LABORATOIRE D'INFORMATIQUE DE NANTES-ATLANTIQUE

An introduction to the ATLAS Model Management Architecture

Jean Bézivin [1], Frédéric Jouault[1 2], David Touzet[1]

[1] Laboratoire d'Informatique de Nantes-Atlantique
2, rue de la Houssinière - B.P. 92208
44322 Nantes Cedex 3, France

[2] TNI-Valiosys
120, rue René Descartes
Technopôle Brest Iroise - BP 70801
29608 Brest Cedex, France

— ATLAS GDD —

RESEARCH REPORT

N° 05.01

February 2005

Jean Bézivin, Frédéric Jouault, David Touzet

An introduction to the ATLAS Model Management Architecture

24 p.

Les rapports de recherche du Laboratoire d'Informatique de Nantes-Atlantique sont disponibles aux formats PostScript® et PDF® à l'URL :
http://www.sciences.univ-nantes.fr/lina/Vie/RR/rapports.html

Research reports from the Laboratoire d'Informatique de Nantes-Atlantique are available in PostScript® and PDF® formats at the URL:
http://www.sciences.univ-nantes.fr/lina/Vie/RR/rapports.html

An introduction to the ATLAS Model Management Architecture

Jean Bézivin, Frédéric Jouault, David Touzet

jean.bezivin@univ-nantes.fr, frederic.jouault@univ-nantes.fr,
david.touzet@univ-nantes.fr

Abstract

The concept of “model” is today considered as a promising technology in domains such as data and software engineering. In the field of model engineering, models are now viewed as firts-class entities. This new approach makes it possible to envision the integration of models into engineering processes. Such an integration will however require a set of dedicated tools enabling to perform standard model operations onto handled models. We strongly believe that, in order to achieve usability for large communinities of users, model-based tools have to rely on a common and well-defined theorical modeling framework.
This report addresses both theorical and implementation issues. We propose the idea that a common set of principles may be mapped to different implementation contexts. We illustrate our approach with AMMA, our current proposal for a model-based conceptual architecture.

General Terms: Design, Theory, Standardization

Additional Key Words and Phrases: model, model engineering, mda, omg, definition

1 Introduction

When we consider most areas of computer science, we may observe that there has been an initial period of development, based on ad-hoc principles, followed by a rapid rise in abstraction allowing radical improvement of the practices in the field. The move from the first to the second period was usually triggered by the proposal of a new key abstraction and a corresponding set of principles. In the database area for example, the pioneering work of E. F. Codd is widely recognized as one of the great technical innovations of the 20th century. The relational database organization provided a theoretical framework within which a variety of important problems could be attacked in a scientific manner. This new abstraction was described in an IBM technical report of 1969 and became mainstream in the seventies. Similarly in software engineering, K. Nygaard and O. Dahl proposed in 1965 to unify many notions including data and procedures into the concept of an object, and the related technology was widely transferred to industry in the eighties. Object technology will also probably be seen as another great invention of the 20th century. What seems now to be happening is a similar movement based on the concept of "model". Model engineering will change the future practices in domains like data engineering and software engineering if we are able to prove that it is based on sound principles and that it is amenable to implementation and wide usage. This dual test of feasibility (conceptual and operational) was brightly passed by relational technology in data engineering and by object technology in software engineering. This paper proposes some initial thinking about the ability of model engineering to achieve similar success. We have then two questions to answer: 1) what are the foundations and basic principles of this new proposal and 2) how could these principles be mapped onto operational systems so that large user communities may start using the technology.

Model engineering applies to several domain areas like distributed real-time embedded systems, as proved by the growing scientific literature in this application field. This report however does not target any specific area, but is intended to show the wide spectrum of applicability of the proposed ideas. Partial proof of feasibility of these ideas is being demonstrated by the Atlas research group[1] contribution to the Eclipse/GMT project[2].

In the second section of this report, we present a possible foundation for model engineering. We then proceed to demonstrate how some of these principles may be mapped onto an operational system. AMMA (ATLAS Model Management Architecture), which is our current proposal for such a conceptual architecture, will be described in the third section. We choose, as a target platform, one of the currently most popular programming frameworks, the Eclipse system[3], extended by some basic facilities for handling models (EMF) [3]. The mapping from basic principles to this implementation object-based platform is captured by what we call a conceptual model engineering architecture. We show, in the fourth section, how AMMA may be mapped onto the Eclipse/EMF framework. We discuss advantages, limitations and possible extensions of our approach in the conclusion.

[1] ATL home page: http://www.sciences.univ-nantes.fr/lina/atl/
[2] Eclipse/GMT home page: http://eclipse.org/gmt/
[3] Eclipse home page: http://www.eclipse.org/

2 Definitions and concerns

The aim of this section is to introduce the basic principles model engineering is based on. To this end, we first provide a description of the different artifacts composing the model-driven architecture, and the way they are related. We then identify some of the issues model-based platforms will have to address.

2.1 Principles and definitions

Models are now commonly used to provide representation of real-world situations. In the scope of a modeling architecture, the term system is used to refer to a real-world situation. A model is then said to *represent* a system. Figure 1 provides an example of a relational model that defines a possible representation for a set of books in a library. In this example, the system is associated with the represented books while the model corresponds to the Book table structure. This structure is defined in terms of tables, columns, and columns types, which correspond to our relational *model elements*. On the right side of Figure 1, 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.

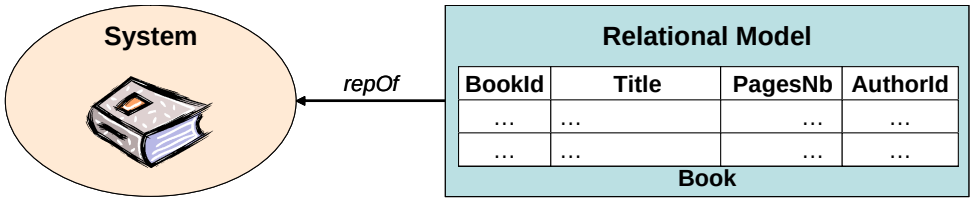

Figure 1: The “representation” relation between a system and a model

Each model is defined in conformance to a metamodel. Metamodels define languages enabling to express models. This means that 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 (cf. Figure 2). Representation and conformance relations are central to model engineering [4].

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 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. Figure 3 explicits 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 relational metamodel provides a specific way to define data-centered models. However, many other metamodels may be similarly specified (UML metamodel, Petri nets metamodel,

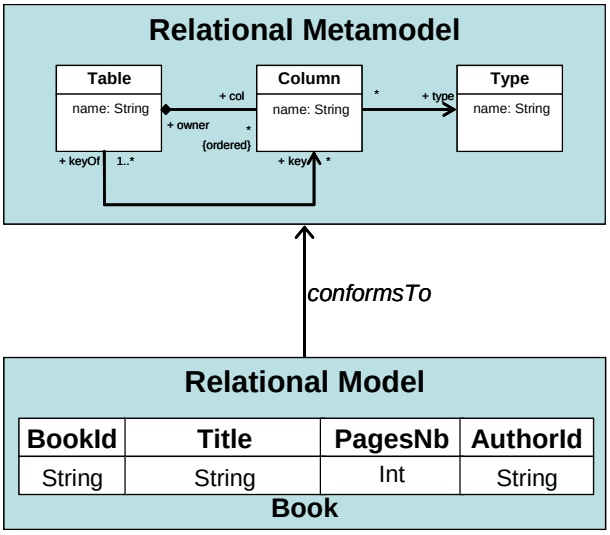

Figure 2: The “conformance” relation between a model and its metamodel

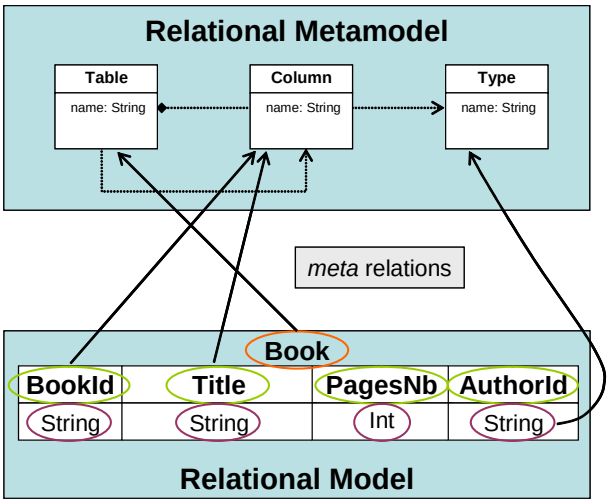

Figure 3: The “meta” relation between model and metamodel elements

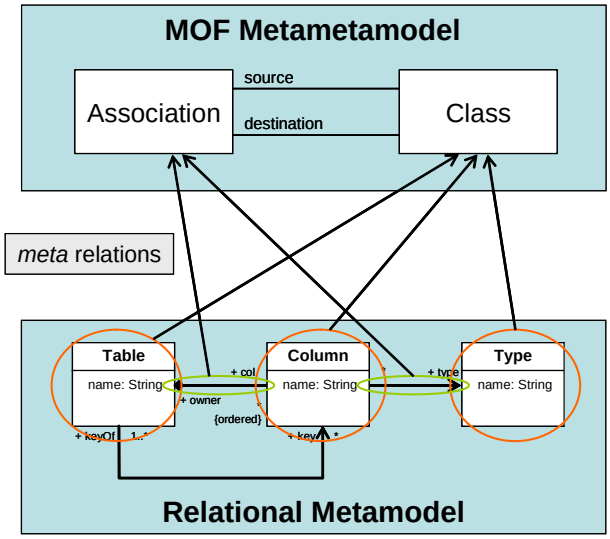

Figure 4: “meta” relations between M2 and M3 levels

etc.), providing thus different complementary ways to define models. The growing number of metamodels has emphasized the need for an integration framework for all available metamodels. The answer was to provide 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 [13]. The relational metamodel conforms to the MOF metametamodel.

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 explicited in Figure 4. 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.

The MOF metametamodel is self-defined. In other words, it is defined in conformance with its own language. It is said to conform to itself. This implies that MOF elements have their

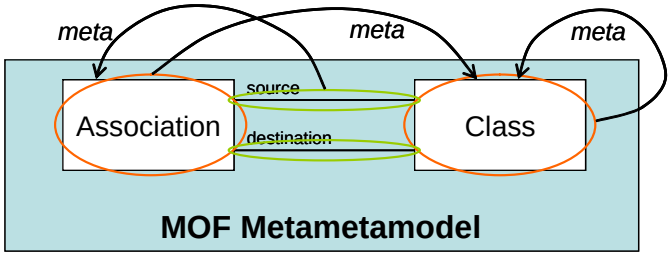

Figure 5: Self definition of the M3 level

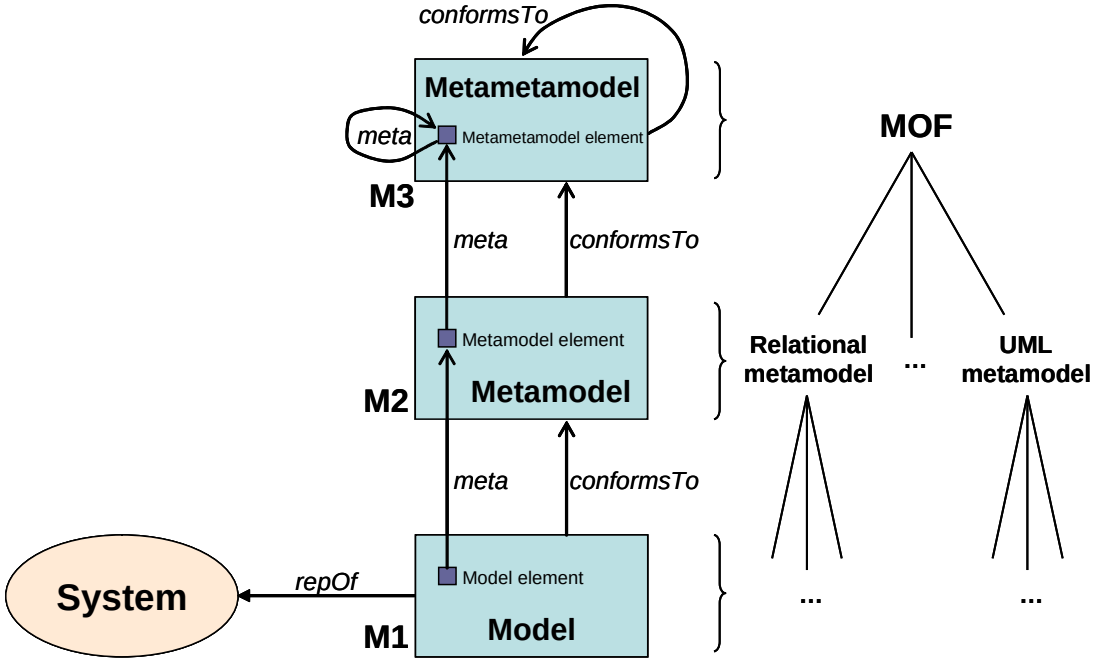

Figure 6: The 3 layers of the model-driven architecture

metaelement defined in the MOF itself. Therefore, a MOF element can be its own metaelement. This situation is illustrated in Figure 5: the Association and the Class elements are both typed by the MOF Class element (i.e. the Class element is its own metaelement), whereas link elements are typed by the MOF Association element.

We have now defined the three different layers composing the model-driven architecture. Figure 6 provides an overview of this architecture. The highest layer is the one of the metametamodel (it is called M3 for the three M's of metametamodel). In the scope of the OMG architecture [19], the metametamodel is the MOF. However, other metametamodels are available in different contexts. For instance, the EBNF provides a convenient metametamodel when dealing with programming languages. The following layer (M2) is the one of metamodels. At this level, we find the different metamodels (relational, UML, etc.) that have been defined in conformance with the metametamodel. The last layer (M1) is the one of models. Each model conforms to a given metamodel defined at layer M2[4].

In the model engineering area, artifacts of the model-driven architecture are considered as first class entities. Suitable tools are hence required in order to handle models and metamodels conveniently. We identify, in the next section, a number of features that should be made available by operational model-based systems.

[4]Some also consider a fourth layer, called M0, for terminal instances but this is not our reading of the OMG architecture [19]. We interpret M0 as being the real world and M1-3 to be the modeled world.

2.2 Issues in handling models

In the field of model engineering, users have to deal with entities such as models, metamodels, model elements, etc. They may be particularly interested in being able to handle models and metamodels as basic elements by means of dedicated loading/saving (from/to persistent storages) and editing facilities. For instance, a developer may be interested in programmatically creating a new model from scratch by adding model elements to it. In the same way, modification and deletion operations should also be available on models, thus enabling a user to modify an existing relational model by removing a column from a table definition.

Working with models also supposes underlying systems to be able to manage a number of metadata. Beside update logs and authoring information (that may be associated with handled models), the metadata have to describe existing relationships between artifacts of the model-driven architecture. As an example, the conformance relation between a model (M1) and its metamodel (M2) can be explicited in the scope of these metadata. Relations between models may also be expressed this way. Further discussion on the need for metadata management may be found in [7].

Co-existing metamodels now make it possible to define different models that represent a same system. For instance, the library system we represented by a relational model may also be represented by means of a UML model. The growing number of available metamodels highlights the need for tools enabling to establish semantic relations between model elements, and to transform a model conforming to a metamodel A into a model conforming to a metamodel B.

Finally, an ideal model engineering platform is also supposed to deal with data types. Models can indeed be specified by means of a number of domain-specific formats (database definition, Java classes, etc.). These domain-specific formats are associated with various Technical Spaces (TS) [11]. Bridges between models conforming to different metamodels should be provided by transformation facilities. In a similar way, an operational platform has to embed tools providing gateways between the model engineering and other TSs.

In this section, we have introduced the basic principles of model engineering and identified a set of desirable features for model-based operational environments. The next section is devoted to the description of AMMA, our model engineering prototype.

3 AMMA: a model engineering platform

The basic principle of a model engineering platform is to consider models as first class entities. Using a model-based platform should allow significant benefits. For instance, 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.

In this section, we view AMMA as a model engineering platform, i.e. a framework offering basic facilities to manipulate models, and onto which a variety of different tools (industrial or research tools, legacy or advanced tools, etc.) may be plugged. In the next section we'll take the slightly different view of considering rather AMMA as a conceptual architecture mapped onto Eclipse/EMF, i.e. a higher abstract interpretation of Eclipse/EMF basic facilities.

AMMA, our model engineering platform prototype can be seen as a software bus adapted to the basic model engineering principles. It may have both local and distributed implementations. A model engineering platform is primarily intended for tool integration. Several tools are usually available on such a platform. AMMA is based on four basic blocks (presented in Figure 7) 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.).

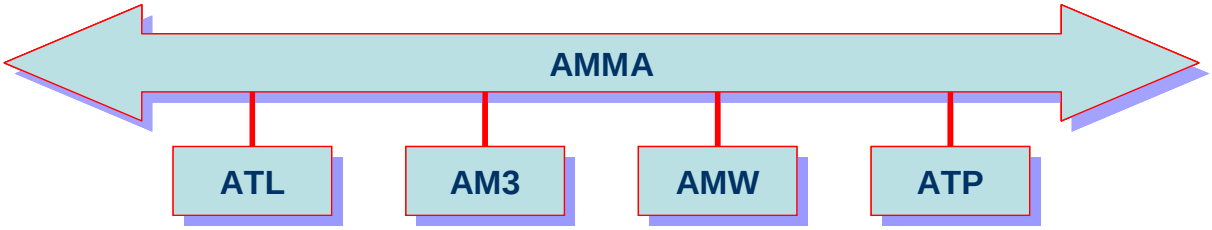

Figure 7: The AMMA platform

3.1 ATL: The Atlas Transformation Language

A model transformation language is used to define how a set of source models is visited to create a set of target models [15]. 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. Figure 8 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 conforms to the ATL transformation language). Following subsections present the ATL language, its execution engine, and the ATL development environment provided within the AMMA platform.

3.1.1 The ATL language

The ATL language is a hybrid of declarative and imperative constructs. The expression language is based on OCL 2.0 [14]. 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, which filters even more the set of matched

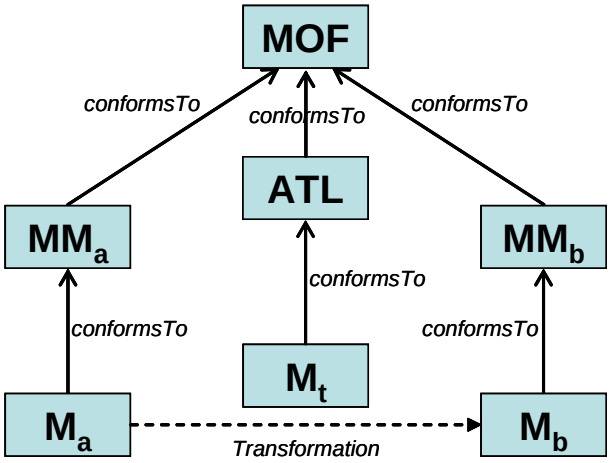

Figure 8: An ATL transformation

elements (e.g. all classes with a name beginning by 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. 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 assignments, looping constructs, conditional constructs, etc. By this mean, complex algorithms can be implemented imperatively.

3.1.2 The ATL engine

The ATL engine has been implemented as a Virtual Machine (VM). The main advantage we see in this approach is flexibility. As a matter of fact, AMMA is a research platform and, as such, ATL is constantly evolving. A single low-level implementation makes it possible to work on high-level transformation language concepts while being rather independent of the tools being used. For instance, the execution engine was first written to use the Netbeans/MDR model handler [12], but it now also run on top of Eclipse/EMF [3]. The only parts that had to be changed are the VM, the ATL compiler, and the related tools being executed on it. Among other interesting aspects, the use of a stack-based instruction set makes compiling OCL expressions quite simple.

The ATL VM is a stack machine that uses a simple instruction set, which can be divided in three subsets. The first one is composed of instructions that perform model elements handling: creation, access and assignment of properties, and operations call. The second one contains

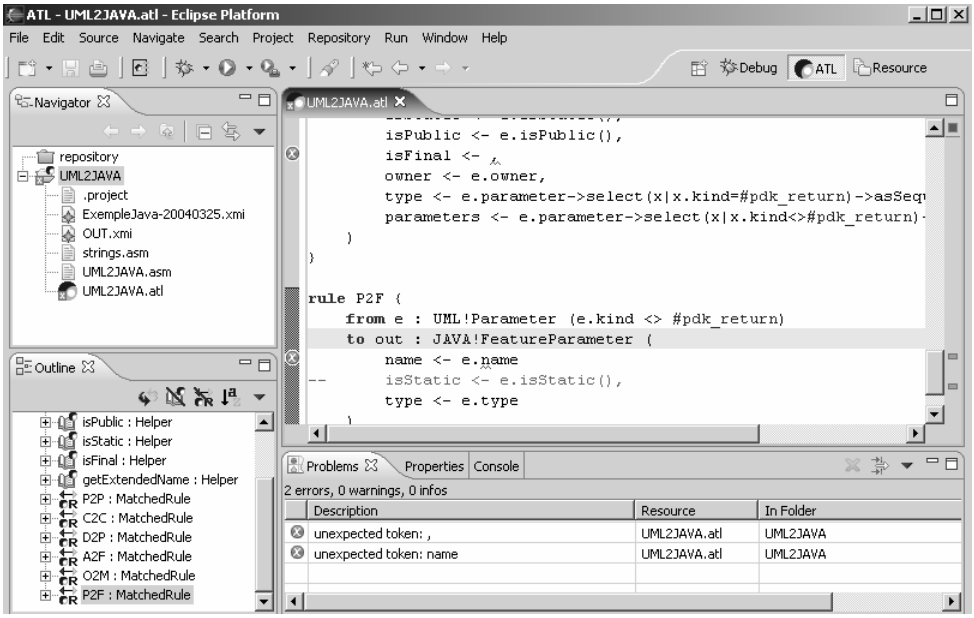

Figure 9: The ATL IDE

control instructions: goto, if, collection iteration instructions, etc. There is also a set of stack handling instructions to push, pop, and duplicate operands. The primitive types are implemented by a native library based on the OCL standard library. All operations on primitive types are handled in the same way they are defined in the OCL specification, that is through operation calls of this library.

3.1.3 ATL Tools

An Integrated Development Environment (IDE) has been developed on top of Eclipse [10] in order to ease the transformation writing process (cf. Figure 9). ATL Development Tools (ADT) provide several tools usually present in such an environment [1]. There is a syntax-highlighting editor synchronized with an outline presenting a view of the abstract syntax of the currently edited transformation program. Wizards have been developed 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 features a simple tutorial that can be used to show all these features. Most of the ATL IDE components [1] behave the same way as their Java Development Tools (JDT) counterpart in Eclipse.

3.2 AMW: The Atlas ModelWeaver

With the growing number of available metamodels, being able to establish relations between model elements has become a key feature for modeling platforms. The Atlas ModelWeaver aims

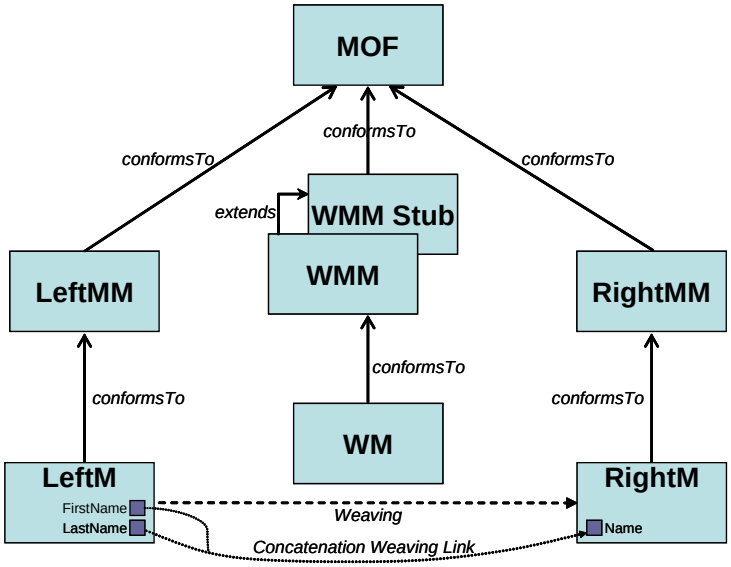

Figure 10: The model weaving scheme

to assist designers in the definition of semantic links between elements of different models (or metamodels).

3.2.1 Model Weaving

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. Let us consider an example inspired by the work of Bernstein [2, 17]. We have two address books to merge and we get both *LeftM* and *RightM* models. In *RightM*, we only have the class *Name* whereas, in the second one, we have the classes *FirstName* and *LastName*. We need here to establish a complex link, stating that these are related by an expression of concatenation.

Concerning the set of links to be generated, the following issues have to 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.

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. Figure 10 describes a simple

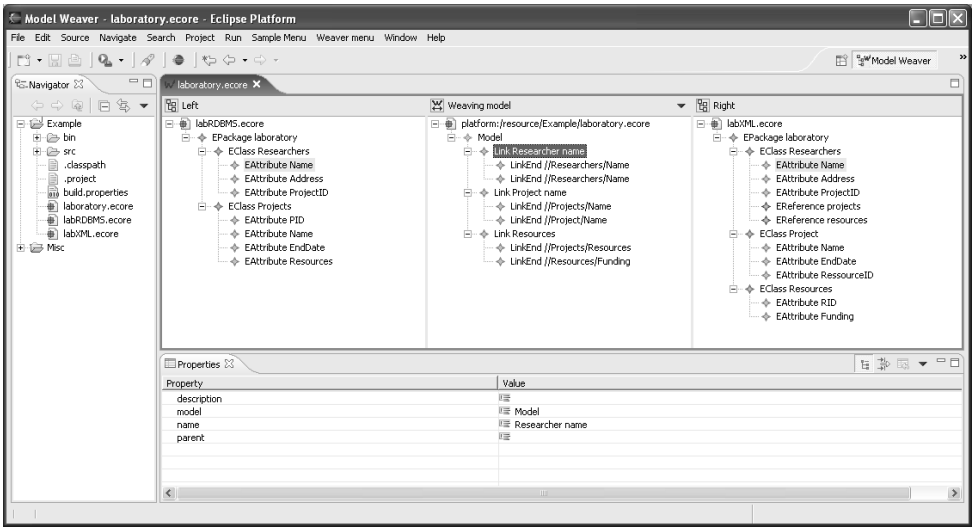

Figure 11: The AMW IDE

model weaving scheme in which an explicited weaving link (of type *Concatenation*) associates two source elements (*FirstName* and *LastName*) with an only target element (*Name*).

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 that 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, the Atlas ModelWeaver, able to generically deal with weaving tasks.

3.2.2 AMW tools

The ModelWeaver tool in AMMA reuses part of the infrastructure of the ATL IDE, itself based on the Eclipse Platform [1]. It aims to provide generic facilities for model weaving tasks. For this purpose, the ModelWeaver design is based on the notions of metamodel extensions and Eclipse plugins.

The implementation of the ModelWeaver was governed by the idea that the GUI of the model weaving tool should be simple, but also partially generetable. To this end, we designed a three parts interface (see Figure 11). From the left part, one can select any class or association 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 [6] and may give an idea of the user interface of AMW. 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.

3.3 AM3: The Atlas MegaModel 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” [5]. The megamodel can be viewed as a model which elements represent and refer to models and metamodels. 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.

To illustrate our purpose, we can consider an example related to transformations. An ATL transformation is a model, conforming to a given metamodel. So, if a model M_b has been created from a model M_a by using transformation M_t, then we can keep this global information in the megamodel. Supposing the transformation model M_t has a link to its reverse transformation M_t^{-1}, the memorized information can be used for reverse engineering (from a modified model M_b) or for consistency checks. Being stored in a repository, a given transformation M_t will have no meaning if the three links are not provided to the source and target metamodels and to the transformation metamodel itself.

Another well-known 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 here 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 (previously named ”meta” in this report). 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.

Moreover, 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 and when and why and may be what for, etc., who has updated it and the history of updates, etc.

3.4 ATP: The Atlas Technical Projectors

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 (XML Model Interchange is the OMG open standard for model exchanges [16]) 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.

Besides, even when dealing with model engineering 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 simple proprietary textual concrete syntax to type metamodels in. Although there are quite a lot of tools to draw UML diagrams, and although some of them actually export valid metamodels in XMI, simple additional textual tools could be useful if they are sufficiently generic. A gateway between XMI and KM3 representations has been built by developing injectors and extractors that enable models translations within the model engineering TS.

4 Implementing AMMA on top of EMF

The AMMA platform has been implemented on top of the Eclipse/EMF framework. We describe, in this section, the way AMMA tools have been mapped onto EMF facilities.

4.1 Eclipse and EMF

Eclipse is an open-source generic framework for developing applications. It can be used as a development tool, can be extended, some parts of it can be integrated into standalone applications, etc. The flexibility of the framework, the source code availability, and the integration level already achieved by some tools, such as JDT and CDT for Java and C/C++ development, make Eclipse very attractive.

The Eclipse Modeling Framework (EMF) is a model engineering extension for Eclipse [3]. It enriches Eclipse with model handling capabilities. To this end, it implements the basic principles described in the second section of this report in the form of a model handling API. EMF provides support for operations such as XMI serialization and loading, creation and deletion of model elements, property assignment, and navigation. Based on these facilities, model engineering platforms can be built on top of EMF, either as Eclipse features, or as separate tools that only use the EMF API. However, EMF is more than just a single model engineering block: it is a model engineering platform. It provides advanced features such as customizable Java code generation for programmatic or editor-based model handling, XML Schema injection and extraction, etc. EMF can be seen as a convenient platform for Java developers who want to benefit from model engineering facilities while still using a generic programming language.

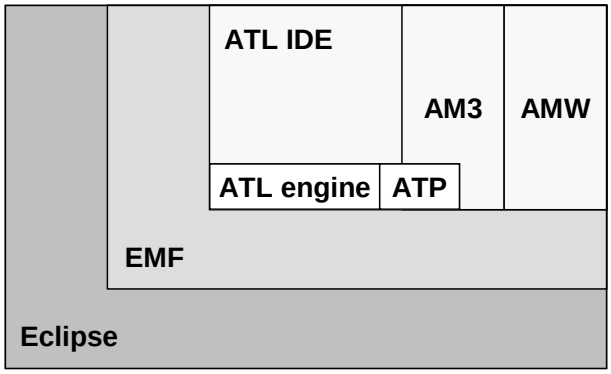

Figure 12: Architecture of the AMMA platform

4.2 AMMA on top of EMF

The AMMA platform is aimed at more model-centric activities, following the line "models as first-class entities". For instance, whereas code generation is a main component of EMF, we consider it as a specific use of projectors. The architecture of the current EMF-based AMMA implementation is described in Figure 12. 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.

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 Conclusion

The main 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. 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. The most publicized is the OMG MDA, but we may also mention generative programming [8], model integrated computing [18], software factories [9], and many others. Our attitude has been to find the set of basic principles common to all the dominant model engineering approaches and to make them explicit. Then we are in a position to clearly separate the principles, standards, and tools levels.

Usually, MDA advocates the separation from platform-dependent and platform-independent aspects. It would be a pity if MDA tools did not apply these principles to themselves. We have already experimented with porting the ATL engine from MDR/NetBeans to Eclipse/EMF. It would be nice if the cost of moving from an execution environment to the next one could be made low by using model engineering techniques. Having an explicit definition of the principles and a separate mapping of these principles onto executable platforms helps keeping the moving cost within reasonable bounds.

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.

Finally, what also appears in this presentation is the high complementarity 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.

Acknowledgements

We thank all members of the ATLAS team that are contributing to this work, and particularly Patrick Valduriez, Freddy Allilaire, Marcos Didonet del Fabro. We also thank all students that have been involved in the implementation of the various blocks of the AMMA platform. The design of the ATL tool has been done in collaboration with the TNI-Software company. The design of the AMW tool has be done in cooperation with the Sodifrance group and TNI-Software. Some of the ideas presented in this report have been developed within the context of the Modelware project[5].

References

[1] F. Allilaire and T. Idrissi. ADT: Eclipse Develoment Tools for ATL. In *Proceedings of the Second European Workshop on Model Driven Architecture (EWMDA-2)*, Canterbury, UK, September 2004.

[5] Modelware (IST European project 511731) home page: http://www.modelware-ist.org/

[2] P. A. Bernstein, A. Y. Levy, and R. A. Pottinger. A Vision for Management of Complex Systems. Technical report MSR-TR-2000-53, Microsoft Research, Redmond, USA, 2000.

[3] F. Budinsky, D. Steinnberg, E. Merks, R. Ellersick, and T. J. Grose. *Eclipse Modeling Framework*. Addison Wesley, 2004.

[4] J. Bézivin. In search of Basic Principles of Model Engineering. *Novatica/Upgrade*, 5(2):21–24, April 2004.

[5] J. Bézivin, S. Gérard, P. A. Muller, and L. Rioux. MDA Components: Challenges and Opportunities. In *Metamodelling for MDA, First International Workshop*, York, UK, November 2003.

[6] J. Bézivin, F. Jouault, and P. Valduriez. First Experiments with a ModelWeaver. In *Proceedings of OOPSLA & GPCE Workshop*, October 2004.

[7] J. Bézivin, F. Jouault, and P. Valduriez. On the need for Megamodels. In *Proceedings of OOPSLA & GPCE, Workshop on best MDSD practices*, Vancouver, Canada, 2004.

[8] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools and Applications*. Addison Wesley, June 2000.

[9] J. Greenfield, K. Short, S. Cook, and Kent S.. *Software Factories*. Wiley, 2004.

[10] W. J. Heuvel. Matching and Adapation: Core Techniques for MDA-(ADM)-driven Integration of new Buisness. In *Proceedings of Model-Driven Evolution of Legacy Systems (MELS'04)*, Monterey, Canada, September 2004.

[11] I. Kurtev, J. Bézivin, and M. Aksit. Technical Spaces: an Initial Appraisal. In *Proceedings of CoopIS, DOA'2002 Federetad Conferences, Industrial track*, 2002.

[12] M. Matula. NetBeans Metadata Repository. http://mdr.netbeans.org/, March 2003.

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

[14] OMG/OCL. UML 2.0 OCL Specification, OMG Document PTC/2003-10-14. http://www.omg.org/, October 2003.

[15] OMG/RFP/QVT. MOF 2.0 Query/Views/Transformations RFP, OMG Document AD/2002-04-10. http://www.omg.org/, 2002.

[16] OMG/XMI. XML Model Interchange (XMI), OMG Document AD/1998-10-05. http://www.omg.org/, October 1998.

[17] R. A. Pottinger and P. A. Bernstein. Merging Models Based on Given Correspondences. In *Proceedings of the 29th VLDB Conference*, Berlin, Germany, 2003.

[18] D. Schmidt. Model driven Middleware for Component-based Distributed Systems. In *Proceedings of EDOC'2004, Invited talk*, Monterey, Canada, September 2004.

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

LABORATOIRE D'INFORMATIQUE DE NANTES-ATLANTIQUE

An introduction to the ATLAS Model Management Architecture

Jean Bézivin, Frédéric Jouault, David Touzet

Abstract

The concept of "model" is today considered as a promising technology in domains such as data and software engineering. In the field of model engineering, models are now viewed as firts-class entities. This new approach makes it possible to envision the integration of models into engineering processes. Such an integration will however require a set of dedicated tools enabling to perform standard model operations onto handled models. We strongly believe that, in order to achieve usability for large communinities of users, model-based tools have to rely on a common and well-defined theorical modeling framework.
This report addresses both theorical and implementation issues. We propose the idea that a common set of principles may be mapped to different implementation contexts. We illustrate our approach with AMMA, our current proposal for a model-based conceptual architecture.

General Terms: Design, Theory, Standardization

Additional Key Words and Phrases: model, model engineering, mda, omg, definition

LINA, Université de Nantes
2, rue de la Houssinière
B.P. 92208 — F-44322 NANTES CEDEX 3