An M3-Neutral infrastructure for bridging model engineering and ontology engineering

J. Bézivin[(1)], V.Devedzic[(2)], D. Djuric[(2)], J.M. Favreau[(1)], D. Gasevic[(2)], , F. Jouault[(1)]

[(1)]ATLAS Group (INRIA & LINA), University of Nantes
2, rue de la Houssinière - BP 92208 44322 Nantes Cedex 3, France
http://www.sciences.univ-nantes.fr/lina/atl/
[(2)]GOOD OLD AI Group, FON – School of Business Administration, University of Belgrade
POB 52, Jove Ilića, 11000 Belgrade, Serbia and Montenegro
http://goodoldai.org.yu

Abstract. In this paper we start from the conjecture that a convenient organization of various technical spaces in three "metamodeling" layers offers a convenient working environment. The main message of this work is that it is possible to consider software engineering and ontology engineering as two similarly organized areas, based on different metametamodels (M3-level). Consequently, building bridges between these spaces at the M3-level seems to offer some significant advantages that will be discussed in the paper. The initial results presented in this paper may also apply to other technical space as well.

1. Introduction

Model driven engineering (MDE) is being considered as an important departure from traditional techniques in such areas as software engineering, system engineering and data engineering. In software engineering, the MDA™ approach proposed by OMG in November 2000 allows separation of platform dependent from platform independent aspects in software construction and maintenance. More generally, MDE is proposing to use models to capture specific aspects of a system under construction or maintenance, not only the business and platform aspects.

Recently a new variant of MDE called Software Factories [Greenfield, 2004] has been proposed by Microsoft. The software factories approach is different from MDA mainly because it focuses more on small, well defined and engineered DSLs (Domain Specific Languages) than on the existence of a huge multi-purpose modeling language like UML 2.0. Apart from that, the Microsoft approach is completely in line with the general MDE ideas.

Technical spaces have recently been introduced as a means to structure the solution space and to figure out how to work more efficiently by using the best possibilities of different technologies, including MDA [Kurtev et al 2002]. Nowadays, using only a single

technology in solving different engineering problems is usually not enough. For example, software engineers can benefit from ontological engineering, or database developers can find useful improvements in using the XML technology.

In this paper we propose the idea that it should be possible to establish generic coordination between different technical spaces by making explicit the M3-level properties and providing domain-independent transformation facilities at this level. This would be more efficient than providing ad-hoc, case-by-case transformation between various DSLs belonging to the same or different technical spaces.

This paper is thus organized as follows. In section 2 we introduce some general considerations on the three-layer conjecture. Section 3 presents the domain of ontology engineering. In Section 4, we show how the idea of defining bridges between these spaces at the M3-level may bring a lot of significant economies and other advantages. Section 5 provides more in-depth thinking about mappings between ontology engineering and model engineering. In particular it introduces the conceptual notion of projectors to define bridges between technical spaces. In section 6 we compare the proposed M3 mapping techniques to more conventional M2-mappings. Finally we conclude by summarizing the project goals and sketching possible extension paths.

2. The 3-Layer Conjecture

In this section we recall the main characteristics of the three-layer conjecture and introduce the term of technical space. We are also discussing Model Driven Architecture (MDA), an important technical space widely accepted by software industry.

2.1. MDA basics

In November 2000 the OMG proposed a new approach to interoperability named MDA™ (Model-Driven Architecture) [9]. MDA is one example of a much broader approach known as Model Driven Engineering encompassing many popular research trends like generative programming, domain specific languages, model-integrated computing, model management and much more.

The central part of MDA is the layered architecture that has a number of standards defined at each of its layers (see Figure 1). Most of MDA standards are developed as metamodels using metamodeling. The topmost layer (M3) is called metametamodel and the OMG's standard defined at this layer is Meta-Object Facility (MOF). This is a self-defined language intended for defining metamodels. In terms of MDA, a metamodel makes statements about what can be expressed in the valid models of a certain modeling language. In fact, a metamodel is a model of a modeling language [Seidewitz, 2003]. Examples of the MDA's metamodels are UML and Common Warehouse Metamodel (CWM). The MDA's metamodel layer is usually denoted as M2. At this layer we can

define a new metamodel (e.g., a modeling language) that would cover some specific application domains (e.g., ontology development, object-oriented programming, data base handling, etc.). The next layer is the model layer (M1) – the layer where we develop real-world models (or domain models). In terms of UML models, that means creating classes, their relations, states, etc. There is an XML-based standard for sharing metadata that can be used for all of the MDA's layers. This standard is called XML Metadata Interchange (XMI).

The bottom layer is the instance layer (M0). There are two different approaches to explaining this layer, and we note both of them:

1. The instance layer contains instances of the concepts defined at the model layer (M1), e.g. objects in programming languages.
2. The instance layer contains things from our reality – concrete (e.g. Mark is an instance of the Person class, Lassie is an instance of the Dog class, etc.) and abstract (e.g. UML classes – Dog, Person, etc.) [Atkinson & Kühne, 2003]. Other authors also mentioned that difference, like Bezivin [Bezivin, 2004], who says that the M0 layer covers program executions as well.

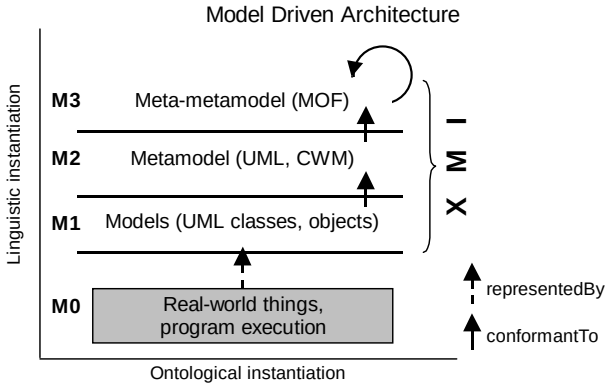


Fig. 1. The four-layer Model Driven Architecture and its orthogonal instance-of relations: linguistics and ontological

In this paper we advocate the second approach, but we should give more details about its impact on UML. In UML, both classes and objects are at the same layer (the model layer) in the MDA four-layer architecture. Actually, MDA layers are called linguistic layers. On the other hand, concepts from the same linguistic layer can be at different ontological layers. Hence, UML classes and objects are at different ontological layers, but at the same linguistic layer. We give an excerpt from the Petri net ontology [Gašević & Devedžić, 2004] as an illustration of this approach (see Figure 2). There is the concept of modules in Petri nets that means components of Petri net models we can reuse in other different Petri net models. In fact, Petri net modules are similar to UML classes – it is a sort of template that we use for creating module instances. However, both of them we

modeled using UML classes, but there was a need to make difference between them. So, we modeled modules as a metaclass, while module instances as a regular class. Note that we did not change UML metamodel by defining Module metaclass, but we just created ontological metaclass. That means, modules are at the O2 ontological layer, while module instances are at the O1 ontological layer. Finally, module instances represent (model) real Petri net modules. As Petri nets themselves are modeling tool, they are at the O0 ontological layer, but sill at the M1 (i.e. model) linguistic layer.

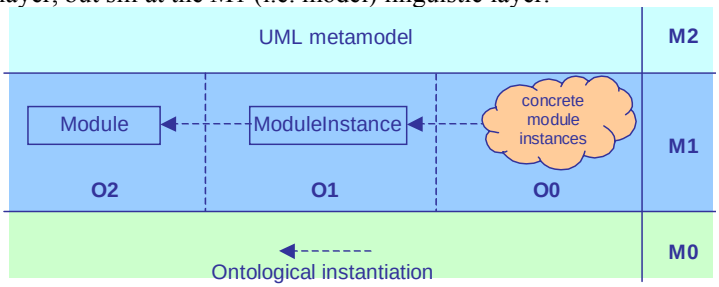


Fig. 2. An Illustration of ontological layers: An excerpt of the Petri net ontology consisting of three ontological layers (O2, 01, O0) at the same linguistic layer.

2.2. Technical spaces

Technical spaces were introduced in [Kurtev et al 2002], in the discussion on problems of bridging different technologies. 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). Each technical space can be organized on a metametamodel (explicit or implicit) and a collection of metamodels. For the OMG/MDA the MOF and the collection of standard metamodels and UML profiles play this role.

The basic assumption in MDE is the consideration of models as first class entities. A model is an artifact that conforms to a metamodel and that represents a given aspect of a system. These relations of conformance and representation are central to model engineering [2]. A model is composed of model elements and conforms to a unique metamodel. This metamodel describes the various kinds of contained model elements and the way they are arranged, related and constrained. A language intended to define metamodels and models is called a metametamodel.

The OMG/MDA proposes the MOF (Meta Object Facility) as such a language. The Eclipse metametamodel is part of EMF and is compatible with MOF 2.0. This language has the power of UML class diagrams complemented by the OCL assertion and navigation language.

There are other representation systems that may also offer, outside the strict MDA or object-oriented boundaries, similar model engineering facilities – other technical space [3]. They are often based on a three level organization like the metametamodel, metamodel and model of the MDA. One example is grammarware [5] with EBNF, grammars and programs but we could also consider XML documents, Semantic Web, DBMS, ontology engineering, etc. A Java program may be considered as a model conforming to the Java grammar. As a consequence we may consider strict (MDA)-models, i.e. MOF-based like a UML model but also more general models like a source Java program, an XML document, a relational DBMS schema, etc.

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 (e.g. in the XML technical space by using XSLT, in ontological engineering through transformations that can be mapped into XSLT, etc.). 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.

The main role of the M3-level is to define the representation system for underlying levels. The MOF for example is based on some kind of non-directed graphs where nodes are model elements and links are associations. The notion of association end plays an important role in this representation system. Within the grammarware space we have the specific representation of abstract syntax trees while within the XML document space we also have trees, but with very different set of constraints.

Associated to the basic representation system, there is a need to offer a navigation language. For MDA the language that plays this role is OCL, based on the specific nature of MDA models and metamodels. OCL for example know how to handle association ends. For the XML document space, the corresponding notation is XPath that takes into account the specific nature of XML trees. As a matter of fact OCL is more than a navigation language and also serves as an assertion language and even as a side-effect free programming language for making requests on models and metamodels. At the M3-level when the representation system and corresponding navigation and assertion notations are defined, there are also several other domain-independent facilities that need to be provided. In MDA for example generic conversion bridges and protocols are defined for communication with other technical spaces:

- XMI (XML Model Interchange) for bridging with the XML space
- JMI (Java Model Interchange) for bridging with the Java space
- CMI (Corba Model Interchange) for bridging with the Corba space

Obviously these facilities may evolve and provide more capabilities to the MDA technical space. We may even see many other domain-independent possibilities being available at the M3-level like general repositories for storing and retrieving any kind of model or metamodel, with different access modes and protocol (streamed, by element navigation, event-based, transaction based, with versioning, etc.).

3. Ontologies, ontology languages and their architecture

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 means for knowledge sharing and reuse among knowledge-based systems. However, even an early survey of the field of ontologies [Fridman-Noy, 1997] has identified a number of application classes that benefit to a large extent from utilizing ontologies although some of them are not necessarily knowledge-based systems in the traditional sense. Later on, researchers have recognized explicitly that ontologies are not just for knowledge-based systems, but for all software systems – all software needs models of the world, hence can make use of ontologies at design time [Chandrasekaran et al, 1999]. Nowadays, ontologies and ontological engineering span such diverse fields as qualitative modeling, language engineering, database design, information retrieval and extraction, knowledge management and organization, ontology-enhanced search, possibly the largest one, e-commerce (e.g., Amazon.com, Yahoo Shopping, etc.), and configuration [McGuinness, 2002].

3.1. Ontological engineering

The engineering part of developing ontologies comprises a complex set of activities that are conducted during conceptualization, design, implementation and deployment of ontologies. Ontological engineering covers a whole range of topics and issues, such as the basics (philosophical and metaphysical issues and knowledge representation formalisms), methodology of ontology development, recent Web technologies such as XML [Bergholz, 2000] and its relatives [Klein, 2001], business process modeling, commonsense knowledge, systematization of domain knowledge, Internet information retrieval, standardization, evaluation, ontology integration with agents and applications, and many more [Devedžić, 2002]. It also gives us design rationale of a knowledge base, helps us define the essential concepts of the world of interest, allows for a more disciplined design of a knowledge base, and enables us to accumulate the knowledge about it. The disciplines tightly interwoven with ontological engineering include modeling, metamodeling, and numerous fields of software engineering.

3.2. Ontology languages

Several ontology languages have been developed during the last few years [Gómez-Pérez & Corcho, 2002]. In the first time, the most used ontology languages were Lisp-like ones, like Knowledge Interchange Format (KIF), Loom, Algernon, etc. By combining Semantic Web and XML technologies, many XML-based ontology languages have been defined like Ontology Exchange Language (XOL), SHOE, and Ontology Markup Language

(OML). On the other hand, we have the languages Resource Description Framework (RDF) and RDF Schema as general languages for the description of metadata on the Web [Corcho et al, 2001]. The Web Ontology Language (OWL) is a current semantic markup language for publishing and sharing ontologies on the WWW adapted 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.

We have tried to identify different meta-layer in definitions of ontology languages as it is given for MDA. However, there is not any similar widely accepted architecture, so we draw the architecture in Figure 3. Generally, ontologies covers knowledge that represents (or models) some domains from the reality. In order to find relations with MDA-based modeling languages we can say that an ontology: from one hand, consists of ontology structure: definitions of concepts, their mutual relations, properties, and restrictions (i.e. axioms); on the other hand, consist of instances (individuals) created in accordance to the ontology structure. Accordingly, we can say that the place of ontologies is the M1 (model) layer. We can say that ontology languages are defined at the M2 layer similarly to MDA's metamodels. In this case we call those definitions metaontology. Examples of metaontologies are: the Frame ontology developed by Gruber [Gruber, 1993] or the specification of the OWL language [Bechhofer et al, 2004]. The main difference from MDA's metamodels is that there is no any general language that we use for defining ontology languages, or more simply, there is no any explicitly defined metametaontology at the M3 layer. However, we can assume one of the present ontology languages (e.g. OWL) as a metametaontology since we can define semantics of other ontology languages by using that presumed metametaontology. Of course, the meaning of this figure is not completely true, as some of ontology languages do not make differences between meta-layers, while in some of them we can define that a class is an instance of the other class (e.g. OWL Full). However, developers mainly see ontology languages as we do, so we can say that the architecture in Figure 3 represents a pragmatic architecture of ontologies languages that allow us to find similarities to the MDA architecture.

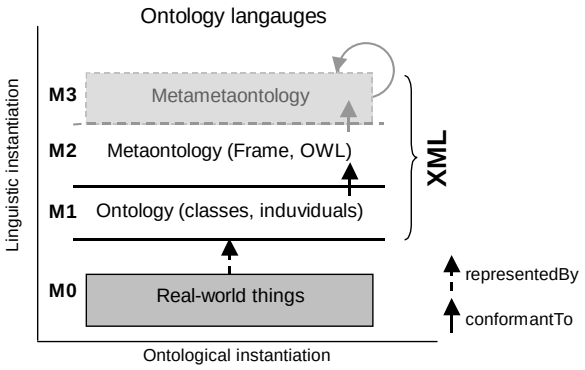


Fig. 3. Meta-layers in terms of ontological engineering

3.3. MDA-based ontology languages

Since most of practitioners are mainly unfamiliar with ontology development techniques, many researches have proposed using well-known tools and techniques (e.g. UML) for ontology development as a solution of this problem [Kogut et al, 2002]. The problem of using UML for ontology development has been firstly addressed in [Cranefield, 2001]. In fact, this was a pioneering work in integrating MDA and ontologies. Currently, there is an OMG initiative aiming to define a suitable language for modeling Semantic Web ontology languages in the context of MDA [OMG ODM, 2003]. In the context of that initiative we present in Figure 4 our proposal for such an architecture [Djurić et al, 2005]. The key components in the architecture are:

- Ontology Definition Metamodel (ODM);
- Ontology UML Profile (OUP) – a UML notation for ontology definition;
- Two-way mappings between OWL and ODM, ODM and OUP, and from OUP to other UML profiles.

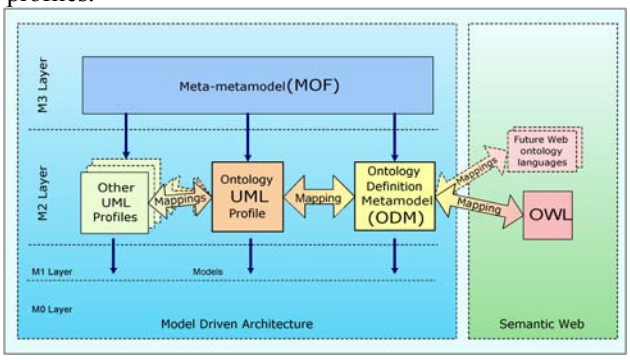


Fig. 4. Ontology modeling in the context of MDA and the Semantic Web

However, the idea of using MDA concepts for ontological engineering is not only present in the OMG's initiative, but some important ontology tools (i.e. editors) also use the same principles. Probably the leading ontological engineering tool Protégé has a formally defined MOF-based metamodel. This metamodel is extensible and adaptable. This means that Protégé can be adapted to support a new ontology language by adding new metaclasses and metaslots into a Protégé ontology. Introduction of these new metamodeling concepts enable users to add necessary ontology primitives (e.g. the Protégé class has different features from OWL class). In that way it can, for instance, support RDFS [Noy et al, 2000] or OWL.

3.4. Ontology technical space

As we have already explained MDA technical space we describe ontology technical spaces as well. Starting from the main intention of ontologies to be used for sharing knowledge we can say that ontologies do not exist without any knowledge about surrounding technical spaces. According to the Semantic Web initiative ontologies are based on XML and RDF(S), and thus we use XML technologies for sharing ontologies. This also assumes that ontologies are related with grammarware and EBNF as MDA is. There is another relation between ontologies and grammarware – APIs for manipulation of ontologies (e.g. Protégé's API, Jena, etc.), or repositories of ontology-aware knowledge in the form of databases (e.g. DBMS technical space). On the other hand, the presented ontology metamodels (ODM and OUP) are MOF-compliant languages defined in the context of the MDA's metamodeling architecture. In that way, ontologies are connected with MDA technical space. Since we can use JMI for accessing to MOF-based repositories we have an alternative way that connects ontology technical space with grammarware technical space. Of course, we have not listed all possible technical spaces the ontology technical space has connection with. Our idea is to show the importance of defining relations between ontologies and other different technical spaces. In the rest of the paper we discuss the transformations between model engineering and ontological engineering.

4. Relations between technical spaces

Generally, each technical space has different purposes. For instance, ontological engineering technical space covers shared knowledge; while MOF-based modeling languages (e.g. UML) do not necessarily represent domain knowledge. However, we can easily recognize some overlaps between different technical spaces. That means, the same real-world thing can be captured by different technical spaces. For example, UML models

in model engineering technical space represent things from the reality. On the other side we use XMI for sharing UML models, so the same UML model can be also represented in the grammarware (i.e. EBNF) technical space.

4.1. Classification of technical spaces

Following the previous discussion we can classify technical spaces into two parallel groups of technical spaces:

1. Conceptual technical spaces – are about conceptual (abstract or semantic) things, like models, ontologies, mathematical logics, etc. They are not interested in techniques for representation of sharing their abstractions. However, we must have some techniques to materialize (or serialize) those technical spaces.
2. Concrete technical spaces – have techniques that allow us to have more material (i.e. physical, syntactical) representation of conceptual things. The examples of these technical spaces are different kind of grammars (e.g. KIF and XML for ontologies, XMI for MOF-based models). Also, MOF-based repositories are often based on DBMSs for storing metamodels, so we represent the model engineering (conceptual) technical space in the context of the DBMS (concrete) technical space.

Very often developers have misunderstanding of these two kinds of technical spaces. For example, most of beginners in ontological engineering do not understand difference between XML and ontologies. The problem is that most of current ontology editors and systems is based on XML, but generally speaking the same ontology can be represented by many different grammars. We can represent the same ontology semantics by many XML schemas [Decker et al, 2000]. Furthermore, those grammars should not be XML-based. One example is the regular-text based M3 notation. Finally, we can store the ontologies into DBMSs, so we can represent ontologies in the DBMS technical space.

4.2. Mappings between technical spaces

According to the previous classification we give the following conclusions regarding mappings between different conceptual technical spaces:

1. Mappings between two conceptual technical spaces. We can only define mappings at the conceptual level. The mappings defined at the conceptual level give recommendation for their implementation.
2. A physical implementation of mappings between two conceptual technical spaces always must be done through a concrete technical space.

From the fist statement we recognize logical relations between two technical spaces. That means, we find out epistemological equivalences between them. Since we are trying to have mappings at the M3 layer we need to represent each technical space as a three-layer architecture. Afterwards we specify their mutual mappings at the M3 layer. Those

mappings should be prorogated to lower meta-levels (i.e. M2 and M1) of the three-layer architecture.

The second conclusion states that we always must use concrete technical space for implementing mappings between conceptual spaces. For example, implementation of mappings between ontological engineering and model engineering can be through the EBNF technical space as both of these technical spaces have XML bindings. However, the statement about implementation is also valid for mappings inside the same technical space. For example, in case of mappings between ODM and UML (i.e. inside the model engineering technical space) we could implement through either the EBNF technical space or the DBMS technical space. The EBNF technical space we use in a case we are dealing with XMI documents. In this case we can use either XSLT or a programming language as an implementation tool. The DBMS technical space we use when we have a MOF-based repository stored in a DBMS. We can use a MOF2 QVT language in this case if the repository supports one of them.

5. Mappings between ontological engineering and model engineering

In this section we try to identify ways for connecting different technical spaces in the main emphasis on the relations between ontological engineering and model engineering. This problem is mainly being solved partially by defining a pair of transformations between languages from two different technical spaces we are going to connect to (e.g. ODM from MDA technical space and OWL from ontology technical space). That mainly means developing transformations according to M2 layer, so that we can transform models at M1 layer. In fact, this is represented by principle of metamodel-driven model transformations [Bézivin, 2001]. One important question is: should we always define as a pair of mappings between different languages belonging to different technical space, or can we find some commonalities among all of them? We believe that if we can achieve mappings between technical spaces at the M3 layer we can get many significant economies and other advantages. The most important one is the decrease of the number necessary transformations for bridging different technical spaces.

5.1. Epistemological relations between technical spaces

In order to clarify mappings between ontological engineering and model engineering we start from the organizational architectures of their languages (Figure 1 and Figure 3).

From those figures we can deduce that either of them consist of three layers. Epistemologically, those layers are equivalent[1]:

ME.M3 ⇔ OE.M3

ME.M2 ⇔ OE.M2

ME.M1 ⇔ OE.M1

In order to illustrate these statements we use the MOF-based ontology language (ODM) as a language from the model engineering technical spaces. It is important to note that ODM is defined at M2 layer. On the side of ontological engineering we have a definition of an ontology language (e.g. OWL) at the M2 layer. Concrete real world models are at M1 layer and they consist of classes and their instances for both model engineering and ontology engineering technical spaces. That means, we must not have one ontological layer at M1 layer according to [Atkinson & Kühne, 2003], and we have two ontological layers: one for classes and one for class instances (i.e. objects).

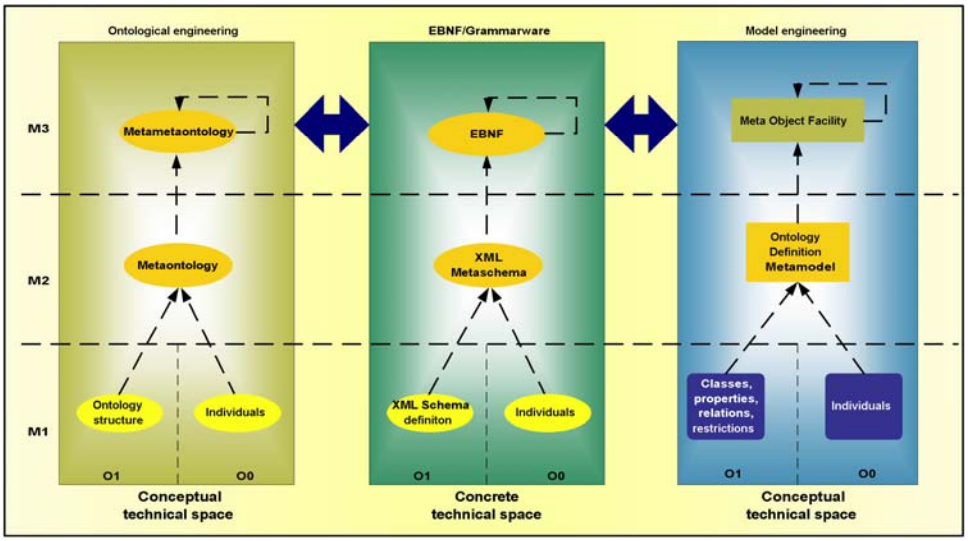


Fig. 5. Mappings between ontological engineering and model engineering through the EBNF technical space

5.2. Pragmatic relations between technical spaces

We see here the high potential impact of considering these technical spaces as explicit and semi-formal entities. In most of these spaces we have for example internal

[1] In this example we use the following notation: name-of-technical-space.meta-level, namely ME.M3 means the M3 layer of the model engineering technical space.

transformation tools (e.g. XSLT and XQuery for XML, QVT for MDA, etc.). Some of these internal transformation tools are general and other are specialized (a compiler can be seen as a specialized transformation tool of the EBNF/Grammarware space). These transformation tools have evolved in their own context to fit with specific objectives and the main representation system of the corresponding space and there is no reason to change that. Now we have to consider another kind of transformation: across technical space boundaries. Let us call these transformation projectors in order to distinguish them from other transformations internal to one technical space.

The responsibility to build projectors lies in one space. The rationale to define them is quite simple: when one facility is available in another space and that building it in a given space is economically too costly, then the decision may be taken to build a projector in that given space. There are two kinds of projectors according to the direction: injectors and extractors. Very often we need a couple of injector/extractor to solve a given problem.

In order to illustrate this situation, let us look at the MDA technical space. The main entity there is a model (a metamodel may be considered as a kind of model). A model contains very useful and focused information, but by itself it is very dull and has no much capability. If we want MDA models to be really useful we have to give them these capabilities. There are two ways to do this: either to build them in the MDA space or to find them in another space. In the latter case what we'll have to provide is some set of projectors.

An MDA model is a graph (non directed graph with labeled edge ends). Since there was no possibility to exchange MDA models, the OMG started a RFP called SMIF (Serial Model Interchange Format). The objective of SMIF was to find a serialization scheme so that any kind of MOF model could be exchanged by simple means (by mail, or a USB key, etc.). After some months of study, the group leading this initiative identified several solutions based on well known graph serialization algorithms. The solution was then to select and standardize some of these algorithms and to suggest building software extensions to handle these standards as part of the major CASE tools. This was the time when some people realized the importance that XML was taking and the growing availability of XML tools in various industrial environments. Many people then realized that it would be economically much more interesting to define standard serialization in XML, i.e. that instead of serializing graphs on text flow, it was more interesting to serialize graphs as trees and the let the remainder of the work being handled in the XML space. As a consequence a bidirectional projector was defined by the XMI convention.

Each MDA projector has a specific goal, i.e. it consists in providing new facilities to models that are available in other technical spaces. XMI brings the capability of global model exchange to the MDA space and this capability is found in the XML space. Global model exchange means only the possibility to have batch-style of communication between tools. This is an interesting facility, but in many occasions it is not sufficient because we have to provide a fine grain access to model elements. XMI is of no use to do this. Here again the problem of adding new capabilities to models arose. Building intra-MDA tools for doing this was considered as very costly. So, as part of the Java community process program, a standard projector with the Java technical space was defined under the name

JRS #40. The capability to access models elements in MDA was given with the help of the Java technical space. This projector is known today under the name JMI.

As we may see, every projector has a specific purpose. In the UML 2.0 initiative, the diagram interchange part deals partially with the separation of content and presentation for MDA models [OMG, 2003]. In order to help model presentation, specific tools could have been added to the MDA space, but with a high implementation cost. Here again a solution was found in the XML space, by using the SVG standard for scalable vector graphics. Although the solution is limited to only certain kind of models, here again we see the interest of using important investments of other technical spaces to bring economically and rapidly functionalities to a given space (here the MDA) with the help of projectors.

Many other examples could be found showing the need for a very precise definition of the goal of any projector. For example, after the introduction of XMI, it was rapidly found that this projector was not bringing the facility of easy textual reading to the MDA space. Many solutions were possible, including applying XSLT transformation to XMI-serialized models to make them more usable for human operator (considering that XMI is sufficient for computer operators). Then the OMG decided to address this problem separately and a solution involving the EBNF space was defined under the name HUTN (Human Usable Textual Notation). HUTN offers three main benefits: (1) It is a generic specification that can provide a concrete HUTN language for any MOF model; (2) the HUTN languages can be fully automated for both production and parsing; (3) the HUTN languages are designed to conform to human-usability criteria. Later it was found that HUTN was not sufficient and that the set of projector should be completed in a bidirectional way with the anti-Yacc proposal (Hearnden, 2002) by DSTC. In the same spirit, SINTEF is today studying more general kinds of projectors between the MDA and the textual flat-file technical space (Oldevik, 2004).

So we can see all the gain that could be reaped from the homogeneous consideration of bridges between technical spaces with the help of generic projectors. There are many activities presently going on in this area with technical spaces like data base (SQL projectors, E/R diagram projectors), in the OS technical space (Unix projectors), in the legacy technical spaces (Cobol, ADA, PL/1 projectors to name only a few of them), in the natural language processing technical space for requirement engineering applications, etc.

One goal of the collaboration between the ATLAS group in Nantes and the GOOD OLD AI group in Belgrade is to define and build a set of generic projectors between the model engineering and the ontology engineering technical spaces.

6. Comparing the proposed M3 bridging to present ways of mappings

6.1. M3 level as representation ontologies

In this paper we have advocated the interest of factoring out the technical bridge work by considering relations at the M3 level. This is based on the conjecture that each space could exhibit such a M3 level. Much evidence can be found to support such a conjecture.

In this three-space conjecture, we assume the existence at level M3 of some form of representation ontology with a set of associated facilities. There is not infinity of possibilities at level M3 like there are at level M2. The representation systems are based on known algebraic structures with well defined properties. Very often these structures are based on trees, graphs, hypergraphs or categories. Mapping between these algebraic structures have been studied and their properties are known. For example transforming a graph into a tree is a reasonably known operation, even if there are many ways to perform it. Hypergraphs have been found very convenient to express some situations in visual languages, but mapping between graphs and hypergraphs have also been studied.

Of course each category comes into a variety of flavors. For example abstract syntax trees and XML trees are not exactly similar trees, but their properties may be studied and compared. MOF models are non directed graphs with labeled edges extremities, somewhat different from what others would se for expressing models as directed labeled graphs. Here again the precise conversion algorithms should be expressed at the M3 level and not reinvented at the M2 level.

So, in some cases we can find entities in various technical spaces that nearly correspond semantically. In this case the gain is very high because we can apply generic M3-based projectors to do automatic conversion. We could for example make correspondence between a Java grammar, a Java XML schema, a Java metamodel or a Java ontology at a very low cost. At the other extreme of the spectrum, if we have two M2-based entities with no direct correspondence, there is no more we can do. For example if we want to match *MusicML* and *MathML*, there is no obvious way to do this.

So we see that there are really three levels of technology involvement:

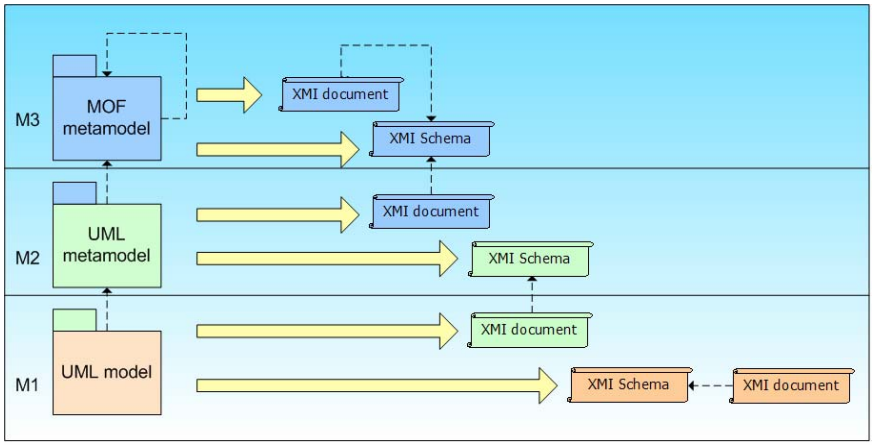
a) Specialized: someone works only inside a given technology space, with a M2 commitment. For example a given team only works with the UML modeling language or the Java programming language considering that these offer sufficient expression capabilities for their work.

b) Generic: someone works inside a given technical space, with several M2-based commitments. This is happening mainly in the XML space, the software factories MDE space (with different DSLs), etc.

c) Universal: someone works with the possibility of mapping his/her problem on several technical spaces. A given information system problem for example may be more easily and generally solved by an XML solution, a Java solution, an MDA solution, an ontology

solution. This level provides the best level of agility to solve computer engineering problems. This is the solution we are advocating here. It presupposes the existence of well designed generic projectors between the various spaces so that a given user may never become captive of a given technology.

The figure below can illustrate that current techniques understand that mappings from conceptual TS to concrete TS can be at different meta-levels. According to our approach we will have only one transformation between conceptual TS (model engineering) to a concrete TS (XML TS), so we will avoid such a complicated procedures.



Mapping MDA metametamodel, metamodels, and models to XMI

The following section illustrates a diversity of techniques that can be used for mappings between different technical spaces. The main problem is that all of them are relayed on the M2 layer. This suggests a metric evaluation of benefits of the approach we advocate in the paper: at M2 layer we have 2N mappings for N metamodels we want to connect to. In our approach we will have only one pair of mappings at M3 layer.

6.2. Current ways for transformations between technical spaces and their lacks

It is obvious from the previous descriptions that we cannot provide direct mappings between the MDA technical space and the OWL technical space. In fact, this transformation can only be defined through the XML technical space. It is important to define a *pair* of transformations in order to enable two-way mapping (one transformation for either direction) between all OWL ontologies and all ontologies represented in an MDA-based ontology language. The transformations can be based on “meta-definitions”

of OWL (i.e. on its meta-ontology) and an MDA-compliant language (i.e. a metamodel). This transformation principle is compliant with the principle of metamodel-based model transformation [Bézivin, 2001]. Table 2 gives some guidelines on how to make transformation between each pair of the languages discussed above.

Source language \ Target language	ODM		OUP		OWL
ODM	–		XML TS / XSLT	MDA TS / QVT	XML TS / XSLT
OUP	XML TS / XSLT	MDA TS / QVT	–		XML TS / XSTL
OWL	OWL TS / RDQL	XML TS / XSLT	OWL TS / RDQL	XML TS / XSLT	–

7. Conclusions

We have presented here some initial work on the ways of bridging model engineering and ontology engineering. The main idea is that the M2-based solution seems quite costly and may not scale-up. We thus suggest using M3-based projectors to provide a general solution to this problem. Many benefits may be reaped from this approach. This paper has only presented our first investigations in this area, with some documentation evidence. We are planning to go ahead with the practical study of this increasingly important engineering problem.

The notion of technical space has been found to be essential in this investigation. However we must bring more conceptual formality to this study. A technical space is defined by a three-level organization, by a top-level representation ontology sometimes called a metametamodel, by a collection of related M2-level domain specific languages (metamodels or schemas or grammars, etc.) and by a number of facility implemented by widely available tools.

Building generic bridges at the representation level (i.e. the M3-level) seems a very promising engineering practice. We have provided some illustrations in support of this hypothesis. There is still much work to be done in this area. However if the general framework is shown feasible in these areas of model and ontology engineering, it may probably also be applied to many other areas as well.

Several of the ideas presented in this paper are supported by initial prototypes developed in the research groups in Belgrade and Nantes. Work is going on in developing these prototypes, learning from previous experiments and building up a collaborative research on the subjects discussed in this paper.

8. Acknowledgements

Parts of this work has been supported by the "Interop" European network of excellence and by the "Modelware" IST European project 511731.

9. Bibliography

[Atkinson & Kühne, 2002] C. Atkinson and T. Kühne, Profiles in a strict metamodeling framework, *Science of Computer Programming*, Vol. 44, No. 1 (2002) 5-22.

[Atkinson & Kühne, 2003] C. Atkinson and T. Kühne, Model-Driven Development: A Metamodeling Foundation, *IEEE Software*, Vol. 20, No. 5 (2003) 36-41.

[Bechhofer et al, 2004] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, and P. F. Patel-Schneider, L. A. Stein, OWL Web Ontology Language Reference, *W3C Recommendation* (2004) [Online]. Available: http://www.w3.org/TR/2004/REC-owl-ref-20040210/

[Bergholz, 2000] A. Bergholz, Extending Your Markup: An XML Tutorial, *IEEE Internet Computing*, Vol. 4, No. 4 (2000) 74-79.

[Berners-Lee et al, 2001] T. Berners-Lee, J. Hendler, and O. Lassila, The Semantic Web, *Scientific American*, Vol. 284, No. 5 (2001) 34-43.

[Bézivin, 2001] J. Bézivin, From Object Composition to Model Transformation with the MDA, *In Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems*, Santa Barbara, USA (2001) 350-355.

[Bézivin, 2004] Bézivin, J.: In search of a Basic Principle for Model Driven Engineering, Novatica/Upgrade, Vol. V, N°2, (April 2004), pp. 21-24, http://www.upgrade-cepis.org/issues/2004/2/upgrade-vol-V-2.html

[Booch, 2004] Booch G., Brown A., Iyengar S., Rumbaugh J., Selic B.: The IBM MDA Manifesto The MDA Journal, May 2004, http://www.bptrends.com/publicationfiles/05-04%20COL%20IBM%20Manifesto%20-%20Frankel%20-3.pdf

[Brachman, 1979] R.J. Brachman, On the Epistemological Status of Semantic Networks, *in Findler, N.V. ed. Associative Networks: Representations and Use of Knowledge by Computers* (Academic Press, 1979) 3-50.

[Chandrasekaran et al, 1999] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins, What Are Ontologies, and Why Do We Need Them?, *IEEE Intelligent Systems,* Vol. 14, No. 1 (1999) 20-26.

[Cranefield, 2001] Cranefield, S. Networked Knowledge Representation and Exchange using UML and RDF. Journal of Digital information, 1(8), 2001. http://jodi.ecs.soton.ac.uk

[Decker et al, 2000] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Ederman, and I. Horrocks, "The Semantic Web: The Roles of XML and RDF," *IEEE Internet Computing*, Vol. 4, No. 5 (2000) 63-74.

[Devedžić, 2002] V. Devedžić, Understanding Ontological Engineering, *Communications of the ACM*, Vol. 45, No. 4 (2002) 136-144.

[Djurić et al, 2005] D. Djurić, D. Gašević, and V. Devedžić, Ontology Modeling and MDA, *Journal on Object Technology*, Vol. 4, No. 1 (2005) forthcoming.

[Fridman-Noy, 1997] N. Fridman-Noy and C.D. Hafner, The State of the Art in Ontology Design: A Survey and Comparative Review, *AI Magazine*, Vol. 18, No. 3 (1997) 53-74.

[Gašević & Devedžić, 2004] Gašević, D., Devedžić, V., "Reusing Petri Nets through the Semantic Web," *In Proceedings of the 1st European Semantic Web Symposium*, Heraklion, Greece (Lecture Notes in Computer Science, Vol. 3053, Springer-Verlag, 2004) 284-298.

[Gómez-Pérez & Corcho, 2002] A. Gómez-Pérez and O. Corcho, Ontology Languages for the Semantic Web, *IEEE Intelligent Systems*, Vol. 17, No. 1 (2002) 54-60.

[Greenfield, 2004] Greenfield, J., Short, K. , Cook, S., Kent, S. Software Factories : Assembling Applications with Patterns, Models, Frameworks and Tools. Wiley Publishing, September 2004, ISBN 0-471-20284-3

[Hearnden, 2002] David Hearnden, Kerry Raymond, Jim Steel, Anti-Yacc: MOF-to-textD. Hearnden, K. Raymond, J. Steel. AntiYacc: MOF-to-text. In Proceedings 6th IEEE International Enterprise Distributed Object Computing Conference, (EDOC 2002), pp 200-211.

[Klein, 2001] M. Klein, Tutorial: The Semantic Web - XML, RDF, and Relatives, *IEEE Intelligent Systems*, Vol. 16, No. 2 (2001) 26-28.

[Kogut et al, 2002] Kogut, P., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Kokar, M. and Smith, J. UML for Ontology Development. The Knowledge Engineering Review, 17(1), 2002. 61-64.

[Kurtev, 2002] Kurtev, I., Bézivin, J., Aksit, M.: Technological Spaces: An Initial Appraisal. Int. Federated Conf. (DOA, ODBASE, CoopIS), Industrial track, Irvine, 2002.

[McGuinness, 2002] D. L. McGuinness, Ontologies Come of Age, *In D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster (eds.) Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential* (MIT Press, Boston, 2002) 171-194.

[Oldevik, 2004] Jon Oldevik, Tor Neple, Jan Øyvind Aagedal, "Model Abstraction versus Model to Text Transformation," In Proceedings of the Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations, Canterbury, England, September 7th-8th 2004

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

[OMG, 1997] OMG/MOF: Meta Object Facility (MOF) Specification. OMG Document AD/97-08-14, September 1997. Available from www.omg.org

[OMG, 1998] OMG/XMI: XML Model Interchange (XMI) OMG Document AD/98-10-05, October 1998. see also OMG XMI Specification, v1.2. OMG Document formal/02-01-01, (2002), http://www.omg.org/cgi-bin/doc?formal/2002-01-01

[OMG, 2002] OMG/RFP/QVT: MOF 2.0 Query/Views/Transformations RFP, OMG document ad/2002-04-10. Available from www.omg.org

[OMG, 2003] UML 2.0 Diagram Interchange, January 6 2003, http://www.jeckle.de/files/UML2DIRevSub.pdf

[Seidewitz, 2003] E. Seidewitz, "What Models Mean," *IEEE Software*, Vol. 20, No. 5 (2003) 26-32.

[Soley, 2000] Soley, R. & the OMG staff: MDA, Model-Driven Architecture, (November 2000), http://www.omg.org/mda/presentations.htm