity, i.e., most tool vendors have focused their efforts on automating the production of code from visual models. However, this only tackles one of the two main aspects of MDD.

The level of *long-term productivity* depends on the longevity of a primary software artifact. The longer an artifact remains of value, the greater the return derived from the effort that went into the creation of that artifact. Thus, a second and strategically even more important aspect of MDD is reducing the sensitivity of primary artifacts to change. Four main fundamental forms of change (I)-(IV) are of particular importance:

I) **Personnel**
As long as certain vital development knowledge is stored only in the minds of developers, such information will be lost in the all too frequent event of personnel fluctuations. To ensure that software artifacts outlive the tenure of their creator it is, therefore, essential that they be made accessible and useful to as wide a range of people as possible. In addition to being presented in as concise a way as possible, they should also take a form which maximizes the ease with which they can be understood by all interested stakeholders (including customers). The implication for the technical level is that primary software artifacts can be expressed using a *concise and tailorable presentation*.

II) **Requirements**
Changing requirements have always been a big problem in software engineering, but never more so than today. Not only must new features and capabilities be supplied at an ever increasing rate, but the impact of these changes on existing parts of the system must be low in terms of both maintenance efforts *and* in terms of disrupting online-systems. Today's enterprise applications cannot simply be taken offline for extended periods of time in order to be extended. Instead, changes must be realized while a system is running. At a technical level, this implies the need to support the *dynamic addition of new types at runtime.*

III) **Development Platforms**
Development platforms are also in a state of constant evolution. Yet, primary software artifacts are dependent on the particular tool that created them, i.e., their lifetime is limited by the corresponding tool lifetime. This strongly suggests that artifacts should be decoupled from their development tools. Consequently, another technical requirement is that tools should store artifacts in formats that can be used by other tools, in other words, they should support *high-levels of interoperability.*

IV) **Deployment Platforms**
Yet another growing problem for software organizations is keeping up with the rate of platform evolution. No sooner have developers mastered one new platform technology then another one comes along to take its place[1]. To increase the lifetime of primary software artifacts it is therefore necessary to shield them from changes at the platform level. Technically this means that it is necessary 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.*

All these different forms of change can occur concurrently, so the techniques used to accommodate them must be at least compatible with one another, and at best complement each other synergistically. However, before one proceeds to pick and enhance existing techniques for this purpose, it is useful to summarize which technical requirements they should support.

The need for *concise* models (I) can be addressed by providing a visual modeling language (see bullets (1)-(3) below). Requiring models to be *tailorable* (I) and to enable the *addition of new types at runtime* (II) implies that the modeling language needs to be dynamically extensible (see (4) below). Finally, *interoperability* between tools (III) and *user definable mappings* from models to other models or code (IV) must be supported (see (5) & (6) below).

In summary, a model driven development supporting infrastructure must 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.

[1] This is the form of change which is usually associated with the MDD/MDA approach.

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.
6. concepts to facilitate user defined mappings from models to other artifacts (including code).

Having clarified what capabilities we would like an MDD infrastructure to provide, we can turn our attention to how these requirements can be satisfied. In particular, we identify what approaches need to be integrated and what basic form they should take.

3 TOWARDS AN MDD INFRASTRUCTURE

As established above it is clear that visual modeling is one of the technological foundations for a technological MMD support, addressing requirements (1)-(3). It has a long record of success in engineering disciplines, including software engineering, since it makes effective use of human visual perception.

The technology which has the best track record at supporting the flexible choice of modeling concepts (1) and extendable languages (4) (in its static form) is object-orientation. Object-oriented languages enable language users to extend the set of available types. Thus, object-orientation is generally accepted as one of the other key foundations of model driven development.

Finally, the approach which has been the most effective at addressing the issues defined by requirements (5)-(6) is the use of metalevel descriptions. Beyond that, metalevel descriptions are also vital for supporting both static and dynamic aspects of requirement (4). Only with a metalevel above user types can models be fully customized to a certain domain or class of stakeholders, and only with a metalevel above classes is it possible to add new types dynamically at runtime[2]. Thus, all four forms of change essentially call for a descriptive metalevel which can be systematically accessed and manipulated by humans and tools.

The challenge that we face in creating an infrastructure for model driven development is therefore to optimally integrate visual modeling, object-orientation, and metalevel description. In the following we first analyze the existing approach for doing this and then suggest some enhancements that better address the complete list of technical requirements (1)-(6).

3.1 Traditional MDD Infrastructure

Figure 1 illustrates the traditional four layer infrastructure that underpins the first generation of MDD technologies–namely the UML [3] and the MOF [4].

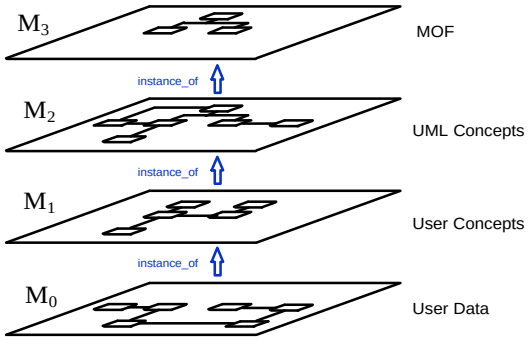

Figure 1 Traditional OMG Modeling Infrastructure

This infrastructure consists of a hierarchy of model levels, each (except the top) being characterized as "an instance" of the level above. The bottom level, also referred to as M_0 is said to hold the "user data", i.e., the actual data objects the software is designed to manipulate. The next level, M_1, is said to hold a "model" of the M_0 user data. This is the level at which user models reside. Level M_2 is said to hold a "model" of the information at M_1. Since it is a model of a (user) model, it is often referred to as a metamodel. Finally, level M_3 is said to hold a model of the information at M_2, and hence is often characterized as the meta-metamodel. For historical reasons it is also referred to as the MOF (Meta Object Facility).

This venerable four layer architecture has the advantage of easily accommodating new modeling standards (e.g., the CWM) as instances of MOF at the M_2 level. MOF aware tools can thus support the manipulation of new modeling standards and enable information interchange across MOF compatible modeling standards.

Although it has been a successful foundation for the first generation of MDD technologies, this traditional infrastructure does not scale up well to handle all technical requirements (1)-(6). In particular, four key problems can be identified.

a) There is no explanation of how entities within the infrastructure relate to the real world. Are the real

[2] Not counting techniques that emulate this effect.

world elements accommodated within the infrastructure or do they lie outside?

b) There is an implication that all instance-of relationships between elements are of fundamentally the same kind. Is this a valid approach or should one be more discriminating?

c) There is no explicit principle for judging at which level a particular element should reside. Using a single instance-of relationship to define the metalevels[3] always seems to introduce inconsistencies in one way or the other. Is it at all possible to use instantiation to define metalevel boundaries and if so how?

d) There is a preference for the use of metalevel description (sometimes in the form of stereotypes) to provide predefined concepts. How can the other way of supplying predefined concepts—libraries of (super)-types, to be used or specialized—be integrated within the architecture?

To address these problems it is necessary to adopt a more sophisticated view of the role of metamodeling in MDD and to refine the simple "one-size-fits-all" view of instantiation. Regarding all instance-of relationships as being of essentially the same form and playing the same role does not scale up well for all requirements (1)-(6). Instead, it is helpful to identify two separate orthogonal dimensions of metamodeling, giving rise to two distinct forms of instantiation [6, 7]. One dimension is concerned with language definition and hence makes use of *linguistic instantiation*. The other dimension is concerned with domain definition and thus uses *ontological instantiation*. Both forms occur simultaneously, and serve to precisely locate a model element with the language-ontology space.

3.2 Linguistic Metamodeling

As stated before, technical requirements (1)-(3) essentially state the need for a language definition capability (see Table 1). Much of the recent work on enhancing the infrastructure has therefore focused on the use of metamodeling as a language definition tool. With this emphasis, the linguistic instance-of relationship is viewed as being dominant, and levels M_2 and M_3 are essentially viewed as being language definition layers.

This approach relegates ontological instance-of relationships, which relate user concepts to their domain types, to a secondary role. In other words, whereas linguistic instance-of relationships cross (and in fact form the basis for) linguistic metalevels, ontological instance-of relationships do not cross such levels, but relate entities within a given level. Figure 2 shows this latest interpretation of the four layer architecture as embraced by the forthcoming UML 2.0 and MOF 2.0 standards. Although the latter take a predominantly linguistic view, it is very useful to nevertheless let ontological (intra-level) instantiation establish its own kind of ontological (vertical) meta level boundaries, as indicated by the different shades within level M_1 in Figure 2.

concept	*purpose*	*UML solution*
abstract syntax	the concepts from which models are created → (1)	class diagram at level M_2
concrete syntax	concrete rendering of these concepts → (2)	UML notation, informally specified
well-formed-ness	rules for the application of the concepts → (1)	constraints on the abstract syntax (e.g., using OCL)
semantics	description of the meaning of a model → (3)	natural language specification

Table 1 Language Definition

As well as making the existence of two orthogonal metadimensions explicit, Figure 2 also illustrates the relationship between model elements and corresponding real world elements. The M_0 level is no longer inhabited by user objects, but rather by the real world elements that they model[4]. Note that the real Lassie is said to be represented by object Lassie, i.e., "instance-of" is no longer used to characterize the real Lassie as an instance of Collie[5]. User objects (i.e., model representatives of real world objects[6]) now occupy the M_1 level, along with the types of

[3] Sometimes know as the strictness condition for metamodelling.

[4] The lightbulb is meant to denote the mental concept "Collie".

[5] It is possible to make such a statement, but it remains a derived property of the real Lassie (via object Lassie).

[6] Note that user data is also considered to be part of the real world, even though it is artificial and may even represent other real world elements.

which they are (ontological) instances. From level M_1 on, every level is regarded as a model expressed in the language defined at the level above.

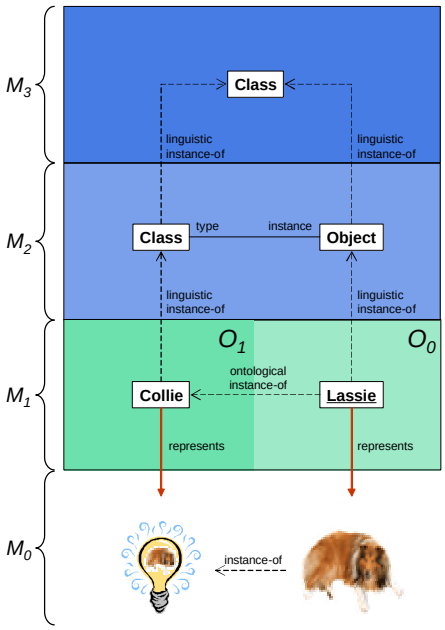

Figure 2 Linguistic Metamodeling View

3.3 Ontological Metamodeling

Although linguistic metamodeling[7] addresses many of the technical requirements, it is not sufficient on its own. In particular, requirement (4) requires the capability to dynamically extend the set of domain types available for modeling, and this in turn requires the capability to define domain metatypes (i.e., types of types). We refer to this form of metamodeling as *ontological metamodeling* since it is concerned with describing what concepts exist in a certain domain and what properties they have.

Figure 2 actually already contains an example of ontological instantiation. The model element Lassie is an ontological instance of Collie, and thus resides at a lower ontological level than Collie. This expresses the fact that in the real world, the mental concept Collie is the logical type of Lassie.

Figure 2 only contains two ontological model levels (O_0 & O_1) both contained within the linguistic level M_1, but this dimension can be naturally extended to give further ontological levels. Figure 3 features another ontological level (O_2), showing that Collie can be regarded as an instance of Breed. An ontological metatype such as Breed not only distinguishes types like Collie and Poodle from other types—such as CD and DVD—but can also be used to define breed properties, for example, where a particular breed first originated or from what other breed it was developed.

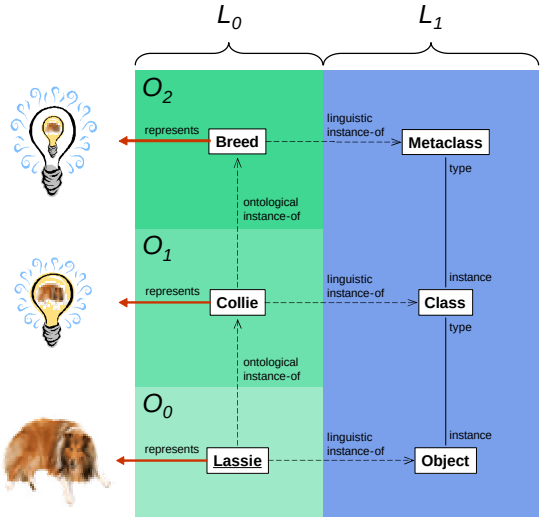

Figure 3 Ontological Metamodeling View

Figure 2 also makes another very important point regarding the relationship of the two meta dimensions (linguistic and ontological), since it is in fact a 90 degree clockwise rotation of Figure 2 (with level O_2 added). Instead of arranging the *linguistic* metalevels horizontally, so as to suggest that the metalevels of the traditional infrastructure are linguistic, Figure 3 arranges the *ontological* metalevels horizontally so as to suggest that the traditional metalevels are ontological. Both arrangements are equally valid because the traditional infrastructure made no distinction between ontological or linguistic instantiation, and thus made no choice about the meaning of its metalevels.

Not only are both arrangements (or viewpoints) equally valid, they are equally useful. Just as ontological metmodeling is relegated to a secondary role when the linguistic viewpoint is emphasized, the linguistic metmodeling is relegated to a secondary role when the ontological viewpoint is emphasized. Ideally, therefore, ontological and linguistic metamodeling should be given equal importance in an MDD infrastructure, and neither should be subservient to the other. Unfortunately this is not the case in most of the recent infrastructure proposals.

In the case of the UML2.0/MOF2.0 it is the linguistic

[7] Previously also referred to as „physical“ metamodeling [EssenceReference].

dimension which is emphasized. Levels O_0 & O_1 exist within M_1 but are not explicitly separated by a metalevel boundary. Ontological metamodeling is not excluded per se, but the encouraged mechanisms for enabling it—profiles and stereotypes—have known limitations. While it is possible to express that Collie is an instance of Breed (see Figure 4), the full arsenal of M_1 modeling concepts is not available for stereotype modeling, e.g., a visual model of associations and generalization relationships between metaconcepts.

Figure 4 Ontological metamodeling through stereotypes

However, being able to freely model with metaconcepts (i.e., make full use of an O_2 level) has long been recognized as being useful. Being able to make use of metaconcepts such as TreeSpecies [5] or Breed is a big advantage. Figure 5 shows perhaps one of the most mature and established examples of ontological metamodeling, the biological taxonomy for living beings. Metaconcepts such as Breed, Species, etc. serve to allow new classes of creatures to be added to the taxonomy. In a software system, they would facilitate the dynamic addition of new types at runtime. Note that it is not possible to cast Breed, Species, etc. as supertypes at the O_1 level. While it makes sense to say that Lassie is a Collie, Dog, etc. it does not make sense to say that Lassie is a Breed, Species, etc. Also, it is not a problem to accommodate level O_3 (see Figure 5) within the ontological meta-dimension, while stereotypes are not designed to support this.

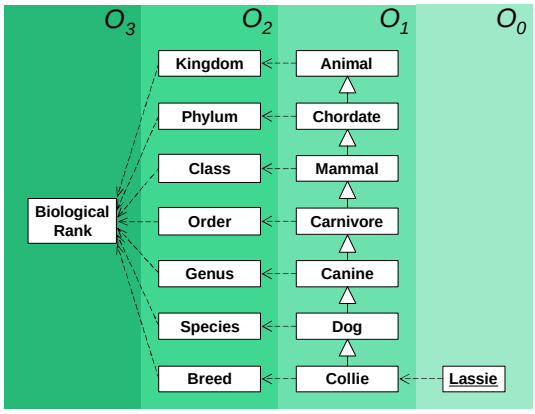

Figure 5 Ontological Metamodel of Biological Classification

Ontological metamodeling is particularly important for model driven development because it is explicitly called for in two of the main strategies for model transformation defined in the MDA Users guide [8]. First, it is the basis for the marking mechanism which is envisaged as one of the key ways to support the user-driven definition of model transformation, that is, to enable the use of technical requirement (6). Second, it serves as the basis for defining mappings in the "framework-based" version of type level transformation [8]. This assumes the existence of an ontologically predefined set of superclasses (with associated predefined mappings) which users specialize with their own application classes.

4 FINAL REMARKS

In this article we have defined a concrete set of requirements that an ideal MDD infrastructure should support, and have argued that the explicit distinction of two orthogonal forms of metamodeling—linguistic and ontological—is the key to fixing some of the problems in the first generation MDD infrastructure and to scaling it up to satisfy all of the identified requirements.

The forthcoming revision of the OMG's MDD infrastructure in the UML2.0/MOF2.0 standards represents a significant step forward in this regard in that for the first time it accommodates two distinct forms of instantiation. However, two significant problems remain.

First, although the distinction is present, it is not made explicit enough. For instance, while linguistic metalevel boundaries are recognized, ontological boundaries do not exist. Moreover, when coupled with the fact that stereotypes are the strongly preferred mechanism for metamodeling, there is a strong suggestion linguistic metamodeling is the only meaningful form of metamodeling.

Second, in the current profile mechanism there is still a major bias in favor of defining predefined concepts at the meta-level (i.e. as part of the modeling language, given the current infrastructure preoccupation with linguistic metalevels) rather than as regular user types at the M_1 level. This is despite the fact that libraries or frameworks at the M_1 level have established a strong track record for making predefined concepts available for direct use or specialization by users. In fact, this form of reuse and predefinition is explicitly exploited in the MDA User Guide as a means for defining reusable type mappings.

Despite this reservation, the new OMG MDD infrastructure does represent a significant step

forward and provides most of the identified technical capabilities. We hope that the discussion in this paper might help improve subsequent versions of the infrastructure in the future.

4 REFERENCES

1. J. Mukerji & J. Miller (ed), Model Driven Architecture, http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01, July 2001.
2. David S. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing, OMG Press, 2003
3. Object Management Group, Unified Modelling Language 1.5 specification, http://doc.omg.org/formal/03-03-01, March 2003
4. Object Management Group, Meta-Object Facility 1.4 specification, http://doc.omg.org/formal/02-04-03, April 2002
5. J. Odell, Power Types, *Journal of Object-Oriented Programming*, May 1994.
6. Colin Atkinson and Thomas Kühne, Rearchitecting the UML Infrastructure ACM journal "Transactions on Modeling and Computer Simulation", Vol. 12, No. 4, 2002.
7. 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.
8. OMG, MDA Guide V1.0, http://doc.omg.org/formal/03-05-01, May 2003.