JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2006**

Vol. 6, No. x, Nov-Dec 2006

The Tao of Modeling Spaces

Dragan Djurić, University of Belgrade, Serbia and Montenegro
Dragan Gašević, Simon Fraser University Surrey, Canada
Vladan Devedžić, University of Belgrade, Serbia and Montenegro

Abstract

The paper introduces modeling spaces in order to help software practitioner to understand modeling. Usually software engineers often think of a specific kind of models – UML models, but there are many open questions such as: Should we assume that the code we write is a model or not; What are models and metamodels, and why do we need them; What does it mean to transform a model into a programming language. Unlike current research efforts that answer to those questions in rather partial ways, we define a formal encompassing framework (i.e. Modeling spaces) for studying many modeling problems in a more comprehensive way. We illustrate the benefits of that framework for explaining present dilemmas practitioners have regarding models, metamodels, and model transformations.

1 INTRODUCTION

Recent software engineering trends like UML modeling and Model Driven Architecture rely heavily on models, metamodels, and model transformations. Most of such efforts focus on specific benefits, but largely ignore the way software practitioners usually understand modeling. In fact, when talking about models, software engineers often think of a specific kind of models – UML models. However, there are many open questions such as:

- Should we assume that the code we write is a model or not?
- What are models and metamodels, and why do we need them?
- What does it mean to transform a model into a programming language?

So far, researchers answered most of such questions focusing on individual aspects of modeling [Seidwitz03]. However, it is also possible to define a formal encompassing framework for studying many modeling problems in a more comprehensive way. We call that framework *modeling spaces*. It has direct implications to software engineering processes and activities. Our work is directly inspired by the idea of technical spaces [Kurtev02] defined as a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. Although technical spaces are rather useful approach, they have been loosely defined and have not given clear answers to the

Cite this article as follows: Djurić, D., Gašević, D., Devedžić, V.,: "The Tao of Modeling Spaces", in *Journal of Object Technology*, vol. 6 no. x, Nov-Dec 2006, pp. xx-xx.

previous questions. We do not argue against technical spaces, but we try to define borders between technical spaces and modeling spaces. We complement the work on technical spaces in that way.

In the next section we discuss about some current modeling technologies (Model Driven Architecture and ontologies) as well as technical spaces. In section 3 we try to give a wider context of modeling in terms of some non-technical fields (e.g. painting and music), while section 4 introduces the basic terms that define modeling spaces. Section 5 further clarifies modeling spaces, whereas section 6 discusses some practical issues related to modeling spaces. In section 7 we depict transformations between modeling spaces. In section 8 we define relations between modeling spaces and technical spaces, while section 9 gives a full definition of modeling spaces using UML. Finally, section 10 shows an example of the use of modeling spaces for explaining relations between Meta-Object Facility (a part of MDA) and XML Metadata Interchange (also a part of MDA).

2 RELATED WORK

Two competing, but also complementary, modeling paradigms have been in the focus of Software Engineering lately: Model-Driven Architecture (UML, MOF, object-oriented) and the Semantic Web (OWL, RDF, ontologies). There is a benefit in mixing these approaches to increase the benefits of using them separately. For example, ontologies have a greater expressiveness and are closer to non-technical people while object-oriented approaches are better implemented and technically better supported. Mixing these in the various phases of a system life cycle could help the system to better respond to user's needs. MDA is also a very user-friendly example of structuring meta-levels, so we will mostly base our explanations on these two well-known approaches.

Model Driven Architecture

Model Driven Architecture (MDA) is an ongoing software engineering effort under the auspices of OMG. It defines three viewpoints (levels of abstraction) from which a certain system can be analyzed. Starting from a specific viewpoint, we can define the system representation (viewpoint model). The representations/models/viewpoints are *Computation Independent Model* (CIM), *Platform Independent Model* (PIM) and *Platform Specific Model* (PSM) [Miller03]. MDA is based on a four-layer metamodeling architecture and several complementary OMG standards, Figure 1. These standards are *Meta-Object Facility* (MOF) [OMGMOF203], *Unified Modeling Language* (UML) [OMGUML203] and *XML Metadata Interchange* (XMI) [OMGXMI02], and the layers are: meta-metamodel layer (M3), metamodel layer (M2), model layer (M1), and the real world layer (M0).

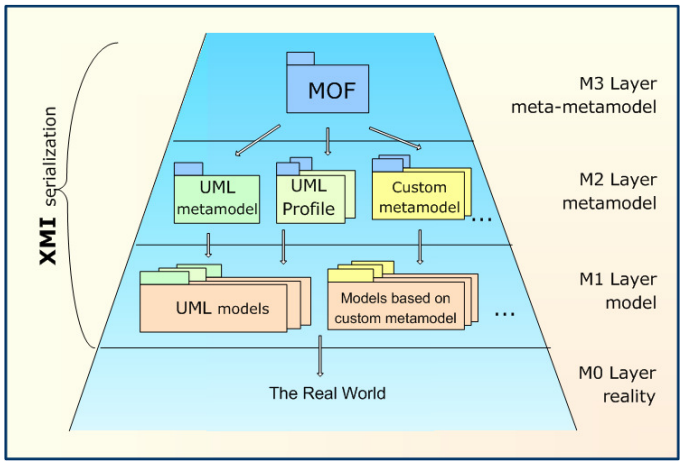


Figure 1 - MDA four-layer MOF-based metadata architecture

The topmost layer in this architecture (meta-metamodel, MOF) defines an abstract language and framework for specifying, constructing and managing metamodels. It is the foundation for defining any modeling language, such as UML or even MOF itself. MOF also defines a framework for implementing repositories that store metadata (e.g. models) described by metamodels [OMGMOF203]. The purpose of the four layers with a common meta-metamodel is to support multiple metamodels and models and their scaling – to enable their extensibility, integration and generic model and metamodel management.

All metamodels (both standard and custom) defined by MOF are positioned at the M2 layer. One of these is UML, a graphical modeling language for specifying, visualizing and documenting software systems. With UML profiles, we can extend basic UML concepts (Class, Association, etc.) with new concepts (by the use of stereotypes) and adapt them to specific modeling needs. The models of the real world, represented by concepts defined in the corresponding metamodel at the M2 layer (e.g. UML metamodel) are at the M1 layer. Finally, at the M0 layer are things from the real world. Although the "traditional" approach in MDA is to consider real-world things as instances of model elements, this was changed in newer approaches to a more natural approach, where model is a "snapshot" of reality [Atkinson03].

Another foundation standard of this architecture is XMI, which defines mapping from MOF-defined metamodels to XML documents and Schemas. XML, which has good software tools support, enables meta-metamodel, metamodel and model sharing through XMI.

Semantic Web and Ontologies

Ontologies have been around for quite some time now. Since early 1990s researchers in the domain of artificial intelligence and knowledge representation have studied ontologies as a means for knowledge sharing and reuse among knowledge-based systems. One of the central roles of ontologies is to establish further levels of interoperability, i.e. semantic interoperability, between agents and applications on the emerging Semantic

Web [Berners-Lee01], as well as to add a further representation and inference layer on top of the Web's current layers [Decker00] [Hendler01]. When used on the Web, ontologies specify standard terms and machine-readable definitions.

Semantic Web architecture is a functional, non-fixed architecture. Berners-Lee defined three distinct levels that incrementally introduce expressive primitives: metadata layer, schema layer and logical layer. In Figure 2 we can see how languages are arranged on these layers. The Resource Description Framework (RDF) and the RDF Schema as general languages for the description of metadata on the Web [Corcho01] are positioned in the Metadata layer and the Schema layer. The Web Ontology Language (OWL) (in the Logical layer) is a current semantic markup language for publishing and sharing ontologies on the WWW adopted by W3C [Bechhofer et al, 2004]. OWL is developed as a vocabulary extension of RDF and is derived from the DAML+OIL Web Ontology Language. OWL has three variants: OWL Lite, OWL DL, and OWL Full. OWL is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF and RDFS by providing additional vocabulary along with a formal semantics.

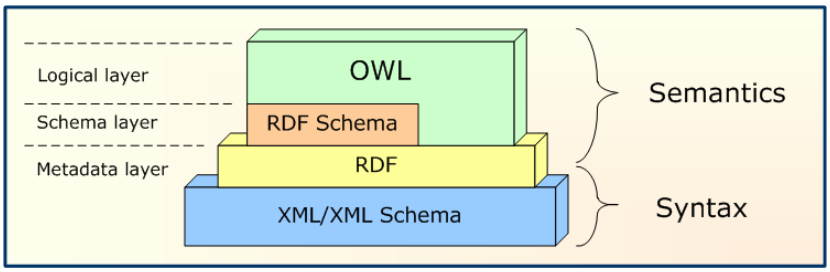
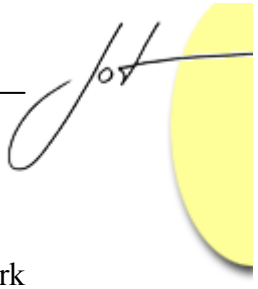


Figure 2 – Ontology languages in the Semantic Web Architecture

Compared to the MDA 4-layer architecture, the Semantic Web language architecture could be a little confusing, since it promotes non-fixed layer architecture. However, we show later in this paper that these non-fixed layers can be put in MDA-like layers depending on the context in which they are used, making them more understandable.

Currently, there is an OMG initiative aimed at defining a suitable language for modeling Semantic Web ontology languages in the context of MDA [OMGODM03] [Djurić05]. The initiative is titled Ontology Definition Metamodel (ODM). It should consist of: 1. ODM – a MOF-based metamodel for ontologies; 2. Ontology UML Profile; 3. Two-way mappings between OWL and ODM, ODM and OUP, and from OUP to other UML profiles. In order to achieve all those mappings between different ontology languages we should take into account differences caused by the fact that those languages are based on different platforms (i.e. Semantic Web and MDA). Accordingly, we need many tools to provide those mappings. One approach to this issue is to apply the concept of technical spaces.



Technical Spaces

Technical spaces have been recently introduced as a means to figure out how to work more efficiently by using the best possibilities of different technologies [Kurtev02]. A technical space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. Although some technical spaces are difficult to define, they can be easily recognized (e.g. XML, MDA, and ontology technical spaces in the case of approaching MDA and OWL). In order to get a synergy of different technical spaces we should create bridges between them, and some of these bridges are bi-directional. The bridges can be created in a number of ways. Note that technical spaces can be classified according to their layers of abstraction (e.g. MDA and ontological engineering are high-level spaces, whereas XML and databases are low-level spaces). The Semantic Web integrates XML and ontological engineering technical spaces.

This approach can be a good starting point for the future research on defining more abstract transformations between different technical spaces. One possible direction is to define all transformations in terms of MDA as well as by employing transformation techniques like Meta Object Facility Query/Views/Transformations (MOF QVT) [OMGMOF2-QVT02]. MOF OVT is a platform-neutral part of MDA aiming to define a language for querying and transforming models as well as viewing metamodels. Furthermore, we can use other technologies for bridging different technical spaces such as: programming languages, XSLT, RDF Query Languages (RDQL) etc. Examples of approaching different languages using technical spaces can be found in [Kurtev03] [Bézivin03] [Gašević05].

3 NON-TECHNICAL MODELS IN A LAYERED MODELING ARCHITECTURE

If we search a dictionary for the word “model”, we get several definitions – most of them just referring to most often used special cases of models; for instance: a replica of an item, a person wearing clothes at a fashion show, a drawing of a building, etc. Few of these definitions are generally applicable. Fortunately, there is a simple and general definition – a model is a simplified abstraction of reality [Hagget67]. Using that definition, we can explain why these special cases are models. A person wearing clothes at a fashion show does not represent herself/himself, but the appearance of any person wearing that clothes; thus it is a model. A drawing of a building is not just a sheet of paper with lines, but also a simplified abstraction of a building containing only data that is important in a given context.

Figure 3 shows some common examples of models put in a layered architecture conceptually inspired by MDA– a famous painting (Mona Lisa, also known as La Gioconda, painted by Leonardo da Vinci in the 16th century), and a part of a written music score (of the song “Smoke on the Water”). A noble Renaissance woman and a rock song are things from the real-world, at the M0 layer. A painting and a music sheet are

obviously abstractions of real world things. Therefore, they are models and can be put at the M1 (model) layer. Metamodels are used to define these models. In the case of a music score, which is a written interpretation of a song, the metamodel (M2) is a set of concepts – stave, note, etc. – and rules that define what each note means and how we can arrange them on five staves. In this context, the meta-metamodel (M3) includes self-defined concepts that also define stave, note etc. Although this architecture is imprecise and definitely not perfect from the perspective of music theory, at least it captures a formal interpretation of music.

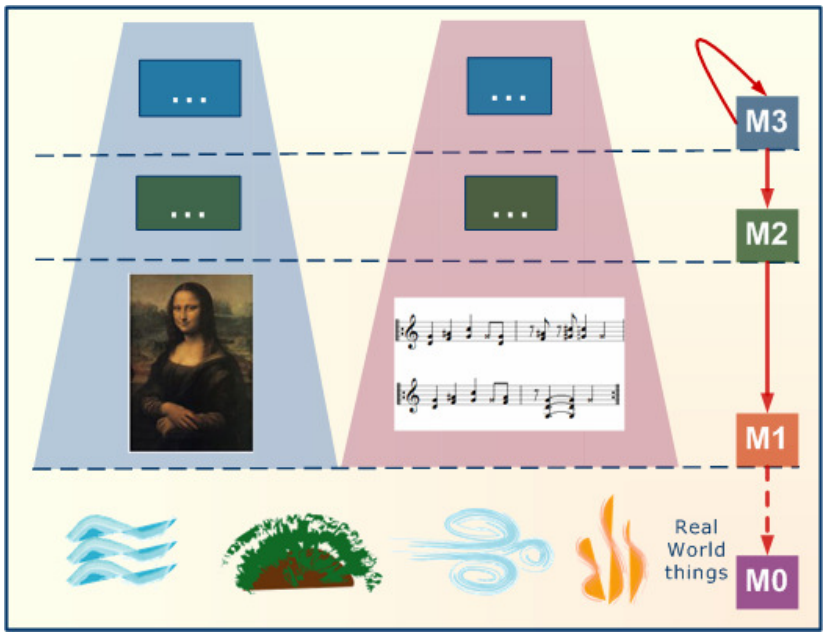


Figure 3 – A few common models put in a MDA-inspired layered architecture

Things get harder in the case of the painting. Is it possible to specify a metamodel that can define such a complex and ambiguous model as a masterpiece painting? A simplistic view based on physical appearance only may lead to the definition of this metamodel in terms of concepts like line, circle, color, etc. The meta-metamodel would then be a set of concepts used to define line, circle, color and their meanings. However, this painting, like any other artwork, is much more than just lines and colors. It has much to do with human psychology and experience and can be interpreted in many ways. It is much more difficult, if not impossible, to define a formal metamodel or meta-metamodel in this case. We may anticipate that they exist, but they are extremely complex and implicit.

Another important issue is that, although Mona Lisa or written notes are models, they are also things from the real world. We can hold them in our hands (if the guards let us do this, in the case of Mona Lisa) and they can be items entered in the information system that stores information about art.

4 MODELING SPACES ESSENTIALS

The previous analysis leads to two important conclusions. First, something can be taken as a model if it is an abstraction of things from the real world, but it is simultaneously a thing from the real world. Whether we take it as a model or as a real/world thing depends on the context, i.e. on the point of view. Second, models can be defined using metamodeling concepts formally or implicitly. Since implicit metamodels cannot be precisely defined using formalisms, as in the case of art, in the rest of this discussion we analyze only formal models. Nevertheless, much of the conclusions can also be applied to implicit metamodels.

Figure 4 shows a general modeling architecture that was inspired by MDA and is in fact its generalization. In such a modeling architecture, the M0 layer is the real world as in [Bézivin04] and [Atkinson03]. It includes all possible things that we try to represent using the models residing at the M1 layer. That representation is more or less abstract and simplified, depending on how rich our models are. Models are defined using concepts defined in metamodels, so each metamodel determines how expressive its models can be. M2 is the layer where the metamodels are located. The metamodels are also defined using some concepts. The set of concepts used to define metamodels resides at the separate M3 layer at the top of this architecture and is called meta-metamodel. Meta-metamodel is nothing more than a metamodel that is used by convention to define other metamodels; it also defines itself. The architecture is generalized to comprise not only models and metamodels based on an object-oriented meta-metamodel like MOF (see sidebar 1), but also other systems (for instance: ontologies, Semantic Web technologies or non-technical representations as shown in Figure 3).

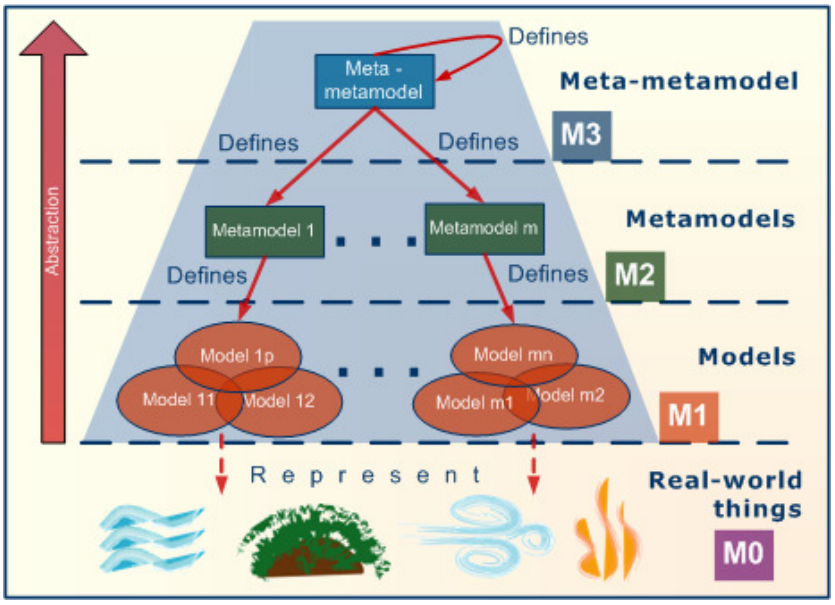


Figure 4 – General four-layer modeling architecture

A *modeling space* is a modeling architecture defined by a particular meta-metamodel. Metamodels defined by the meta-metamodel and models defined by those metamodels

represent the real world from one point of view, i.e. from the point of view of that modeling space. As the meta-metamodel defines the core concepts used in defining all other metamodeling concepts, it is defined by itself. If it was defined by some other concepts, it would not be a meta-metamodel; it would be an ordinary metamodel in some other modeling space.

Figure 5 shows a few examples of well-known modeling spaces. The most straightforward example from this picture is the MOF modeling space. It is defined by the MOF meta-metamodel, which in turn is defined by itself. It defines various metamodels, for instance Unified Modeling Language [OMGUML203] or Ontology Definition Metamodel [Djurić05], that are used to describe models that represent things from the real world. The same reality is described in the context of other modeling spaces, like RDF(S) or EBNF spaces. Many software engineers would associate the terms like model and modeling exclusively with UML aristocracy, taking EBNF-based models (Java, C#, C++ code) as more technical, flattened artifacts and ignoble citizens. However, Java (or C++, or some other) code is a model, since it represents some simplified abstraction of reality. The same is with XML code, databases, books, etc – they are all models, but modeled in terms of different modeling spaces, defined by different meta-metamodels.

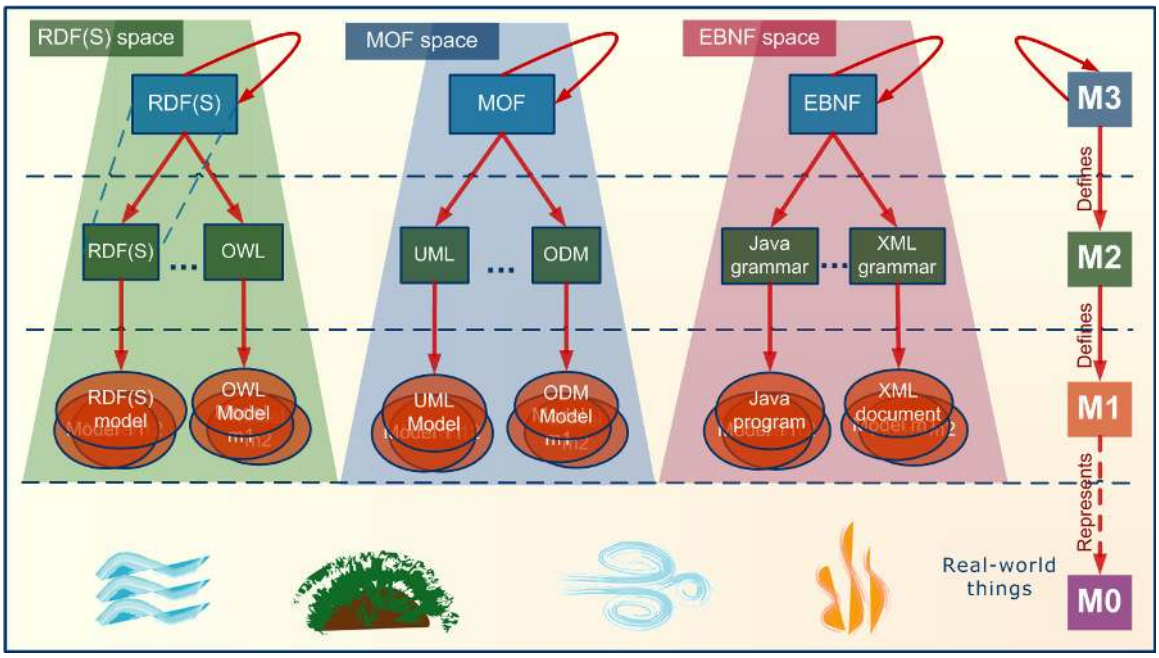


Figure 5 – RDFS, MOF and EBNF modeling spaces

If we model the real world in a certain modeling space, we will use some models. If we model the same reality in another modeling space, we will describe it with different kinds of models, highlighting other characteristics when abstracting from reality. The models from the first modeling space will be a part of reality that we can model using the models from the second modeling space.

Figure 6 clarifies this duality by an example of the same thing being simultaneously a model and a real-world thing. Along the vertical axis, the world is modeled in the MOF

modeling space. Along the horizontal axis is the EBNF space hierarchy, which is a real-world thing in the MOF space. An interesting observation here is that any modeling space, like the EBNF space or even the MOF space itself, is a part of the real world from the MOF-space point of view. In general, the way we model some business system or another "real" domain is pretty much the same as the way we model meta-metamodels, metamodels or models from another modeling space. Of course, these models involve a certain level of abstraction, so there is a possibility of losing some information.

For many software engineers, this duality is complicated to understand at first. Try with a couple analogies. Ghosts do not really exist (well, we hope so!), but they are things from the real world. Some people believe there is life outside of the Solar system, some claim it is a pure fiction, but life outside of the Solar system is a thing from the real world. Otherwise, we would not be able to model it using movies, literature, video games, etc. The fact that M1-M3 layers are fiction and above the M0 layer does not mean that meta-metamodels, metamodels and models are things outside of reality. Everything is in the real world; we just use a convention to put some things in layers, depending on the context.

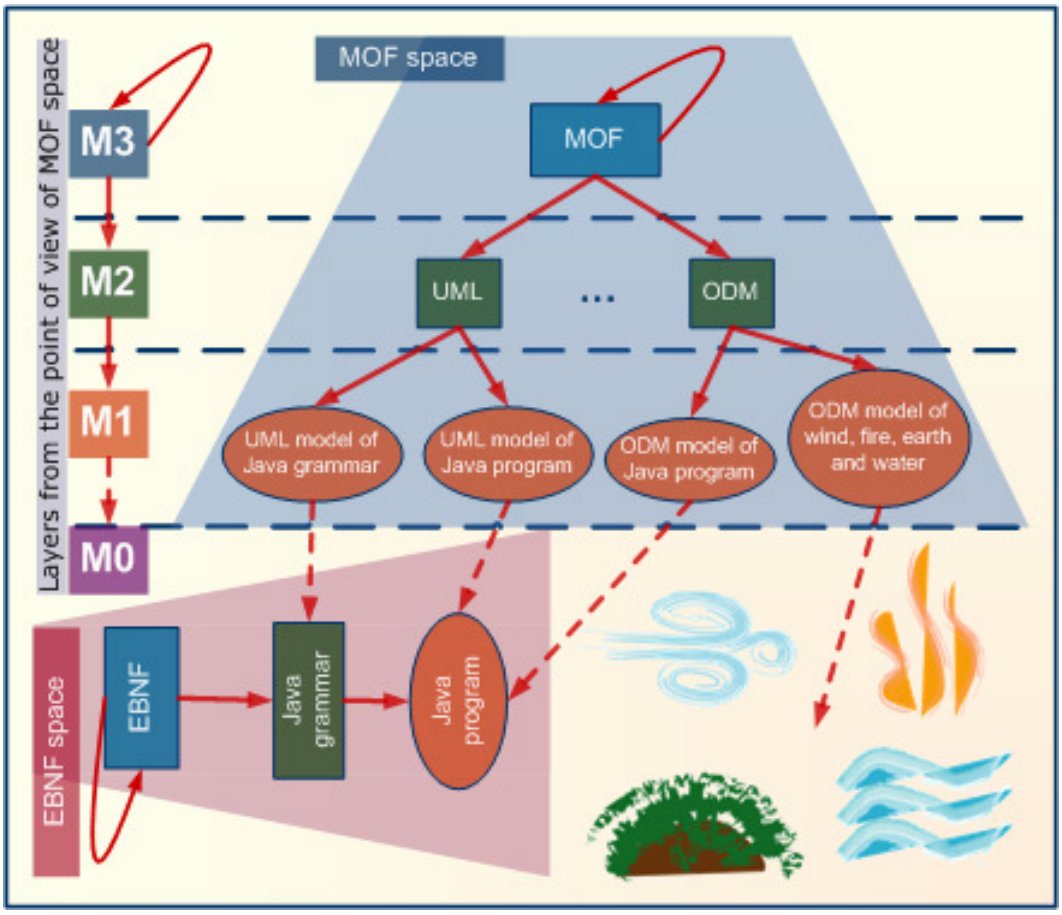


Figure 6 – MOF MS sees EBNF MS as a set of things from the real world

5 MODELING SPACES ILLUMINATED

Modeling spaces can be defined in more or less abstract manner. Some modeling spaces are focused on conceptual (abstract or semantic) things, like models, ontologies, mathematical logics, etc. They are not interested in techniques for representation or sharing their abstractions. We call them *conceptual modeling spaces*. However, we must have some techniques to materialize (or serialize) those modeling spaces. We can do this using *concrete modeling spaces*, which are equipped with notation. Examples of those materializations are some syntax or databases.

Take the MOF space as an example of a conceptual modeling space. The basic concepts of the MOF meta-metamodel – like Class, Association, Attribute, Package and the relations among them – are expressed using those concepts themselves. We can draw them using UML diagrams, but a group of boxes and lines is not a MOF model – it is a drawing of a MOF model. We can serialize them into XMI, enabling computers to process them and programs to share them – but then we leave the MOF modeling space and enter the EBNF modeling space. One can argue that these drawings, i.e. UML diagrams, or models serialized into XMI, are inside the MOF modeling space because they represent MOF concepts. Indeed, they do represent concepts from the MOF space. Simultaneously, they represent other things from the real world. It means that the MOF concepts are modeled in another modeling space.

EBNF is an excellent example of a concrete modeling space. Theoretically well founded, it arguably has some "semantics" primarily for type checking, and has a syntax that is formally specified using a grammar. However, it lacks semantics; when we parse the expression `name = "Mona Lisa"`, we get a syntax tree that does not know that it deals with the name of a painting. We always need some external interpretation of what its abstract syntax means. Actually, this interpretation is given in the corresponding models from other technical spaces that were serialized into the BNF form.

Being able to represent bare syntax, concrete modeling spaces need some means to express the semantics, i.e. the meaning of the data they carry. Conceptual modeling spaces, on the other hand, are able to represent semantics, but need a means to represent their information physically. It is obvious that they should complement each other's representation abilities to create models that have both a semantics and a syntax. One of the most interesting examples of this symbiosis of various modeling spaces can be found in OMG Model Driven Architecture.

We can find similar "modeling patterns" in spoken languages, where people of different nationalities use different languages to express the same meaning. Regardless of the fact that they have the same things in mind (i.e. the same semantics), they need some regulations to share the meanings. Spoken languages have their own syntax for sharing semantics. Just like we can model the same semantics by different modeling spaces, we can talk about the same thing using different spoken languages. Similarly, definitions of mathematical languages (i.e. logics) include two parts: syntax and semantics.

There are two types of usage scenarios for different modeling spaces, both shown in Figure 7:

- *Parallel spaces* – one modeling space models the same set of real-world things as another modeling space, but in another way. In this case, the relation between these modeling spaces is oriented towards pure transformation, bridging from one space to another. Examples of such parallel modeling spaces are MOF and ECore (ECore is a meta-metamodel very similar to but also different from MOF; although ECore is a little simpler than MOF, they are both based on similar object-oriented concepts).
- *Orthogonal spaces* – some modeling space models concepts from another modeling space, taking them as real-world things, i.e. one MS is represented in another MS. This relation is often used in round-trip engineering to facilitate different stages of modeling some system. For example, in order to make a Java program we could first use Java UML profile to create classes and method bodies, then transform this UML model into Java code, and complete the Java program using a Java IDE. Orthogonal modeling spaces are also used when a conceptual modeling space is implemented using a certain concrete modeling space – for example, when one develops a MOF-based repository to run in a Java virtual machine.

Figure 7 shows some of the modeling spaces included in the MDA standards, as well as their relations. It also shows another similar conceptual modeling space, the ECore modeling space (a part of the Eclipse Modeling Framework, EMF) and its relation to the MDA modeling space.

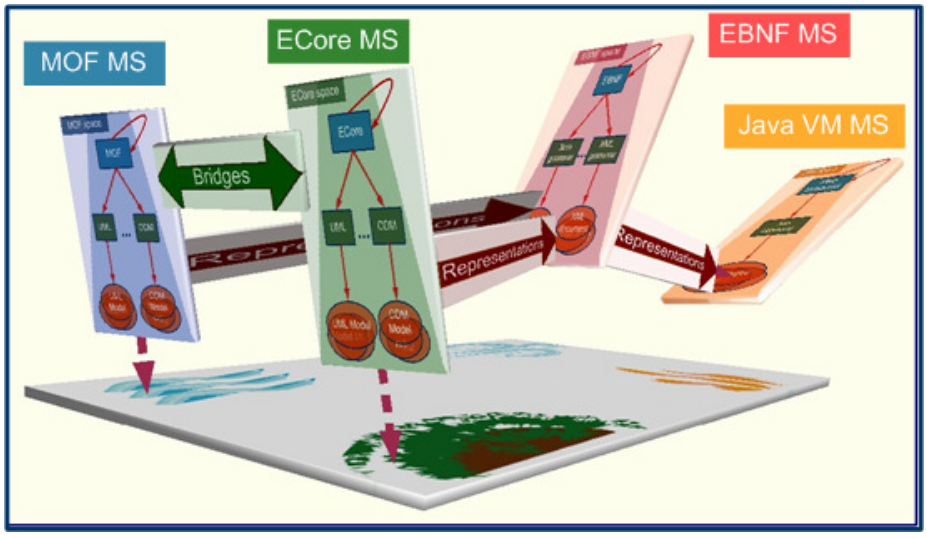


Figure 7 – Two conceptual modeling spaces model the real world in parallel, while a more concrete modeling space represents them

The MOF modeling space is the shining star of MDA. Its meta-metamodel, MOF, defines the core concepts used to specify all the metamodels that MDA is based on (UML, CWM, MOF itself) and many other domain-specific metamodels. Software engineers are often confused about how these concepts are represented. MDA includes a standard for

representation (or serialization) of MOF-based models using XMI (which is an XML-based representation of objects), so when software engineers see XMI they often think it is UML (or MOF). The truth is a little different. The XMI document they are looking at is an XML document that models concepts from the MOF modeling space in the XML modeling space using XML concepts like element, node, etc. XML is then modeled with concrete XML syntax in the EBNF space. The same is with Java Metadata Interface (JMI) standard [Dirckze02]; it models MOF concepts using Java constructs enabling the implementation of MOF-based repositories. Both MOF XMI and JMI standards are in fact standards for modeling the MOF space models in other, concrete modeling spaces.

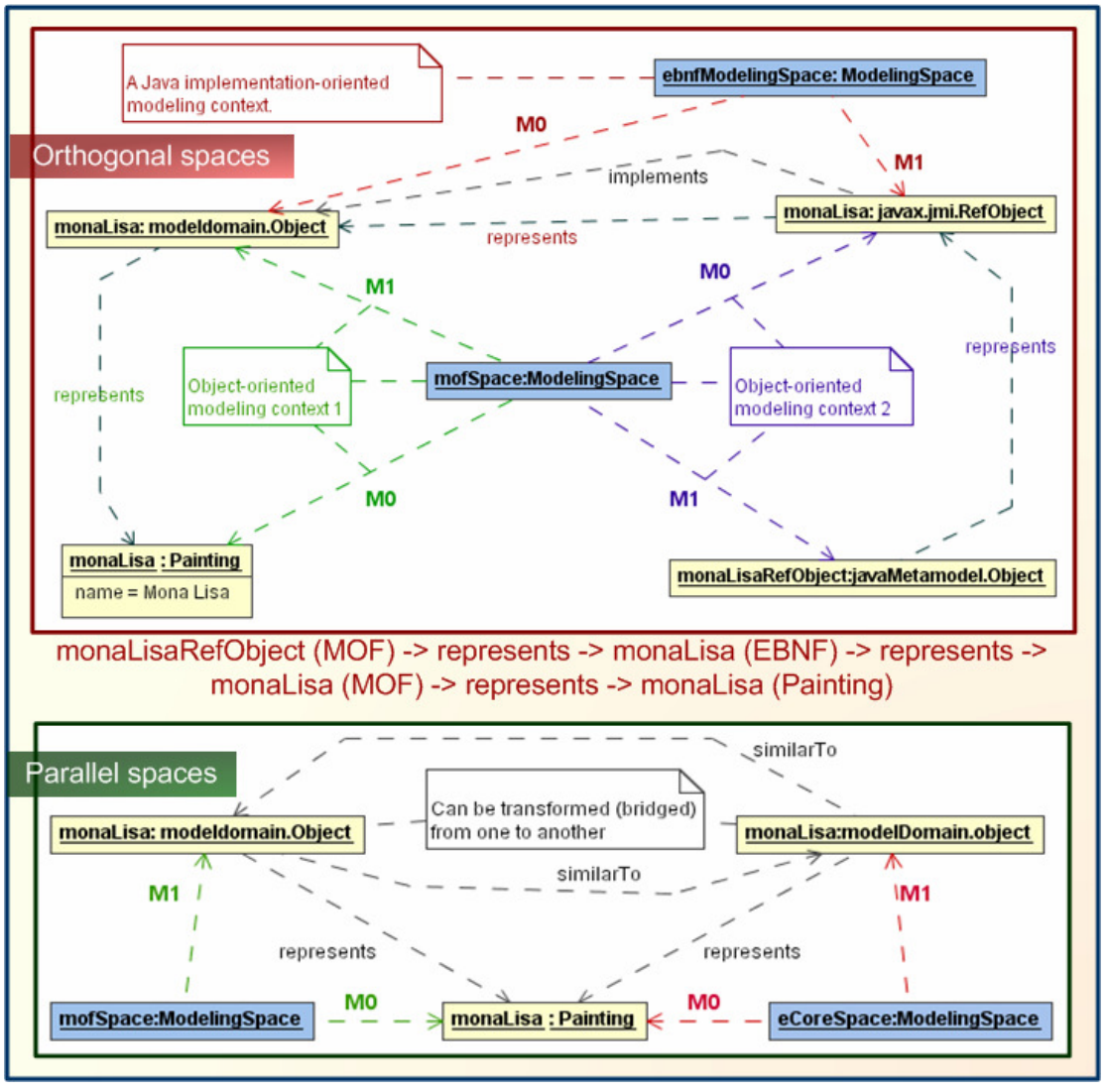
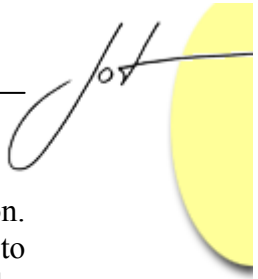


Figure 8 – Orthogonal and parallel modeling spaces

In Figure 8A we can see several modeling spaces included in MDA and their orthogonal, representation-oriented relations that, depending on the context, form a "modeling circle" that can often be confusing. Mona Lisa, the painting, is modeled in the MOF space using the concept of Object. This concept exists only as an idea – the MOF space is a



conceptual modeling space, hence it needs some kind of syntax for representation. Concrete modeling spaces like those based on EBNF (Java, XML) can be used to represent the MOF concepts. That is the second stage of representation in Figure 8 – the object monaLisa from the MOF space is modeled in the EBNF space as the RefObject monaLisa (RefObject is a part of JMI specification). These concrete concepts from XMI and JMI can be (and often are) also modeled using their corresponding MOF-based metamodels or UML profiles, bringing them back to the MOF modeling space. In Figure 8, the monaLisaRefObject is an instance of the corresponding concept from the MOF-based Java metamodel in the MOF modeling space. The MOF space is involved two times in this "chain" of representations. First the model from the MOF space (monaLisa:modeldomain.Object) is at the M1 layer, but later it descends to the M0 layer in the hierarchy although the other MOF model is used at the M1 layer.

However, MDA is not the only standard for model-driven architecture – there is also EMF. Figure 8B shows the same real-world concept, the Mona Lisa painting modeled in two different modeling spaces in parallel. It is often necessary to completely shift from one modeling space to another by means of bridges; there is less space for confusion because one modeling space is translated into another without changing the modeling layers. In this case, metamodels from the MOF modeling space (UML, ODM, etc.) translated into the corresponding ECore-based metamodels will still be at the M1 layer.

6 PRACTICAL ISSUES

Why should we care about all these things?

Engineers can benefit from modeling spaces by getting a better understanding of the vast diversity of things that can be modeled. Moreover, this framework provides a sound modeling foundation for software developers because it explains the roles of different software modeling technologies and how to combine them.

In practice, developers commonly use one or more integrated and complex tools that involve multiple modeling approaches. Remember your last round trip with UML, Java/C# code, specialized technologies like J2EE, database schemas, XML files etc.? You do not always need to think about the relationships between these; sometimes it is a matter of choosing the right option from the menu of your favorite integrated development environment. However, you might find that, even when the tool you use supports changes from one modeling aspect to another, getting the big picture is a pretty complex task. Things get even more complex in situations when the support provided by tools is not straightforward and you need a lot of effort to figure out how to transform one model to another. Sometimes you must do it by hand or you must develop a transformation tool for your specialized case.

Using a number of modeling approaches greatly increases the developers' ability to better describe the problem using the right approach, but is also more complex to comprehend in the whole. Moreover, it can be difficult for software developers to adopt the practice of using a wide spectrum of modeling approaches because they may seem too different and

too unrelated to each other. In order to save time and effort working under tight development schedules, practitioners tend to focus on individual aspects of modeling. For all these reasons, a common view of different modeling approaches is highly desirable.

7 TRANSFORMATIONS

Usage scenarios for parallel spaces most often pertain to conceptual modeling spaces that model the same reality using different concepts. Each of these modeling spaces is implemented in some other, more concrete modeling space, as a represented reality. In order to exchange models between conceptual modeling spaces, it is necessary to provide transformations from one space to another. These transformations are also models [Bézivin03], and should be developed in a modeling space that can represent both the source and the target modeling spaces. Moreover, the transformation also has to be represented in some concrete modeling space orthogonal to the source and the target modeling spaces. That concrete modeling space leads the conceptual model of transformation to its implementation.

Figure 9 shows an example of the previous discussion – parallel conceptual modeling spaces MOF and ECore, and the space orthogonal to them, EBNF, which represents MOF and ECore using XMI. MOF and ECore model the real world in parallel, using some modeling languages (UML, ODM, or other) that are defined using different meta-meta concepts. At the conceptual level, we could establish a transformation from one language to another, e.g. UML to ODM and vice versa, in the same modeling space. An example of a transformation modeling language for such purposes in MOF is Query-View-Transformation (QVT) [OMGMOF2-QVT02]. We can also establish a transformation between modeling spaces, a bridge that transforms concepts from one modeling space into the corresponding concepts belonging to another modeling space. For example, UML concepts defined using MOF can be transformed into the equivalent UML concepts defined using ECore. The UML Class defined using the MOF Class becomes the UML Class defined using the ECore's EClass, and so on.

Both MOF and ECore spaces are represented in other, more concrete modeling spaces. They can be implemented using repositories, serialized into XMI etc., which involves many modeling spaces. For the sake of simplicity, we have skipped a few steps and have shown them as Java program codes and XML documents in the EBNF space. Models from the MOF space are modeled in Java code according to JMI standard [Dirckze02], and in XML according to the MOF XMI. ECore models are modeled in Java in compliance with the EMF framework, and in XML according to the EMF XMI, which is similar but not the same as the MOF XMI. A bridge is also a model; it can be also represented in a concrete modeling space representing meta-metamodels that should be bridged. Examples include an XSLT that transforms a MOF XMI document into an EMF XMI document, a set of Java classes that adapt JMI interfaces to EMF interfaces, and a Java program that does a batch transformation from a JMI-based code to an EMF-based one.

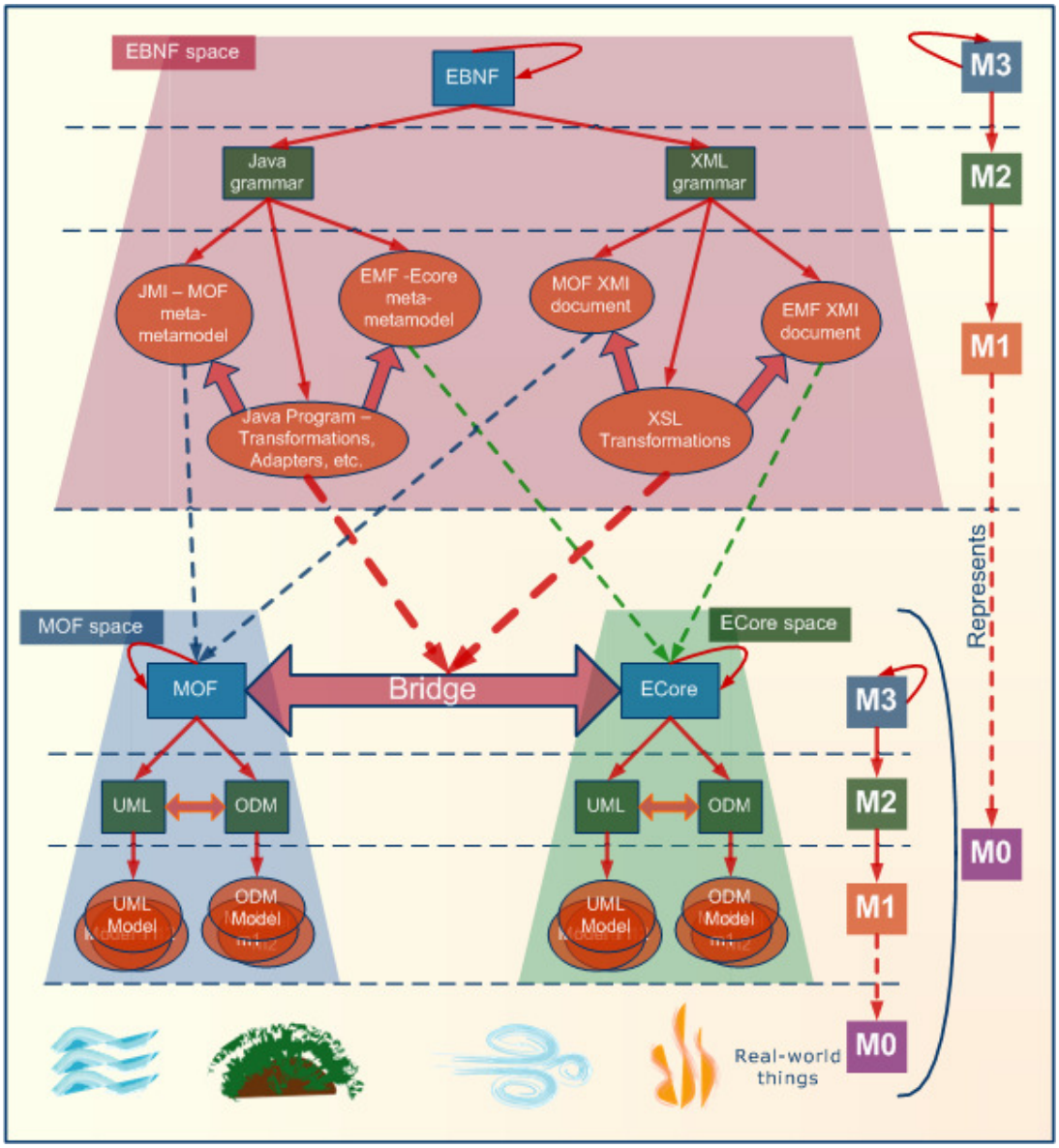


Figure 9 – A bridge between two conceptual spaces and its implementation in a concrete space

As Figure 9 shows explicitly, a single bridge models transformation between two modeling spaces at layer M3, between meta-metamodels. Transformations between metamodels situated in a single modeling space at M2 layer are internal to that modeling space. However, they can be implemented through some concrete modeling spaces (e.g. EBNF for XSLT).

8 MODELING SPACES AND TECHNICAL SPACES

Modeling space is a concept inspired by the concept of *technical space*. Technical space was defined as a working context with a set of additional concepts, body of knowledge,

tools, required skills, and possibilities [Kurtev02]. This fuzzy definition fails in defining technical spaces precisely, although it is often possible to identify them: MDA TS, Ontology Engineering TS, XML TS, Java TS, etc. We can define technical spaces using modeling spaces. In this discussion, we stick to technical spaces in the field of software engineering.

Let us firstly consider a well-known technical space – MDA. We have already noted that MDA includes a set of modeling spaces (MOF, XML, EBNF) and various relations among them that are indeed models. The MDA TS also includes various know-how, literature, etc., which also belong to some modeling spaces (though these modeling spaces are implicit). Do tools belong to a modeling space? In our opinion, software tools can be considered as models, although they are intended to implement some models belonging to some modeling space.

A technical space is a working context that includes various related modeling spaces. Most often a technical space is built around some modeling space, whereas the role of other modeling spaces is supportive (e.g., implementation), or implicit (literature, know-how). For example, the MOF modeling space is at the center of the MDA TS. However, the MDA TS also partially includes other modeling spaces: XML and EBNF in the area of XMI representation, EBNF in the area of repository implementation (JMI), an implicit modeling space that includes literature, etc. Transformations, for example to plain Java, C++ or VB code, are also models belonging to one or several modeling spaces that are partially included in the MDA TS.

Figure 10 shows overlapping between the MDA TS and the Java TS in some modeling spaces. The MDA TS and the Java TS are simplified in the figure for the sake of simplicity; many other modeling spaces can be fully or partially included in these two technical spaces. The MDA TS is built around the MOF modeling space, which resides completely in the MDA TS. The MDA TS also includes JMI based programs, which are parts of the EBNF modeling space, because JMI is intended for implementation of MOF-based repositories in Java. On the other hand, Java TS includes, among others, a part of the EBNF modeling space related to Java grammar and Java programs, including those that are JMI-based. Additionally, Java TS includes parts of the MOF modeling space related to Java metamodel and Java UML Profile, and two-way transformations from these MOF-based models to Java code. These transformations are also a part of the MDA TS. Recall that those transformations are also modeled, so they belong to some modeling spaces as well. Some researches are trying to identify a way to enable transformations between different modeling spaces at the M3 layer using just one two-way transformation for all three layers [Bézivin05].

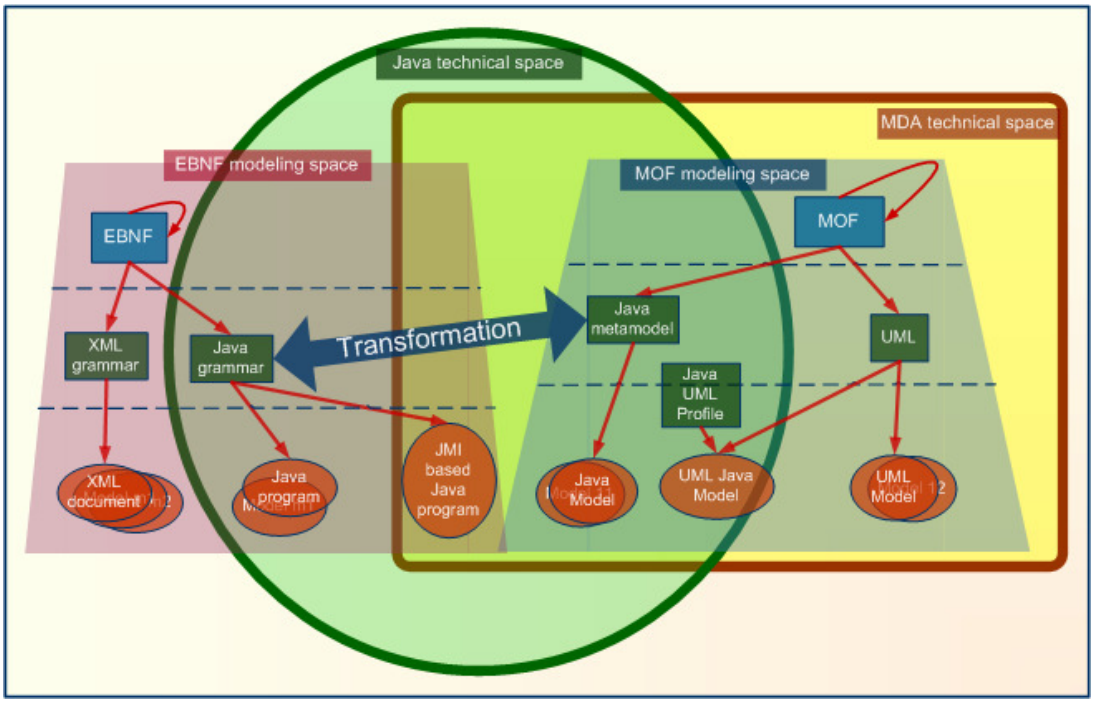


Figure 10 – Technical space comprises one or more modeling spaces

It follows from the above discussion that one technical space includes one or more modeling spaces and that each modeling space is a part of one or more technical spaces, whereas a technical space is a means for grouping modeling spaces that have something in common or simply need to interact. The bridge connecting two modeling spaces is also a means for connecting surrounding technical spaces.

9 A UML MODEL OF MODELING SPACES

In order to have a more formal definition of modeling spaces we decided to define them using UML, since UML is a well-known modeling language in the software community. The description of the modeling spaces discussion from this article is shown in Figure 11 using UML Class diagrams.

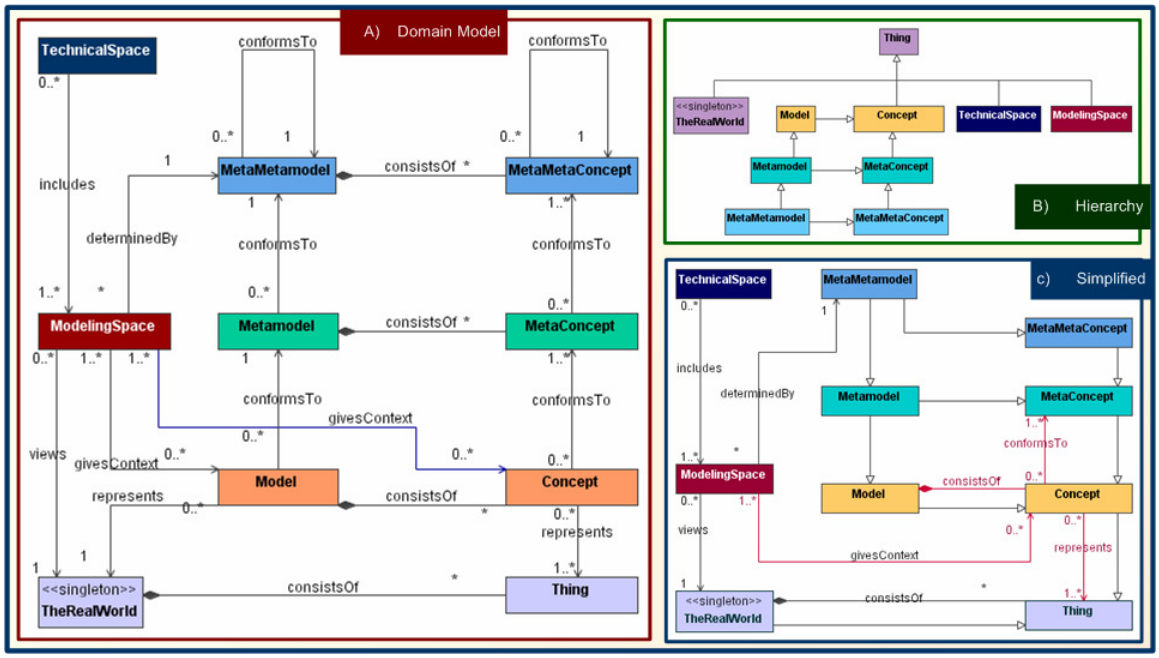


Figure 11 – Modeling spaces modeled in UML. A) domain model, B) hierarchy, C) simplified domain model

Diagram A in Figure 11 shows the initial domain model resulted from this survey. Diagram B shows the inheritance tree of the most important concepts. We can observe some similar relations, for example multiple consistsOf or conformsTo, so we can apply a few common object-oriented techniques to get a simplified "design" of the Modeling Spaces domain model shown in diagram C. Note that diagram C does not show all the inheritance relationships, for example that ModelingSpace inherits from Thing, nor OCL constraints, for the sake of simplicity.

10 MOF IN THE EYES OF EBNF IS XMI

In this section, we try to exemplify the usefulness of modeling spaces for understanding some more practical problems software engineering have to face in their everyday practices. Let us consider XML Metadata Interchange (XMI), an XML based standard for sharing MDA metadata [OMG XMI, 2004]. Although this standard sounds very well-known, most practitioners are confused with this term. We try to explain XMI using Figure 12. The confusion comes from the presence of different XML Schemas, and all of them are called XMI. We usually encounter two kinds of XMI documents, more precisely two XML Schemas defining XMI:

- XML schema for MOF metamodels
- XML schema for UML models

The first one defines the syntax for sharing both MOF-based metamodels and the MOF definition itself. So, we use one schema at two different MDA layers, M3 and M2, thus

for sharing both metametamodels and metamodels. For instance, MOF is defined by itself, so we use MOF XML Schema for describing the XMI document comprising the MOF specification, and this documents is also a part of the MOF standard. Similarly, there is the standard XMI document containing the UML metamodel. However, UML is a modeling language that developers use for describing different models. It is obvious there is a need for an XML Schema for exchanging UML models, and indeed there is the standard one. The name for that XML Schema is the UML XMI Schema. The UML tools such as IBM/Rational Rose, Poseidon for UML, Together, etc. support it, but some researchers report that we always lose some information when sharing UML models between two UML tools [Ambler03].Are the things we mentioned so far complex enough? Yes, we have two different XML Schemas, and use the name XMI for both of them. But, is it the end of XML/XMI Schemas? No, it is not, and some questions raise naturally: What about regular UML models; and is there any relations between UML models and XML documents real-world applications use. Developers know that we never employ UML XML Schema as an XML language in domain applications. We always define domain XML languages, i.e. domain XML Schema. Accordingly, there is a set of rules for mapping UML models into XML Schemas [Grose02]. Thus, one can generate an XML Schema for each UML models, while model instances (i.e. objects) can be shared in accordance with those schemas. Considering the presumption that both UML objects and UML classes are at the same MDA layer (the M1 layer), we regard the generated XML schemas as well as their instance XML documents that are placed at the M1 layer.

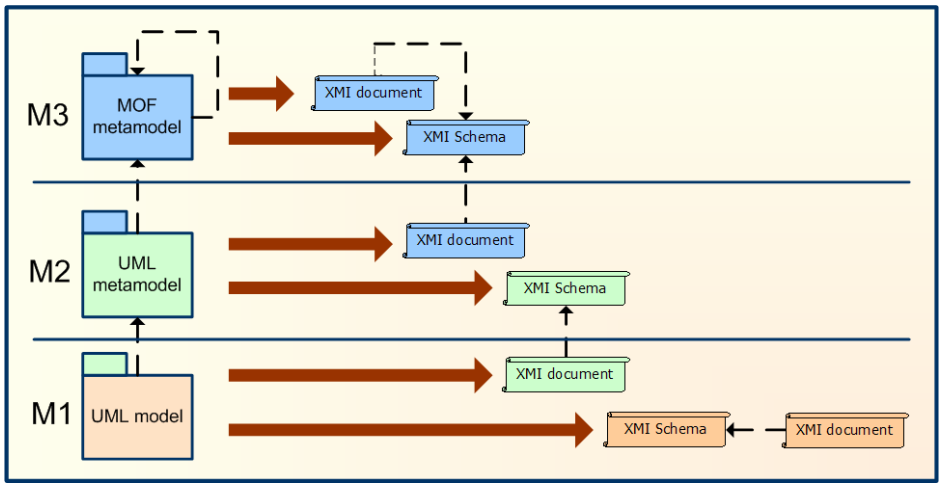


Figure 12 – Mapping MDA metametamodel, metamodels, and models to XMI

Since we have a set of rules for generating XML Schemas from UML models, we can apply the same principle to the upper MDA layers (M2 and M3), so we can produce an XML Schema for each MOF-based metamodel. Using that principles, we generated XML Schema for ODM [Djurić05], while Protégé developers produced XML Schema for the MOF-compatible metamodel of the Protégé ontology editor (http://protege.stanford.edu). Summarizing the story about XMI, we can say XMI is a set of rules for schema generation as well as object serialization.

Although the aforementioned facts about XMI exemplify the relations between some MDA layers and their XML documents or schemas, it is not clear enough how those two modeling spaces see one another. The fist modeling space is MOF – a conceptual one, and the second one is EBNF – a concrete modeling space. In terms of modeling spaces we have a case of orthogonal modeling spaces, since the EBNF modeling space models the MOF modeling space. In Figure 13 we illustrate this case where MOF artifacts are regarded as real-world things modeled using XML. Note that all the previously mentioned XML Schemas and documents are places at the M1 layer of the EBNF modeling space. One could expect this since all of them are models of the MOF meta-layers. It is also important to note that we do not have any equivalencies between the XML meta-schema (i.e. XML grammar) since the M1 layer is intended to be used for representing thing from the reality. In this case MOF conjecture is the modeled reality. Finally, we can say that Figure 13 is equivalent to the content of Figure 12, but it clarifies the way how the EBMF modeling space sees the MOF modeling space.

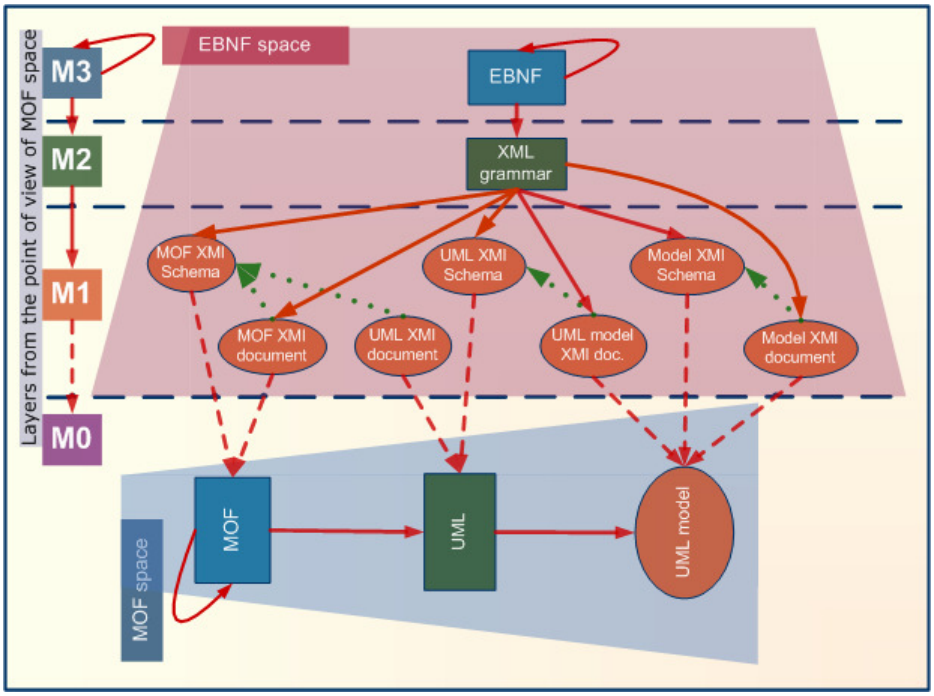


Figure 13 – How EBNF sees MOF – the same as Figure 12

11 CONCLUSIONS

Modeling spaces abstract and generalize a vast amount of diverse modeling approaches, and can help engineers get the big picture of what underlies the software they make or use. Modeling spaces also clarify structures of different modeling approaches, their similarities and dissimilarities, mutual relations, how they work together. By making a clear difference between conceptual and concrete modeling spaces developers can successfully select mechanisms to automate transfer and sharing of information,

knowledge, and even other modeling spaces between different projects and applications. Likewise, understanding specific modeling spaces helps select suitable modeling and development tools in a specific project.

In the future we are planning to employ the approach of modeling spaces to identify a way to enable transformations between different modeling spaces at the M3 layer using just one two-way transformation for all three layers [Bézivin05]. The idea is to avoid building metamodel-based model transformations for each pair of domain languages defined in different modeling, but just to have a bi-directional transformation between modeling spaces at the M3 layer.

12 REFERENCES

[Ambler03] Ambler, S.W., "Agile Model Driven Development Is Good Enough," *IEEE Software*, vol. 20, no. 5, 2003, pp. 71-73.

[Atkinson03] Atkinson, C., Kühne, T., "Model-Driven Development: A Metamodeling Foundation" *IEEE Software*, vol. 20, no. 5, Sep/Oct, 2003, pp. 36-41.

[Berners-Lee01] Berners-Lee, T., Hendler, J., and Lassila, O., "The Semantic Web," *Scientific American*, vol. 284, no. 5, 2001, pp. 34-43.

[Bézivin03] Bézivin, J., Dupé, G., Jouault, F., Pitette, G., Rougui, J.E., "First experiments with the ATL model transformation language: Transforming XSLT into XQuery," *In Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of Model Driven Architecture*, Anaheim, CA, USA, 2003.

[Bézivin05] Bézivin, J., Devedžić, V., Djurić, D., Favreau, J.M., Gašević, D., and Jouault, F., "A M3-Neutral infrastructure for bridging model engineering and ontology engineering," 1st International Conference on Interoperability of Enterprise Software and Applications, Geneva, Switzerland, 2005 (forthcoming).

[Bézivin04] Bézivin, J., "In Search of a Basic Principle for Model Driven Engineering," *Upgrade*, vol. 5, no. 2, 2004, pp. 21-24.

[Corcho01] Corcho, O., Fernández-López, M. and Gómez-Pérez, A., "Technical Roadmap v1.0," OntoWeb Consortium Deliverable D1, http://www.ontoweb.org/download/deliverables/D11_v1_0.pdf, 2001.

[Decker00] Decker, S., Melnik, S., van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Ederman, M. and Horrocks, I., "The Semantic Web: The Roles of XML and RDF," *IEEE Internet Computing*, vol. 4, no. 5, 2000, pp. 63-74.

[Dirckze02] Dirckze, R., "Java Metadata Interface (JMI) Specification Version 1.0," http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html, 2002.

[Djurić05] Djurić, D., Gašević, D., Devedžić, V., "Ontology Modeling and MDA", *Journal of Object Technology*, vol. 4, no. 1, 2005, pp. 109-128.

[Gašević05] Gašević, D., Djurić, D., Devedžić, V., "Bridging MDA and OWL," *Journal of Web Engineering*, vol. 4, no.2, 2005, pp. 118-143.

[Grose02] Grose T., Doney, G., Brodsky, S., *Mastering XMI – Java Programming with XMI, XML, and UML*, John Wiley & Sons, Inc., New York, USA, 2002.

[Hagget67] Hagget, P. and Chorley, R.J., "Models, Paradigms and New Geography," *In Models in Geography*, Methuen & Co., London, UK, 1967.

[Hendler01] Hendler, J., "Agents and the Semantic Web," *IEEE Intelligent Systems*, vol. 16, no. 2, 2001, pp. 30-37.

[Kurtev03] Kurtev, I. and van den Berg, K. (2003), "Model Driven Architecture based XML Processing," *In Proceedings of the ACM Symposium on Document Engineering*, Grenoble, France, 2003, pp. 246-248.

[Kurtev02] Kurtev, I., Bézivin, J., Aksit, M. "Technological Spaces: An Initial Appraisal", In Proceedings of the Confederated International Conferences CoopIS, DOA, and ODBASE 2002, Industrial track, Irvine, CA, USA, 2002. http://wwwhome.cs.utwente.nl/~kurtev/files/TechnologicalSpaces.pdf

[Miller03] Miller, J., Mukerji, J. (eds.), "MDA Guide Version 1.0.1," OMG Document: omg/2003-06-0, http://www.omg.org/docs/omg/03-06-01.pdf, 2003.

[OMGMOF203] Meta Object Facility (MOF) 2.0 Core Specification version 2.0 Final Adopted Specification, OMG Document ptc/03-10-04, http://www.omg.org/cgi-bin/apps/doc?ptc/03-10-04.pdf, 2003.

[OMGMOF2-QVT02] MOF Query/Views/Transformations First Revised Submission, OMG Document ad/03-08-03, http://www.omg.org/cgi-bin/apps/doc?ad/03-08-03.pdf, 2003.

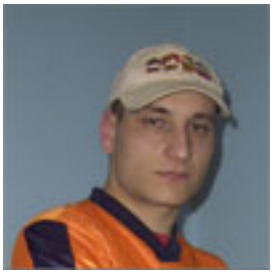
[OMGODM03] Ontology Definition Metamodel Request for Proposal, OMG Document: ad/2003-03-40, http://www.omg.org/cgi-bin/doc?ad/2003-03-40,(2003)

[OMGUML203] Unified Modeling Language: Superstructure, version 2.0, Final Adopted Specification, OMG Document ptc/03-08-02, http://www.omg.org/cgi-bin/apps/doc?ptc/03-08-02.zip, 2003.

[OMG XMI, 2002] OMG XMI Specification, v1.2, OMG Document formal/02-01-01, http://www.omg.org/cgi-bin/doc?formal/2002-01-01, 2002

[Seidwitz03] Seidewitz, E., "What Models Mean," *IEEE Software*, vol. 20, no. 5, 2003, pp. 26-32.

About the author(s)



Dragan Djurić ,PhD, is a free scientist and a member of the GOOD OLD AI research group. His interests mostly include modeling, enterprise software architecture, object-oriented development, Java platform, Semantic Web, and intelligent information systems. He can be reached at dragan@dragandjuric.com.

Dragan Gašević is a postdoctoral fellow at the School of Interactive Arts and Technology, Simon Fraser University Surrey, BC, Canada. His research interest mainly include ontologies, ontology mappings, Semantic Web, learning technologies, knowledge engneering and software engineering. He can be reached at dgasevic@acm.org.

Vladan Devedžić is an associate professor of computer science at the Department of Information Systems, FON - School of Business Administration, University of Belgrade, Serbia and Montenegro. His main research interests include software engineering, intelligent systems, knowledge representation, ontologies, Semantic Web, intelligent reasoning, and applications of artificial intelligence techniques to education and medicine. He can be reached at devedzic@etf.bg.ac.yu.