Modeling in the Large and Modeling in the Small*

Jean Bézivin, Frédéric Jouault, Peter Rosenthal, and Patrick Valduriez

Atlas Group, INRIA and LINA, University of Nantes,
2, rue de la Houssinière - BP92208, 44322 Nantes Cedex 3, France
FirstName.LastName@univ-nantes.fr
Patrick.Valduriez@inria.fr

Abstract. As part of the AMMA project (ATLAS Model Management Architecture), we are currently building several model management tools to support the tasks of modeling in the large and of modeling in the small. The basic idea is to define an experimental framework based on the principle of models as first class entities. This allows us to investigate issues of conceptual and practical interest in the field of model management applied to data-intensive applications. By modeling in the small, we mean dealing with model and metamodel elements and the relations between them. In this sense, ATL (ATLAS Transformation Language) allows expressing automatic model transformations. We also motivate the need for the "ModelWeaver" which handles fine-grained relationships between elements of different metamodels with a different purpose than automatic model transformation. By modeling in the large, we mean globally dealing with models, metamodels and their properties and relations. We use the notion of a "MegaModel" to describe a registry for models and metamodels. This paper proposes a lightweight architectural style for a model-engineering platform as well as a first prototype implementation demonstrating its feasibility.

1 Introduction

Following the seminal work of Deremer and Kron in 1976 [9], we believe that the situation in the modeling area today is quite similar to the situation described at that time in the programming area. Starting from this similarity, we distinguish in this paper the two related activities of "modeling in the large" and "modeling in the small" which we illustrate with specific examples. The term “Megamodel” has been chosen to convey the idea of modeling in the large, establishing and using global relationships and metadata on the basic macroscopic entities (mainly models and metamodels), ignoring the internal details of these global entities. There is probably not going to be a unique monolithic modeling language (like UML 2.0) but instead an important number of small domain specific languages (DSLs) [6], [10] and this will only be possible if these small DSLs are well coordinated. To avoid the risk of fragmentation [19], we need to offer a global vision, which can be provided by the activity of modeling in the large. On the contrary, there will always be an important need to precisely define associations between model or metamodel elements, i.e. looking inside the

* This work is performed in the context of the "ModelWare" IST European project 511731.

U. Aßmann, M. Aksit, and A. Rensink (Eds.): MDAFA 2003/2004, LNCS 3599, pp. 33 – 46, 2005.

global entities. This activity of modeling in the small will be illustrated here by the two related but different examples of model transformation and model weaving.

This paper is organized as follows. Section 2 recalls the main characteristics of the MDE approach (Model Driven Engineering) and illustrates them within the particular example of the AMMA (Atlas Model Management Architecture) project. Section 3 presents model transformation operations with a focus on ATL (Atlas Transformation Language). Section 4 describes model weaving operations and their implications in the context of the ATLAS Model Weaver (AMW), another important tool in the AMMA platform. In particular, we discuss the conceptual differences between model transformation and model weaving. Section 5 describes global model management facilities and shows their practical impact with the help of the ATLAS MegaModel Management tool (AM3) that is intended to support modeling in the large activities in the AMMA platform.

2 AMMA: The Atlas Model Management Architecture

AMMA consists of two main sets of tools, one set of tools for modeling in the small (model transformation and model weaving) and another set of tools for modeling in the large based on what we call megamodels [5].

2.1 Models

A model is an artifact that conforms to a metamodel and represents a given aspect of a system. These relations of conformance and representation are central to model engineering [3]. A model is composed of model elements and conforms to a metamodel. This means that the metamodel describes the various kinds of contained model elements and the way they are arranged, related and constrained. A language intended to define metamodels is called a metametamodel.

In November 2000, the OMG proposed a new approach to interoperability named MDA™ (Model Driven Architecture) [20]. In MDA, the metametamodel is MOF [15] (Meta Object Facility) and the transformation language is based on the QVT 1.0 (Query View Transformation) specification [16]. MDA is one example of a much broader approach known as Model Driven Engineering (MDE), encompassing many popular research trends such as generative programming [8], domain specific languages, model integrated computing, model driven software development, model management and much more.

A basic principle in MDE is to regard models as first class entities. Besides the advantage of conceptual simplicity, it also leads to clear architecture, efficient implementation, high scalability and good flexibility. As part of several projects, open source platforms are being built with the intention to provide high interoperability not only between recent model based tools, but also between legacy tools.

There are other representation systems that may also offer, outside the strict MDA or even MDE boundaries, similar model engineering facilities. We call them technical spaces [13]. They are often based on a three-level organization similar to the metametamodel, metamodel and model of the MDA. One example is grammarware [12] with EBNF, grammars and programs, but we could also consider XML docu-

ments, Semantic Web, database systems, ontology engineering, etc. A Java program may be viewed as a model conforming to the Java grammar. As a consequence, we may consider, in the OMG scope, strict (OMG)-models (i.e. MOF-based like a UML model); but we may also consider outside of this scope more general models such as a Java source file, an XML document, a relational database schema, etc. A strict OMG-model may be externalized as an XMI document and conforms to MOF-conforming metamodel. In our approach we deal with OMG-models but also with non-OMG models, based on other metametamodels.

2.2 Open Platforms for MDE

The advantage of using a model-based platform is that it allows many economies of scale. For example model and metamodel repositories may handle efficient and uniform access to these models, metamodels and their elements in serialized or any other mode. Transactional access, versioning and many other facilities may also be offered, whatever the kind of considered model: executable or not, product or process, transformation, business or platform, etc. An MDE platform is primarily intended for tool integration. Several tools are usually available on such a platform.

AMMA defines a lightweight architectural style for MDE platforms. It is based on the classical view of the software bus, adapted to the basic model engineering principles, and may support local or distributed implementations. A local platform is conceptually similar to a software factory as described in [10]. Most of the tools available in our current implementation of AMMA, that will be described later, are open source tools, such as ATL, AWM, ATP and AM3. More than tools, these represent minimal functional blocks in the abstract platform architecture. There is a set of conventions, standards and protocols for plugging or unplugging MDE tools from the AMMA platform. As an example, XMI is one standard for exchanging models and metamodels in serialized formats. Many other conventions will however allow other forms of communication between tools operating on a platform.

The AM3 tool described in Section 5 defines the way metadata on a given platform is managed in AMMA (registry on the models, metamodels, tools, services and all other global entities accessible at a given time in a given scope). The fact that these metadata are externally handled by megamodels allows achieving simplicity of the MDE platform. Extending the scope of a local platform to a given distributed environment may be performed by operations on the connected megamodels. The use of a megamodel allows keeping the architecture of the software bus very simple because the complexity is mainly handled externally by these megamodels conforming to specific and adapted metamodels. The megamodel will typically describe artifacts (models, metamodels, transformations, etc.), tools and services available in a given scope. The management of tools and services may borrow ideas from the Web service area (WSDL, UDDI, etc.) but we don't wish to reinvent heavyweight stream-based and event-based CORBA-like protocols for handling model-management tool interoperability on top of the Web. Instead, in the spirit of model engineering, we prefer simple, adaptive and extensible solutions, based on generative approaches and borrowing their power from the handling of metadata outside of the platform itself, in these well-defined megamodels.

3 ATL: The Atlas Transformation Language

This section presents the ATLAS model Transformation Language (ATL) and its environment: an execution virtual machine and an IDE. ATL provides internal MDE transformations, but we may also need facilities for handling external specific external formats. This is handled by the way of projectors described at the end of the section.

3.1 Model Transformation Languages

A model transformation language is used to define how a set of source models is visited to create a set of target models. The language defines how the basic operations on models can be performed using a specific set of language constructs (declarative rules, imperative instruction sequences, etc.).

More complex transformation scenarios can be expressed using this simple definition. The set of source models can include a parametric model used to drive the transformation on a specific path: this is the equivalent of the command line options given to UNIX tools. Among target models, there can be such models as trace models. In the context of model transformation, traceability is the process of collecting information on a running transformation for later use. There are different kinds of traceability ranging from the simple (and heavy) recording of every action performed to lighter, more specialized and abstract traces which only keep links between some source and target elements of interest.

3.2 ATL

ATL is a model transformation language, which has 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. For instance, a transformation program can be the source or the target of another model transformation. ATL has been designed as an answer to the QVT RFP [16] and is consequently in the MDA space. However, we have ongoing work on M3 level independence: enabling the possibility to write transformations for any MDE platform.

ATL is a hybrid of declarative and imperative constructs. While the recommended style to write transformations is declarative, imperative concepts are implemented to let the transformation writer decide which style is the more appropriate depending on the context. The expression language is based on OCL 2.0 (Object Constraint Language).

In declarative ATL, a transformation is composed of rules. Each rule specifies a set of model element types (coming from the source metamodels), which are to be matched, along with a Boolean expression, used to filter more precisely the set of matched elements (e.g. all classes with a name beginning with a “C”). This constitutes the source pattern, or left-hand side, of the rule. The target pattern, or right-hand side, is composed of a set of model element types (coming from the target metamodels). To

each of them is attached a set of bindings which specifies how the properties of the target element are to be initialized. These declarative rules are named *matched rules*.

Imperative constructs in ATL can be specified in several places. An imperative block can be added to any declarative rule to conveniently initialize target elements requiring complex handling. Procedures, which are named *called rules* in contrast with the declarative *matched rules*, can be placed in the transformation and be called from any imperative block. Some procedures may bear the flags entrypoint or endpoint to specify that they should be executed either before or after the declarative rules are. The content of imperative blocks consists of sequences of instructions among: assignment, looping constructs, conditional constructs, etc. Complex algorithms can therefore be implemented imperatively if necessary.

A hybrid language is interesting because it can be used declaratively whenever possible. This means that some parts of a transformation and even full transformations, depending on their complexity, can be simply expressed. It is however possible to revert to a more classical all-purpose imperative language when the declarative constructs are not sufficient. However, we may then loose interesting properties that come with the use of declarative constructs. This is why it is planned to define several classes of ATL transformations, such as: declarative-only, imperative-only, hybrid, etc. Specific tools depending on the class of the transformation will use constraints to check whether a given model belongs to the class it supports. Thus, a transformation reverser that generates the opposite of a given transformation may only accept declarative transformations.

3.3 The Execution Virtual Machine

There are several practical solutions to implement ATL. We chose to define a Virtual Machine (VM) for different reasons. The main advantage we see in this approach is flexibility. As a matter of fact, AMMA is a research project and as such, ATL is constantly evolving to explore new advanced possibilities. A single low-level implementation makes it possible to work on high-level transformation language concepts while being rather independent of the actual tools used. For instance, the execution engine was first written to use the Netbeans/MDR model handler but it now can also work on Eclipse/EMF [7].The only part that had to be changed is the VM, since the ATL compiler and related tools run on top of it. Besides, despite the fact that our implementation has not been developed with performance in mind, the principal work to do to have a faster execution of ATL transformations is to write a new machine with less stringent flexibility requirements. It is of course still possible to develop a native code (or even Java bytecode to benefit from its portability) compiler later if necessary.

Among other interesting aspects, the use of a stack-based instruction set makes compiling OCL expressions quite simple. Moreover, other languages can also be compiled to our virtual machine. We use this to bootstrap several tools including the ATL compiler.

The ATL VM is a stack machine that uses a simple instruction set, which can be divided into three subsets. The first one is composed of instructions that perform model elements handling: creation, property access and assignment, operation call. The second one contains control instructions: goto, if, collection iteration instructions. There is also a set of stack handling instructions to push, pop and duplicate operands.

The primitive types are implemented by a native (meaning: part of the VM, actually in Java) library based on the OCL 2.0 standard library. All operations on primitive types are handled through operation calls to this library, e.g. 1+2 is performed as 1.+(2), the same way it is defined in the OCL specification.

While the choice of OCL as a navigation language for ATL has initially been made, other alternatives may be considered later without impacting the global architecture. Furthermore, the OCL part of ATL is being reworked to be pluggable. This means that it will be possible to reuse it in other languages for diverse purposes. One of our first experiments will be with a constraint-based language to express well-formedness rules on models. It will, for instance, be used to define the different classes of ATL transformation.

3.4 The ATL IDE

In order to ease the transformation writing process, we developed an Integrated Development Environment (IDE) for ATL on top of Eclipse [11]: ATL Development Tools (ADT) [1]. It provides several tools usually present in such environments. There is a syntax-highlighting editor synchronized with an outline presenting a view of the abstract syntax of the currently edited transformation program. We also developed wizards to create ATL projects for which a specific builder compiles ATL transformations.

A launch configuration is available to launch transformations in run or debug mode. In the latter, the execution can be debugged directly in Eclipse. The accompanying documentation tutorial can be used to show usage of all these features. Most of the ATL IDE components [1] behave the same way as their Java Development Tools (JDT) counterpart in Eclipse. Developers used to any modern IDE should not be lost when using ADT, which is illustrated in Figure 1.

3.5 Projectors

There are quite a lot of peripheral tools that are also useful to actually perform some model transformation work in relation with other technical spaces. We have grouped these tools under the name ATP (ATLAS Technical Projectors). Among these tools, we have identified a very important subset which we call injectors and extractors tools. As a matter of fact, there is a very large amount of pre-existing data that is not XMI [17] compliant but that would greatly benefit from model transformation. This data needs injection from its technical space (databases, flat files, EBNF, XML, etc.) to the MDE technical space. 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. The ATP goal is to host, in an organization as regular as possible, all drivers for external tool formats. It is an alternative to defining ad-hoc solutions for a lot of bridges with MDE-models usually named for example Model2Text, Text2Model, Model2EBNF, EBNF2Model, Model2SQL, SQL2Model, Model2XML, XML2Model, Mode2Binary, Binary2Model, etc.

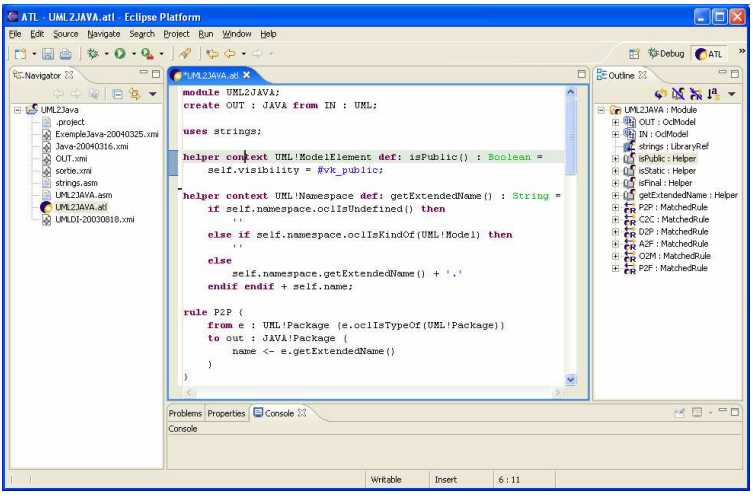

Fig. 1. A view of ATL Development Tools (ADT)

Besides, even when dealing with MDE-based tools, it may be convenient to use simple textual representations rather than always using a complex ad-hoc tool or meta-tool. We designed the Kernel Metametamodel (KM3) to this end. It is a simpletextual concrete syntax to represent metamodels. Although there are quite a lot of tools to draw UML diagrams and although some of them actually export valid metamodels in XMI, we came to the conclusion, after much experimentation, that an additional simple textual tool for metamodel representation is really useful.

4 AMW: The Atlas ModelWeaver

In order to provide a naive description of the ModelWeaver, let us suppose we have two metamodels *LeftMM* and *RightMM*. We often need to establish links between their related elements. There are many occasions when we need such functionality in a MDE platform as will be discussed later. Concerning the set of links the following issues have to be considered:

- The set of links cannot be automatically generated because it is often based on human decisions or 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.

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

Each link instance has to be typed conforming to a given *WMM*. There is no unique type of link. Link types should provide weaving tools with useful information. Even if some links contain only textual descriptions, these are valuable for tools supporting documentation, manual refinements or performing heuristics.

4.1 Motivating Examples

In software engineering practices, the "Y organization" sometimes called the 2TUP (Two Tracks Unified Process) has often been proposed as a methodological guide. The OMG has promoted this idea in the MDA proposal where a Platform Independent Model (PIM) should be weaved with a Platform Definition Model (PDM) to produce a merged Platform Specific Model (PSM).

Let us suppose we have a PIM for a bank containing the class *BankAccountNumber*. Suppose we have a PDM for an implementation platform containing classes *LongInteger* and *String*. One of the most important events in the software development chain is to take design decisions. One such design decision here would be for example to establish that the *BankAccountNumber* should be implemented using a *String* instead of a *LongInteger*. We will not discuss here the validity of this decision. However, we would like to ensure that this decision is well recorded, with the corresponding author, date, rationale, etc. Furthermore this decision is probably based on previous decisions and further decisions will be based on it.

What we see here is that a metamodel for design decisions would be most useful with several properties and links associated to each design decision. We can understand also that it would be very improbable to have an automatic weaving algorithm since this is most often a human decision based on practical know-how. Of course the user deciding of the weaving actions should be guided and helped by intelligent assistants that may propose her/him several choices. These helpers may be sometimes based on design patterns or more complex heuristics.

Let us take another example inspired by the work of Ph. Bernstein [2], [18]. We have two address books to merge and we get both metamodels *LeftMM* and *RightMM*. In *LeftMM* we have the class *Name* and in the second one the classes *FirstName* and *LastName*. Here we need to establish a more complex link stating that these are related by an expression of concatenation.

4.2 Extensible Metamodels

One may assume that there is no standard metamodel for weaving operations since most developers define their own. However, most often a given weaving metamodel will be expressed as an extension of another weaving metamodel that allows building a general weaving tool.

The ModelWeaver tool in AMMA reuses part of the infrastructure of the ATL IDE based on the Eclipse Platform [1]. We suppose there is a stub weaving metamodel and this is extended by specific metamodel extensions. The important goal is not to have to build a specific tool for each weaving task or use case. The two notions on which we are basing the design are metamodel extensions and Eclipse plugins.

The main idea of the implementation is that the GUI of the weaving tool is simple and may be partially generated. From the left part, one can select any class or associa-

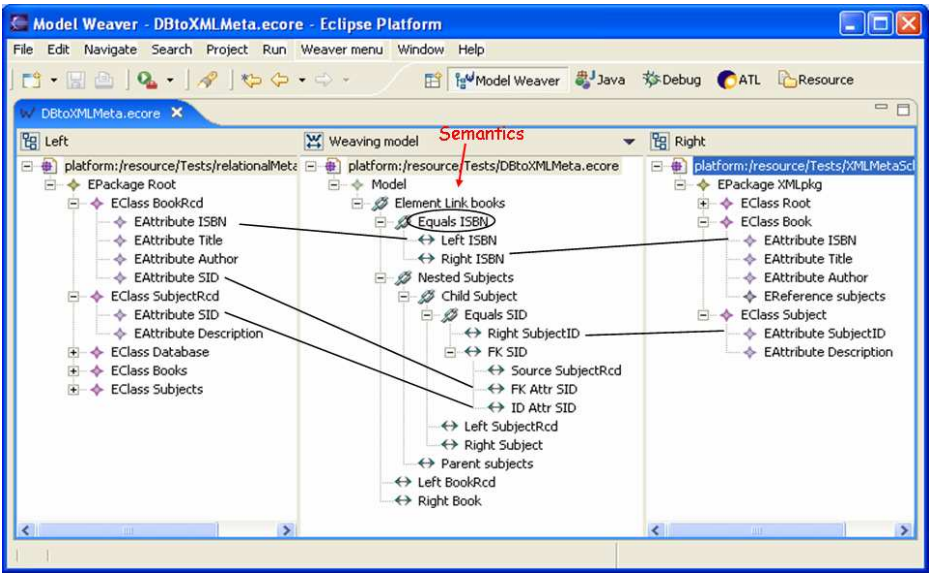

Fig. 2. First prototype for Atlas Model Weaver (AMW)

tion of the left metamodel and from the right part one can similarly select any class or association of the right metamodel. In the central part appear all the main elements of the weaving metamodel. Selecting a triple thus means creating a weaving link in the resulting weaving model.

Proceeding in this way, we get a generic weaving tool, adaptable to any kind of left, right and weaving metamodels. Of course, many design alternatives are being explored in the actual building of this tool. An initial prototype has been built and may give an idea of the user interface of AMW (see Figure 2).

As may be inferred from this prototype, a typical weaving session starts by uploading the weaving metamodel. From this metamodel, part of the tool GUI may be automatically generated. Then the left and the right metamodel may be chosen and the weaving work may proceed.

4.3 Weaving Rationale

One question often asked is why we need model weaving operations in addition to model transformations. This question raises at least the following issues:

- Issue of "arity": Usually a transformation takes one model as input and produces another model as output, even if extensions to multiple input and output may be considered. In contrast, a model weaving takes basically two models as input plus one weaving metamodel.
- Issue of "automaticity": A transformation is basically an automatic operation while a weaving may need the additional help of some heuristics or guidance to assist the user in performing the operation.

- Issue of "variability": A transformation is usually based on a fixed metamodel (the metamodel of the transformation language) while there is no canonical standard weaving metamodel.

Although one may argue that there may be several levels of abstraction in transformations (e.g. specifications and implementations of transformations), these three mentioned issues allow concluding that transformation and weaving are different problems. The first experiments with ATL and AMW confirmed this conclusion. In some particular cases however, a weaving model may be itself transformed into a transformation model.

Many research efforts like [22] are presently starting to investigate the relations between aspect-oriented programming and model engineering. This will be an important source of inspiration for weaving metamodels in the future.

One important open research issue that will be addressed later is how to integrate user guidance and domain dependent heuristics [11] in a model weaver. At this point of the research we have yet no hint on how to integrate this kind of knowledge as independent models. It is likely that those heuristics will have to be coded as Eclipse plugins in a first stage.

5 AM3: The Atlas MegaModel Management Tool

The Atlas MegaModel Management, AM3, is an environment for modeling in the large. With the macroscopic angle, models or metamodels are considered as a whole together with tools, services and other global entities.

Connected to an open platform, tools will exchange models. But tools may also be considered as models. A tool implements a number of services or operations. Each service or operation is also represented as a model. An operation may have input and output parameters, each being considered as a model. The interoperability platform may be organized as an intelligent model exchange system according to several principles and protocols such as the classical software bus or even more advanced architectures. To facilitate this exchange, the platform may use open standards such as XMI (XML model Interchange), CMI (CORBA Model Interchange), JMI (Java Model Interchange), etc.

Each time a given tool joins or leaves the platform, the associated megamodel is updated. There are also plenty of other events that may change the megamodel like the creation or suppression of a model or a metamodel, etc. Within one platform (local or global), the megamodel records all available resources. For each platform, we suppose there is an associated megamodel defining the metadata associated to this platform. For the sake of simplicity, we shall suppose that a local platform may be connected to another remote platform. This connection may be implemented by extension operations applied to the related megamodels.

5.1 Motivating Examples

To illustrate our purpose, we start by mentioning some situations, which one could find useful to get macroscopic information on models. By macroscopic, we mainly

mean relations that consider models as wholes and not their elements. Of course this is a point-of-view related consideration since we are talking about elements of megamodels.

A well-known global example of global link is the conformance relation between a model and its metamodel. This is often considered as an implicit link, but we suggest that this could also be explicitly captured in a megamodel with many advantages. One interesting property of this global conformance relation between a model and its metamodel is that it may be viewed as summarizing a set of relations between model elements and metamodel elements (a relation we often name the "meta" relation). One can clearly see here the coexistence between global model level relations and local element based relations. In some cases, one is not interested in the local element level relations because the global relation provides sufficient reliable information on what is actually needed.

Another example is related to transformations. Recall that in our MDE landscape, a transformation is a model, conforming to a given metamodel. So, if a model Mb has been created from a model Ma by using transformation Mt, then we can keep this global information in the megamodel. Supposing the transformation model Mt has a link to its reverse transformation Mt^{-1}, the memorized information can be used for reverse engineering (from a modified model Mb) or for consistency checks. Being stored in a repository, a given transformation Mt will have no meaning if the three links are not provided to the source and target metamodels and the transformation metamodel itself.

There is a whole set of information that could be regarded as global metadata. For example, we could associate to a model the information of who has created it, when, why, what for, who has updated it and the history of updates, etc. To a metamodel we can associate its goal, the place where we can find its unique original definition, alternate definitions, its authors, its history, its previous and next versions, etc. A very naive implementation idea, for example, would be to imagine using a CVS for different versions of a metamodel. The notion of megamodel goes much beyond this kind of facilities, with a very low implementation cost and a much more regular organization.

The idea of model driven architecture has sometimes been presented with megamodels as described in [4] from where the diagram of **Fig. 3** is extracted. What is conveyed there is that a PSM is a model, in relation with a given PIM and with a given PDM (Platform Description Model). From a PIM, it should be possible to know which PSMs are available and to what platform they correspond. In simple cases, the

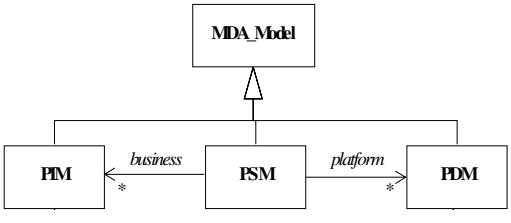

Fig. 3. Extended MDA classification

description of the process that produced the PSM from the PIM and the PDM could also be explicitly defined. Although somewhat idealized, these illustrations show how megamodels may be used beyond mere documentation of architectural and process approaches. The megamodel captures the idea of a "MDA component" as originally presented in [4], but allows going much beyond this proposal. In its present state it also allows to take into account solutions like the RAS OMG specification (Reusable Asset Specification).

5.2 Use Cases for the Megamodel Manager

Let us consider a use case with two MDE platforms installed in specific organizations, one in Nantes and one in Oslo, for example. Within each organization, there are different tools for capturing (Rational XDE, Poseidon, etc.) and storing/retrieving models and metamodels. Some tools may be common. Each platform has been initialized and whenever a tool is plugged or unplugged, the local megamodel is updated. Also, each tool has the possibility/responsibility to update the local megamodel upon achievement of specific operations. Examples of these operations may be:

- a user has created a model with a modeling tool such as Poseidon or Rational XDE,
- a user has created a metamodel in KM3 format with a textual editor,
- a user has modified a metamodel,
- a user has created a transformation in ATL,
- a user has created a new model by running an ATL transformation,
- etc.

Let us suppose a user in Oslo wishes to generate a list of contacts in a given database from a set of 5000 contacts contained in his/her local Microsoft Outlook system. The first action would be to look on the local Oslo platform if there is a metamodel for Microsoft Outlook available. If the metamodel is available in Oslo, the user will look for a corresponding injector able to act as a driver to transform Outlook into MDE formats. If these correspond exactly to his/her needs, then they will be used as is. Otherwise they will be adapted. After that, the user will do the same for the target database (metamodel and extractor). Finally the local platform will be queried for a suitable transformation. If none corresponds, one will be built or adapted with the help of suitable browsing/editing tools. In case of local unavailability, platforms that have links with the Oslo platform will be queried, for example the Nantes platform. The actions will be applied on this new platform. Alternatively, the user may consider his/her platform to virtually correspond to all components available on the Oslo or Nantes platform by a simple extension operation.

The problem of modeling in the large is related to several issues like typing [21], packaging, tool integration and interoperability, etc. Uniform access to local and remote resources may be facilitated by a megamodel-based approach as presented here. A resource may be a model, a metamodel, a transformation, a process, a service, a tool, etc. The first implementation of AMMA mainly deals with local platforms, but several extensions to distributed environments are already considered. One particular goal is to consider that professional modelers will exchange metamodels, transformations, etc.

like other exchange music or video. As a consequence, peer-to-peer architectures will be studied as an important alternative way to implement the MDE platform.

6 Conclusion

In this paper, we have presented the main tools that are being progressively integrated in the current AMMA prototype. Even if the development status of these tools is different, they share the common principle of models as first class entities and they have been designed to collaborate in a complementary way.

Our work on the design implementation and first use of the AMMA platform has led us to consider two kinds of activities in MDE: modeling in the large and modeling in the small. The corresponding operations have been illustrated by four tools at different levels of maturity: ATL, AMW, ATP and AM3. Model transformation has been recognized as an essential operation in MDE, but a lot of work yet remains to establish the exact application domain for QVT-like tools. Model weaving is presently in search of recognition, but when one considers how active this subject has been in the past in knowledge, data, and software engineering, it is very likely that applying model engineering principles to this field will bring important results in the future. As for megamodel management, it is probably the most recent question raised in the MDE field. Here again, there are a lot of examples of successful usages of this global approach in related domains and it is very likely that global registries will soon be considered essential for dealing with the management of an increasing number of global entities.

What we have tried to achieve with the AMMA platform is a balanced integration of these complementary aspects. The idea of considering models as first class entities has been the key principle to reach this goal. The AMMA experimental implementation of an MDE platform has allowed a first level of validation in the separation between the activities of modeling in the small and modeling in the large. There remains, however, a considerable amount of research effort yet to be done before these ideas translate into mainstream engineering platforms.

Acknowledgements

We would like to thank F. Allilaire, M. Didonet del Fabro, T. Idrissi, D. Lopes and G. Sunye for their contributions to the AMMA project.

References

[1] Allilaire, F., Idrissi, T. ADT: Eclipse Development Tools for ATL. EWMDA-2, September 2004, Kent, http://www.cs.kent.ac.uk/projects/kmf/mdaworkshop/

[2] 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

[3] 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
[4] 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
[5] Bézivin, J., Jouault, F., Valduriez, P.: On the Need for Megamodels. OOPSLA & GPCE, Workshop on best MDSD practices, Vancouver, Canada, 2004
[6] Booch, G., Brown, A.W., Iyengar, S., Rumbaugh, J., Selic, B.: An MDA Manifesto. Business Process Trends/MDA Journal, May 2004.
[7] Budinsky, F., Steinnberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework, EMF, The Eclipse series, ISBN 0-13-142542-0, 2004
[8] Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools and Applications. Addison-Wesley, Reading, MA, USA, June 2000
[9] Deremer, F., Kron, H.: Programming in the Large versus Programming in the Small. IEEE Trans. On Software Eng. June 1976, http://portal.acm.org/citation. cfm?id=390016.808431
[10] Greenfield, J., Short, K., Cook, S., Kent, S.: Software Factories, Wiley, ISBN 0-471-20284-3, 2004
[11] Heuvel, W.J.: Matching and Adaptation: Core Techniques for MDA-(ADM)-driven Integration of new Business. Applications with Wrapped Legacy Systems. MELS, 2004
[12] Klint, P., Lämmel, R., Kort, J., Klusener, S., Verhoef, C., Verhoeven, E.J.: Engineering of Grammarware. http://www.cs.vu.nl/grammarware/
[13] 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/
[14] Lemesle, R.: Transformation Rules Based on Metamodeling. EDOC,'98, La Jolla, California, 3-5, pp.113-122, November 1998
[15] OMG/MOF: Meta Object Facility (MOF) Specification. OMG Document AD/97-08-14, September 1997. http://www.omg.org
[16] OMG/RFP/QVT: MOF 2.0 Query/Views/Transformations RFP. OMG document ad/2002-04-10. http://www.omg.org
[17] OMG/XMI: XML Model Interchange (XMI) OMG Document AD/98-10-05, October 1998. http://www.omg.org
[18] Pottinger, R.A., Bernstein, P.A.: Merging models Based on Given Correspondences, Proc. 29th VLDB Conference, Berlin, Germany, 2003
[19] Schmidt, D.: Model driven Middelware for Component-based Distributed Systems. Invited talk, EDOC'2004, Monterey, Ca., September 2004
[20] Soley, R.: OMG staff Model-Driven Architecture. OMG document. November 2000. http://www.omg.org
[21] Willink, E.D. OMELET: Exploiting Meta-Models as Type Systems. EWMDA-2, Canterbury, England, September 2004
[22] Wu, H., Gray, J., Roychoudhury, S., Melnik, M. Weaving a Debugging Aspect into Domain-Specific Language Grammars. ACM, 2004