Bridging the MS/DSL Tools and the Eclipse Modeling Framework

Jean Bézivin, Guillaume Hillairet, Frédéric Jouault, Ivan Kurtev, William Piers

ATLAS Group (INRIA & LINA, University of Nantes)

{bezivin | g.hillairet | f.jouault | ivan.kurtev | william.piers}@gmail.com

ABSTRACT

Model Driven Engineering is based on a number of principles that may be applied in different contexts. Nowadays several environments employ the MDE principles: Model Driven Architecture (MDA™), Eclipse Modeling Framework (EMF), Microsoft Domain-Specific Language tools (MS/DSL), and many more. Focusing only on one context and ignoring other environments and platforms, based on different conventions, standards or protocols would be unwise because one of the desired properties of models is their ability to be exchanged between different contexts. Due to their abstraction expression level, models should ideally be more adaptable to various operational environments than conventional code. In other words, OMG models and Microsoft models among others should be able to be exchanged between the corresponding environments. In this paper we focus on exchange of models created in these two major industrial platforms: EMF and Microsoft DSL. The capability to exchange models between an EMF and a corresponding MS/DSL based system requires an abstract understanding of both architectures and a precise organization of the interoperability scheme. This paper describes the first results of a project in this area and presents the lessons learnt in this work.

General Terms

Design, Languages.

Keywords

MDE, EMF, MS DSL, ATL, model transformations, technical spaces.

1. INTRODUCTION

In November 2000 the OMG proposed a new approach to interoperability named MDA™ (Model-Driven Architecture) [13]. MDA is one realization of the broader vision of Model Driven Engineering (MDE) that encompasses current research trends related to generative and transformational techniques in software engineering, system engineering, and data engineering. Considering models as first class entities and any software artifact as a model or as a model element is one of the basic principles of MDE. The key ideas of MDE are germane to other approaches such as domain-specific languages (DSLs), software factories, model-integrated computing (MIC), model-driven software development (MDSD), model management, language-oriented programming [3], and many more.

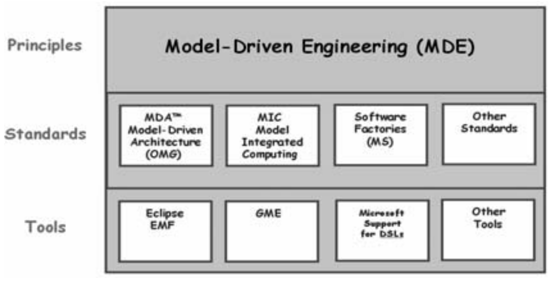
Principles
Model-Driven Engineering (MDE)
Standards
MDA™ Model-Driven Architecture (OMG)
MIC Model Integrated Computing
Software Factories (MS)
Other Standards
Tools
Eclipse EMF
GME
Microsoft Support for DSLs
Other Tools

Figure 1 MDE principles, standards and tools.

The OMG MDA initial proposal implements the principles of MDE around a set of OMG standards such as MOF, XMI, OCL, UML, CWM, and SPEM. Based on similar principles but sometimes on different standards (Figure 1), several other related technical spaces [9] have also been proposed like Microsoft Software Factories Tools (MS/DSL) [6], Generic Modeling Environment (GME) [14], Coral [12] and many others.

These spaces are not isolated from each other and should be allowed to communicate by exchanging artifacts. In this paper we will focus on two of those spaces and the possible links between them: MS/DSL and Eclipse Modeling Framework (EMF). The concept of bridge is used to denote the capability of communication between two spaces. If a bridge is available between spaces then it is possible to import an artifact from one space to the other and vice-versa.

To implement a bridge between MS/DSL and EMF we use the functionalities provided by the AMMA (ATLAS Model Management Architecture) model engineering platform. AMMA is built on top of the Eclipse Modeling Framework and consists of four different components. The component that plays the most important role in the bridge implementation presented in this paper is a model transformation language named ATL (ATLAS Transformation Language).

The bridge between MS/DSL and EMF spans two levels: metamodel and model level. At the level of metamodels it allows to transform MS/DSL domain models to EMF metamodels. At the level of models the bridge allows transforming MS/DSL models conforming to domain models to EMF models conforming to EMF metamodels. At both levels the bridge operates in both directions. A chain of ATL-based transformations is used to implement the bridge at these two levels. The benefit of using such a bridge is the ability to transpose MS/DSL work in EMF platform, and vice-versa.

While pursuing this objective we realized the need of systematic methods to build bridges between similar or different technical spaces. The lessons learnt in the realization of the bridge between EMF and MS/DSL environments are very helpful in building of other exchange schemes between similar platforms, for example between GME and EMF.

This paper is organized as follows. In section 2 we describe the platforms used in this work: MS/DSL, EMF and AMMA. Section 3 presents the metametamodels of these platforms and the basic functionality of the bridge. Section 4 explains the operation of the bridge at the metamodel level, and Section 5 shows the operation at the model level. Section 6 concludes the paper.

2. EMF, MS/DSL and the AMMA Platform

The two main industrial references of MDE are currently EMF and MS/DSL Tools. AMMA extends the EMF facilities in a number of ways. By design AMMA integrates the notion of technical space, i.e. the possibility to interoperate with other MDE and non-MDE environments. This section provides a brief introduction to EMF, MS/DSL Tools and AMMA.

2.1 EMF

EMF is a modeling framework and code generation facility for building tools and applications based on a structured data model. The following description is adapted from [4]. On the base of a model specification expressed in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. Models can be specified using annotated Java, XML documents, or modeling tools like Rational Rose, and then imported into EMF. An important feature of EMF is that it provides the foundation for interoperability with other EMF-based tools and applications. EMF consists of three fundamental parts:

- **EMF** - The core EMF framework includes a metamodel (Ecore) for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.
- **EMF.Edit -** The EMF.Edit framework includes generic reusable classes for building editors for EMF models.
- **EMF.Codegen** - The EMF code generation facility is capable of generating everything needed to build a complete editor for an EMF model. It includes a GUI from which generation options can be specified, and generators can be invoked. The generation facility leverages the JDT (Java Development Tooling) component of Eclipse.

Three levels of code generation are supported:

- **Model** - provides Java interfaces and implementation classes for all the classes in the model, plus a factory and package (meta data) implementation class.
- **Adapters** - generates implementation classes (called ItemProviders) that adapt the model classes for editing and display.
- **Editor** - produces a properly structured editor that conforms to the recommended style for Eclipse EMF model editors and serves as a point from which to start customizing.

2.2 MS/DSL

The Microsoft Tools for Domain-Specific Languages is a suite of tools for creating, editing, visualizing, and using domain-specific data for automating the enterprise software development process. These new tools are part of a vision for realizing software factories [6]. The latest version is May 2005 CTP release for Visual Studio 2005 Beta 2.

As described in [10], the suite of tools is supported by a code framework that makes it easier to define domain models and to construct a custom graphical designer hosted in Visual Studio. The suite consists of:

- A project wizard for creating a fully configured solution in which one can define a domain model that consists of a designer and a textual artifact generator. Running a completed solution within Visual Studio opens a test solution in a separate instance of Visual Studio, allowing testing the designer and artifact generator.
- A format and an updated graphical designer for defining and editing domain models.
- An XML format for creating designer definitions, from which the code for implementing designers is generated. This allows defining a graphical designer hosted in Visual Studio without any hand coding.
- A set of code generators, which take a domain model definition and a designer definition as input, and produce code that implements both components as output. The code generators also validate the domain model and designer definition and raise errors and warnings accordingly.
- A framework for defining template-based artifact generators, which takes data (models) conforming to a domain model as input, and outputs text based on the template. Parameters in the template are substituted using the results of running a C# script embedded in the template.

2.3 AMMA

Currently built on top of EMF, AMMA [2] brings additional functionalities and could be implemented on top of other MDE environments. AMMA has both local and distributed implementations and is based on four blocks providing a set of model processing facilities:

- Atlas Transformation Language (ATL) [1][8]defines model transformation facilities for a QVT-like language: a transformation virtual machine, a metamodel based compiler, an editor based on Eclipse, and a debugging environment.
- Atlas ModelWeaver (AMW) makes it possible to establish links between the elements of two (or more) different models.
- Atlas MegaModel Management (AM3) defines the way the metadata is managed in AMMA (registry on the models, metamodels, and tools).
- Atlas Technical Projectors (ATP) defines a set of injectors/extractors enabling to import/export models

from/to foreign technical spaces (for instance Java classes, relational models).

2.4 Global overview of bridging

Before explaining the implementation of the bridge between MS/DSL tools and EMF we give the overall idea behind the project. The bridge performs transformations at two levels: M2 level (metamodel level) and M1 level (model level). Figure 2 provides an overview of the approach.

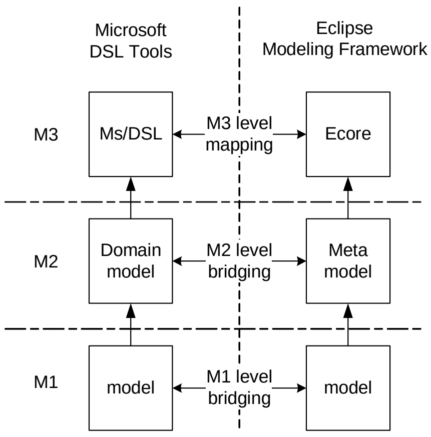

Figure 2 Global overview of the two level bridging

To make a complete bridge between MS/DSL and EMF we must establish the correspondence between those platforms, at three metamodeling levels: M3, M2 and M1. The M3 level mapping allows establishing correspondences between metametamodels. With these correspondences we can perform the M2 level bridging that transforms domain models to Ecore metamodels. The M1 level bridging transforms MS/DSL models to Ecore models and vice-versa.

3. M3-level mapping

To enable mapping between MS/DSL and EMF we need a definition of each system at the M3 (i.e. metametamodel) level. Microsoft does not specify an explicit metametamodel for DSL designer, so we had to discover it by observation. The metametamodel of AMMA platform is called KM3. KM3 is used as a mediator between the Ecore and the DSL metametamodel. If we build a mapping between MS/DSL metametamodel and KM3 we can reuse the existing mapping between KM3 and Ecore to obtain the mapping between MS/DSL and Ecore as a composition of two mappings. KM3 and the MS/DSL metametamodel are explained in the rest of this section.

3.1 KM3: Kernel MetaMetaModel

KM3 [1], [5] is a metametamodel close to Ecore and EMOF 2.0. A simplified version is presented in Figure 3. We use it instead of Ecore because we need to work with several other metametamodels such as MOF 1.4. KM3 uses a Java-like textual notation for expressing metamodels. A similar approach is also proposed by Emfatic [7], a language for representing Eclipse Modeling Framework Ecore Models in textual forms but the focus of KM3 is much broader than simply EMF. Of course several ATL transformations of KM3 notation to or from other notations are available.

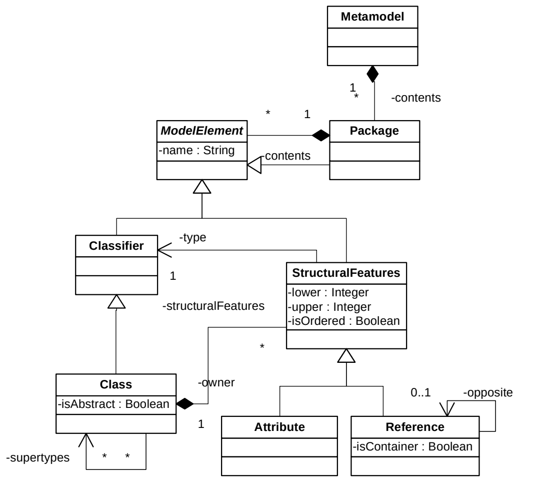

Figure 3 Simplified version of the KM3 metametamodel

KM3 may thus be used as a pivot between various metametamodels as illustrated by Figure 4.

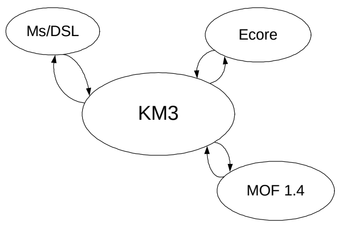

Figure 4 Use of KM3 as a pivot

The AMMA platform provides mappings between KM3 and other metametamodels. This enables easy mapping of a newly added metametamodel to the existing ones.

3.2 DSL metametamodel

The equivalent of a metamodel in the Microsoft world is called a "domain model". It is composed of a class hierarchy and relationships. From our initial experience with Microsoft's tools we inferred a DSL metametamodel. The result is expressed in Ecore and is shown in Figure 5.

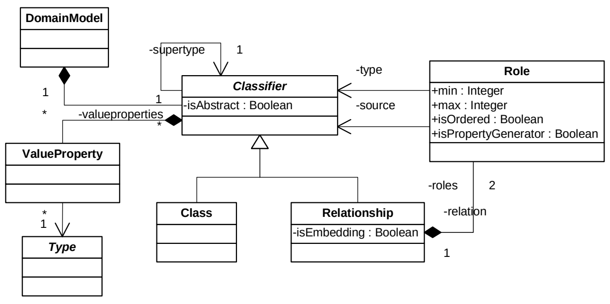

Figure 5 Simplified version of the DSL metametamodel

A relationship may be a simple reference or an embedded relation (similar to composition in UML terms). It has properties and may have a supertype. A relationship has two roles, but a future version of DSL Tools may propose relationships with *n* roles.

3.3 Comparison between KM3 and DSL

With those diagrams, we can compare KM3 and DSL with each other. The following observations may be made:

- KM3 and DSL Classes are almost equivalent, and have the same characteristics, except supertypes: KM3 allows multiple inheritance whereas DSL does not.
- A KM3 Attribute is equivalent to a DSL ValueProperty.
- DSL roles can be mapped to KM3 References, but the latter are not affiliated with a Relationship like in DSL. References are contained in their owner and linked to their opposite reference. If the owning relationship of a pair of roles is embedded then one of the associated KM3 references is a container.
- DSL Relationships and DSL Classes have the same properties: relationships may be linked to each other, have a supertype and attributes. There is no direct equivalent of DSL Relationship in KM3. We can consider simple relationships (with no supertype or attribute) as corresponding to a pair of references while complex relationships correspond to classes.
- On the basis of this analysis we can establish mappings between DSL and KM3. Mappings are used to perform M2-level bridging and are operationalized in ATL language.

4. M2-level bridging

After establishing the mappings between DSL and KM3, we can use ATL to transform domain models into KM3 models and into EMF metamodels with an additional transformation from KM3 to Ecore. A detailed overview of the M2-level bridge is given in Figure 6.

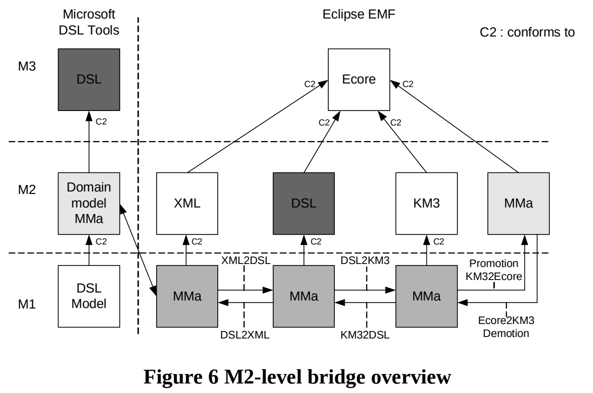

Figure 6 M2-level bridge overview

We have already shown the DSL metametamodel expressed in Ecore. KM3 may also be expressed as an Ecore metamodel. The ATL language is capable of specifying transformations on Ecore models conforming to Ecore metamodels. Therefore we can transform DSL domain models into KM3 models. KM3 models may be transformed into Ecore metamodels (recall that there exist a reusable transformation between KM3 and Ecore). In summary, we can apply a chain of ATL transformations to implement the M2 level bridge. However, first we have to obtain a representation of a DSL domain model as an Ecore model.

The transformation chain for the first direction of the bridge contains four steps. First, a domain model MMa (defined in DSL Tools) is transformed to an XML representation conforming to the XML metamodel. After that, the model is transformed using ATL to conform to DSL metametamodel expressed in Ecore, and then to a model conforming to KM3. The final step is the promotion of this model using KM32Ecore transformation, which creates the MMa EMF metamodel conforming to Ecore. This operation is called promotion since the result lies at a higher level in the metamodeling hierarchy compared to its source. The inverse transformations from Ecore to DSL are also defined.

In fact we build two transformation chains: from DSL to KM3 and from KM3 to DSL. Since KM3 acts as a pivot between our metametamodels we already have the transformations to and from Ecore. The following subsections explain in detail these transformation chains.

4.1 First transformation chain: DSL to Ecore

To make the transformation we proceed in three steps. They are detailed below.

4.1.1 First step: XML2DSL

We have to get information from a *.dsldm* file into a DSL metamodel. We inject the *.dsldm* file into an XMI file conforming to an XML metamodel, using an XML injector. An ATL transformation is then applied to capture information from the XML file to a model which conforms to the DSL metametamodel previously described.

The main work of this transformation is a mapping between *.dsldm* features (concepts, relationships, roles, enumerations, properties, etc.) and our DSL model.

4.1.2 Second step: DSL2KM3

In this step we transform the DSL model to a KM3 metamodel, using another ATL transformation. By splitting the transformation from XML to KM3 in two steps we first reduce the complexity of the transformation and second, we achieve a reusability of the transformation from XML to DSL and vice-versa.

In the DSL2KM3 transformation, DSL classes are mapped to KM3 classes, like simple types and properties. We encountered several problems due to the differences between DSL and KM3 metamodels explained in section 3.3:

- In DSL, a Relationship is defined like a Class, with the same properties.
- In KM3, a Relationship between two Classes is encoded by two references into adjoining classes.

We found two solutions to this problem:

Solution 1: We can ignore all specific properties of the Relationship except roles and type of containment.

Solution 2: We can turn the Relationship into a KM3 class, and keep the attributes, supertypes, and other features.

In both cases some information is lost. In solution 1 we loose Relationship features. In solution 2 we loose track of the fact that the source element is a Relationship. Transformation of the resulting KM3 metamodel back into a DSL metamodel produces a DSL metamodel, which contains less information than the original one. We may cope with this issue by keeping the information that cannot be represented in KM3 into an additional model. This information would then be used by an enhanced version of KM32DSL.

To apply our solutions, we can classify DSL Relationships in two types:

- Simple Relationship:

If a relationship does not have any supertype or attribute then we can ignore the name and transform the relationship into a couple of KM3 references. Using solution 1 the roles of a relation are mapped to KM3 references.

- Complex Relationship:

If the relationship has attributes, supertypes, or subtypes, we turn it into a KM3 class (using solution 2), and we create two pairs of references to link it to the classes referenced by its roles, like in Figure 7.

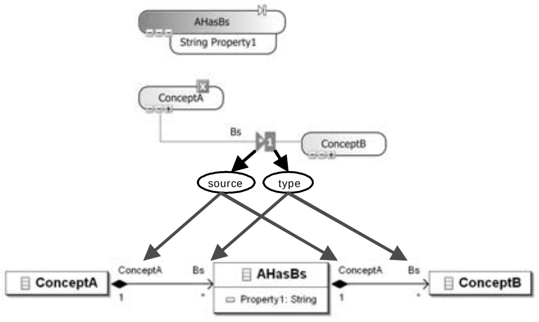

Figure 7 An example of Complex Relationship treatment

4.1.3 *Third step: KM32Ecore*

Using the existing transformation KM32Ecore, we get a metamodel conforming to EMF. We can now use this model within any EMF compatible tool, for example, Omondo's EclipseUML plug-in. With this last step the first direction of the M2-level bridge is completed.

4.2 Second transformation chain: Ecore to DSL

We start this transformation with a model which conforms to KM3, and want to transpose it into a DSL tools model. To achieve the transformation we proceed in three steps detailed below.

4.2.1 *First step: KM32DSL*

When we transform our initial model (which is expressed in KM3) into a model which conforms to our DSL metamodel, we find problems similar to the one explained in section 4.1.2.

KM3 doesn't implement Relationships, so we have to create them from pairs of KM3 References.

Some attributes of DSL don't have correspondence in KM3, so we create them using default values, and we generate single identities.

4.2.2 *Second step: DSL2XML*

We transform the result from the previous step into a model conforming to XML, which defines a *.dsldm* like file. Figure 8 shows a simple metamodel of *.dsldm* file. The gray boxes show what is added from scratch when we make the transformation and the white ones show what is supported by our DSL metametamodel.

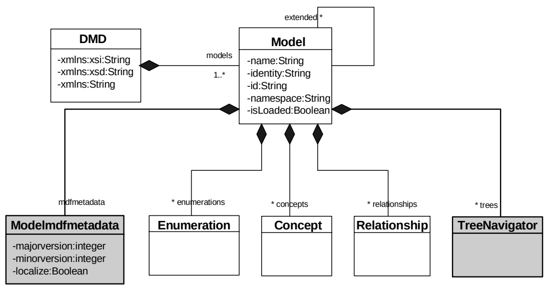

Figure 8 Simple metamodel of .dsldm file

We use an ATL transformation, which creates an XML model similar to a *.dsldm* file.

To be able to build diagrams as trees, DSL Tools needs to satisfy some constraints which are specified in the tree navigators. For instance we must signal if a class needs to appear several times, or if a class needs to be considered as a root class (if it must appear on the left side of the diagram).

4.2.3 *Third step: XML2Text*

We use existing XML to text transformation to create a *.dsldm* file that will be included into a "blank project" for DSL Tools, by replacing the *.dsldm* file.

5. M1-level bridging

M1 level bridge allows transforming DSL models to and from EMF models. We describe such a tool in this section.

First we have to know how models are viewed by the two technologies. Basically a model has to conform to a domain model in context of MS/DSL and to an Ecore metamodel for EMF.

To store models, Microsoft uses XML documents that conform to an XML schema, which does not directly map to the domain model. With EMF, the models are stored in XMI format and explicitly conform to a metamodel.

To implement the M1-level bridge we have to transform a DSL model file to a model that conforms to a metamodel defined under EMF. To do this we have to make a metamodel for the DSL models.

The problem is that if we write a transformation between the DSL metamodel and the metamodel in EMF, we'll have to write a transformation per EMF metamodel. This approach is not general. We need a transformation that works for any EMF metamodel. How this is done is explained below in section 5.4.

We will use ATL as in the M2-level bridge. The process will be performed in three steps, as shown in Figure 9. For simplicity we only describe the transformation chain from DSL to EMF:

- The first step consists of injecting the DSL model file *Ma* to an XML model.
- In the second step we transform the XML model into a model conforming to the DSL models metamodel (named *DSLModel* in Figure 9). DSLModel was defined to match Microsoft's schema as closely as possible.
- In the third step we apply an ATL transformation that takes as input a DSL model *Ma* and the metamodel *MMa* defined under EMF and output a model *Ma* that conforms to the model *MMa*.

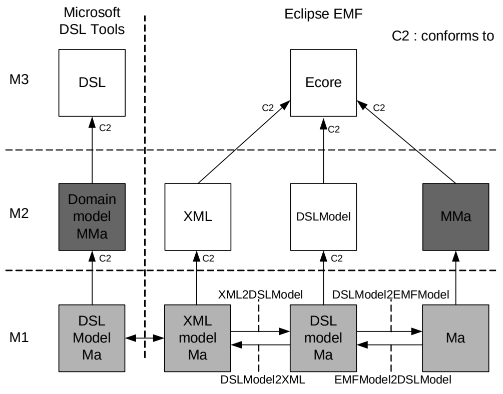

Figure 9 M1-level bridging overview

In the remaining part of this section we give details on the DSL models metamodel and the three steps outlined above.

5.1 The DSL models metamodel

We created a metamodel close to the schema used by Microsoft to store DSL models as XML files. This metamodel is called DSLModel and is shown in Figure 10.

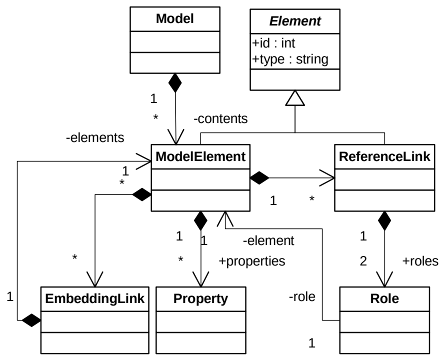

Figure 10 The metamodel DSLModel

In this metamodel the attribute *type* of class *Element* contains the name of the class from the domain model that is instantiated in the model. A model element has properties that correspond to attributes in domain model, a reference link is a reference relationship and an embedding link is an embedded relationship.

5.2 First step: Inject XML file to XML model

The first step injects the XML file containing the DSL model definition into an XML model that conforms to the XML metamodel.

5.3 Second step: XML2DSLModel

The second step transforms the XML model to a model that conforms to DSLModel. We use an ATL transformation to do it (*XML2DSLModel* in Figure 9).

5.4 Third step: DSLModel2EMFModel

The last step has to produce a model conforming to its EMF-metamodel. This metamodel varies so if we write a transformation between DSLModel and the metamodel it will work for only one concrete EMF metamodel. To obtain a generic transformation definition that works for every target metamodel we proceed in two steps shown in Figure 11.

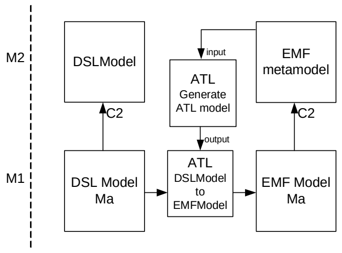

Figure 11. Overview of Step 3

The first step is to apply an ATL transformation (*Generate ATL model* in Figure 11) that takes as input an EMF metamodel and generates an ATL transformation (*DSLModel to EMFModel* in Figure 11). The generated transformation contains the necessary rules to produce a model conforming to the EMF metamodel. This solution is generic and works for any EMF metamodel. The second step is to apply the generated transformation in order to obtain an EMF model *Ma* from the DSL model *Ma*.

In the first step we use the fact that transformation definitions are also models and therefore may be generated by a transformation. In other words we create a higher-order transformation to solve the problem with the generality of our solution.

6. Conclusions

This paper has reported on an experiment of practical interest: building operational bridges between two industrial MDE platforms: MS/DSL tools and EMF. Since EMF is often considered as one of the major implementations of OMG standards, the resulting tools are of high potential utility to make the MS world and the OMG world communicate in the domain of model engineering.

We have seen that achieving model interoperability is much more complex than simply defining a local serialization format in a local context like XMI.

However the value of this work goes much beyond mere practicality. It shows that different kinds of models may be exchanged between different technical spaces. It suggests a general way to building bridges between different spaces. Knowing the rising fragmentation in current technologies, this issue will probably become increasingly important in the future.

Interoperability is a horizon that moves farther and farther as one tries to come closer. The conviction that there will never be a unique operating system, a unique programming language, a unique database management system or a unique network framework has convinced organization like OMG to consider heterogeneity as inherent to IT and to pursue interoperability as a main goal. The limits of code-based or middleware-based interoperability have been reached with proposals such as CORBA and IDL to hide operating system or programming language heterogeneity in distributed applications. The existence of several de-facto middleware-like frameworks is now a reality.

Model-based interoperability is trying to approach the problem at a higher abstraction level. Using one unique and rather monolithic language like UML 2.0 is not very realistic. Instead the solution is to use a metalanguage building system like MOF, allowing defining a set of domain-specific languages through metamodels like UML, SPEM, CWM, EDOC. This seemed to be the current state of the art in model engineering until recently.

Unfortunately the situation is no more so clear and the MOF-based metamodel building system proposed by OMG will probably face several new competitors. Some of them like EMF are really close but other are making quite different assumptions on what should be in a metametamodel. Similarly to CORBA that had to cope with different middleware proposals, MOF, XMI and OCL may also have to cope with different other metamodeling proposals, sometimes with apparent advantages. The present work addresses at the same time some conceptual and practical issues raised by this situation.

The naive answer to this situation could be to build an additional M4 metamodeling layer, but obviously this is not a serious suggestion. Instead we have to prepare a systematic and methodic bridging approach between different representation systems (such as MOF, MS/DSL, XML, EBNF). This is possible because the number of such technical spaces (based on trees, graphs, or other simple algebraic structures) is limited and by considering all these spaces as having a three-level organization (M1, M2, M3), it is possible to keep this diversity of technical space representation under control.

The only alternative to this solution would be to wait that one technical space prevails on the other ones by making them obsolete. By experience and observation we know that this will never happen. We thus propose to improve bridging schemes such as the one presented in this paper and to hide as much as possible their implementation complexity to the end-users. When a user will be allowed to work indifferently with an XML-schema, an EBNF-grammar, a MOF-metamodel or a MS/DSL Domain Model, and when the differences in representation will be made transparent, he/she may be able to work on solving the real problems and not curing the artificial problems introduced by increasingly fragmented and complex technology.

7. Acknowledgements

We would like to thank Freddy Allilaire, David Touzet, Patrick Valduriez and all the members of the AMMA group for their numerous contributions to this document. This work has been supported in part by the IST European project "ModelWare" contract 511731) and by a grant from Microsoft Research (Cambridge).

8. References

[1] ATL, ATLAS Transformation Language Reference site http://www.sciences.univ-nantes.fr/lina/atl/ including KM3: Kernel Metametamodel definition.

[2] Bézivin, J, Jouault, F, and Touzet, D: Principles, Standards and Tools for Model Engineering. In: Proceedings of the Using metamodels to support MDD Workshop, 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2005). See also research report http://www.sciences.univ-nantes.fr/lina/atl/www/papers/RR-LINA2005-01.pdf. http://www.sciences.univ-nantes.fr/lina/atl/publications/ http:/www.sciences.univ-nantes.fr/lina/atl/www/papers/AMMA_ICECCS05.pdf

[3] Dmitriev, S. Language Oriented Programming: The Next Programming Paradigm, OnBoard, november 2004, http://www.onboard.jetbrains.com/is1/articles/04/10/lop/mps.pdf

[4] Eclipse Modeling Framework http://www.eclipse.org/emf/

[5] GMT, General Model Transformer Eclipse Project, http://www.eclipse.org/gmt/ including KM3: Kernel MetaMetaModel definition.

[6] Greenfield, J., Short, K., Cook, S., Kent, S., Software Factories, Wiley, ISBN 0-471-20284-3, 2004.

[7] IBM Emfatic Language for EMF Development AlphaWorks, http://www.alphaworks.ibm.com/tech/emfatic

[8] Jouault, F., and Kurtev, I., Transforming Models with ATL, to appear in proceedings of Model Transformations in Practice Workshop, October 3rd 2005, part of the MoDELS 2005 Conference

[9] Kurtev, I., Bézivin, J., Aksit, M. Technical Spaces: An Initial Appraisal. CoopIS, DOA’2002 Federated Conferences, Industrial track, Irvine, 2002 http://www.sciences.univ-nantes.fr/lina/atl/publications/

[10] Microsoft Domain-Specific Language (DSL) Tools, May 2005 CTP Release for Visual Studio 2005 Beta 2. http://lab.msdn.microsoft.com/teamsystem/workshop/dsltools/default.aspx

[11] MS/DSL Tools Walkthrough 1 downloadable from http://lab.msdn.microsoft.com/teamsystem/workshop/dsltools/default.aspx

[12] Porres, I. The CORAL Platform. http://mde.abo.fi/tools/Coral/

[13] Soley, R., and the OMG staff, Model-Driven Architecture, OMG Document, November 2000, http://www.omg.org/mda

[14] Vanderbilt Institute for Software Integrated Systems (ISIS) Web-based Open Tool Integration Framework (WOTIF) http://www.isis.vanderbilt.edu/Projects/WOTIF/default.html