Model Driven Engineering: An Emerging Technical Space

Jean Bézivin

Atlas Group: INRIA, and LINA
University of Nantes
2, rue de la Houssinière - BP92208
44322 Nantes Cedex 3, France
`Jean.Bezivin@univ-nantes.fr`

Abstract. As an emerging solution to the handling of complex and evolving software systems, Model Driven Engineering (MDE) is still very much in evolution. The industrial demand is quite high while the research answer for a sound set of foundation principles is still far from being stabilized. Therefore it is important to provide a current state of the art in MDE, describing what its origins are, what its present state is, and where it seems to be presently leading. One important question is how MDE relates to other contemporary technologies. This tutorial proposes the "technical space" concept to this purpose. The two main objectives are to present first the basic MDE principles and second how these principles may be mapped onto modern platform support. Other issues that will be discussed are the applicability of these ideas, concepts, and tools to solve current practical problems. Various organizations and companies (OMG, IBM, Microsoft, etc.) are currently proposing several environments claiming to support MDE. Among these, the OMG MDA™ (Model Driven Architecture) has a special place since it was historically one of the original proposals in this area. This work focuses on the identification of basic MDE principles, practical characteristics of MDE (direct representation, automation, and open standards), original MDE scenarios, and discussions of suitable tools and methods.

Keywords: Model Driven Engineering; MDE; MDA; Metamodeling; Technical Spaces;

1 Introduction

In November 2000 [32] the OMG proposed a new approach to interoperability named MDA™ (Model-Driven Architecture). MDA is one example of the broader Model Driven Engineering (MDE) vision, encompassing many popular current research trends related to generative and transformational techniques in software engineering, system engineering, or data engineering [6], [11]. 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 many other approaches such as domain specific languages (DSLs), software factories, model-integrated computing (MIC), model-driven software development (MDSD), model management, language-oriented programming and much more. The OMG MDA initial proposal may be defined as the realization of MDE principles around a set of OMG standards like MOF, XMI, OCL, UML, CWM, and SPEM. Most of these acronyms will be referenced later in the document. Their important number is due to the initial normative aspect of the field. A list of some common ones is provided in an appendix.

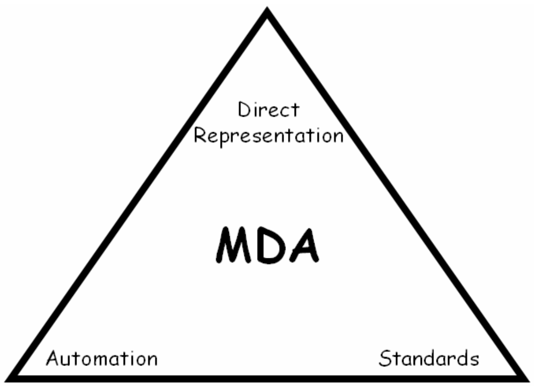


Fig. 1. The three IBM manifesto tenets

The IBM manifesto [15] makes the claim that MDA-based approaches are founded on three ideas: Direct representation, Automation and Standards. Direct representation allows a more direct coupling of problems to solutions with the help of Domains Specific Languages (DSLs). Automations means that the facets represented in these DSLs are intended to be processed by computer-based tools to bridge the semantic gap between domain concepts and implementation technologies and not only for mere documentation. This should be complemented by the use of open standards that will allow technical solutions to interoperate. These three complementary ideas are central to the development of MDE approaches. Models should be exchangeable and for this we need to agree on consensual standards. Many models would however need to be written by human agents, very often non computer scientist agents. In this case, these models (or programs) will be written in domain specific languages, restricted in size and precisely defined. The mapping of these models written in precise DSLs onto operational technology by using generative and transformational techniques is one important aspect of MDE. Another major issue is to solve the fragmentation problem resulting from the coexistence of a high number of small DSLs. The answer to this problem is the existence of a global representation system (for example the MOF OMG M3 level), and the support of libraries of various correspondences between models (e.g. transformation or weavings). However this is not sufficient and we need also to invent registries and global links between entities like models and metamodels to escape fragmentation problems.

Historically in year 2000, the MDA had a specific goal: preserving the IT investments of companies through the constant and rapid evolution of platforms. At that time the middleware and the component solutions alone were no more in a position to achieve this goal. The proposal was thus to capture in PIMs (Platform Independent Models) the part of the investment that should not be affected by major or minor changes in platforms. The idea was then that it should be possible, by some means, to generate PSMs (Platform Specific Models) from these PIMs. How this problem could be concretely solved was not completely clear at the time. The main idea was that a PIM could be expressed in UML and that, through the supposedly stability of UML versions in time, the corresponding assets could be preserved over long periods. The concrete means to generate PSMs from PIMs were not precisely stated at the time since the number of such target platforms was rather limited and similar (mainly CORBA, J2EE/EJB and DotNet). The scope of these target platforms was then broadened, the notion of models (including PIMs and PSMs) was extended beyond mere UML models, and the generation of PSMs from PIMs was suggested to be automated by model transformations using the newly defined QVT standard [29].

More than five years after, the situation has much evolved. Separating platform dependent from platform independent aspects is no more seen as the unique goal. The major problem is now the separation and combination of concerns in the construction and maintenance of information systems. Among these concerns, platform dependent and independent aspects remains important in the agenda, but these are more and more considered as a special case of a general problem including for example separation of functional and non-functional requirements. MDA and DSL solutions are now more and more closely related. What MDA is bringing to DSLs is this idea of using a collection of metamodels to capture the various facets of a system under construction or under maintenance. What DSLs is bringing to MDA is that a unique general purpose language, even a very large one like UML 2.0, is not able and will never be able to capture all the needs of the designers, administrators, and users of a given system.

The notion of direct representation [15] is very important. This means that instead of performing themselves directly certain tasks in general purpose languages like Java, computer scientists may instead concentrate on defining specialized languages and handling these to final users that will be able to express precisely their contributions in these languages in a non-ambiguous manner. The computer scientists will be in charge of mapping these contributed expressions (often of a declarative nature) into executable structures i.e. target platforms. We recognize here the common objective of MDA and DSLs. But a DSL may address a lot of needs, corporate or organizational for example. A typical DSL, that has been very successful for a long time, is Excel that addressed in early stages, with products like VisiCalc, the domain needs of basic accounting. Excel is now an example of a language defined by computer specialists to solve the problems of non specialists. Using such tools, many tasks may be solved now by non-specialists, without the help of specialists that are becoming more and more language engineers. The form of these languages may vary in their concrete appearance and they could be defined by conventional grammars, by DTDs or XML schemas, by ontologies, by graphical representation or by metamodels. One central contribution of MDE is about the possible expression of DSLs by metamodels.

The three tenets mentioned in the IBM manifesto are essential to MDE. However, to make it practical we need to extend this definition in two directions First we need to build MDE on a sound set of principles. Next we need to implement MDE on practical platforms of wide usage. We propose an initial set of kernel principles that could serve as a proposal for a foundational set of principles. We also suggest an architectural style that could be used as a guide to implement these principles on current industrial platforms.

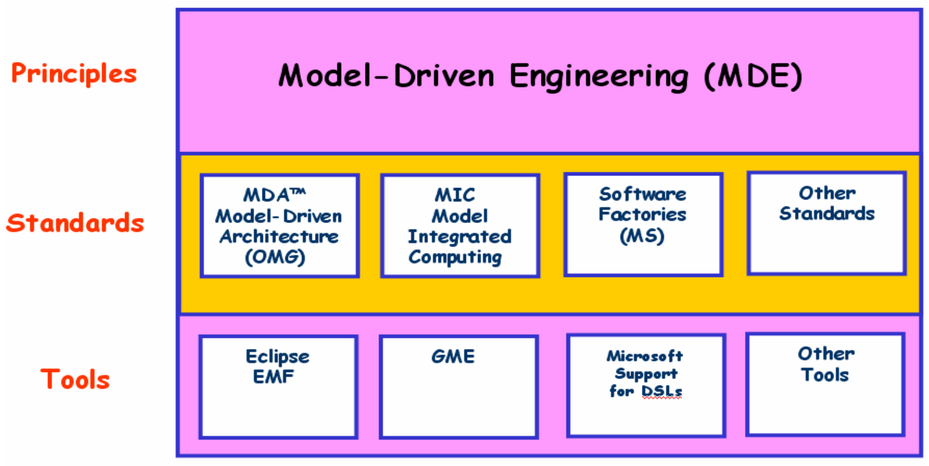

Principles
Model-Driven Engineering (MDE)
Standards
MDA™ Model-Driven Architecture (OMG)
MIC Model Integrated Computing
Software Factories (MS)
Other Standards
Tools
Eclipse EMF
GME
Microsoft Support for DSLs
Other Tools

Fig. 2. Principles, Standards, and Tools

This text proposes some ideas on the present rapid evolution of the MDE scene. The basic set of MDE principles is based on two concepts (system and model) and two basic relations (conformance and representation). This allows giving a first definition of what is a model in the MDE context. In another section we propose to situate MDE with respect to other possible solutions. Our organization of the solution space is based on the notion of "technical space". This will help us to give an extended definition, more general and precise, of what is a model. In the rest of the document we come back to the strict MDE technical space. We also propose an initial inventory of possible operations on models. Model transformation will obviously be one important example of an operation on models. The consequences of a transformation being itself considered as a model will be much emphasized. We also propose an architectural style for implementing an MDE platform respecting the basic principles. This will be given in the form of an abstract architecture composed of four complementary functional blocks addressing the issues of model transformation, model weaving, global model management, and model projection onto other spaces. As an illustration of this architectural style, we will present the AMMA prototype [1] available in the Eclipse GMT project [19].

2 Basic MDE Principles

2.1 Prerequisites

In order to discuss the broad view of MDE, we need a suitable common notation. Among many possibilities, we use the now conventional UML class diagrams and the Object Constraint Language. UML and OCL do not bring new expressive power to MDE, but they are standards (OMG standards) that may facilitate interoperability of solutions and common understanding.

2.2 Introduction

A model is a complex structure that represents a design artifact such as a relational schema, an interface definition (API), an XML schema, a semantic network, a UML model or a hypermedia document [4]. In the present section we will give a more limited definition of a model, in the context of MDE only, as a graph-based structure representing some aspects of a given system and conforming to the definition of another graph called a metamodel. As we shall see later there are several contextual and complementary definitions of what a model is. We are not interested here by a theoretical definition of a model, but by an engineering one, i.e. a definition that will help users to implement and maintain systems. The parallel between object technology and model engineering that was made in [7] may be relevant here. The definition of an "object" that was given by pioneers like Dahl, Nygaard, Kay, Meyer and others had nothing to do with philosophy but this was an engineering definition that is still of high interest to the profession today. Similarly we are presently looking for an operational engineering definition of a "model" that could play a similar role in the coming period.

2.3 Basic Entities

The present trend in model engineering [7] is to consider that models are first class citizens. This approach seems to be the only possibility to deal with ever-increasing complexity in information and software systems. It will hopefully allow to separate and to combine different aspects in a more regular way. Among these aspects we may mention platform dependent and independent features. As a corollary of this principle stating “that everything is a model”, we may infer for example that “a model transformation should also be considered as a model”. The basic principle and its corollaries build the foundation of third generation model transformation frameworks. However there is still an important amount of work to be done before fully understanding MDE environments and putting them to work.

In [31], Ed Seidewitz writes: “…In any case, without this well-grounded foundation, our models are, in the end, just pictures that don’t really mean anything at all…”

Models are now commonly used to provide representation of real-world situations. A model is said to *represent* a system. Fig. 3 provides an example of a relational model that defines a possible representation for a set of books in a library. On the right side of Fig. 3, we have a relational representation of part of the world (a library). Other different representations of this same library are possible, e.g. an event-based representation capturing book creation, lending, returning, destruction, etc.

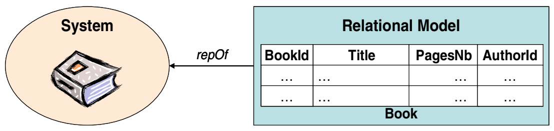


Fig. 3. The "representation" relation between a system and a model

Each model is defined in conformance to a metamodel. Metamodels define languages enabling to express models. A metamodel describes the various kinds of contained model elements, and the way they are arranged, related, and constrained. A model is said to *conform to* its metamodel. Thus, our Book relational model conforms to the relational metamodel (Fig. 4). Representation and conformance relations are central to model engineering [7].

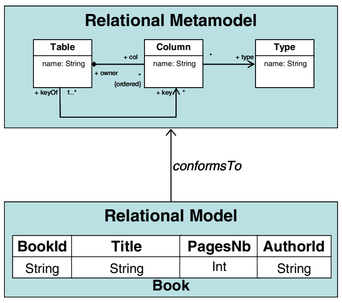


Fig. 4. The "conformance" relation between a model and its metamodel

As models, metamodels are also composed of elements. Metamodel elements provide a typing scheme for models elements. This typing is expressed by the *meta* relation between a model element and its *metaelement* (from the metamodel). We also

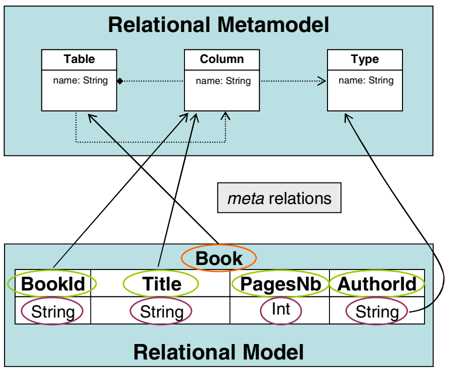


Fig. 5. The "meta" relation between model and metamodel elements

say that a model element is *typed* by its metaelement. A model conforms to a metamodel if and only if each model element has its metaelement defined within the metamodel. Fig. 5 makes explicit some of the *meta* relations between Book model elements and relational metamodel elements: the Book element is typed by the Table metaelement, BookId and Title are typed by the Column metaelement, and String is typed by the Type metaelement.

The growing number of metamodels has emphasized the need for an integration framework for all available metamodels by providing a new item, the metametamodel, dedicated to the definition of metamodels. In the same way models are defined in conformance with their metamodel, metamodels are defined by means of the metametamodel language. A metamodel is said to conform to the metametamodel. As an example, we can consider the MOF (Meta-Object Facility), which is the OMG proposal for metamodels definition [28]. The relational metamodel may conform to the MOF metametamodel.

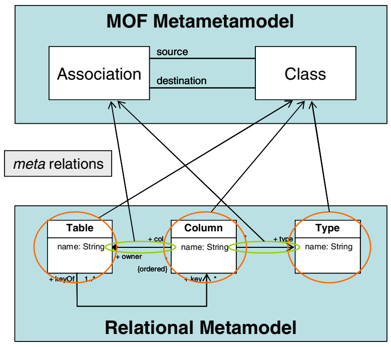


Fig. 6. The "meta" relation between M2 and M3

As models and metamodels, the metametamodel is also composed of elements. A metamodel conforms to the metametamodel if and only if each of its elements has its metaelement defined in the metametamodel. Some *meta* relations between relational metamodel elements and MOF elements are made explicit in Fig. 6. Thus, the Table, Column, and Type elements are typed by the MOF Class element, whereas relational links elements are associated with the MOF Association element.

2.4 Extensions

The previous characterization of MDE principles given above is minimal. It does not cover all aspects necessary for a workable definition. Other relations between metamodels like a clean and precise "extension" relation may be necessary. One reason this has not been completely and consensually defined is that there are ways to solve this problem when you use specific metamodels like UML. The notion of profile for example allows some kind of extensibility in this context.

2.5 Structuring Metamodels

Before engaging in the following section about comparing technologies, we see here that the strong concept in model engineering is the concept of a metamodel. As a consequence a metamodel brings engineering facilities different from grammars, XML schemas, ontologies, etc. Even if the notion of metamodel is still evolving in structure and application, we can say that a metamodel helps defining a language (DSL). This covers however a lot of different facets, some of them being described below.

Basically, as we have seen, a metamodel is a graph composed of concepts and relationships between these concepts. From a usage perspective, a metamodel is a concrete representation of a shared conceptualization. Some of these conceptualize-tions may be normative (e.g. OMG) and some are not. A metamodel acts as a filter to extract pertinent elements from a system in order to build a corresponding model. Any feature (concept or relationship) not present in the metamodel will be ignored when building the model representing the system.

Metamodels are used to define formalisms or languages (DSLs). To define a formalism, we may need to provide different kind of information for example structure knowledge, assertion knowledge, execution knowledge, display knowledge, etc. The fact that these information are provided by separate parts of a metamodel contribute to a clear separation of concerns. The fact that they are provided by separate metamodel parts combined together goes one step beyond in the direction of modularity and reusability.

Let us consider for example a classical PetriNet formalism. A Petri net is a bipartite directed graph with two kinds of nodes: Places and Transitions. It is an edge-labeled and node-labeled graph. A number of Tokens may be associated to each Place. We may define a Petri Net in four steps:

- The structural knowledge may be captured by a class diagram with concepts of *Pnet* (the global graph), *Place*, *Transition*, *Token* and relations *basicRelation* and *numberOfToken*.

- The assertional knowledge may be captured by OCL descriptions stating that the value attribute of *Token* may never be negative and that a *basicRelation* may link a *Place* to a *Transition* or a *Transition* to a *Place* but never a *Place* to a *Place* or a *Transition* to a *Transition*.

- The execution knowledge may be captured by the following description:

```
function fireable (t:Transition)
  {return true if every directly incoming Place has at least one Token else false}
context Pnet action;
  repeat
        select from pNet one arbitrary Transition t such that fireable(t);
        decrement the number of tokens for every incoming Place of t;
        increment the number of tokens for every outcoming Place of t;
  until no Transition t in pNet verifies fireable(t);
```

- The display knowledge may be captured by the following description:

- represent a *Transition* by a *Rectangle*
- represent a *Place* by a *Circle*
- represent an *Edge* by an *Arrow*

Now we may see in this description that any part is based on a separate metamodel for example the OCL metamodel or the AS (Action Semantics) metamodel or a drawing metamodel composed of concepts *Arrow*, *Circle* or a *Rectangle*. So the formalism of Petri Nets is defined not by a single but by an aggregation of metamodels. This way of "decorating" a model with another one based on a different metamodel is quite powerful and goes beyond the classical distinction between abstract and concrete syntaxes. It allows achieving separation of contents from presentation. If we wish to define an extension to Petri nets, for example colored Petri nets, then reusability may be achieved thanks to this clean separation.

We just mentioned in this example that a model may be decorated with assertions to make it more precise, but that it may also be decorated with execution annotations to provide it with some animation capabilities (e.g. simulation but not only). Models are not naturally executable, but by using some available language with precisely defined execution semantics, it is possible to animate them. In [5] the language to write execution annotations was Smalltalk but variants of Java have also been used. The OMG invest efforts in trying to standardize action semantics for UML or even for MOF.

A more regular definition of a DSL may be given as a coordinated set of models. Among these, a central domain metamodel would define the central concepts like *Place*, *Transition* or *Token* in the example above. Most of the other models will be correspondence or transformation models mapping the domain metamodel onto other DSLs, in order to provide various concrete syntaxes, to define executability or other properties. This external way to define executability by a mapping to another executable language (like Java for example) is very general.

2.6 Summary

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 [7]. 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. Models may be decorated in various ways in order to associate additional properties. The declaration itself is a model, i.e. conforms to a metamodel. The precise mechanisms for composing the various models are not yet completely understood.

3 Engineering: Structuring the Solution Space

Technical spaces were introduced in [24], 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. It is also a model management framework usually based on some algebraic structures like trees, graphs, hypergraphs, categories, etc. Although technical spaces may be difficult to define formally, they can be easily recognized (e.g. XML, MDA). In the three-level conjecture, each technical space can be seen as based 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.

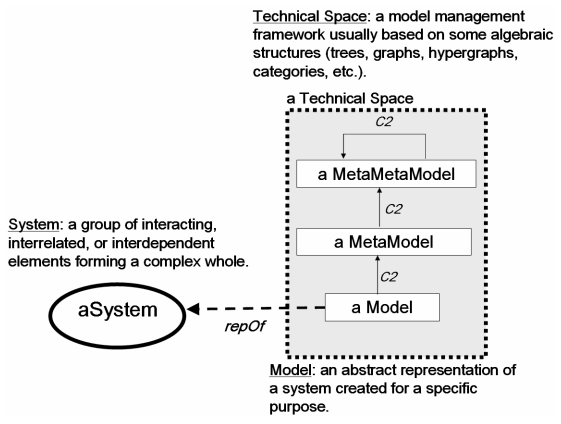

Technical Space: a model management framework usually based on some algebraic structures (trees, graphs, hypergraphs, categories, etc.).
a Technical Space
C2
a MetaMetaModel
C2
a MetaModel
C2
a Model
System: a group of interacting, interrelated, or interdependent elements forming a complex whole.
aSystem
repOf
Model: an abstract representation of a system created for a specific purpose.

Fig. 7. Systems, models and technical spaces

As illustrated in Fig. 7, the basic notions that we consider are now Systems, Models and Technical Spaces (abbreviated TSs). When we talk about a model, we should say which kind of model we are referring to. For example we could say that an XML-document is an XML-model or that a Java program is a Java-model. Proceeding in that way saves a lot of time by solving a lot of endless discussions about a Java program being or not being a model. The notion of model is a contextual one and to be non-ambiguous we need to prefix the model by its context. A TS denotes such a notion of a context. To take once again the object analogy, Smalltalk objects, Eiffel objects and C++ objects were different kinds of objects, not even able to communicate directly in the absence of some kind of Middleware support like CORBA. However Smalltalk programmers were used to talk about objects in their particular context. Similarly a MDA-model is a model that conforms to a metamodel that conforms to the MOF. When the context is clear, we may talk about a model, often meaning here MDA-model. Such a model will have specific properties, i.e. being able to be serialized in the XMI 2.1 format. When we talk about a Microsoft/DSL-model like in Fig. 8, this will be a different kind of model, not based on the MOF or directly serializable in XMI [30]. We may generalize this prefixing convention when we have to talk about models pertaining to different TSs. If we consider TSs that are organized according to the three level conjecture, we may even talk about a Java program as EBNF/Java/myProg or about an XML document as XML/MusicML/myMusic as naturally as we could talk above ECORE/UML2.0/ myModel or about MOF2.0/CWM/MyData.

Several TSs may thus be considered as based on a three level organization like the metametamodel, metamodel and model of the MDA. One example is grammarware [23] with EBNF, grammars and programs but we could also consider XML documents, RDF documents, Semantic Web, DBMS, ontology engineering, natural

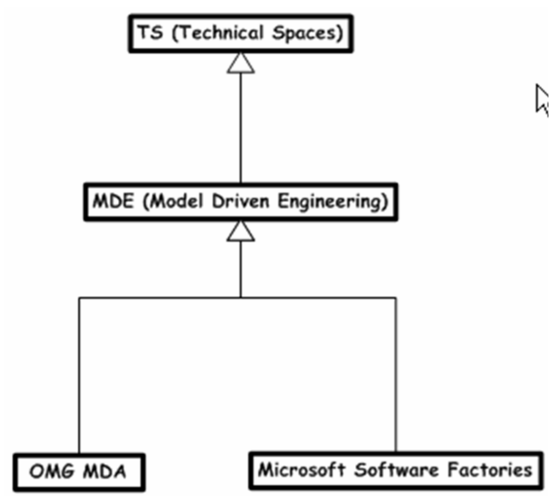


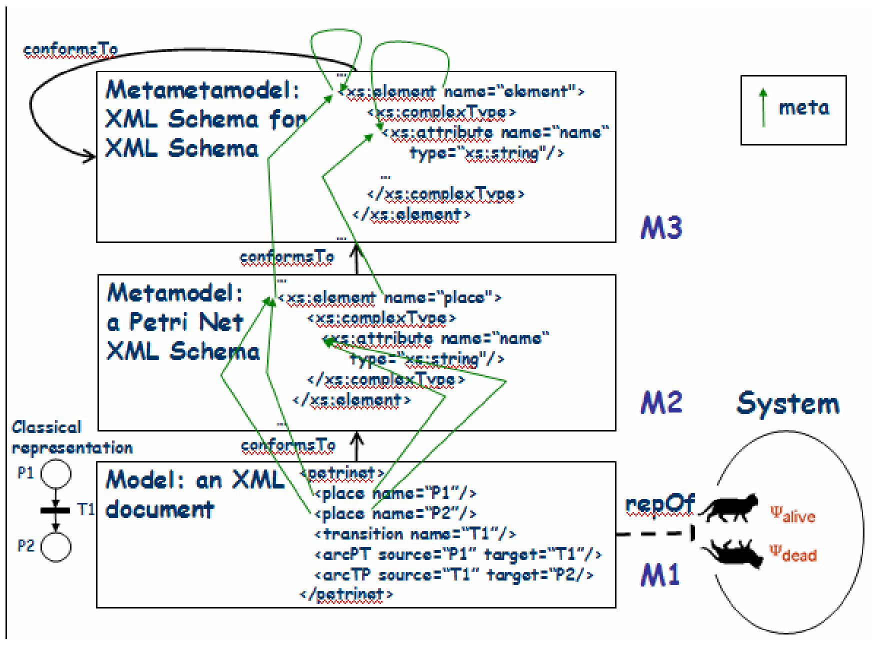
Fig. 8. Two MDE Technical Spaces

language processing systems, etc. In order to get a synergy of different technical spaces we should create conceptual and operational bridges between them, and some of these bridges are bi-directional.

The main role of the M3-level in a TS is to define the representation structure and a global typing system for underlying levels. The MOF for example is based on some kind of non-directed graphs where nodes are classes 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, for example with possibilities to have direct references from one node to another node (REFs and IDREFs). In Fig. 9 we see how a simple system may be represented as an XML document corresponding to a Petri Net XML schema. We represent in this figure the *conformsTo* relation between the document, the schema, and the schema definition. We also represent the fine grained *meta* relations presented earlier (section 2) between elements and metaelements.

As we can see, there are a lot of similarities between the XML TS and the MDA TS. To get even more convinced, we may compare this situation with a similar one expressed in the MDE TS. Here, in Fig. 10, we have chosen another specific variant of MDE called sNets based on typed, reflective, and partitioned semantic networks [8], [9].

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 navigation 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 as we have seen earlier and may be even used as a side-effect free



```
...
<xs:element name="element">
  <xs:complexType>
    <xs:attribute name="name"
      type="xs:string"/>
    ...
  </xs:complexType>
</xs:element>
...
```

```
...
<xs:element name="place">
  <xs:complexType>
    <xs:attribute name="name"
      type="xs:string"/>
  </xs:complexType>
</xs:element>
...
```

```
<petrinet>
  <place name="P1"/>
  <place name="P2"/>
  <transition name="T1"/>
  <arcPT source="P1" target="T1"/>
  <arcTP source="T1" target="P2/>
</petrinet>
```

Fig. 9. Three level structure in an XML TS

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 TS. 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.).

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 internal 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 main representation system of the corresponding space. There is no reason to change that. Now we have to consider another kind of transformation: across technical space boundaries. We call these transformations "projectors" in order to distinguish them from other transformations internal to one technical space.

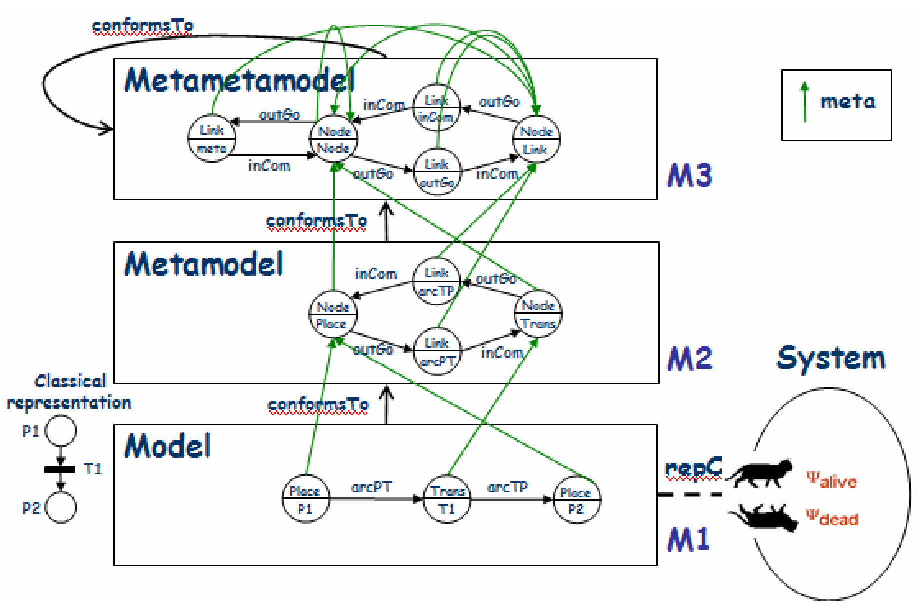


Fig. 10. Three level structure in the sNet MDE TS

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 will 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 initiated a RFP called SMIF (Stream-based 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 the XML TS 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 directly serializing graphs on text flow or binary streams, it was more interesting to serialize graphs as trees and the let the remainder of the work being handled in the XML TS. 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 TSs. 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 very costly. So, as part of the Java community process program, a standard projector with the Java technical space was defined under the name JSR #40. The capability to access models elements in MDA was given with the help of the Java TS. This projector is known today under the name JMI (Java Metadata Interface Specification [33]).

As we may see, each projector has a specific purpose. In the UML standard, the diagram interchange part deals partially with the separation of content and presentation for MDA models. 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 TSs 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 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. In the same spirit, OMG is today studying more general kinds of projectors between the MDA and the textual technical space (Model to Text RFP).

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

4 Some Examples of Technical Spaces

In this part, we give some rapid examples of TSs related to model engineering.

4.1 The OMG MDA Technical Space

We have already mentioned many of the characteristics of this MDA TS which was one of the first to explicitly state its clear foundation on some notion of concrete model. It should be noted that this TS borrowed much inspiration from the CDIF achievements as well as from the Microsoft OIM framework. CDIF and OIM are two examples of previous TSs, now extinct.

A typical presentation of the OMG/MDA organization is shown in Fig. 11. This may be used to illustrate the various roles that UML is playing in the global picture.

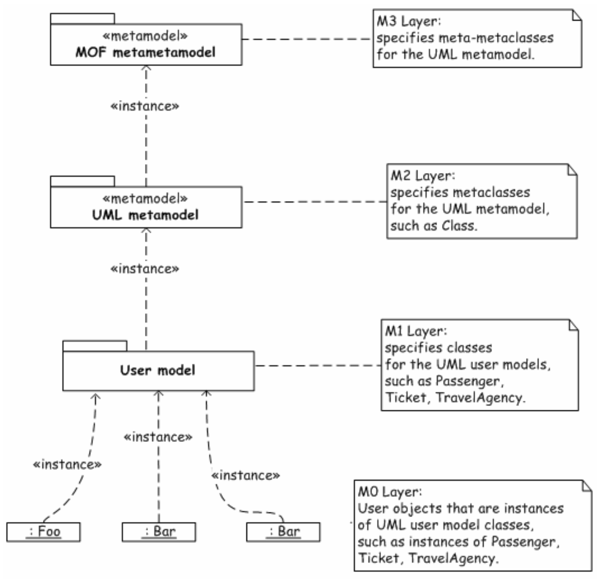

«metamodel»
MOF metametamodel
M3 Layer: specifies meta-metaclasses for the UML metamodel.
«instance»
«metamodel»
UML metamodel
M2 Layer: specifies metaclasses for the UML metamodel, such as Class.
«instance»
User model
M1 Layer: specifies classes for the UML user models, such as Passenger, Ticket, TravelAgency.
«instance»
«instance»
«instance»
: Foo
: Bar
: Bar
M0 Layer: User objects that are instances of UML user model classes, such as instances of Passenger, Ticket, TravelAgency.

Fig. 11. Typical illustration of OMG MDA Organization

There has been a lot of reorganization at OMG on the occasion of the move to UML 2.0. The idea was to achieve some simplification by taking this opportunity to align other standards as well (OCL 2.0, MOF. 2.0, XMI 2.0, etc.). The result may be considered as mitigated. For various reasons there have always been two camps at OMG, according to the role devoted to UML. For the MOF camp, UML is only one ordinary metamodel among many (CWM, SPEM, etc.) while for others UML has a special and central role in MDA; the latter view UML as a rather universal language covering most of the software engineering needs, either directly or through its profile extension mechanism. The two camps have made a working compromise stating that 1) the UML conforms to MOF but also that 2) MOF is aligned on UML. Keeping the balance between these two political views has always been a complex exercise. The fact that UML is separated in infrastructure and superstructure was a help in defining the alignment, but is not sufficient. MOF itself is now composed of two parts, EMOF (for essential MOF) and CMOF (for complete MOF).

One sub-area of the MDA work at OMG is called ADM (Architecture-Driven Modernization) and deals with model-based reverse engineering and software modernization. In this very active area, the notion of TS projector between legacy spaces and the MDA space are of paramount importance. ADM mainly deals with the utilization of metamodeling techniques for recovery PIMs from PSMs corresponding to platform of the past.

4.2 The EMF Technical Space

In theory EMF (Eclipse Modeling Framework [17]) and OMG/MDA are aligned and should be considered as only one TS. As suggested in Fig. 2, MDA may be viewed as a set of standards while EMF should be an implementation based on these same standards. In practice this is not completely true and the two may be somewhat evolving independently. The M3 level in EMF is called ECORE (see Fig. 12), and corresponds approximately to EMOF mentioned above. Another view is to consider EMF as a sophisticated projection of MDA onto the Java TS, and to a lesser extent onto the XML TS.

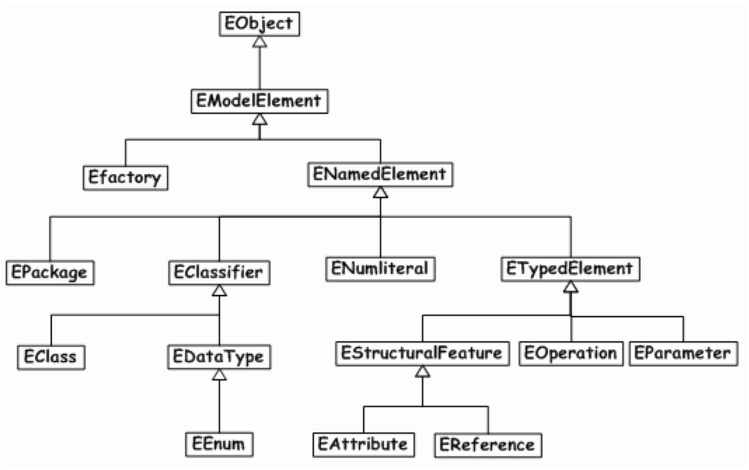


Fig. 12. The EMF ECore metametamodel

4.3 The Microsoft DSL Tools Technical Space

A general description of the concept of software factories has been presented in [20]. Starting from there, several sets of tools are being regularly released as beta-versions since December 2004. This allows us to understand in which direction the modeling activities are leading at Microsoft.

In order to define a DSL toolkit for a specific purpose (e.g. for a business of deigning airports), one will proceed as follows:

- Define the 'object model' (abstract syntax or metamodel) of the language -- that is, the concepts and relationships you want to handle in it.
- Define a graphical concrete syntax for the language -- the boxes, lines, etc that represent the concepts on-screen.
- Create a graphical editor for the language that you will use to design a specific airport.
- Develop code generators that will create software, configuration files, reports and other artifacts from the graphical model.

The choice of Microsoft DSL Tools has been to map mainly to the XML technical Space for handling models and metamodels. Executability is provided by mapping to the Dot Net TS. The M3 level at Microsoft is left implicit, but could be reified somewhat as illustrated in Fig. 13. An operational bridge between EMF and Microsoft Software factories may be found in [12].

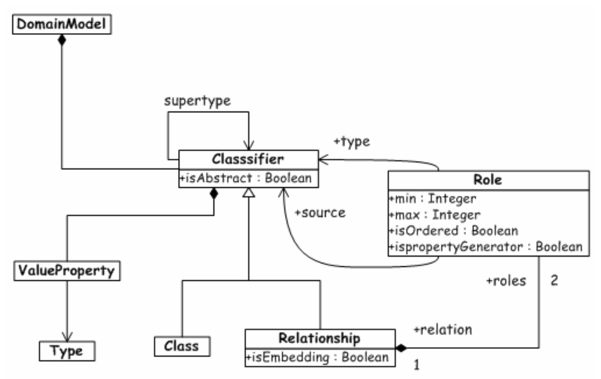


Fig. 13. A tentative to explicit the M3 level of Microsoft DSL Tools (simplified)

4.4 Other Technical Spaces

There are plenty of other technical spaces besides the three MDE ones that we have just briefly introduced. A list of these spaces is obviously not realistic here. However it is important to recognize them when they are involved in relation to MDE. The major one is probably the XML Document space that has taken considerable importance in the last decade. We have seen how the OMG has established links with this space through standards like XMI. This is even more important in Microsoft DSL Tools that are making more usage of XML mappings.

Another very important technical space is programming languages, e.g. Java. We can even say that EMF is mainly concerned with the bridging of MDA and Java. This had previously been achieved with JMI in other environments like MDR/NetBeans, but to a much less ambitious scale.

An interesting reading about technical spaces in the domain of web services is [21]. Although not naming explicitly the concept of technical space, this paper considers three complementary ones, namely objects, SQL and XML. The author notices that each of these solutions has strengths and weaknesses when applied to the inside and outside of web service boundary. He then concludes that the strength of each of these solutions in one area is derived from essential characteristics underlying its weakness in the other area. In other words, the multiplicity of technical space is not only a fact of life but it has also many positive effects.

Bridging to another technical space is interesting if it has something interesting to bring. We have already seen the advantage of using XML or Java instead of reinventing the wheel.

Interesting bridges could also be built with ontology engineering and web semantic. Some functionalities may be easier to provide in a TS with a M3 based on OWL than on the MOF. For example name management seems superior in OWL where a given object may be referred by several different names. Also in OWL the possibility to infer from the properties of an individual that it is a member of a class may be of interest. Through this example we understand more clearly the fundamental relations between representation and reasoning. Reasoning on a model is usually considered a more complex operation than just querying this model. It would be unwise to try implementing in the MDA TS all the reasoning facilities available in the ontology engineering or description logic TS for example. It seems much more valuable to build specific projectors when needed.

Having surveyed the basic MDE principles and having placed them in the context of multiple TSs, it remains now to prove that this approach may lead to real and usable implementations. We will use the example of AMMA (ATLAS Model Management Architecture), a platform built in our team to demonstrate the feasibility of these model-centric approaches to software engineering, system engineering, and data engineering.

5 Architectural Style for an MDE Platform

This section will describe an architectural style for MDE composed of four functional blocks illustrated with prototypes running in the AMMA platform:

- Model transformation (ATL)
- Model weaving (AMW)
- General model management (AM3)
- Model projection to/from other technical spaces (ATP)

This architectural style and the feasibility of its implementation will be illustrated by the description of the AMMA platform. The architecture of the current EMF-based AMMA implementation is described in Fig. 14. The transformation tool of AMMA, ATL, uses the basic features of EMF to handle both source and target models and metamodels, as well as the transformation model and metamodel. An Integrated Development Environment (IDE) has been developed for ATL on top of Eclipse. Based on EMF, it makes use of many other features, such as the code editor and the code debugging frameworks. AMW, the AMMA model weaving tool, uses more advanced EMF features. Since it is built as a model editor, AMW can benefit from editing domains facilities for complex model handlings (including undo-redo). It also reuses some components of the Eclipse default views to display models.

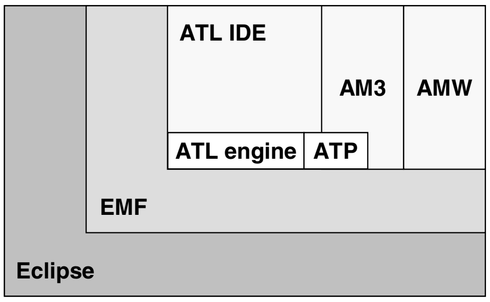


Fig. 14. Architecture of the AMMA platform

Eclipse is mostly used as an IDE for software development. As such, it includes facilities enabling to navigate the code, to keep track of the files that need rebuilding, etc. The megamodel tool (AM3) is used as a model-oriented extension of these abilities. As a matter of fact, using the relations between models (such as the source and target relations between a transformation model and its source and target metamodels), and between models and tools (such as those provided by ATP), AM3 makes it possible to easily carry on complex weaving, transformation and projection tasks.

5.1 MMA: A Model Engineering Platform

AMMA has both local and distributed implementations and is based on four blocks (Fig. 15) providing a large set of model processing facilities:

- the Atlas Transformation Language (ATL) defines model transformation facilities;
- the Atlas ModelWeaver (AMW) makes it possible to establish links between the elements of two (or more) different models;
- the Atlas MegaModel Management (AM3) defines the way the metadata is managed in AMMA (registry on the models, metamodels, tools, etc.);
- the Atlas Technical Projectors (ATP) defines a set of injectors/extractors enabling to import/export models from/to foreign technical spaces (Java classes, relational models, etc.).

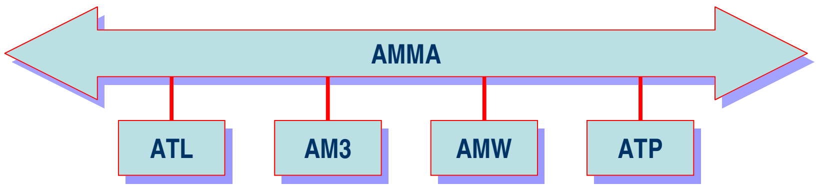


Fig. 15. The AMMA platform

5.2 ATL: Transforming Models

5.2.1 ATL Presentation

ATL is a model transformation language, having its abstract syntax defined using a metamodel. This means that every ATL transformation is in fact a model, with all the properties that are implied by this. Fig. 16 provides the scheme of the transformation of a model M_a (conforming to MM_a) into a model M_b (conforming to MM_b) based on the M_t transformation (which itself conforms to ATL transformation language).

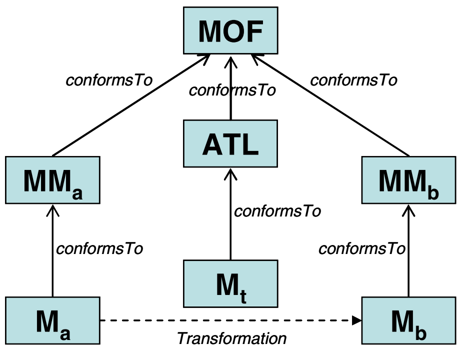


Fig. 16. An ATL transformation

What appears on Fig. 16 is the postulate of the existence of this common family of model transformation languages. This is exactly what OMG is presently trying to define through MOF/QVT. A given transformation operation is thus represented as follows:

$$Mb \leftarrow f\,(MMa, MMb, Mt, Ma)$$

This means that a new target model Mb based on metamodel MMb is obtained from the source model Ma based on metamodel MMa, by applying a transformation Mt based on the standard transformation language.

5.2.2 The ATL Metamodel

The ATLAS transformation language is defined by the way of a metamodel (Fig. 16) taking inspiration from the OCL 2.0, which may be considered here as an assertion and as a navigation language at the same time.

ATL transformations are stored in QVTUnits. QVTUnits are composed of QVTOperators, which are composed of TransformationDescription. A TransformationDescription is an abstract class, which has two sub classes: Action and Context.

Context: The context is used to store the variables and models manipulated by the transformation.

Action: The Action class is the element of the language that will describe the action needed to perform a transformation. The actions have to be executed in specific order, which is defined in the position attribute. This notion of order is necessary. We cannot for example set an attribute to a class if the class has not been created.

There are three types of Actions: CreateInstance, PropertyOperation, AddTransientLink.

The rest of ATL metamodel is the description of Expression sub-classes. ATL expression classes are a copy of a part of OCL Expression sub classes. An important extension has been made: the QueryTransientLinkExp that is a sub-class of CallExp. It is used to navigate through the transient links.

5.2.3 An Example of a Transformation in ATL

Several examples of model transformation in ATL are provided as an open source contribution on the Eclipse/GMT Website. This is a tentative to build a first library of reusable model transformations. Among the fifty examples currently available, we chose one particular for illustrative purpose here: http://www.eclipse.org/gmt/atl/atlTransformations/#Java2Table. The complete code and documentation is available from the Web site. This example aims to compute a static call graph from a Java program and to present it in a tabular way (in an Excel spreadsheet or in an HTML Table). The following comments on this example are typical of ATL transformations.

a) Although a conventional transformation from UML 2.1 to UML 2.1 (e.g. refactoring) with source models and metamodels in XMI, target models and metamodels in XMI may be written in ATL without much difficulties, many examples are usually more diverse and more specific.
b) Here we describe in the source metamodel only a small subset of Java programs. More precisely we consider that a Java program is composed of class definitions, each one being composed of methods definitions and each method definition in turn being composed of a number of method calls. The other characteristics of a Java program are not captured by this metamodel.
c) The process of practically getting this Java metamodel expressed in XMI is rather complex, besides the fact that XMI exists in many non compatible versions. As a standard procedure we should define a specific UML class diagram with a standard tool like Poseidon, then get the corresponding XMI output file and input it to a "model promotion" tool like UML2MOF available in the MD/NetBeans tool suite. The resulting XMI output file could serve in the transformation as the definition of the java metamodel. As we can see this procedure is rather cumbersome. As an alternative we have defined a DSL for specifying metamodels called KM3 (Kernel MetaMetaModel [2]). This is a textual language with a Java-like syntax and basic support available in the Eclipse/GMT project, for example XMI conversion tools. ATL accepts the source and target metamodels in KM3.
d) As already noticed, the source metamodel does not cover much details of the Java syntax and this is an advantage on using a Java metamodel corresponding for example to the full Java grammar. We see here one additional characteristics of model transformation: the metamodels should be tailored to the transformation task at hand. Using an over-dimensioned source Java

metamodel would have made the transformation more complex and less secure. The metamodels play the role of type and the models of variables. It is of high importance to use the most accurate metamodel for reasons of clarity and reliability of the transformation. A theoretical scheme would allow building a transformation μ taking as input a metamodel Ms and a transformation α and producing as output a new metamodel Mt, reduction of Ms to the only exact needs of transformation α.

e) Now that we have discussed the source metamodel characteristics, we have to face the real situation that Java programs are naturally and usually expressed as plain source text programs and not as XMI representations. As a matter of fact, there is a very restricted number of information naturally expressed in standard XMI in the real world. So what we have to consider here is a bridge between the Java and the MDA TS, i.e. a projection. We suppose that such a projection exists in the ATL projection library (see ATP below). However if we look at the actual Java2Table example, we realize that such projections have not been realized directly but instead that the author found more convenient to cross another TS (XML) to achieve the transformation. Among various reasons for this decision, the existence of the JavaML DTD that allowed to consider all the class definitions in one single file. This is an example of a possible implementation choice.
f) Now that we have discussed the source model, metamodel and projector we may turn our attention to the corresponding target items. The first work is to define a metamodel for Excel, obviously not provided with the tool. Here again we notice that we don't need a full metamodel but a very simplified one, tailored to our transformation needs. Formulas are not needed but a *Spreadsheet* could be considered as composed of *Rows*, each being composed of a *Cell* with a contained value.
g) Once we have defined the target metamodel, we need to build the corresponding projectors. This could be implemented with specific MS tools like Visual Basic or more likely again through the XML import/export facilities available in the MS Office suite. In the process of doing this we realize that the target metamodel could as well correspond to HTML or XHTML tables. As a consequence this is the final implementation choice in the provided example. More precisely the target metamodel is an abstract definition of tabular presentation. The result of this transformation could then be chained to another transformation generating specific XHTML or Excel tables, with the metamodels specific to these tools. Of course chains of transformations are important in many practical situations.

5.3 AMW: Weaving Models

Model weaving operations are performed between either metamodels (two or more), or models. They aim to specify the links, and their associated semantics, between elements of source and target models. Concerning the set of links to be generated, the following issues may be considered:

- this set of links cannot be automatically generated because it is often based on human decisions. The generation can however be partially automated by means of heuristics;
- it should be possible to record this set of links as a whole, in order to use it later in various contexts;
- it should be possible to use this set of links as an input to automatic or semi-automatic tools.

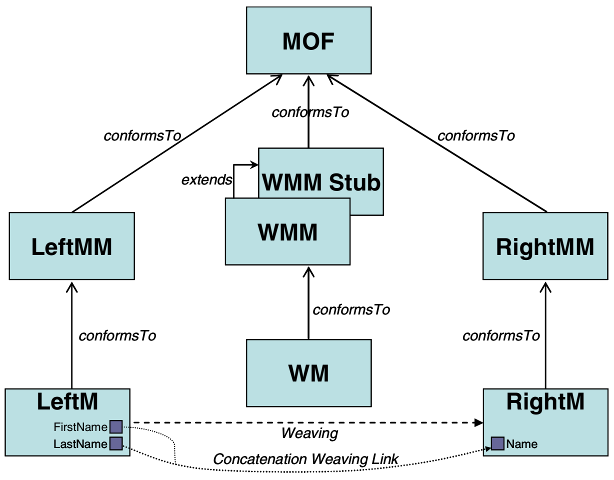


Fig. 17. The model weaving scheme

As a consequence, we come to the conclusion that a model weaving operation produces a precise model *WM*. Like other models, this should conform to a specific metamodel *WMM*. The produced weaving model relates to the source and target models *LeftM* and *RightM*, and thus remains linked to these models in a megamodel registry.

Each link element of the produced weaving model *WM* has to be typed by an element of a given *WMM* metamodel. There is no unique type of link. Link types should provide weaving tools with useful information. Even if some links contain textual descriptions, these are valuable for tools supporting documentation, manual refinements or applying heuristics.

One may assume that there is no standard metamodel for weaving operations since most developers define their own. However, we suppose there is a stub weaving metamodel, and that this stub is extended by specific metamodel extensions. Thus, a given weaving metamodel may be expressed as an extension of another weaving metamodel. This allows building a general weaving tool able to generically deal with weaving tasks. Fig. 17 describes a simple model weaving scheme in which an explicit weaving link (of type *Concatenation*) associates two source elements (*FirstName* and *LastName*) with an only target element (*Name*).

Mapping heterogeneous data from one representation to another is a central problem in many data-intensive applications. Examples can be found in different contexts such as schema integration in distributed databases, data transformation for data warehousing, data integration in mediator systems [25], data migration from legacy systems [14], ontology merging [18], schema mapping in P2P systems [22], workflow integration [27], mapping between context and ontologies [16].

A typical data mapping specifies how data from one source representation (e.g. a relational schema) can be translated to a target representation (e.g. a XML schema). Although data mappings have been studied independently in different contexts, there are two main issues involved. The first one is to discover the correspondences between data elements that are semantically related in the source and target representations. This is called schema matching in schema integration [3] and many techniques have been proposed to (partially) automate this task, e.g. using neural networks. After the correspondences have been established, the second issue is to produce operational mappings that can be executed to perform the translation. Operational mappings are typically declarative, e.g. view definitions or SQL-like queries. Creating and managing data mappings can be very complex and time-consuming if done manually. Recent work in schema integration has concentrated on the efficient management of data mappings. For instance, Clio [26] provides techniques for the automatic generation of operational mappings from correspond-dences obtained from the user or a machine learning technique. ToMAS [34] also provides techniques for the automatic generation of operational mappings as well as their consistency management while schemas evolve. This work is significant as it can be the basis to general purpose data integration tools.

5.4 AM3: Global Model Management

The Atlas MegaModel Management tool, AM3, is an environment for dealing with models or metamodels, together with tools, services and other global entities, when considered as a whole. For each platform, we suppose that there is an associated megamodel defining the metadata associated to this platform. Within the content of a given platform (local or global), the megamodel records all available resources. One may also refer to these resources as "model components" [10]. The megamodel can be viewed as a model which elements represent and refer to models and metamodels [13]. Represented as models, available tools, services, and services parameters are also managed by the megamodel. There are plenty of events that may change the megamodel, like the creation or suppression of a model, or a metamodel, etc. A megamodel is associated with a specific "scope" and conforms to a specific metamodel.

5.5 ATP: Projection Between Technical Spaces

The Atlas Technical Projectors, ATP, define a set of injectors and extractors, which can be seen as import and export facilities between the model engineering Technical Space and other TSs (databases, flat files, XML, etc). Indeed, a very large amount of pre-existing data that is not XMI compliant would greatly benefit from model transformation. In order to be processed by a model engineering platform, this data needs injection from its TS to the model engineering TS. The need for extraction is also quite important: many existing tools do not read XMI. A simple example is the Java compiler. What we need here is code generation, which may be seen as a specific case of model extraction. Many other TSs require both injectors and extractors: database systems provide another example in which database schemes have to be generated from model definitions.

5.6 Conclusions

What appear in this presentation are the high complementarities between all four presented functional blocks (ATL, AMW, AM3, and ATP). There are plenty of applications that make use of these four kinds of functionalities at the same time.

6 Conclusions

We have presented in this paper our definition of MDE basic principles and our view of an MDE implementation architectural style. The basic principle on which this work is based (Models as first class entities) is common to many current research communities (Model Management, Model Integrated Computing, etc.) and similar goals and means may be found in other TSs. This is summarized in Fig. 18.

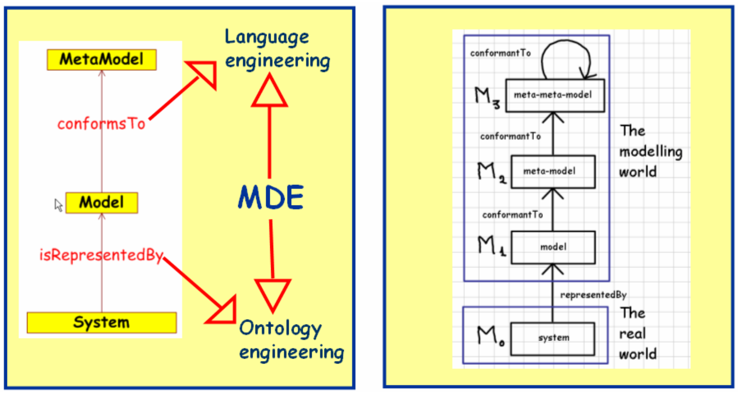


Fig. 18. Summarizing the Basic Principles

We have taken here a broad view of model engineering as encompassing not only the MDA™ OMG proposal or the Microsoft SoftwareFactories/DSL view, but also other approaches like Model Integrated Computing, Generative Programming, Model Management and many more. We distinguished the three levels of principles, standards, and tools to facilitate the discussion. We suggested the idea that there may exist a common set of principles that could be mapped to different implementation contexts through the help of common standards. We have illustrated our claim with AMMA, an architectural organization that is currently mapped onto the EMF extension of the Eclipse platform.

One contribution of this work has been to propose a precise and minimal definition of a conceptual MDE technical space. This space may be considered as a general graph where partitions are composed of model, metamodel and metametamodel entities. We have not committed here to a particular kind of graphs. The OMG/MOF graphs, the EMF/Ecore graphs or the Microsoft/SoftwareFactories/DSL graphs are not completely identical but we believe these systems share one common set of principles and definitions corresponding to the MDE abstract global typing system presented here. As a consequence this work should be useful not only to relate different technical spaces like XML, Grammarware, etc., but also to compare variants of the MDE space.

One contribution of this work is the AMMA conceptual architecture, seen as an intermediary level between model engineering basic principles and executable systems running on operational platforms like EMF/Eclipse. The main advantage of proceeding in this way is that we may more clearly evaluate the gap between principles and implementation. From our initial experimentations, we came to the conclusion that building a model engineering platform is much more demanding than simply providing a RPC-like mechanism for allowing tools to exchange models in serialized format (e.g. XMI-based), with the corresponding services and protocols (e.g. Web Service-based). The present state of AMMA with the four functional blocks is only one step in this direction and still needs many extensions.

There are many variants of model engineering. Our attitude has been to find the set of basic principles common to all the dominant model engineering approaches and to make them explicit. We are then in a position to clearly separate the principles, the standards, and the tools levels.

One of the contributions of our approach is also to take explicitly into account the notion of technical space. Instead of building a lot of different ad-hoc conversions tools (modelToText, textToModel, ontologyToModel, modelToOntology, XMLToText, textToXML, modelToSQL, SQLToModel, etc.), we have proposed, with the notion of projectors (injectors or extractors), a general concept that may be used in various situations. These projectors can be selected as either front-ends or back-ends for classical transformations.

Acknowledgements

I would like to thank Freddy Allilaire, Marcos Didonet del Fabro, Frédéric Jouault, Ivan Kurtev, 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).

References

1. ATL, ATLAS Transformation Language Reference site http://www.sciences.univ-nantes.fr/lina/atl/
2. ATLAS Group KM3: Kernel MetaMetaModel. Available at http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/doc/atl/index.html

3. Batini, C., Lenzerini, M., and Navathe, S. B. 1986. A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys 18, 4, 323–364.
4. Bernstein, P.A., Levy, A.Y., Pottinger, R.A., A Vision for Management of Complex Systems, MSR-TR-2000-53, Microsoft Research, Redmond, USA, ftp://ftp.research.microsoft.com/pub/tr/tr-2000-53.pdf.
5. Bézivin J., Lemesle R. The sBrowser: a Prototype Meta-Browser for Model Engineering. Proceedings of OOPSLA'98, Vancouver, Canada, 18-22 October 1998. (http://www.metamodel.com/oopsla98-cdif-workshop/bezivin2/)
6. Bézivin, J. From Object Composition to Model Transformation with the MDA TOOLS'USA 2001, Santa Barbara, August 2001, Volume IEEE publications TOOLS'39. http://www.sciences.univ-nantes.fr/info/lrsg/Recherche/mda/TOOLS.USA.pdf
7. 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/up5-2Presentation.pdf
8. Bézivin, J. Lemesle, R. Towards a true reflective modeling scheme LNCS, ISSN: 0302-9743, Vol. 1826/2000, http://www.springerlink.com/media/3G267U4QVH5RRJ47VBFT/Contributions/2/8/4/W/284W7VGQC302VR5W.pdf
9. Bézivin, J. sNets: A First Generation Model Engineering Platform. In: Springer-Verlag, Lecture Notes in Computer Science, Volume 3844, Satellite Events at the MoDELS 2005 Conference, edited by Jean-Michel Bruel. Montego Bay, Jamaica, pages 169-181.
10. Bézivin, J., Gérard, S. Muller, P.A., Rioux, L. MDA Components: Challenges and Opportunities, Metamodelling for MDA, First International Workshop, York, UK, (November 2003), http://www.cs.york.ac.uk/metamodel4mda/onlineProceedingsFinal.pdf
11. Bézivin, J., Gerbé, O. Towards a Precise Definition of the OMG/MDA Framework ASE'01, San Diego, USA, November 26-29, 2001 http://www.sciences.univnantes.fr/lina/atl/publications/ASE01.OG.JB.pdf
12. Bézivin, J., Hillairet, G., Jouault, F., Kurtev, I., Piers, W. bridging the MS/DSL Tools and the eclipse EMF Framework. OOPSLA Workshop on Software Factories, http://softwarefactories.com/workshops/OOPSLA-2005/Papers/Bezivin.pdf
13. Bézivin, J., Jouault, F., Valduriez, P., On the Need for Megamodels, OOPSLA & GPCE, Workshop on best MDSD practices, Vancouver, Canada, 2004.
14. Bisbal J., Lawless D., Wu B., Grimson, J. Legacy Information Systems: Issues and Directions. IEEE Software, September/October 1999, pp. 103-111, Vol. 16, Issue 5. 1999.
15. 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
16. Bouquet P., Giunchiglia F., Van Harmelen F., Serafini L., Stuckenschmidt H.: Contextualizing Ontologies. Journal of Web Semantics, 1(4):1-19, 2004
17. Eclipse Modeling Framework (http://www.eclipse.org/emf/)
18. Ehrig M., York Sure: Ontology Mapping - An Integrated Approach. ESWS 2004: 76-91
19. GMT, General Model Transformer Eclipse Project, http://www.eclipse.org/gmt/
20. Greenfield, J., Short, K., Cook, S., Kent, S., Software Factories, Wiley, ISBN 0-471-20284-3, 2004.
21. Helland, P. Data on the outside versus data on the inside.2005 CIDR Conference.
22. Kementsietsidis A., Arenas M., Miller, R. J. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In Proceedings of the SIGMOD International Conference on Management of Data (SIGMOD'03), San Diego, USA, pages 325-336. 2003.

23. Klint, P., Lämmel, R. Kort, J., Klusener, S., Verhoef, C., Verhoeven, E.J. Engineering of Grammarware. http://www.cs.vu.nl/grammarware/
24. 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/
25. Lenzerini M., Data integration: a theoretical perspective, Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, June 03-05, 2002, Madison, Wisconsin
26. Miller, R. J., Hernandez, M. A., Haas, L. M., Yan, L.-L., Ho, C. T. H., Fagin, R., and Popa, L. 2001. The Clio Project: Managing Heterogeneity. SIGMOD Record 30, 1, 78–83.
27. Omelayenko B. RDFT: A Mapping Meta-Ontology for Business Integration, In: Proceedings of the Workshop on Knowledge Transformation for the Semantic Web (KTSW 2002) at the 15-th European Conference on Artificial Intelligence, 23 July, Lyon, France, 2002, p. 76-83
28. OMG/MOF Meta Object Facility (MOF) Specification. OMG Document AD/97-08-14, September 1997. Available from www.omg.org
29. OMG/RFP/QVT MOF 2.0 Query/Views/Transformations RFP, OMG document ad/2002-04-10. Available from www.omg.org
30. OMG/XMI XML Model Interchange (XMI) OMG Document AD/98-10-05, October 1998. Available from www.omg.org
31. Seidewitz, E. What do models mean? IEEE Software, IEEE Software,September/October 2003 (Vol. 20, No. 5)
32. Soley, R., and the OMG staff, Model-Driven Architecture, OMG Document, November 2000, http://www.omg.org/mda
33. Sun Java Community Process JMI Java MetaData Interface Specification Available from ftp://ftp.java.sun.com/pub/spec/jmi/asdjhfjghhg44/jmi-1_0-fr-spec.pdf
34. Velegrakis Y., Miller R. J., Popa L., Adapting Mappings in Frequently Changing Environments, Int. Conf of Very Large Databases (VLDB), Sep 2003.

Appendix: Acronyms

Due to the initial normative aspect of the field, we have used an important number of acronyms in this document. We provide below a list of more common ones with their definitions.

ADM	Architecture-Driven Modernization
AS	Action Semantics
CDIF	CASE Data Interchange format
CORBA	Common Object Request Broker Architecture
CIM	Computation Independent Model
CWM	Common Warehouse Metadata
DTD	Document Type Definition
EAI	Enterprise Application Integration
EBNF	Extended Backus-Naur Form

EDOC	Enterprise Distributed Object Computing
EJB	Enterprise Java Beans
HUTN	Human Usable Textual Notation
IDL	Interface Definition Language
JSR	Java Specification Request
JMI	Java MetaData Interface Specification
MDA	Model Driven Architecture (OMG™)
MDE	Model Driven Engineering
MDD	Model Driven Development (OMG™)
MDSD	Model Driven Software Development
MDSE	Model Driven Software Engineering
MIC	Model Integrated computing
MOF	Meta-Object Facility
OCL	Object Constraint Language
OIM	Open Information Model
OMA	Object Management Architecture
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform specific Model
RFP	Request for Proposal
RAS	Reusable Asset Specification
RUP	Rational Unified Process
SMIF	Stream-based Model Interchange Format
SPEM	Software Process Engineering Metamodel
TS	Technical Space
UML	Unified Modeling Language (OMG™)
XMI	XML Model Interchange
XML	eXtensible Markup Language