# Model-Based DSL Frameworks

Jean Bézivin, Frédéric Jouault, Ivan Kurtev, Patrick Valduriez
ATLAS (INRIA & LINA)
University of Nantes
2, rue de la Houssinière BP 92208
44322, Nantes, France
{jean.bezivin | frederic.jouault | ivan.kurtev}@univ-nantes.fr; patrick.valduriez@inria.fr

PRELIMINARY VERSION

## Keywords
Model-Driven Engineering, MDA, DSL Engineering, Tool-based approaches.

## Abstract
*More than five years ago, the OMG proposed the Model Driven Architecture (MDA™) approach to deal with the separation of platform dependent and independent aspects in information systems. Since then, the initial idea of MDA evolved and Model Driven Engineering (MDE) is being increasingly promoted to handle separation and combination of various kinds of concerns in software or data engineering. MDE is more general than the set of standards and practices recommended by the OMG's MDA proposal. The main idea is to consider models as first class entities. In MDE the concept of model designates not only OMG models but a lot of other artifacts like XML documents, Java programs, RDBMS data, etc. Today we observe another evolutionary step. A convergence between MDE and DSL (Domain Specific Language) engineering is rapidly appearing. The notion of domain is common to these two study areas and they share a lot of common goals and practices. In the same way as MDE is a generalization of MDA, the DSL engineering may be viewed as a generalization of MDE. One of the goals of this paper is to explore the potential of this important evolution of engineering practices. In order to anchor the discussion on practical grounds, we present a set of typical problems that could be solved by classical (object-oriented and others), MDE, or DSL-based techniques. Solutions to these problems will be based on current platforms (EMF, AMMA, GME, etc.). This paper illustrates how powerful model-based frameworks, allowing to use and build a variety of DSLs, may help to solve complex problems in a more efficient way. This will in turn lead us to suggest a tool-oriented approach that may be considered as another conceptual extension to MDE and DSL engineering.*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*Conference'06*, Month 1–12, 2006, City, State, Country.
Copyright 2006 ACM 1-58113-000-0/00/0004…$5.00.

## 1. INTRODUCTION
As an emerging solution for handling complex and evolving software problems, Model Driven Engineering (MDE) is still very much in evolution [7]. The industrial demand is quite high while the research answer for a sound set of foundation principles is still far from being stabilized. 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 [48].

This paper focuses on the identification of the basic MDE principles, and the applicability of the related ideas, concepts, and tools to solve current practical problems. Of particular interest also is the present convergence of MDE and DSL (Domain Specific Language) engineering [19]. Both MDE and DSL share the idea that language engineering may help in domain modeling. DSL Engineering is positioned at a more abstract level, using different technical solutions like MDE (sometimes called Modelware), Grammarware [36], XML solutions, etc. MDE mainly uses metamodeling capabilities to implement families of languages in specific application domains.

Observing that MDE is more and more related to DSL engineering, we suggest that MDE principles and tools may be considered as a convenient support technique for building DSL frameworks that may solve existing and newly emerging complex problems. We illustrate this claim with the example of the AMMA (ATLAS Model Management Architecture) framework, an open-source effort of more that 15 person-years that is now being used in a variety of application areas [3]. The various tools are contributed as open source to the Eclipse GMT project [24]. Some of the more stable components, like the ATL transformation language environment, are currently used on more than 100 sites, both in academy and industry (Thales, Airbus, CS, TNI, JPL, Sodius, etc.). Initially considered as a tool support for MDA, for generating platform specific models from platform independent models, AMMA has evolved to be a DSL building framework. It consists of a set of primitive DSLs that will be presented later in the paper like ATL, KM3, TCS, etc. and offers the capability to build sets of related new DSLs for a given domain of for a family of systems.

This paper is organized as follows. Section 2 provides the basic definitions related to models, tools, and DSLs. Section 3 gives a list of typical problems that could be solved by these new

conceptual tools. Section 4 presents some current tools and how they may contribute to solve these problems. Section 5 presents some related work. Section 6 concludes the paper.

# 2. DEFINITIONS

In MDE models are considered as the unifying concept in IT engineering. Traditionally, models have been used as initial design sketches mainly aimed for communicating ideas among developers. MDE promotes models to primary artifacts that drive the whole development process. The notion of model goes beyond the narrow view of semi-formal diagram thus requiring much more precise definitions and modeling languages.

Models come in various flavors. A UML model, a Java program, an XML or RDF document, a database relational table, an entity-relationship schema are all examples of models. We call all these models λ-models where λ identifies the technology used to create the model. Therefore, we need another unification concept that helps us to denote various modeling technologies at a higher level of abstraction. We call this concept *technical space* [37][14] associated with a given precise *metametamodel*. Many technical spaces represent existing technologies that have not been strictly perceived as modeling frameworks.

In the previous paragraph we have referred to a number of concepts that need precise and consensual definitions. It is the purpose of this section to provide these definitions so that this common vocabulary may allow us later to talk more precisely about problems and solutions.

First we need to distinguish between principles, standards, and tools. Figure 1 illustrates the relations among them.

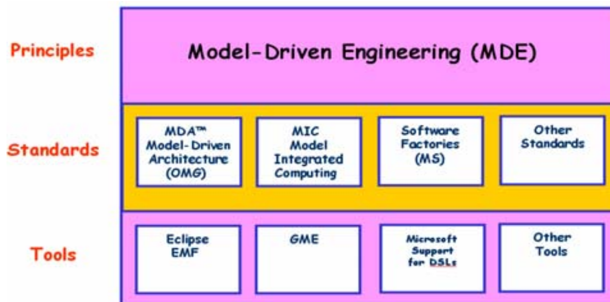


**Figure 1. Principles, Standards, and Tools**

MDE principles provide the conceptual foundation of the approach. They may be applied in multiple ways resulting in different MDE approaches that are often standardized. Every MDE approach/standard relies on a set of tools. Figure 1 enumerates some existing standards and tools.

The central role in the MDE conceptual foundation is played by the notion of model. There are two main definitions of a model corresponding to its internal organization and its potential utilization. Furthermore, the organizational structure of models is constrained by a model called metamodel. A metamodel, in turn, is constrained by a metametamodel. Models, metamodels, and metametamodels together with the relations among them form a metamodeling stack. Figure 2 illustrates the organization of a metamodeling stack.

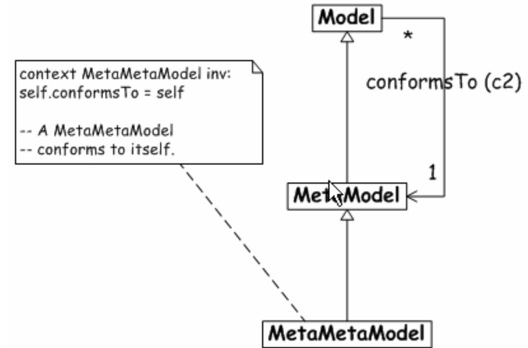


**Figure 2. General organization of a metamodeling stack**

The next section gives precise definitions of the elements in the metamodeling stack.

## 2.1 Model organization definition

The organization of a model reflects its structure. From an organization point of view we perceive models as graphs constrained by other graphs. We propose the following definitions:

**Definition 1.** A directed multigraph $G = (N_G, E_G, \Gamma_G)$ consists of a set of nodes $N_G$, a set of edges $E_G$, and a mapping function $\Gamma_G: E_G \rightarrow N_G \times N_G$.

**Definition 2.** A model $M = (G, \omega, \mu)$ is a triple where:

- $G = (N_G, E_G, \Gamma_G)$ is a directed multigraph
- $\omega$ is itself a model (called the reference model of M) associated to a graph $G_\omega = (N_\omega, E_\omega, \Gamma_\omega)$
- $\mu$: $N_G \cup E_G \rightarrow N_\omega$ is a function associating elements (nodes and edges) of G to nodes of $G_\omega$

The relation between a model and its reference model is called *conformance*. We denote it as *conformsTo*, or simply *c2*.

We may rewrite the previous discussion more precisely as follows, finally explaining the illustration provided by Figure 2:

**Definition 3.** A metametamodel is a model that is its own reference model (i.e. it conforms to itself).

**Definition 4.** A metamodel is a model such that its reference model is a metametamodel.

**Definition 5.** A terminal model is a model such that its reference model is a metamodel.

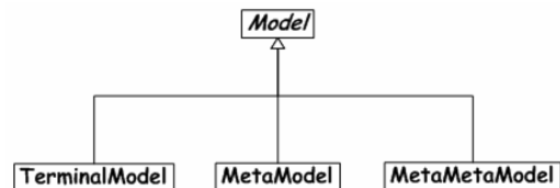


**Figure 3. Classification of models as terminal models, metamodels, and metametamodels**

Figure 3 shows the classification of models implied by the definitions presented so far.

## 2.2 Model utilization definition

The utilization of a model reflects the purpose it is built for and the relation to the real phenomenon it represents. The objective of the model utilization definition is to define the possible usages of

a model. In the present subsection, model will mean "terminal model". We base our second set of definitions on ideas presented by Marvin Minsky in [41]:

*"If a creature can answer a question about a hypothetical experiment without actually performing it, then it has demonstrated some knowledge about the world. … We use the term "model" in the following sense: To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A. … It is understood that B's use of a model entails the use of encodings for input and output, both for A and A*. If A is the world, questions for A are experiments. ... A* is a good model of A, in B's view, to the extent that A*'s answers agree with those of A's, on the whole, with respect to the questions important to B. …"*

An analysis of the expressed view shows that a model is an object always related to another object (object *A* above). We call the latter one a *system*. This leads to a second set of definitions:

**Definition 6.** A system S is a delimited part of the world considered as a set of elements in interaction.
**Definition 7.** A model M is a representation of a given system S, satisfying the substitutability principle (see below).
**Definition 8.** (Principle of substitutability). A model M is said to be a representation of a system S for a given set of questions Q if, for each question of this set Q, the model M will provide exactly the same answer that the system S would have provided in answering the same question.

The relation between a model and a system is called *representationOf* (*repOf*). It is shown in Figure 4.

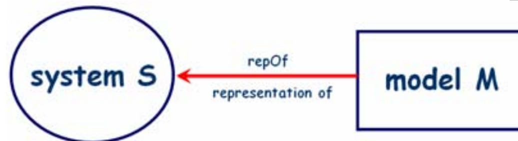


**Figure 4. Utilization definition of a model**

In a set based view, the system is composed of *system elements* and the model is composed of *model elements*. The aforementioned relation *repOf* is thus defined in terms of a partial function $\psi$: S $\rightarrow$ M associating model elements to system elements.

The study of this partial function $\psi$ is one of the big difficulties of model engineering. We may quote here Brian Cantwell Smith in [18]: "*What about the [relationship between model and real-world]? The answer, and one of the main points I hope you will take away from this discussion, is that, at this point in intellectual history, we have no theory of this [...] relationship*". However the increasing importance of understanding the nature of this relation cannot be neglected. This is currently a subject of many efforts in the context of various sciences [2][29][42].

## 2.3 Relations between organization and utilization of a model

The study of the *repOf* representation relation ($\psi$ function) is mostly the responsibility of ontology engineering. The study of the *c2* conformance relation ($\mu$ function) is mostly the responsibility of language engineering. Model engineering may be considered as a synergy between these two fields of language engineering and ontology engineering. This synergy may be made apparent in Figure 5.

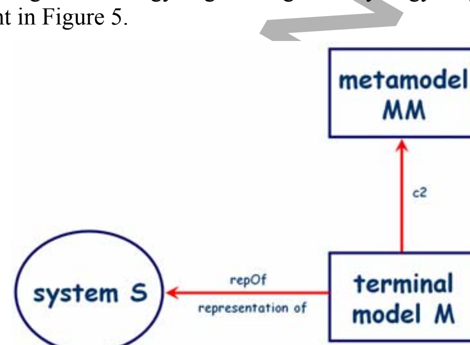


**Figure 5. Dual definition of a model**

Figure 5 illustrates that the organization of a model is closely related to its utilization. The extraction of elements from system S to build model M is guided by the metamodel MM and the purpose of the model. In other words, the metamodel MM acts as a filter that states which elements of the system can be selected to constitute the model M.

## 2.4 Domain Specific Languages

Language engineering is at the hearth of computer science. There is a variety of categories of languages. We discuss here only a small facet of language engineering. A distinction is often made between programming languages and modeling languages. Typical examples are PL/1 and UML. The distinction between these categories has mainly to do with canonical executability. This distinction is currently becoming more and more blur since programs are treated as models and some modeling languages may have the executability property. Another distinction is between General Purpose Languages (GPLs) and Domain Specific Languages (DSLs). PL/1, UML, Java, and C# are examples of GPLs. R [5], SQL [40] or Excel are examples of DSLs.

The distinction between GPLs and DSLs is orthogonal to many other language classifications. For example, there are indifferently visual or textual GPLs or DSLs. Similarly DSLs and GPLs may fall under various categories based on the employed paradigm being object-oriented, event-oriented, rule-oriented, function-oriented, etc. There are examples of imperative and declarative GPLs and DSLs.

A DSL is a language designed to be useful for a delimited set of tasks, in contrast to general-purpose languages that are supposed to be useful for much more generic tasks, crossing multiple application domains. A typical example of DSL is GraphViz [23], a language used to define directed graphs, which creates a visual representation of that graph as a result. Some GPLs have started as DSLs and have sometimes evolved towards genericity to become GPLs. The reverse process has not been observed in the history of programming languages.

Similarly to GPLs, DSLs have the following common properties:

- They usually have a concrete syntax;
- They may also have an abstract syntax;
- They have a semantics, implicitly or explicitly defined;

There are several ways to define these syntaxes and semantics. The most commonly used way for defining the syntax is via

grammar-based systems. In contrast, there are multiple semantic specification frameworks but none has been widely established as a standard.

## 2.5 DSLs and Models

There are strong relations between DSLs and models. We discuss here the possibility of using model-based solutions for defining the syntax and semantics of DSLs.

**Definition 9.** A DSL is a set of coordinated models.

The following comments clarify this definition.

**Domain Definition Metamodel.** As we discussed in the previous section, the basic distinction between DSLs and GPLs is based on the relation to a given domain. DSLs have a clearly identified, concrete problem domain. In contrast, GPLs cover multiple domains. Programs (sentences) in a DSL represent concrete states of affairs in this domain, i.e. they are models. A conceptualization of the domain is an abstract entity that captures the commonalities among the possible state of affairs. It introduces the basic abstractions of the domain and their mutual relations. Once such an abstract entity is explicitly represented as a model it becomes the reference model for the models expressed in the DSL, that is, it is a metamodel. We refer to this metamodel as domain definition metamodel (DDMM). Since the DDMM is a specification of the domain's conceptualization, following the Gruber's definition [27] we may regard it as an *ontology*. This base ontology plays a central role in the definition of the DSL. For example, a DSL for directed graph manipulation will contain the concepts of nodes and edges, and state that an edge may connect a source node to a target node.

Such a DDMM plays the role of the abstract syntax for a DSL.

**Concrete Syntax.** A DSL may have different concrete syntaxes. Each one is defined by a transformation model that maps the DDMM onto a "display surface" metamodel. Examples of display surface metamodels may be SVG or GraphViz, but also XML. An example of such a transformation for a Petri net DSL is the mapping from places into circles, from transitions into rectangles and from place to transition or transition to place relations into arrows. The display surface metamodel will then have the concepts of *Circle*, *Rectangle* and *Arrow*.

**Semantics**. A DSL may have an execution semantics definition. This semantics definition is also defined by a transformation model that maps the DDMM onto another DSL having by itself a precise execution or even to a GPL. The firing rules of a Petri net may, for example, be mapped into a Java code model.

In addition to canonical execution, there are plenty of other possible operations on programs based on a given DSL. Each may be defined by a mapping represented as a transformation model. For example, if one wishes to query DSL programs, a standard mapping of the DDMM onto Prolog may be useful.

## 2.6 Technical Spaces

Technical spaces were introduced in [37], in the discussion on problems of bridging different technologies. This concept was further elaborated in [14] where technical spaces are defined as model management frameworks. The notion of technical space is another important unification concept along with the concept of model. The intention behind it is to denote technologies at a more abstract level in order to allow reasoning about their similarities and differences and possibilities for integration. The following definition is given for technical space:

**Definition 10.** A technical space is a model management framework with a set of tools that operate on the models definable within the framework.

We observe that technical spaces expose an important commonality: they are based on the three-level metamodeling stack thus fitting into the definitions given so far (Figure 6).

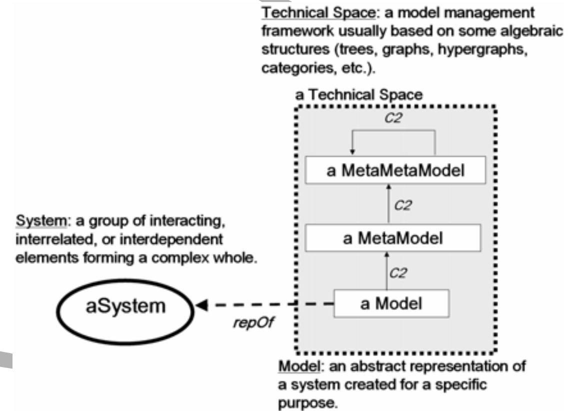


**Figure 6. Systems, models and technical spaces**

Figure 7 gives concrete examples of technical spaces and shows the levels observed in every space.

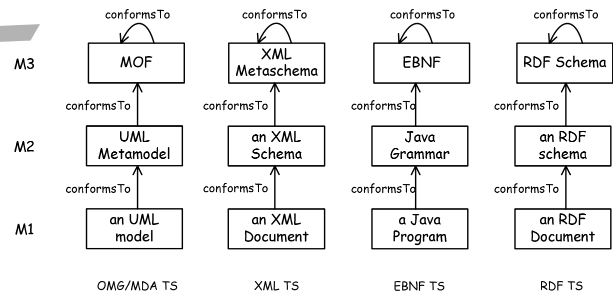


**Figure 7. The three-level model organization in various technical spaces**

This three-level organization is the corner stone for building the model management functionality in a given space. It is mainly based on the fixed metametamodel at M3 and the meaning of the *conformsTo* relation between levels. It should be noted that the *conformsTo* relation is defined differently in different spaces. In OMG/MDA it is defined by the MOF language. In the XML TS it is based on the notion of validity of XML documents. In the EBNF TS, *conformsTo* means that a sentence is syntactically correct according to the grammar rules. Usually, every TS provides tools that check for the presence of the *conformsTo* relation. For instance, in the EBNF TS these tools are language parsers. In XML TS, they are validating XML parsers.

The main role of the M3-level in a TS is to define the representation structure and a global typing system for underlying levels. For example, MOF is based on directed multigraphs where nodes are typed by MOF classes and edges are typed by MOF associations. The notion of "association end" plays an important role in this representation system. Within the EBNF TS we have the specific representation of abstract syntax trees while within

PRELIMINARY VERSION

the XML TS we also have trees but with different set of constraints, for example, with possibilities to have direct references from one node to another node (REFs and IDREFs). The basic representation structure in RDF TS is directed labeled graphs that are typed by an RDF schema.

An important benefit of treating technical spaces as semi-formal entities is the recognition of the various capabilities offered by technical spaces and their combination aimed to solve a given problem. To achieve an effective integration towards a certain goal, however, various technologies should interact with each other. An important requirement for such an interaction is the possibility for transferring an artifact from one space to another space and vice versa. This inter-space transfer is called *bridging*.

Bridging is implemented by a transformation utility called *technical projector*. The responsibility to build projectors lies in one reference space. The rationale to define them is quite simple: when one facility is already available in a given space and building it in another space is economically too costly, then the decision may be taken to build a projector that enables the reuse of the facility. There are two kinds of projectors according to the direction of the transformation relative to the chosen reference space: *injectors* and *extractors*.

# 3. CASE STUDIES AND PROBLEMS

In this section, we present some typical problems. They can be solved with different technologies. We first describe the basic case studies. Then we look for corresponding characteristics in order to establish a list of generic problems.

## 3.1 Telephony Languages

Recently, Voice over Internet Protocol (VoIP) has gained popularity thanks to a combination of reasons including reduced fees compared to standard land-line phones. With this emerging technology several functions that used to be carried out by fixed pieces of hardware are now implemented in software. For instance, wires are replaced by TCP connections allowing users to have a single call identifier wherever they are. Similarly, any computer can now act as a Private Automatic Branch eXchange (PABX) and route calls. One consequence is that telephony routing tables can now be represented as user-specified programs capable of complex behaviors.

To leverage these new possibilities, the Internet Engineering Task Force (IETF) has adopted CPL (Call Processing Language) [38], an XML-based language to control and describe internet telephony services. Other languages, such as SPL (Session Processing Language) [17] have also been developed to play a similar role but with different properties. For instance, CPL is voluntarily limited to be relatively simple and secure. On the contrary, SPL is expressive enough to represent most telephony services while still permitting some properties to be checked on programs. CPL and SPL are two telephony DSLs, but other exist too.

## 3.2 Querying Program Source Code

Today source code is one of the main artifacts in software engineering. Although programming languages are supposed to be a human usable interface with computers, it is often impossible for one person to understand all the details of even modestly sized systems. Therefore, languages and tools to query source code are often necessary. One of the first and still widely used tools is *grep*. It relies on concrete syntax and often on coding style. For instance, line feeds may be inserted almost anywhere in the code but are difficult to handle with standard regular expressions.

More advanced tools are aware of abstract syntax and enable querying relations between entities in the source code (e.g. to get all subtypes of a given Java class). CodeQuest [28] is such a tool targeted at Java. It uses Datalog as a query DSL and is implemented as an Eclipse plugin. CodeQuest makes use of Eclipse Java parsing abilities and comes with a library of predicates to query Java code. Although the approach is generic, the implementation is limited at querying Java source code.

There is a need to unify source code representations at the abstract syntax level independently of the language in which it is expressed. This would enable a single source code query DSL to be used on many languages.

## 3.3 PIM to PSM Transformations

Transformation of Platform Independent Models (PIM) to Platform Specific Models (PSM) was the central problem in the initial vision of Model Driven Architecture [43].

The goal of the MDA approach is to produce software assets that are resilient to changes in the technologies. Such assets should take the form of PIMs, that is, models that do not contain implementation specific details. These models may be transformed to other models that include information specific to the current state of the art implementation technologies (PSMs). PSMs then can be used for code generation. MDA stresses on the importance of PIMs since they are supposed to survive the constant changes in software technologies. If a new technology emerges as a competitor of an existing one (e.g. CORBA vs. Web Services, Java vs. C#) the PIMs remains the stable and reusable entities that are transformed to new PSMs.

Compared to the traditional software development where programs are transformed to machine executable code by a compiler, the transformation and model centric MDA approach is step towards a higher flexibility. Whereas a compiler may be perceived as a transformer with fixed source and target languages, in MDA the languages of PIMs and PSMs are expected to vary. This put a requirement for more flexible transformation approaches capable of handling various source and target languages.

## 3.4 Sensor Data Stream Processing

Following the recent Tsunami disaster, a lot of sensors have been installed in different countries, by different operators, sometimes independently in order to build world-wide real-time surveillance networks. Real-time information about phenomena in the physical world can be processed, modeled, measured, checked, correlated, and mined to permit on-the-fly decisions and actions to be taken on a large scale [49]. Examples include environment monitoring with prediction and early warning of natural disasters, missile detection tracking and interception and many more potential applications in different areas such as medical care, aeronautics, telephony. An airplane, a hospital, or a factory may be equipped with such complex apparatus. The number of involved sensors in such system is exponentially increasing, but also the nature of the data produced and emitted is becoming much more complex as embedded intelligence in the capture device has dramatically evolved. As a result, many systems are becoming world-wide data-centric processing networks. Furthermore the topology of the

PRELIMINARY VERSION

network is constantly evolving with new data sources constantly appearing or disappearing, new data translation, merging, control or measure being added, deleted or updated, etc. Managing such complex systems necessitate the ability to deal with the changing metadata of the various source, targets or intermediary processing nodes in the network.

## 3.5 Bug Tracking

Several tools are often used to ensure the quality of a software product. We may consider the example of “bug-tracing” or “bug-tracking” in the context of software product development. Assume that three teams are currently working on the same product at the same time but on different modules of this product. Teams may be geographically distributed, may have different levels of maturity of the used development process, with different experience of the team members, and they may use different tools. The following situation is typical. Team “A” is developing the first module by using an Excel workbook with a specific format to track bugs. Team “B” is working on the second module and uses Bugzilla [16] which is a free bug-tracking system. Team “C” is developing the third module and uses Mantis Bug Tracker [39] which is another free bug tracking system. The problem is that each team has used a different tool for keeping track of bugs. So in that case, how to succeed in centralizing bug-tracking, i.e. how to be able to interoperate a tool to another without losing critical information about detected bugs? A list of nearly fifty open-source bug-tracking tools may be found on the Web, not counting a lot of commercial products. Each of these tools use similar but often non-compatible data on the bugs found, corrected, validated, etc.

## 3.6 Contract Management

Assume we want to build a contract management tool within an organization. A contract is negotiated between different companies and is directed by a person in charge. There are financial payments made when various parts of the contract are achieved with the production of deliverables. Staff is assigned to the realization of various parts of the contract. The problem is to build a contract management tool that may be used inside the company by various stakeholders. All these users will use a common terminology defined by the domain language supported by the contract management tool. This tool should be considered as belonging to a product line because different variants may have to be built for different contexts. Covered features may include general access control, relation management, project monitoring, project planning, time tracking, statistics, productivity tools, todo management, expense registration, overtime tracking, employee contract management, etc. There is a straightforward way to develop such a contract management tool by producing 100% of the code in classical general purpose programming languages like Java or C#. The way suggested here is rather different. It consists in systematic use of bridging with other tools that implement functionalities (or services) that may serve to construct the facilities of the virtual contract management tool that we wish to build. For example if we need a calendar management, a Gantt chart displaying, a spreadsheet tabular information capture, an accounting reporting, etc., then we may look for specific tools already implementing these facilities (open source tools or proprietary tools already used by the potential users in their environments). There is obviously a service oriented dimension in this approach, but the main idea is to build semantic data conversion bridges between the virtual tool we want to build and the various concrete tools that we are using to build it. The difficulty is that most tools use different conventions and data encoding that need to be harnessed. Of course between the 100% code production and the 100% functionality reuse by tool-bridging, it is likely that an intermediate way may have to be found. However, what we define here as a case study is the possibility to achieve a significant experiment in virtual tool building.

## 3.7 Problem Identification

In these illustrative case studies we may identify several generic problems of interest. In this section we abstract these problems and describe them. The problems are summarized in Table 1.

| | | Case Studies | | | | | |
|---|---|---|---|---|---|---|---|
| | | Telephony Languages | Querying Source Code | PIM to PSM Transformation | Sensor Data Stream Processing | Bug Tracking | Contract Management |
| Problems | Semantics interoperability | X | | | | X | X |
| | Heterogeneous syntaxes | | X | | | X | X |
| | Uniform representation framework | | X | | X | | |
| | Flexible transformations between languages | | | X | | | |
| | Metadata management | | | | X | X | X |
| | Volume scalability | | X | | X | X | |
| | Tool reusability | | | | | | X |
| | Querying heterogeneous data | | X | | | | |
| | Product and Process combination | | | | | X | X |

**Table 1. Problems exemplified in different case studies**

It can be seen that a given problem is usually exemplified in more than one case study. The columns of the table indicate case studies and the rows indicate the problems that we have formulated. The ‘X’ sign indicates in which case study a problem is observed.

The following list summarizes some of the characteristics of the problems mentioned above.

**Uniform representation framework**. The problem of uniform representation framework emerges in the case studies that process data represented in various formats. A major requirement is scalability of the solutions, that is, the possibility to handle an open set of representation formats. In the case study of querying program source code we need a system that represents the concrete syntaxes of programming languages in a uniform way. In the stream-based data processing and conversion the open set of data formats must be accommodated in the same type of uniform underlying representation.

**Semantic interoperability**. In the telephony languages, bug tracking, and contract management case studies there is the common goal of implementing solution for ensuring semantics interoperability between tools or languages. In all these cases we observe different conceptualizations of the same underlying domain. These conceptualizations are usually developed

PRELIMINARY VERSION

independently from each other by different agents. It is possible to have overlap between them but also we may have aspects of the domain that are captured in one tool/language and are missing in another. For example, the concept of bug in Bugzilla has the same meaning as the concept of issue in Mantis but these concepts capture different sets of attributes. Another semantic related problem is observed in the telephony languages case study where the two languages conceptualize the domain at different levels of abstraction. Solving the semantic interoperability problem requires to identify the equivalent and non-equivalent concepts and to resolve the differences between the concepts that refer to the same domain abstraction.

**Heterogeneous syntaxes**. Three case studies deal with various representation formats. They have to handle an open set of heterogeneous syntaxes. A generic system for querying source code requires uniform view over the syntaxes of various programming languages. In bug tracking and contract management we observe the problem of syntactic interoperability along with the problem of semantic interoperability described above. Bug tracking systems may use XML-based or EBNF-based syntaxes as input/output formats. The same heterogeneity is observed in various contract management systems. To achieve tool interoperability in these case studies we need a scalable translation mechanism from one syntax to another. This facility will need to easily allow changes from one representation system to another one, for example from a Java program classical textual representation to an XML corresponding document based on a Java DTD or schema, or to a Java model based on a Java MOF metamodel. Flexibility to apply dynamically these format translation from various contexts is also an important feature.

**Flexible transformations between languages**. This problem is present in most of the case studies to a certain degree but is most apparent in the PIM to PSM transformation case study. Crafting a transformation program on the base of fixed source and target languages is not a big challenge. Much more challenging is developing an open and flexible transformation system. The openness and flexibility properties indicate the ability to handle an open set of source and target languages.

**Metadata management**. The term metadata used here refers to its most general meaning as “data about data”. We stay neutral from any particular form of metadata such as metamodels and metametamodels as defined in section 2. Metadata are of significant importance when different types of data in different formats must be handled. Almost all of the case studies are related to the need of some form of metadata processing. This is most apparent, however, in the case studies on data-centric distributed systems for stream processing, the bug tracking, and the contract management.

**Volume scalability**. Three case studies expose a potential for dealing with large volumes of data that must be processed. The large volume of data that may be considered in stream-based situations or in analysis of large volumes of source code is an important characteristic of these case studies that may necessitate specific solutions. Yet, in the bug tracking system we have potentially a huge number of bugs in case of large software systems.

There is an important aspect observed in the stream-data processing. Many data processing scenarios assume that the data are available prior to the processing. However, in the case of data coming from censors we may have a continuous, eventually infinite stream of data. The problem is then to investigate if the existing techniques for data processing (e.g. transformation and querying) are applicable or new techniques must be invented.

**Tool reusability**. One difference between the bug tracking and the contract management case studies is that the latter implies the building up of a new "virtual" tool from the functionalities available in other concrete tools. We see here a potential for tool reusability as an alternative to class reusability. In other words, it should be possible to build tools without coding, just by establishing bridges with other tools that offer composable functionalities. Obviously this is based on some form of service composition that also needs to consider data integration and semantic interoperability.

**Querying heterogeneous data**. A problem that appears in the source code querying program is the querying of heterogeneous data. How the request may be established or adapted to the nature of the data to be queried is also an important dimension to consider in the problem space. This problem is also relevant to the stream-based data processing case study.

**Product and Process combination**. If we look at the bug tracking and the contract management case studies, we see that there is a strong relation between the data models of bugs or contracts and the lifecycle of these entities. More generally, the product/process aspect separation pattern again applies here. Processes affect the products and the state of the products drives the processes. We need to find generic solution to this problem.

# 4. SOLUTIONS

In this section we outline solutions for the problems identified in the previous section. Solutions are based on the vision that we need domain specific languages to perform various tasks. Consequently, the ability to rapidly define DSLs and to manipulate models expressed in various DSLs is of a key importance. To provide such ability we developed a model-based DSL framework called AMMA. In section 4.1 we give an overall presentation of AMMA followed by descriptions of its components (sections 4.2-4.4). Finally, in section 4.5 we discuss how the problems may be solved by using AMMA.

## 4.1 The Overall Structure of AMMA

AMMA provides facilities for defining domain specific languages. According to Definition 9 a DSL is a set of coordinated models. Among these models are the DDMM, the concrete syntax, the semantics of the language. AMMA provides several DSLs that are used to define the components of other DLSs. They form the core of AMMA. In this work, we focus on a simplified subset of AMMA composed of three DSLs. Figure 8 shows the components of AMMA and how they are used to define DSLs.

PRELIMINARY VERSION

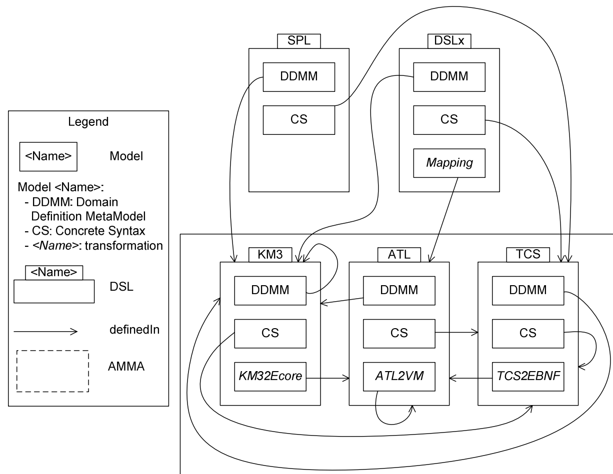


**Figure 8. Structure of AMMA: a model based framework for DSLs**

The core of AMMA consists of KM3, TCS, and ATL languages. KM3 is a DSL for defining metamodels. TCS is a DSL for describing concrete syntaxes of DSLs and how the concrete syntax is related to the DDMM of a given DSL. ATL is a model transformation language.

It can be seen that these three DSLs contain models that are expressed in some other DSL from the core. For example, the DDMM of KM3 is defined in KM3. The concrete syntax of KM3 is defined in TCS. Furthermore, KM3 is mapped to the elements of Ecore by using an ATL transformation (the box *KM32Ecore*). The semantics of ATL is defined as a transformation to the language of the ATL virtual machine (*ATL2VM*), which we described in [32]. This transformation is expressed in ATL.

We can define other DSLs by using the ones provided by AMMA. For example, we experimented with the SPL language (see section 3.2) by defining its DDMM in KM3 and its concrete syntax in TCS. The semantics of the language is not defined since we assumed that it is implemented by already existing tools.

An arbitrary language (denoted as DSLx in Figure 8) can be defined in a similar manner. In the context of DSLx, the box *Mapping* denotes a possible mapping to another DSL or a GPL such as Java.

Currently, AMMA does not provide means for defining semantics of DSLs. In many cases a pragmatic definition of the semantics can be given by providing a mapping from the DDMM of a DSL to the DDMM of another DSL for which there is clearly defined semantics. This mapping can be specified in ATL. We are performing experiments on using Abstract State Machines [15] as formal foundation for specifying dynamic semantics of DSLs. The initial results are promising [9]. This opens the possibility for inclusion of a DSL that captures the ASM mechanisms in the core of AMMA.

It should be noted that Figure 8 is a visual representation of a specific kind of model. Elements of this model are: models, DSLs, the *definedIn* relation between them, and the AMMA framework itself. We call *megamodels* such models, in which some elements represent models or other artifacts (e.g. DSLs, tools and services) as well as relations between them. We are currently working on AM3 [12] (ATLAS MegaModel Management): a generic tool for megamodeling. More details about megamodeling in general and AM3 in particular are available in [12] and [13].

In the subsequent sections we present the three core DSLs: KM3, TCS, and ATL.

## 4.2 KM3: Metamodel Definition Language

Considering the need for DSLs we have been using a language named KM3 (Kernel MetaMetaModel) to define the domain definition metamodel of DSLs. This section briefly presents the rationale of this language.

The KM3 language [33] is intended to be a lightweight textual metamodel definition language allowing easy creation and modification of metamodels. The metamodels expressed in KM3 have good readability properties. These metamodels may be easily converted to/from other notations like Emfactic or XMI.

KM3 has its roots in the complex and evolving relations between modeling and visual languages. The OMG has proposed the MOF language for the definition of its various metamodels (e.g. SPEM, UML, CWM, etc.). The problem was that there was no practical support environment for this language. As a replacement solution the existing UML CASE tools were used. The price to pay for this was an alignment of MOF with a subset of UML (mainly class diagrams). Since this time, the alignment has been more or less maintained through the various versions of UML and MOF. In other words, UML may be considered as a multi-purpose language allowing defining software object-oriented terminal models and also allowing defining MOF metamodels. However, this approach has certain drawbacks. When we need to build a metamodel (for example a source or target model for a transformation), we have first to start building a UML class diagram with certain properties. The result may then be serialized in a first XMI file and then transformed into a second XMI file corresponding to the metamodel. This conversion from a UML model to a MOF metamodel is called a "promotion" and is implemented by some widely available tools like UML2MOF provided in the MDR/NetBeans suite.

We have experimented for some time with this approach. When the number of involved metamodels is limited (i.e. when someone mainly deals with OMG fixed and stable metamodels), there are no major problems. But when we need multiple and evolving metamodels, we found this approach very cumbersome. The only alternative was to define KM3, a textual language for specifying metamodels, including MOF metamodels. After experimenting with this language for two years, we are completely convinced of the practicality of the approach. Public libraries of more than one hundred metamodels expressed in KM3 are now available [50]. ATL (explained in the next section), a QVT-like model transformation language, uses natively KM3 to facilitate the handling of metamodels. Many other projects are based on this format.

Among the properties of KM3 is the possibility to use it for the definition of non-MOF based models. KM3 has also been designed to cross technical spaces.

KM3 has a clear semantics, partially presented in [33]. This semantics is based on the definitions of terminal model, metamodel, and metametamodel presented in section 2. The semantics uses multi-graph structures for representing models and first-order logic to express the required axioms. An

PRELIMINARY VERSION

implementation in Prolog has been developed as a proof of concept. To the best of our knowledge, such a formal definition has not been proposed for the existing metamodeling languages in MDE. As a side effect of this work, we have been able to propose also a precise characterization of a model and a metamodel.

## 4.3 TCS: Language for Defining Concrete Syntaxes

According to Definition 2 (section 2.1), a model is a graph. Tools are required to present such an abstract structure in a user-friendly fashion. TCS (Textual Concrete Syntax) is such a tool. We call concrete syntax the definition of a set of rules allowing the representation of a model. There are several kinds of concrete syntaxes: visual, XML-based, textual, binary, etc. Some are designed to allow convenient transmission and storage of models (e.g. XMI). Others are targeted at users like most visual syntaxes (e.g. class diagrams for metamodels) and some textual syntaxes (e.g. KM3, ATL). Such human usable concrete syntaxes have specific requirements: user-friendliness, low complexity and low verbosity.

TCS is a DSL aimed at specifying context-free textual concrete syntaxes of DSLs. From such specifications, models can be serialized into their textual equivalent and text can be parsed into models. In other words, a TCS specification defines a bidirectional translation utility between a textual representation of a model and its internal representation. The choice of context-free languages was mainly motivated by the observation that programming languages use them extensively. TCS models are used to attach syntactic elements, such as keywords and symbols, to elements of the DDMM of a DSL.

To outline the usage of TCS models, consider the TCS2EBNF transformation in Figure 8. It takes as source a DDMM (expressed in KM3) and a TCS model to generate an EBNF grammar. This grammar is annotated with semantic actions, which build the model in memory while parsing its textual representation. If the grammar of a DSL was directly specified in EBNF (instead of automatically generated), the mappings to the metamodel would also have to be specified in the form of annotations. Annotating a grammar to build a metamodel is a tedious task, which also depends on the parser generator that is used. By providing a dedicated language for this task we can abstract from the underlying parser generator tool (e.g. ANTLR, YACC). Generating an annotated grammar for different parser generators is achieved by providing different transformations on TCS models.

## 4.4 ATL: Model Transformation Language

Model transformation is one of the most important operations on models. The recent efforts in MDE are towards defining DSLs for specification of model transformation programs. In AMMA we provide such a language called ATL (ATLAS Transformation Language).

ATL is a hybrid model transformation DSL. Its declarative part enables simple specification of many problems, while its imperative part helps in coping with problems with higher complexity. The operational context of ATL is represented in Figure 9. A model $M_a$ (conforming to metamodel $MM_a$) is transformed into a model $M_b$ (conforming to metamodel $MM_b$) based on the $M_t$ transformation expressed in ATL. The box ATL represents the DDMM (the abstract syntax) of the language. An ATL program is therefore a model that conforms to this DDMM. More than one source and one target models may be used in practice. Informal semantics of ATL is presented in [31] along with a non-trivial case study. More than forty different scenarios accounting for more than a hundred individual transformations are available on ATL GMT website [51].

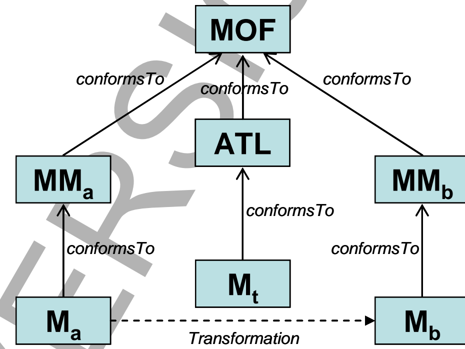


**Figure 9. The operational context of ATL**

The fact that transformation programs are models conforming to the ATL metamodel opens a possibility for interesting applications. For instance, Higher-Order Transformations (HOTs) may be defined that take other transformations as source, target or even both. An example of a HOT is given in [30] to implement traceability in ATL.

The general scenario shown in Figure 9 is similar to the context of the QVT transformation language proposed by OMG. Thus, ATL can be considered as a QVT-like model transformation language. In [32] we explored the relations between ATL and QVT.

The general scheme of a model transformation operation is the generation of output models from given source models by executing a transformation program. We may further identify different schemes that define specific classes of transformations. For example, model composition (model merging) is an operation in which two source models are transformed into one target model following certain constraints. Although ATL is not targeted at a specific class of model transformations such as model merging, other transformation DSLs may be specified by focusing on particular scenarios. We therefore postulate the existence of a family of model transformation DSLs that share the scheme shown in Figure 9.

AMMA provides a generic mechanism for constructing such DSLs. This mechanism is called ATLAS Model Weaving (AMW). It should first be noted that *model weaving* [20] is different from *aspect weaving* [35]. Model weaving is about establishing typed links between model elements. Links themselves form a model and *link types* are therefore defined in a metamodel. Weaving links are more abstract than ATL rules because, whereas ATL has fixed semantics, AMW has user-defined semantics. Consequently, *link types* can be adapted to specific application domains.

The adaptability of AMW to different problem domains is achieved by providing tools working on a core weaving metamodel defining only the abstract notion of *link type*. This core metamodel can be extended by users. Metamodel extension is a complex operation, which will not be discussed here. The basic idea is that user-defined *link types* have to extend the

PRELIMINARY VERSION

abstract *link type* concept defined in the AMW core. Transformations to ATL code can then be used to implement *link types'* semantics.

Furthermore, weaving semantics need not even be executable. This is because AMW application domain is actually broader than transformation specification.

## 4.5 Using AMMA to Solve Problems

In this section we address the problems identified earlier by applying the conceptual framework presented in section 2 and the tools provided by AMMA.

**Uniform representation framework**. The definitions of model, metamodel, and metametamodel given in Section 2 specify a uniform modeling framework. Each model conforms to a reference model itself conforming to a unique metametamodel. Different implementations of this abstract framework may have distinct metametamodels. However, the uniformity of this approach alone is limited to the MDE Technical Space. If we also consider the concepts of Technical Space, and projectors then we broaden the uniformity of the framework. For instance, abstract syntaxes may be represented in MDE (e.g. using KM3), concrete syntaxes in EBNF, and projectors may be used to bridge between these TSs. TCS is an example of such a projector. In summary, our primary unification concepts are model and technical space

To address the problem that necessitates uniform representation framework we may consider various entities in the context of a single TS, for example the MDE TS. Usually, the entities will come from different technical spaces. We may apply injectors to import these entities in the MDE TS. In that way, the uniform representation will be based on the basic representation scheme induced by the metametamodel (see the part of section 2.6 concerning the role of the M3 level).

**Flexible transformations between languages**. We have seen in the previous paragraph that we have a uniform representation framework for models. Languages, or more precisely their abstract syntaxes, are captured as metamodels. The problem of transforming between languages may then be restated as problem of transforming between metamodels. ATL is a DSL designed for this purpose. The ATL metamodel itself is hardwired into the ATL engine along with its execution semantics. The source and target metamodels are, however, specified at runtime. This enables transformations between virtually any metamodels, and therefore languages.

**Semantic interoperability**. Given a uniform representation framework as described previously, semantic interoperability may be approached by specification of mappings between metamodels. Such a mapping can define the relationship between the concepts from several metamodels. In AMMA, we consider two kinds of mappings:

- *Transformations*. A transformation (e.g. written in ATL) is an operational representation of a mapping. It may be executed to automatically transform a model conforming to a metamodel into a model conforming to another metamodel. The way target elements are created from source elements depends on the semantic mapping. With transformations, mappings are specified by the transformation writer.
- *Weavings*. A weaving model (e.g. specified with AMW) can capture semantic relationships between several metamodels as a set of typed links. Actual meaning of this links depends on the semantics of the weaving metamodel. We do not provide a general semantic mapping weaving metamodel as part of AMMA. However, it is possible for the user to define her own weaving metamodel. Additionally, heuristics may be defined to semi-automatically derive weaving links from the metamodels which semantics have to be aligned. This is related to the problem of schema matching in data integration. We do not claim to solve this problem, but rather to provide a tool (namely AMW), which enables a uniform specification of semantic relationships.

To summarize our approach to semantic interoperability we may state that the process of establishing correspondence links between semantic entities is generally semi-automatic. Semantic equivalence, mismatches, and conflicts are judged and resolved by domain experts helped by heuristics. Their decisions are captured in the form of transformations and weavings. AMMA provides tools to automatically execute transformations and to handle weavings as ordinary models.

**Heterogeneous syntaxes**. Heterogeneity may be handled by defining a uniform representation framework for syntaxes and providing bridges between various kinds of syntaxes. The notion of Technical Space based on the three-level architecture provides such a uniform representation framework: context-free syntaxes are represented as grammars, XML-based syntaxes as schemas, etc. Technical Projectors (such as TCS) provide means to bridge heterogeneous technical spaces. AMMA does not provide projectors for every possible situation. However, new projectors may be defined by using AMMA as a building framework.

**Metadata management**. Metadata can take various forms. It can be, for instance, a metamodel, a grammar, an annotation, etc. Whatever their forms are metadata can be uniformly represented as models (e.g. KM3 models, EBNF models). The problem is now to deal with a possibly large number of such models. Megamodeling, which represents models and relations between them as a model, may be used to capture complex relations between various metadata. For instance, a metamodel, a concrete syntax (in TCS or EBNF), and an annotation metamodel may be linked together by megamodel links.

**Volume Scalability.** One possible way to handle large volumes of data is to use tools that are specialized in this task. These are mainly database management systems. In the context of AMMA and the concepts defined so far this is achieved as a projection from one technical space to another. For example, in case of huge files of source code we need a projector from the EBNF technical space to the RDBMS technical space. The latter one provides optimized engines for querying large volumes of data. Once the data are obtained after a query execution we need to extract them in the technical space of initial interest.

This approach, however, may not be applicable in case of continuous and possibly infinite streams of data. To solve this problem we need new techniques for transformations and query execution. This is an interesting direction for future research concerning an important problem.

**Tool Reusability.** This problem is related to two other problems: semantic interoperability and dealing with heterogeneous syntaxes. The solutions proposed for them should be used for this problem as well. Furthermore, often we need to describe a tool chains and flow of data between different tools. Therefore, we

PRELIMINARY VERSION

need a DSL for performing workflow management tasks. Such DSLs exist and it should be possible to incorporate them in our framework for DSL definition. Currently, we perform a practical experiment for solving this problem. Results will be reported in another paper.

**Querying Heterogeneous Data.** One approach to this problem is to represent the data in a uniform way thus eliminating the heterogeneity. The unification concept of model may be used here and then the queries over data are in fact queries over models. Since models are graphs we have the uniformity achieved and then we may run queries over graphs. However, this approach may require translation of the data to be queried from one format to another. That is, again we apply the concept of bridging between technical spaces. In case of large amount of data, however, this may be inefficient.

Another approach is to create a metamodel that provides the structure of the uniform view but no actual data are translated. Queries may be formulated over the uniform view and translated to queries over the original sources. This approach reflects the classical schema integration problem in data engineering. This problem requires query translation. Queries may be considered as models expressed in a query DSL thus opening the possibility to apply model transformations.

It should be noted that we have not experimented with the application of this approach in a practical context so it remains mostly a vision.

If the data to be queried are coming from a data stream then we need novel techniques. This marks another open issue for future research.

**Product and Process Combination.** This problem requires establishing relationships between the process models and product models. Basically, such relationships may be considered as forming a model. What is the semantics and the possible usage of such a model requires further investigation.

# 5. RELATED WORK

In this paper we proposed a model based framework for defining DSLs. There are other approaches that aim at facilitating definition of DLSs. Some of them are based on classical language engineering techniques and others apply MDE principles.

SDRR (Software Design for Reliability and Reuse) [6] is a method based on DSLs, which the authors call Domain Specific Design Language (DSDL). SDRR is targeted at executable DSLs for which computational semantics must be specified in terms of ADL (Algebraic Design Language). *Meaning preserving* program transformations are then used to optimize the definitions. Contrary to AMMA, this approach offers no support for non-executable DSLs and requires a formal semantics to be specified. Having a formal semantics is useful but being able to implement a DSL by simply transforming it to another language is sometimes enough from practical point of view.

GME (Generic Modeling Environment) [25][34] and Microsoft DSL Tools [26] are two approaches for defining DSLs based on MDE principles. The main difference with AMMA is that they are not defined as a set of core DSLs. The Microsoft approach lacks an explicitly specified metametamodel equivalent to KM3 in the case of AMMA. GME focuses on definition of visual syntax whereas AMMA provides TCS which is a DSL for defining textual concrete syntax.

It should be noted that AMMA is not meant to replace these two approaches. Our vision is that the MDE based approaches for defining DSLs should be considered as complementary to each other. We aim at achieving interoperability between these approaches and portability of models across their tools. The notion of technical space and projectors is applied to solve this. We performed case studies in building projectors between AMMA and MS/DSL tools and AMMA and GME. The results of these case studies are reported in [10] and [11] respectively.

There are languages similar to the languages that form the core of AMMA (KM3, TCS, and ATL).

MOF is a standard metametamodel proposed by OMG of which there exist several versions (e.g. 1.4 [44] and 2.0). Both are more complex than KM3 (i.e. they contain more classes). None has a formal semantics. Their standard concrete syntax is XMI, which is based on XML and is, as such, more verbose than KM3 concrete syntax. Human Usable Textual Notation (HUTN) [47] is a standard by OMG that gives a default textual notation to each metamodel.

Ecore [21] is a metametamodel close to MOF 2.0. There is a standard textual notation for EMF: emfatic. One difference with KM3 is that emfatic provides EMF-specific constructs (e.g. to customize Java code generation). One of our experiments has shown that such additional information may be embedded into KM3 comments. Another difference is that Ecore has no formal semantics.

Typed Attributed Graphs [22] are a conceptual framework on which graph transformations are based. They have a precise formal semantics. In contrast to KM3 and the definitions given in section 2, there is no explicit metametamodel: type graphs are not themselves typed.

sNets [8] is one of our past experiments. We have learnt much from them and KM3 is an elaboration of this previous work.

There are two standards by OMG for defining concrete syntax for MOF metamodels: XMI [46] and HUTN. They are therefore closely related to TCS. However, contrary to TCS, both specify an automatic mapping from the metamodel to an XML Schema (for XMI) or an EBNF grammar (for HUTN). XMI is especially adapted to automated serialization of models but relatively difficult to use by human beings because of its verbosity and syntactic constraints (e.g. XMI identifiers). HUTN is more user-friendly than XMI because it uses a simpler textual representation of models but still remains relatively verbose. TCS is capable of reducing verbosity because the syntax is not automatically derived from the metamodel but user-specified. For instance, it is possible to tell TCS to use operators (either with infix or prefix notation) for specific metamodel constructs (i.e. expressions). This is typically something that cannot be automatically guessed.

In the last couple of years we observed a number of proposals for model transformation languages. Some of them are a response to the QVT RFP issued by OMG [45]. ATL is applicable in QVT transformation scenarios where transformation definitions are specified on the base of MOF metamodels. However, ATL is designed to support other transformation scenarios going beyond QVT context where source and target models are artifacts created in various technical spaces.

PRELIMINARY VERSION

Another class of transformation approaches relies on graph transformations theory [1][52]. ATL is not directly based on the mathematical foundation of these approaches. An interesting direction for future research is to formalize the ATL semantics in terms of graph transformation theory. The declarative part of ATL is especially suitable for this. Currently, we are planning to provide formal semantics for ATL in terms of Abstract State Machines.

# 6. CONCLUSIONS

An important contribution of our work has been 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.) covering various formats and data conversions, 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.

Starting from there, the second contribution presented in this paper has been to propose a precise and minimal definition for a conceptual MDE technical space. This technical 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.

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. With the proposed approach we hope to avoid future portability problems in MDE. It would be a pity if the abstraction rise from code to models produces incompatibility between OMG models, EMF models, Microsoft/DSL models, etc. With a clean separation of technical spaces, the precise identification of the inter-space mappings and the definition of efficient support tools for space projectors that are being made available in open source libraries, we are in a good position to deal with different kinds of models, a UML model, a Java program or an XML document being considered as models pertaining to different technical spaces.

Standard compatibility and user efficiency are supported by the toolbox present in the AMMA open source Eclipse platform. AMMA may be considered as an operational mapping onto Java, compatible with the Eclipse Modeling Framework conventions. However the successful practice of these metamodeling tools in the last months has progressively evolved towards the consideration of AMMA as a DSL-building framework. Model engineering is being seen in turn as a successful implementation of a DSL-based platform. We consider now that the main value of AMMA is essentially its collection of core DSLs (ATL, KM3, AMW, TCS, etc.) and its ability to facilitate the building of new DSLs for specific families of systems from the built-in DSLs natively available in the framework. This experimental work is going on with the definition of new DSLs for specific goals, like the specification of dynamic semantics.

Considering model engineering as a way to implement powerful DSL building frameworks is by itself an interesting achievement. Practical experimentation shows that this may allow to solve complex and evolutive systems more easily and efficiently than with conventional technologies like object-oriented programming. We have provided a set of typical case studies where we have done preliminary experiments. Although there is yet no definitive killer application, first results show that the combined MDE/DSL approaches may bring deployable solutions in the short to medium term.

The ultimate validation of the approach would consist in building a new virtual tool for some domain (e.g. bug tracking or project management) from a set of existing tools in different domains. Each domain would be organized around a DSL or a small set of DSLs. Then the functionalities would be acquired from other tools, themselves described with the help of other DSLs. We are presently looking for mounting such an experiment that would help validating more thoroughly the approach. The initial implementation of specific semantic bridges between similar tools seems to provide good indication of feasibility.

# 7. ACKNOWLEDGEMENTS

We would like to thank Freddy Allilaire, Marcos Didonet del Fabro and all the students that have participated in this work. We also acknowledge the support of Microsoft Research Cambridge and of the ModelWare IST European Project 511731.

# 8. REFERENCES

[1] Agrawal A., Karsai G., Kalmar Z., Neema S., Shi F., Vizhanyo A.**The Design of a Simple Language for Graph Transformations**, Journal in Software and System Modeling, in review, 2005

[2] Apostel, L. **Towards the formal study of models in the non-formal sciences**. In H. Freudenthal (Ed.), The concept and the role of the model in mathematics and natural and social sciences. D. Reidel Publishing Company, Dordrecht, the Netherlands, 1960

[3] ATL, **ATLAS Transformation Language Reference site** http://www.sciences.univ-nantes.fr/lina/atl/

[4] ATLAS Group **KM3: Kernel MetaMetaModel**. Available from http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/doc/atl/index.html

[5] Bates, D. & al. **The R language definition for statistical data analysis**. http://stat.ethz.ch/R-manual/R-patched/doc/manual/R-lang.html

[6] Bell, J., Bellegarde, F., Hook, J., Kieburtz, R. B., Kotov, A., Lewis, J., McKinney, L., Oliva, D. P., Sheard, T., Tong, L., Walton, L., and Zhou, T. **Software design for reliability and reuse: a proof-of-concept demonstration**. In Proceedings of the Conference on Tri-Ada '94 (Baltimore, Maryland, United States, November 06 - 11, 1994)

PRELIMINARY VERSION

[7] Bézivin, J. **On the Unification Power of Models**, Software and System Modeling, SoSym Journal, 4(2):171--188, 2005

[8] 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

[9] Bézivin, J., DiRuscio, D., Jouault, F., Kurtev, I., Pierantonio, A. **A practical Experiment to Give Execution Semantics to a DSL for Telephony Services Development**, Submitted for Publication, March 2006

[10] 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

[11] Bézivin, J., Brunette, C., Chevrel, R., Jouault, F., and Kurtev, I. **Bridging the Generic Modeling Environment (GME) and the Eclipse Modeling Framework (EMF)**. Proceedings of the Best Practices for Model Driven Software Development at OOPSLA'05, San Diego, California, USA, http://www.softmetaware.com/oopsla2005/bezivin2.pdf

[12] Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P. **Modeling in the Large and Modeling in the Small. MDAFA'2004**, Springer-Verlag LNCS 3599, pages 33-46.

[13] Bézivin, J., Jouault, F., Valduriez, P., **On the Need for Megamodels**, OOPSLA & GPCE, Workshop on best MDSD practices, Vancouver, Canada, 2004

[14] Bezivin, J., Kurtev, I. **Model-based Technology Integration with the Technical Space Concept**, In Metainformatics Symposium 2005, Esbjerg, Denmark, November 2005, to be published in LNCS volume

[15] Borger, E. **High Level System Design and Analysis using Abstract State Machines**. In *FMTrends 98, Current Trends in Applied Formal Methods*, volume 1641 of *LNCS*, pages 1–43. Springer, 1999

[16] Bugzilla **official site**, http://www.bugzilla.org

[17] Burgy, L., Consel, C., Latry, F., Lawall, J., Reveillère, L., Palix, N. **Language technology for internet-telephony service creation.** to appear In IEEE International Conference on Communications, 2006

[18] Cantwell Smith, B. **The Limits of Correctness**; a paper prepared for the Symposium on Unintentional Nuclear War, Fifth Congress of the International Physicians for the Prevention of Nuclear War, Budapest, Hungary, June 28 – July 1 1985

[19] Consel, C. **Domain Specific Languages**. LABRI, Bordeaux, http://compose.labri.fr/documentation/dsl/

[20] Didonet Del Fabro, M., Bézivin, J., Jouault, F., and Valduriez, P., **Applying Generic Model Management to Data Mapping**, in proceedings of the Journées Bases de Données Avancées (BDA05), France, 2005

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

[22] Ehrig, H., Prange, U., Taentzer, G.: **Fundamental theory for typed attributed graph transformation**. In: Graph Transformations: Second International Conference, ICGT 2004. Volume 3256 of Lecture Notes in Computer Science., Springer-Verlag (2004) 161-177

[23] Gansner, E.R., North, S.C. **An open graph visualization system and its applications to software engineering**. Software - Practice and Experience, Volume 30, Issue 11, Pages 1203-1233, 2000

[24] GMT, **General Model Transformer** Eclipse Project, http://www.eclipse.org/gmt/

[25] GME, **The Generic Modeling Environment**, Reference site. http://www.isis.vanderbilt.edu/Projects/gme/

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

[27] Gruber, T. R. **Toward Principles for the Design of Ontologies Used for Knowledge Sharing**.,International Journal of Human and Computer Studies, 43(5/6): 907-928, 1995

[28] Hajiyev, E., Verbaere, M., de Moor, O. **CodeQuest: Scalable Source Code Queries with Datalog** Programming Tools ECOOP'2006, Nantes, July 2006

[29] Hughes, R.I.G., **The Ising model, computer simulation, and universal physics**. In M. Morgan and M. Morrison (Eds.), Models as mediators. Perspectives on natural and social science. Cambridge University Press, 1999

[30] Jouault, F : **Loosely Coupled Traceability for ATL**. In: Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability, Nuremberg, Germany

[31] Jouault, F., Kurtev, I. **Transforming Models with ATL**, Workshop Model Transformations in Practice, collocated with MoDELS 2005, Montego Bay, Jamaica, October 2005

[32] Jouault, F, Kurtev, I. **On the Architectural Alignment of ATL and QVT**. In: Proceedings of ACM Symposium on Applied Computing (SAC 06), Model Transformation Track, Dijon, Bourgogne, France, April 2006.

[33] Jouault, F., Bézivin, J. **KM3: a DSL for Metamodel Specification** FMOODS 2006, Bologna, Italy, 14-16 June 2006

[34] Karsai, G., Gray, J. **Component Generation Technology for Semantic Tool Integration**. Proceedings of IEEE Aerospace 2000 Conference, Big Sky, MT, March, 2000

[35] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W.G., **An Overview of AspectJ**, Lecture Notes in Computer Science, Volume 2072, Jan 2001, Page 327

[36] Klint, P., Lämmel, R. Kort, J., Klusener, S., Verhoef, C., Verhoeven, E.J. **Engineering of Grammarware**. http://www.cs.vu.nl/grammarware/

[37] 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/

[38] Lennox, J., Wu, X., Schulzrinne, H. **Call Processing Language (CPL): A Language for User Control of**

PRELIMINARY VERSION

**Internet Telephony Services**. RFC 3880, http://www.ietf.org/rfc/rfc3880.txt, 2004

[39] Mantis **Bug Tracker official site**, http://www.mantisbt.org

[40] McJones, P. (ed) **The 1995 SQL Reunion: People, Projects, and Politics**. SRC Technical Note 1997-018, August 1997, Digital System Research Center, http://www.mcjones.org/System_R/SQL_Reunion_95/SRC-1997-018.pdf

[41] Minsky, M. L. **Matter, Mind and Models Semantic** Information Processing**,** MIT Press, 1968

[42] Molenaar, J. **Mathematical modeling and dimensional analysis**. In A. van den Burgh and J. Simonis (Eds.), Topics in Engineering Mathematics. Modeling and Methods, Kluwer Academic Publishers, 1992

[43] OMG. **MDA Guide** version 1.0.1. OMG document omg/2003-06-01, 2003

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

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

[46] OMG/XMI XML **Model Interchange (XMI)** OMG Document AD/98-10-05, October 1998. Available from www.omg.org

[47] OMG. **Human Usable Textual Notation (HUTN) Specification**, Final Adopted Specification. OMG Document ptc-02-12-01, 2002

[48] Soley, R., and the OMG staff. **The Model Driven Architecture Whitepaper**, OMG document, http://www.omg.org/mda

[49] Tham, C-K, Buyya, R. **SensorGrid: Integrating Sensor Networks and Grid Computing**. Available from http://www.gridbus.org/reports/sensor-grid.pdf

[50] The ATL Group, **The Atlantic metamodel zoo in KM3**, http://www.eclipse.org/gmt/am3/zoos/atlanticZoo/

[51] The ATL Group, **ATL Transformation Examples**, http://www.eclipse.org/gmt/atl/atlTransformations/

[52] Varró, D., Varró, G., Pataricza, A. **Designing the automatic transformation of visual languages**. Journal of Science of Computer Programming, vol. 44, pp. 205-227, Elsevier, 2002

PRELIMINARY VERSION