Project acronym: MODELPLEX

Project full title: MODELling solution for comPLEX software systems

Contract n° 034081

Workpackage 2: Framework Building

Deliverables D2.1.a:

"Global Model Management Principles"

Project co-funded by the European Community under the "Information Society Technology" Programme

Contract Number: 034081

Project Acronym: MODELPLEX

Title: MODELling solution for comPLEX software systems

Deliverable N°: D2.1.a

Due Date: 06/2007

Delivery Date: 07/2007 – Resubmission date 03/2008

Short Description:

This report addresses the capture and controlled management of a landscape of multiple possibly heterogeneous modelling artefacts, and the specifications for a supporting tool.

This version is a resubmission of the deliverable as per the request by EC and reviewers in the Annual Review report.

Lead Partner: INRIA

Made available to: PU (Public)

Rev	Date	Author	Checked by	Internal Approval	Description
0.1	21/11/06	Draft – Hugo Bruneliere, INRIA	Jean Bezivin, INRIA		Initial version
0.2	19/12/06	Contributions integrated from: Hugo Bruneliere, INRIA	Jean Bezivin, INRIA		Updates on sections 5.1, 7 and 9.
0.3	24/12/06	Contributions integrated from: Jean Bezivin, INRIA			Updates on various sections
0.4	20/03/07	Contributions integrated from: Hugo Bruneliere, INRIA			Minor editing corrections and reformatting
0.5	27/03/07	Contributions integrated from: Hugo Bruneliere, INRIA			Updates on section 10
0.6	29/03/07	Contributions integrated from: Mikaël Barbero, INRIA	Nick Dowler, ADAPTIVE		Section 7, 8 and 9 updated
0.7	26/05/07	Contributions integrated from: Nick Dowler, ADAPTIVE	Hugo Bruneliere, INRIA		Section 9.4
0.8	02/05/07	Contributions integrated from: Jean Bezivin, INRIA	Hugo Bruneliere, INRIA		Various minor updates
0.9	11/06/07	Contributions integrated from Hugo Bruneliere, INRIA		**1st Internal Reviewer** : Jerome Lenoir, TRT **2nd Internal Reviewer**: Parastoo Mohagheghi,	Updates corresponding to the received reviews Section 10: First analysis of the final requirements assigned to Task 2.1 (after the Oslo meeting)

				SINTEF	Section 11: Adding of an initial calendar for the release of the prototype V1
1.0	25/06/07	Final deliverable	QB	EB	
1.1	18/03/08	Contributions integrated from Hugo Bruneliere, INRIA			Updated version of section 10 with SAP use case information (according to review's comments)
2.0	25/03/08				Final deliverable

Table of Content

1. Executive Summary 6
2. Acronyms and Terminology 7
3. Introduction 7
4. The Need for GMM 8
5. A Conceptual Framework for GMM 10
   5.1. Overview 10
   5.2. Conceptual Definitions 13
6. AM3: an Initial Prototype for GMM 16
   6.1. AM3: a Part of the AMMA Platform 16
   6.2. Overview 17
   6.3. Current Functionalities 17
   6.4. Zoos (Resources) 18
   6.5. Plug-ins (Tools Implementation) 20
7. Specifications Overview of the GMM Support Tool 22
   7.1. Terminology 22
   7.2. Main Characteristics and Functionalities 24
   7.3. Overview of the Architecture 25
8. Requirements for Distribution 27
9. Repository Support for GMM 28
   9.1. Common Repository Interface 28
   9.2. Lightweight Repository 28
   9.3. Database Repository 28
   9.4. Heavyweight Repository 29
10. GMM for End-Users: WP1 Requirements First Analysis 31
11. Initial Calendar: First Version of the GMM Prototype 36
12. Conclusion 37
13. References 38

List of Figures

Figure 1 Definition of Model 10
Figure 2 Organization of the Metamodeling Stack 11
Figure 3 General MDE Framework 11
Figure 4 Global Model Management (GMM) Conceptual Framework 12
Figure 5 “Megamodels” Metamodels 13
Figure 6 Eclipse/GMT AM3 component’s home page 17
Figure 7 AM3 Zoos page 18
Figure 8 "AM3 Resource Navigator" & "OCL Query View" 20
Figure 9 TGE: the KM3 example of use 21
Figure 10 Overview of the AM3 GMM Support Tool architecture 25
Figure 11 Calendar for the release of the GMM prototype first version 36

List of Tables

Table 1 MDE general concepts 14
Table 2 GMM specific concepts 14
Table 3 AM3 zoos 19
Table 4 AM3 Atlantic Zoo mirrors 19
Table 5 AM3 plug-ins 21
Table 6 GMM support tool’s terminology 23
Table 7 Analysis of the requirements which are assigned to Task 2.1 32
Table 8 Analysis of the requirements which implicates Task 2.1 35

1. Executive Summary

Systems are becoming more and more complex. Developing and managing such systems by applying Model-Driven Engineering (MDE) methodologies is becoming a central issue. Because of the high complexity and heterogeneity of these systems, we need to invent new ways to cope with a very large number of involved artifacts. Until now, MDE has essentially concentrated on defining simple operations on models, like capture, transformation, verification, etc. This is for example the outcome of the experience gained in the ModelWare project [13]. But we need much more now. Just to give a motivating example, many problems are solved not only by a unique transformation, but by a long chain of transformations. At the beginning of the chain there may be a conversion from XML (for instance considering an UML model stored in XMI format) and at the end of the transformation there may be another conversion to Java or C# code for example. Each atomic transformation in the chain may have a different source and target metamodel. If we count the number of involved artifacts (terminal models, transformations, metamodels), this is becoming quite significant. Furthermore, some metamodels or transformations may be based on other metamodels or transformations that are available on other sites on the Web. The handling of all these elements may easily bring much more complexity to some MDE processes. We want to avoid these drawbacks and the solution we propose is to apply MDE in order to deal with the accidental complexity that may be generated by MDE processes and related artifacts. The work undertaken in this task is to propose a set of solutions to handle a large number of distributed resources necessary to carry out activities in model engineering. More particularly we propose a global view of the handling of these resources called Global Model Management (GMM).

GMM aims to provide support for modeling in the large, i.e. managing global modeling resources in the field of MDE-oriented software developments. These global resources are usually heterogeneous and distributed. Thus, to access them without increasing the accidental complexity of MDE, we need to invent new ways to create, store, view, access, and modify the global modeling entities that may be involved in developing a practical solution. The goal of this initiative is to provide a simple access to the MDE developer so that he/she may deal with complex problems with simple interfaces.

This deliverable is mainly about defining the general principles and concepts of GMM and about providing a specifications overview for GMM support tools. It also describes the identified needs for GMM and an already existing GMM tool prototype (i.e. AM3). In addition to this, it provides some explanations on the GMM requirements for distribution and repository support as well as a first analysis of the WP1 end-users expectations concerning GMM.

2. Acronyms and Terminology

AM3	ATLAS MegaModel Management
AMW	ATLAS Model Weaver
ATL	ATLAS Transformation Language
DSL	Domain-Specific Language
DTD	Document Type Definition
EMF	Eclipse Modeling Framework
GMM	Global Model Management
MDE	Model Driven Engineering
OCL	Object Constraint Language
SAP	SAP (WP1 partner)
TID	Telefonica (WP1 partner)
TIS	Thales (WP1 partner)
UML	Unified Modeling Language
WGO	Western Geco (WP1 partner)
XML	eXtended Markup Language
XSD	XML Schema Definition

3. Introduction

The idea of *megaprogramming* was introduced by B. Boehm [1] in order to propose a solution to the construction of large-scale software systems. The related idea of *megamodeling* was proposed in order to cope with the accidental complexity that has been observed when building real-life model-driven engineering (MDE) solutions to practical problems [2].

MDE mainly suggests basing the software development and maintenance process on chains of model transformations (Model-to-Model, Model-to-Text or Text-to-Model). A single transformation is often quite easy to handle, but as soon as we tackle real-life situations (as those described by WP1 partners) we are faced with large sets of MDE artefacts (i.e. models, metamodels, transformations, etc) from which we have to assemble a solution. The classical programming level tools are of no significant help here to manage this kind of situation. If we want software systems to be designed from a high number of models, metamodels, transformations, converters and other similar components, we need to provide a Global Model Management environment that will allow MDE users to handle this complexity without major penalties.

Within this deliverable, we will address this specific problem by describing the need for GMM, specifying a conceptual framework and presenting AM3 (ATLAS MegaModel Management) which is, in its current version, a very first prototype for GMM. We will also give a specifications overview of a GMM support tool and explain the GMM requirements in terms of distribution and storage (i.e. repository). Finally, we will start to show how GMM can provide answers to WP1 end-users expectations for their MDE processes.

4. The Need for GMM

As summarized in the previous introductive section, Global Model Management is one of the current main issues concerning MDE. Thus, it is possible to highlight different needs and requirements for using GMM solutions in current software developments and management.

The building of complex software solutions involves more and more models in the broad sense. These numerous models are not only UML models and can be of very various natures (e.g., models can be expressed in XML documents that conform to specific XSD schemas or DTD, it can also be EMF models that conform to different metamodels expressed in Ecore format, etc). Moreover, these models are often linked to each other and are also involved in complex chains of operations which, most of the time, consist of sequences of transformations. As a consequence, the solution providers need facilities (i.e. methodologies and tools) for managing all these models, specifying all the relationships and dependencies that may exist between them and building the complex chains of operations involving all these different modeling artifacts.

In addition to this, because solutions such as the Eclipse Modeling Framework (EMF) or the Microsoft DSL Tools are being increasingly used, we may anticipate that MDE is going to be more and more present in many organizations.

Thus, applying MDE processes to complex real-life use cases (like those presented in WP1 for example) means that we need to handle a huge number of various and varied modeling artifacts.

These artifacts are:

a) **Numerous**. One application may use several hundreds of such components selected between large libraries of available components amounting to tens of thousands of units and even more.

b) **Distributed**. Some components may be available on a Web site different from the one where the application is executed or deployed.

c) **Interrelated**. The artifacts are related by strong semantic links. For example a QVT or ATL transformation should refer to its source and target metamodels. These metamodels may be themselves versions or extensions of other metamodels. A model Mb obtained from a model Ma by an ATL or QVT transformation Mt may record its origin model and its transformation model Mt. Moreover these models Mb and Ma may be related by a traceability relation. These are only a few of the semantic relations that may be found in a set of MDE artifacts.

d) **Heterogeneous**. The artifacts are of different natures. They should be categorized in a very systematic organization, i.e. a typing system. The simplest idea that comes to mind is to consider that all artifacts are models. Then we have a simple solution which consists in stating that each one is typed by its metamodel. We will mainly follow this conjecture

in the present work that all managed artifacts are models, conforming to a precise metamodel.

e) **Complex**. The artifacts may contain a lot of internal elements. Here again, if we follow the simplifying conjecture stated above, the possible nature of elements contained in one artifact is defined by the metamodel.

In order to provide access to these artifacts in the definition of MDE solutions, we need to hide additional generated complexity here. A search into a large distributed library of metamodels or the consecutive chaining of a lot of different operations on models are examples of global model management facilities that should be provided. The simplest way is to apply general MDE principles to the handling of models. This is the main idea of megamodels described below, i.e. models which elements represent model themselves.

5. A Conceptual Framework for GMM

Before being able to provide a GMM environment, we first need to study and design a conceptual framework. Within this section, we propose a set of concepts for GMM and we also present their precise definition.

5.1. Overview

A GMM approach is based on several general concepts. Some of them, corresponding to a conceptual MDE framework, have already been clearly identified, defined, and presented in [3]; some others are introduced here in order to address more specifically the particular problems of GMM.

Within this overview section, we are going to progressively introduce these general concepts in order to lead to the presentation of a complete GMM conceptual framework. We only provide here general descriptions of the concepts; more formal definitions are given in the next section.

We start by explaining the generic core concept of “model”. We propose the simple basic definition shown in Figure 1:

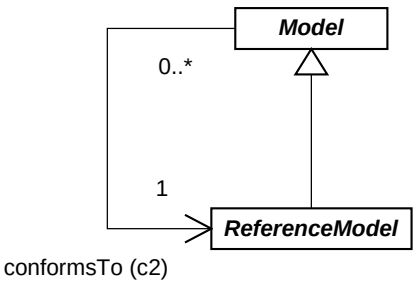

Figure 1 Definition of Model

According to this definition, each model in an MDE approach *conforms to* (or *c2*) a reference model which is itself a model (i.e. which itself conforms to a reference model). By taking into consideration this simple definition of a *model* and several already existing technical spaces, we identify three main levels of modeling that we call M1, M2 and M3. Indeed, we observe that three levels are usually considered in technical spaces, for instance:

- In XML: documents, schemas and the schema of XML Schema,
- In EBNF: programs, grammars and the grammar of EBNF.

Figure 2 summarizes this three-level organization:

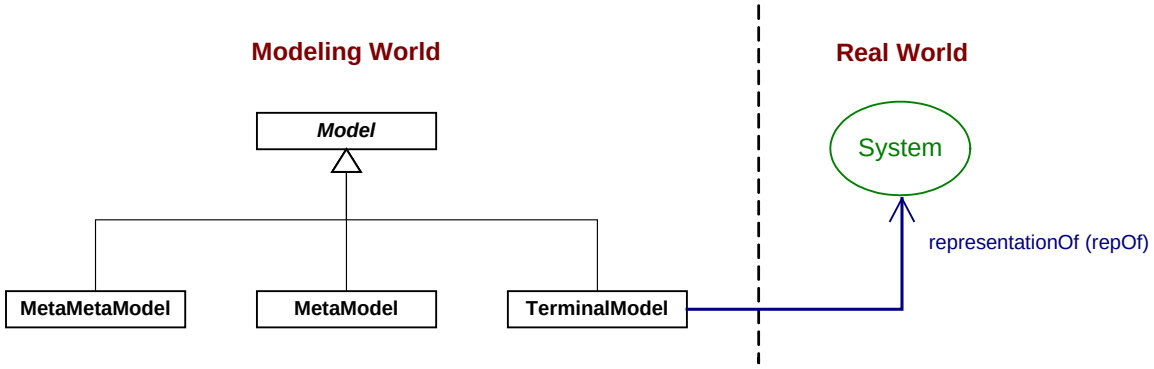

Figure 2 Organization of the Metamodeling Stack

M1 (i.e. *terminal model-level*) consists of all models that are not metamodels and that are directly *representations of* (or *repOf*) real-world systems. M2 (i.e. *metamodel-level*) consists of all metamodels that are not the metametamodel. M3 (i.e. *metametamodel-level*) consists of a unique metametamodel for each given technical space.

By merging the two previously provided definitions (i.e. the definition of *model* and the definition of the *metamodeling stack*), we obtain a complete MDE framework which is totally compatible with the OMG view as illustrated in [4]. This generic and general modeling framework is described in Figure 3:

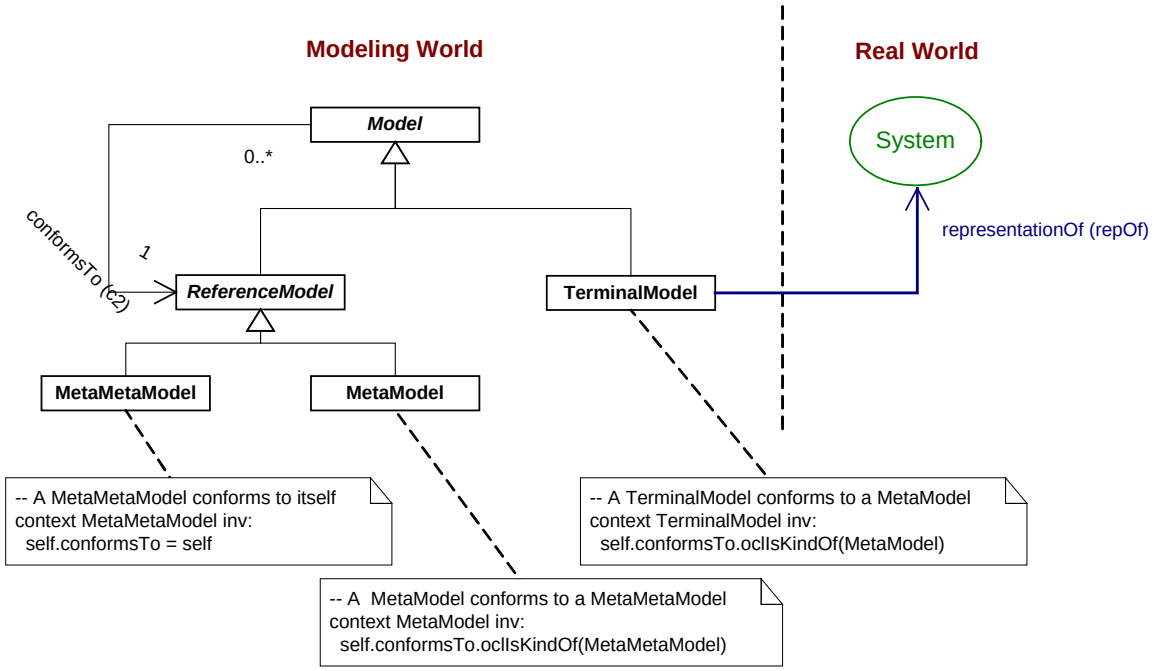

Figure 3 General MDE Framework

In MDE (Model Driven Engineering), models are first class entities. Models are organized in three different categories corresponding to three different levels: *terminal models* (M1-level), *metamodels* (M2-level), or *metametamodels* (M3-level). Each model conforms to its reference model, i.e. a metamodel or a metametamodel. It is important to note that the reference model of a metametamodel is always itself. A terminal model is a representation of a *real world system* and also conforms to a given metamodel (its reference one).

As previously mentioned, more precise definitions of the concepts of *model*, *terminal model*, *metamodel* and *metametamodel* are provided in the next section.

The previously described MDE framework is a general one. In order to deal with GMM-specific problems, we have to propose new dedicated concepts. The GMM framework presented in Figure 4 is built upon this MDE general framework and introduces a reduced set of additional concepts:

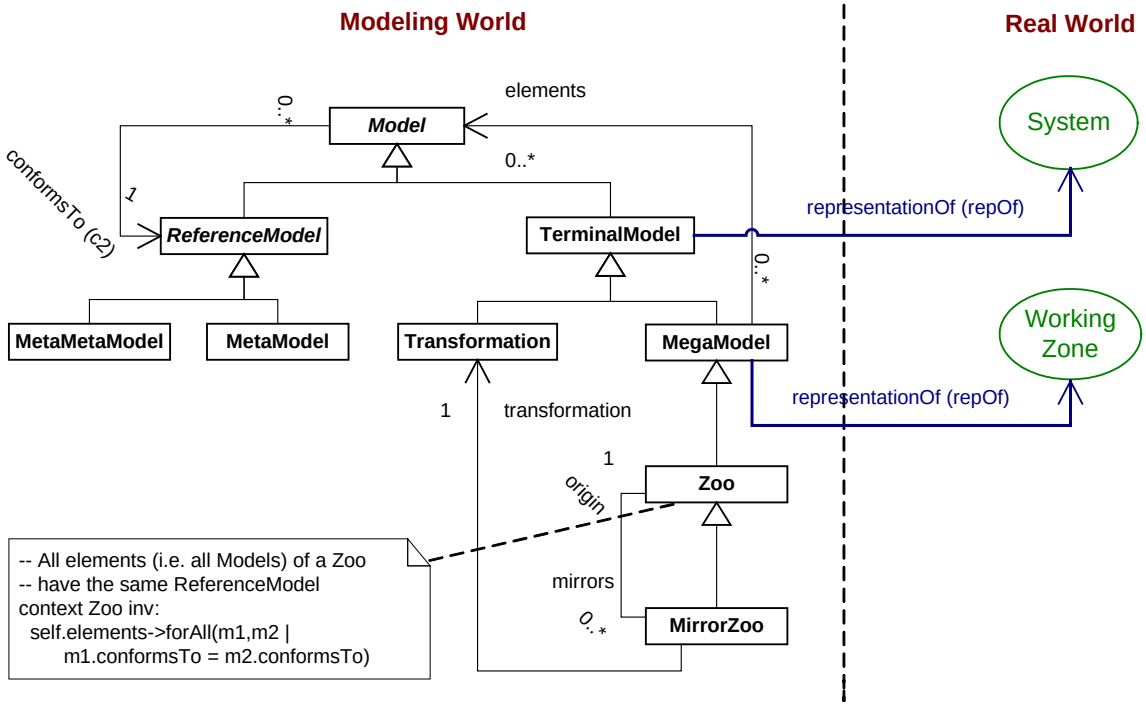

Figure 4 Global Model Management (GMM) Conceptual Framework

Transformations and megamodels are examples of *terminal models* with particular properties.

A *transformation* is a model that can be processed (with a virtual machine for example) in order to build target models from source models, both of them conforming to metamodels that may be different. An associated megamodel can store references to the transformation source and target models and metamodels.

We introduce the concept of *megamodel* [2] in order to provide some kind of registry for models in the context of GMM. A megamodel is a model that contains global entities like terminal models, metamodels, transformations, etc. In our approach, a megamodel is considered as a representation of what we name a “working zone” (i.e. a context composed of various MDE artifacts). For an MDE application developer, the working zone is the complete set of MDE artifacts that he/she may potentially use to build the application.

We also add the concepts of *zoo* and *mirror zoo* which are megamodels with some specific properties. We call *zoo* a megamodel in which all elements (i.e. each model) conform to the same metamodel. As a consequence, we can imagine building zoos of terminal models, metamodels, transformations, etc. We

call *mirror zoo* a zoo generated, from another zoo by a given transformation, such that all its models conform to the transformation output metamodel. So, each zoo may have several mirrors, built using different transformations, which conform to different metamodels.

Megamodels conform to a particular type of metamodels. These metamodels may be original ones but also extensions of already existing ones. This principle is summarized in Figure 5:

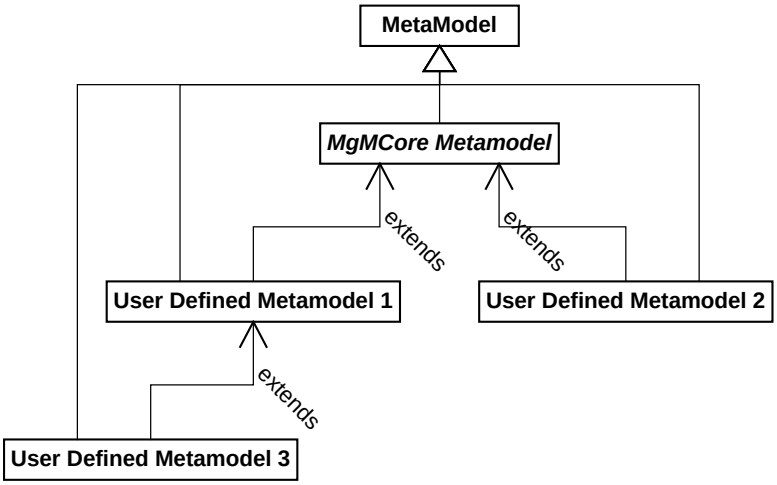

Figure 5 “Megamodels” Metamodels

We believe that there is a basic megamodel metamodel that defines standard resources and behaviors in the context of GMM (we call it *MgMCore* Metamodel in Figure 5). However, users may want to refine this metamodel (or another already defined extension of this metamodel) if they need to specify their own GMM policy. That is the reason why we introduce the "*extends*" relation (and corresponding mechanism) in order to allow them to extend an existing metamodel by creating a new metamodel that conserves the properties of the extended metamodel.

The next subsection gives more precise and formal definitions of the different modeling concepts mentioned and used within this overview.

5.2. Conceptual Definitions

In this section, we elaborate on the concepts previously presented by giving, for each of them, a complete informal definition. Note that most of the following definitions are related to the core concept of *model*:

Concept	Definition
Technical space	A model management framework [5], belonging to the “modeling world”, with a set of tools that operate on the models definable within the framework.
System	A delimited part of the world (the “real world”) considered as a set of elements in interaction. It can be represented in terminal models.
Model	A representation of a given system. For each question of

	a given set of questions, the model will provide exactly the same answer that the system would have provided in answering the same question.
Terminal model (M1)	A model such that its reference model is a metamodel, i.e. it conforms to its reference metamodel. It is a representation of a “real world” system.
Metamodel (M2)	A model such that its reference model is a metametamodel, i.e. it conforms to its reference metametamodel.
Metametamodel (M3)	A model that is its own reference model, i.e. it conforms to itself.

Table 1 MDE general concepts

Now that we have clearly defined the general concept of “model” and all its main related concepts, we must specify the new concepts we propose in order to deal with the general problems of GMM.

Concept	Definition
Working zone	A delimited part of the world (the “real world”) consisting of MDE resources.
Transformation	A terminal model that defines a transformation from a model $M1_A$, conforming to a source metamodel $M2_S$, to a model $M1_B$ conforming to a target metamodel $M2_T$. Its reference model is a transformation metamodel (like the metamodel of the ATL language for example [6]).
Megamodel	A terminal model such that all its elements are references to models (i.e. all kinds of modeling artifacts and modeling tools like terminal models, metamodels, metametamodels…). It is a representation of a “real world” working zone. We consider this concept as the core of our GMM approach but also as a central part of the “modeling in the large” principle discussed in [7].
Zoo	A megamodel such that all models that compose it have the same metamodel (i.e. the same reference model). The kind of modeling artifact that can be found in a zoo may vary (as an example, the “Atlantic zoo” and the “ATL transformation zoo” which are located in [8] are zoos, respectively of metamodels and of transformations). Alternatively, a zoo may be considered as a view on a megamodel. This view may be implemented as a transformation for example. A zoo may have several mirrors, each of them having this zoo as original zoo.
Mirror zoo	A zoo that has been automatically generated, by the execution of a given transformation, from a specified original zoo. There are several types of events that can be the triggers of the generation, or regeneration, of a mirror zoo (as an example, when a modification occurs in the original zoo…).

Table 2 GMM specific concepts

All the concepts detailed in this section are summarized in Figure 4. Note that there are different types of event that can be linked to the various kinds of model (i.e. transformation, megamodel, etc) presented in the previous table. Thus, a *GMM Event* metamodel and the corresponding facilities should be defined in order to be able to handle and manage all kinds of event that may be encountered in a GMM process.

The next section describes the initial version of AM3 (ATLAS MegaModel Management) which is a first proposition of prototype for GMM.

6. AM3: an Initial Prototype for GMM

AM3 (for ATLAS MegaModel Management) is an Eclipse/GMT component whose aim is to provide an environment for Global Model Management.

Within this section, we are going to present the initial state of this component (i.e. at the beginning of the MODELPLEX project): its situation in the AMMA platform, its general description, its current functionalities and the resources and tools it currently provides.

Note that because AM3 is still in a development phase, its available features and resources will considerably evolve during the course of MODELPLEX.

6.1. AM3: a Part of the AMMA Platform

The “ATLAS Model Management Architecture” (AMMA) Platform is an MDE platform developed by the ATLAS Group and built on the top of Eclipse. It is totally integrated into the Eclipse Project: it takes part of the top-level “Eclipse Modeling Project” (EMP) and more precisely of its “Generative Modeling Technologies” (GMT) subproject, which is a research incubator for innovative modeling environments.

The AMMA Platform is currently composed of three GMT components which contain sets of plug-ins for Eclipse and MDE resources:

AM3 for **“ATLAS Megamodel Management”**

ATL for **“ATLAS Transformation Language”**

AMW for **“ATLAS Model Weaver”**

AM3 [8] is the component that provides tools and resources for Global Model Management (GMM). Its initial content is detailed in the next subsection.

ATL [6] is the component that provides the ATL language for performing models transformations and also the KM3 language which is a DSL for defining metamodels. An ATL development environment under Eclipse is implemented in this component. The ATL component is now part of the Eclipse Modeling M2M project (dedicated to Model-to-Model Transformation).

AMW [9] is the component that provides resources and the corresponding Eclipse tooling for defining weaving extensions (i.e. extensions of the basic weaving metamodel) and performing models weaving using these extensions.

6.2. Overview

The AM3 component offers to Eclipse community's users some general documentations, three functional free-downloadable plug-ins and three zoos (with several mirrors for one of the three zoos), as shown in Figure 6.

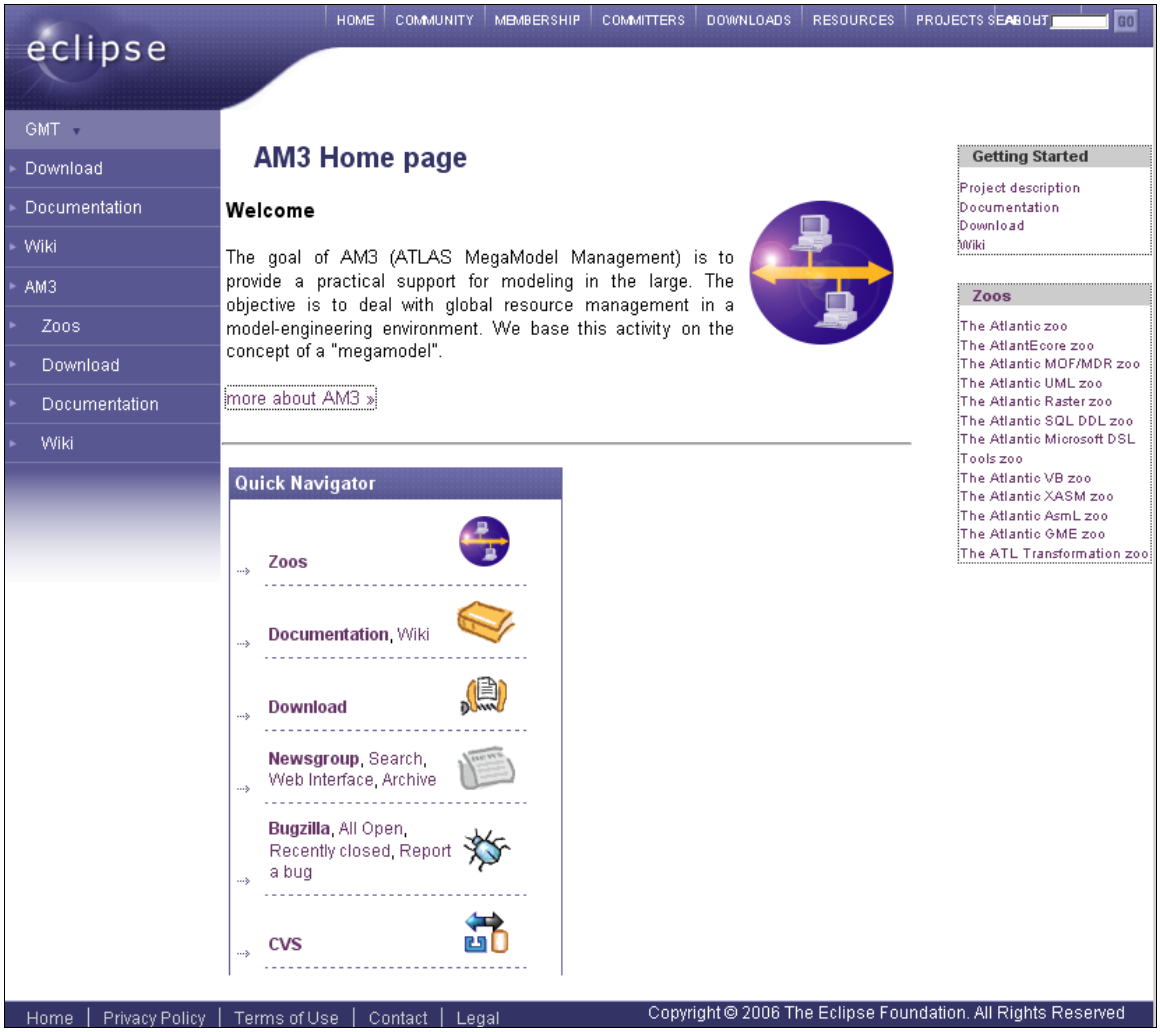

Figure 6 Eclipse/GMT AM3 component's home page

6.3. Current Functionalities

The current implementation of the AM3 project provides several basic features in the context of GMM. In fact, it already allows users to index in a megamodel the most common kinds of modeling artifacts (i.e. most common resources and their associated tools) and so to be able to handle them more easily. The currently supported operations are more detailed in the next two paragraphs of this subsection.

A default *megamodel metamodel* and a basic AM3 megamodel (conforming to this metamodel) are currently defined and have been contributed to AM3. This available megamodel involves only local resources for the time being. It can be interrogated by OCL requests in order to retrieve some particular resources such as models, metamodels, editors, outlines, projectors (injectors or extractors), etc. In addition to that, it is also possible to add new artifacts to this megamodel and

to specify their metadata. Users can define their own megamodels representing their own working zone by using the TCS (Textual Concrete Syntax) provided in AM3 to facilitate the creation of megamodels. Currently, there is only one megamodel loaded in memory at the same time: this megamodel is a merge of all the available megamodels (in the workspace and in the plugins).

Resources found in a megamodel can be directly handled with AM3 or with the other main tools of the AMMA Platform (ATL [6] and AMW [9]). An important feature is the *ATL-specific ANT tasks* that have been implemented in order to allow users to dynamically load and save models (i.e. terminal models and metamodels expressed in Ecore format) and to build and launch complex chains of ATL transformations. A recent initiative to provide an Eclipse minimal workflow engine may also be mentioned here.

6.4. Zoos (Resources)

As previously mentioned, the AM3 project already provides many MDE resources which are mainly at this time, metamodels in different formats and transformations (as it is shown in Figure 7).

Figure 7 AM3 Zoos page

In fact, three zoos and several mirrors are currently freely available in the AM3 project. They are indexed and described in the two following tables.

Zoo	Description
Atlantic zoo	It is composed of metamodels expressed in KM3 (Kernel MetaMetaModel) format. This zoo has several mirrors (see Table 4) that are auto-generated from it by using chains of model transformations in ATL.
ATL Transformation zoo	It is composed of model transformations expressed in ATL (ATLAS Transformation Language) format.
AMW zoo	It is composed of weaving metamodels and extensions of weaving metamodels expressed in KM3 format. For the moment, it provides the AMW weaving core metamodel and some already defined basic extensions metamodels.

Table 3 AM3 zoos

Atlantic Zoo's Mirror	Description
AtlantEcore zoo	It contains metamodels expressed in EMF **XMI 2.0**, conforming to **Ecore**.
Atlantic MOF/MDR zoo	It contains metamodels expressed in MDR **XMI 1.2**, conforming to **MOF 1.4**.
Atlantic UML zoo	It contains UML -class diagram's representations of metamodels expressed in MDR **XMI 1.2**, conforming to **UML**. They are compatible with the Poseidon UML CASE tool.
Atlantic Raster zoo	It contains graphical representations of metamodels expressed in **PNG** bitmaps.
Atlantic SQL/DDL zoo	It contains metamodels' representations expressed in **SQL DDL** (Data Definition Language), conforming to **SQL**. They have been tested with the MySQL DBMS.
Atlantic Microsoft DSL Tools zoo	It contains "domain models" expressed in the **DSL Tools** specific XML format (".dsldm" files). These files are usable under Visual Studio 2005 with the Visual Studio 2005 SDK (including the DSL Tools).
Atlantic Microsoft Visual Basic zoo	It contains metamodels expressed in Visual Basic source code for Visual Studio 2005.
Atlantic XASM zoo	It contains metamodels expressed as abstract machines in XASM which is an open source compiler for Abstract State Machines (ASMs).
Atlantic AsmL zoo	It contains metamodels expressed as abstract state machines in the Microsoft Abstract State Machine Language.
Atlantic GME zoo	It contains metamodels expressed in the Generic Modeling Environment (GME) format.

Table 4 AM3 Atlantic Zoo mirrors

All files contained in these zoos or mirror zoos are directly downloadable in the "Zoos" page of the AM3 Project. Several other mirrors of the Atlantic Zoo, as well as mirrors of the ATL Transformation Zoo, may be created and added in the future to the list of available zoos in this component.

6.5. Plug-ins (Tools Implementation)

The AM3 project also provides, in addition to the MDE resources described in the previous section, a set of tools that allow users to handle these kinds of resources. These tools currently implement the functionalities previously presented in section 6.3 as shown by the next figure:

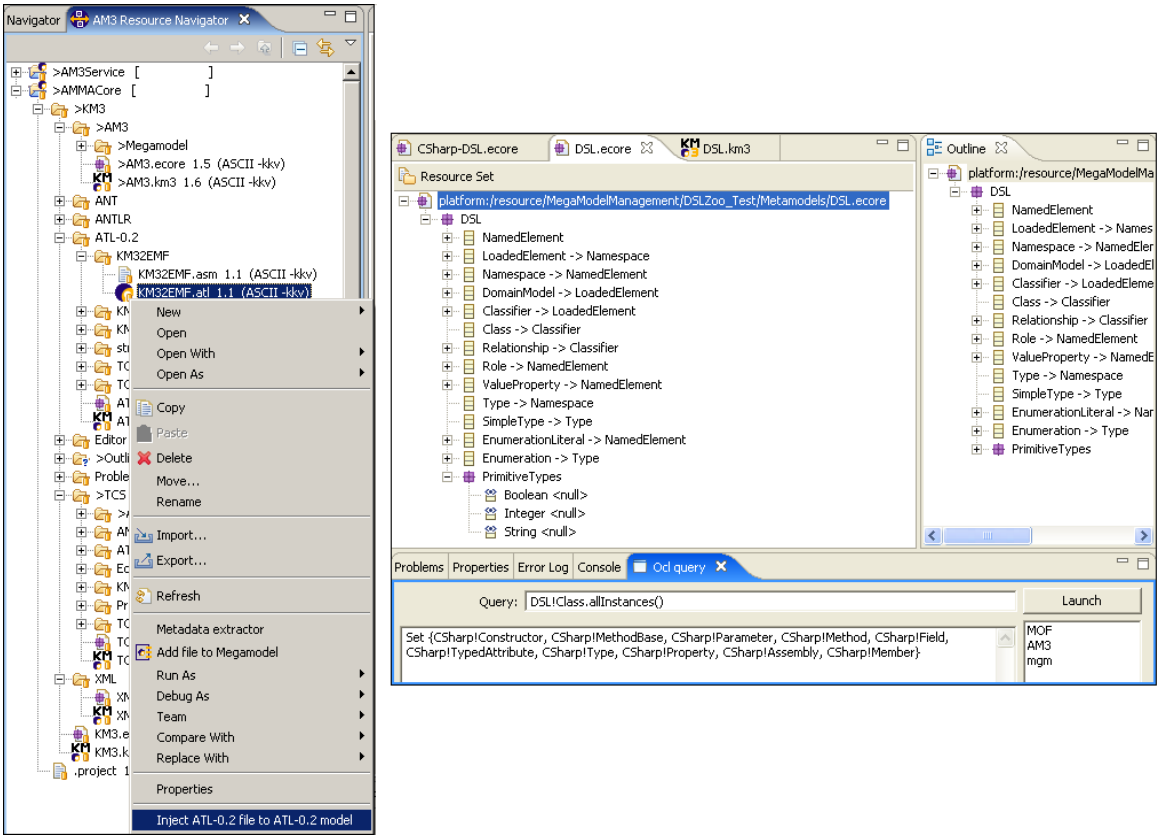

Figure 8 "AM3 Resource Navigator" & "OCL Query View"

In a more technical point of view, the provided tools are encapsulated into three different Eclipse plug-ins which are described in the following table.

Plug-in	Description
org.eclipse.am3.core	The Core plug-in provides resources and the general mechanism (of the megamodel) used to handle and manage these resources in our GMM environment. For the moment, a metamodel of megamodels is defined and loaded by this plug-in as well as a basic AM3 megamodel that conforms to this metamodel. It also provides a TCS (Textual Concrete Syntax) for megamodels definitions.
org.eclipse.am3.ui	The UI (User Interface) plug-in provides a user interface under Eclipse for the Core plug-in. The already implemented “AM3 Resource Navigator” and the still in development “OCL Query View” (see Figure 8) are typical examples of that kind of UI. This user interface would be improved by adding new tools and new functionalities.

org.eclipse.am3.tools.tge	The TGE (Textual Generic Editor) plug-in provides an editor and outline for textual languages. To provide these two features, TGE needs the metamodel of the language, a model conforms to the Generic Editor metamodel, a model conforms to the Generic Outline metamodel and an injector. Figure 9 provides an example of use for the TGE plug-in, as it is already implemented in AM3 for the KM3 language.

Table 5 AM3 plug-ins

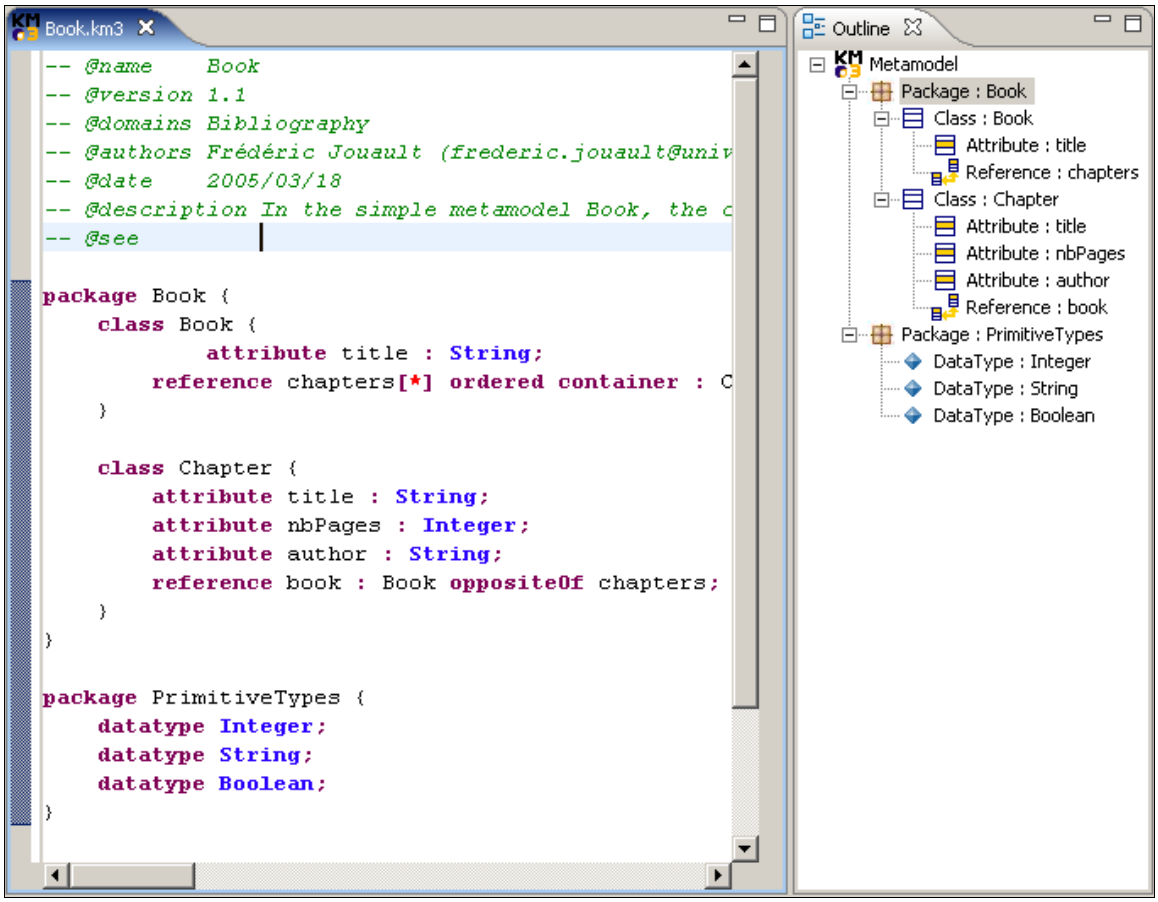

Figure 9 TGE: the KM3 example of use

These three plug-ins, although being already functional, will probably be completed or replaced by new ones during the development of the new GMM support tool.

7. Specifications Overview of the GMM Support Tool

Now that the conceptual framework has been clearly defined and that some experiments have been performed with the AM3 GMM tool's prototype, the objective is to provide a first specifications overview of what a GMM support tool must be.

Within this section we are going to present the general terminology related to a GMM support tool (i.e. how do we name the different items involved), to provide its main characteristics and functionalities (i.e. what it does), to describe its overall architecture (i.e. how it is built) and to give some details on each of its components (i.e. how it works).

7.1. Terminology

This first and important step is about identifying and defining the terms that will be used when describing the GMM support tool's specification. Our goal here is not to add new notions to the conceptual framework (already presented in section 5) but to provide a more technical terminology. By doing this, we want to ensure that everybody is using the same term for talking about the same thing when presenting the specification of a GMM support tool.

Thus, the following table lists the different terms we plan to use and gives, for each of them, a precise definition in the scope of a GMM support tool.

Term	Definition
Storage Method	A model can be stored in various ways: in a file on a file system, in a relational database, in an object database, etc. Each of these digital formats has an absolute or relative locator which provides the ability to retrieve them.
Model Locator	A model locator is an abstract concept whose concrete implementation is a reference being able to be dereferenced. This locator can be relative to the execution context (e.g. the current user home's directory) or absolute (e.g. a fully qualified web address).
Model Identifier	A model identifier is a unique Uniform Resource Identifier (URI) which allows identifying this model. It has no link to the model locator. When requesting a megamodel knowing the searched model, this identifier allows accessing to the model description (i.e. metadata). Most of the time, the model locator will be stored in this metadata. This leads to double indirection for retrieving models.
Model Repository	A model repository is a consistent set of models with a common storage method. For instance, we can have a file system based repository, a database based repository, etc. It provides interfaces for adding, accessing or deleting models from the

	repository. Most of the model repositories are linked to fixed model handlers and do not accept models from other model handlers (for instance, Teneo, Netbeans MetaData Repository, Adaptive Metadata Manager, etc.). A model repository can be considered as a very simple megamodel with an implicit metamodel. The models referenced by the repository (i.e. the "megamodel" elements) can be terminal models, metamodels or metametamodels.
Model Handler	A model handler is a software artifact which allows manipulating models and their content (i.e. model elements). We can cite for instance EMF or JMI implementation in MDR. Its goal is to allow accessing and modifying the information stored into different models. They often provide an integrated implementation of a metametamodel (Ecore for EMF, MOF 1.4 for the JMI implementation of MDR, etc).
Megamodel	A megamodel is a terminal model (the conceptual definition has already been given in Table 2). In a technical point of view, it can be manipulated by model handlers (as it is a model), registered in a repository and so on and so on. Its main purpose is to define metadata on models and to provide facilities for accessing to them.
Versioning	It is the fact of dealing with the problem of having several different versions of the same model. The GMM support tool must provide tooling for bringing a solution to this problem (i.e. managing different model's versions without losing/erasing any information). Versioning of the models may be transparent to the model handler and preferably the storage method. It could be possible that the model identifier can include versioning metadata or be transparent through the model locator.

Table 6 GMM support tool's terminology

7.2. Main Characteristics and Functionalities

A Global Model Management support tool must be independent of any specific implementations of the general concepts previously introduced (section 7.1). As a consequence, it is very important to abstract from these concepts by providing an abstraction layer which describes the platform core concepts and services, and so allows a wide range of different implementations.

Indeed, the main feature of a satisfying GMM support tool is its extension capacity, i.e. the extensibility of its core metamodel of megamodel and of the corresponding services:

- The core metamodel must be essentially composed of the abstract concepts of *entity* and *relationship*. *Entities* represent models in the broad sense (see section 5) whereas *relationships* describe the way these entities (i.e. models) are related the ones to the others.
- The core services must mainly consist in specifying basic adding, modifying, removing and querying facilities on megamodel's entities and relationships.

Depending on the domain we want to model (i.e. the domain or context in which a MDE process is applied), we may want to refine the kind of models that can be managed and the different types of relationships that may exist between us. Thus, a GMM support tool must provide a mechanism for defining domain-specific metamodels which are extensions of the core metamodel of megamodel (directly or indirectly via another metamodel that extends the core one). It is also important to note that the metamodel extensions must often be associated to core services extensions, i.e. query/view-specific facilities extensions, specific model locators, etc.

Another important feature is that, even though the model storage method is strongly related to the model locator's type, a GMM support tool must not be linked to any specific storage method. In other words, model storage methods and model locators have to be as independent as possible. Being tied with some specific implementations would lead to a non scalable or non evolutionary megamodel. For instance, model repositories (sometimes called metadata repository in the literature) can be considered as the very first megamodel and they are tied to some specific storage methods. As a consequence, for scalability and compatibility issues, it is important to take them in account.

A GMM support tool also has to be independent from any model handler: it has to provide a generic interface for allowing different model handler implementations to be used. This is the foundation of GMM whose main objective is to allow managing models coming from various and varied technical spaces.

From a more functional point of view, a megamodel must be associated to a set of services for handling and querying it, as previously mentioned in this section (i.e. retrieve a set of models and/or relationships between models, add/remove a model to/from a megamodel, add/remove a relationship between models to/from a megamodel, etc). This way, it must be possible to define some views on the megamodel and so to be able to reason on it, for instance by performing some kind of analysis or by producing different types of metrics.

7.3. Overview of the Architecture

A general view of the GMM tool support architecture is proposed and depicted on the figure below.

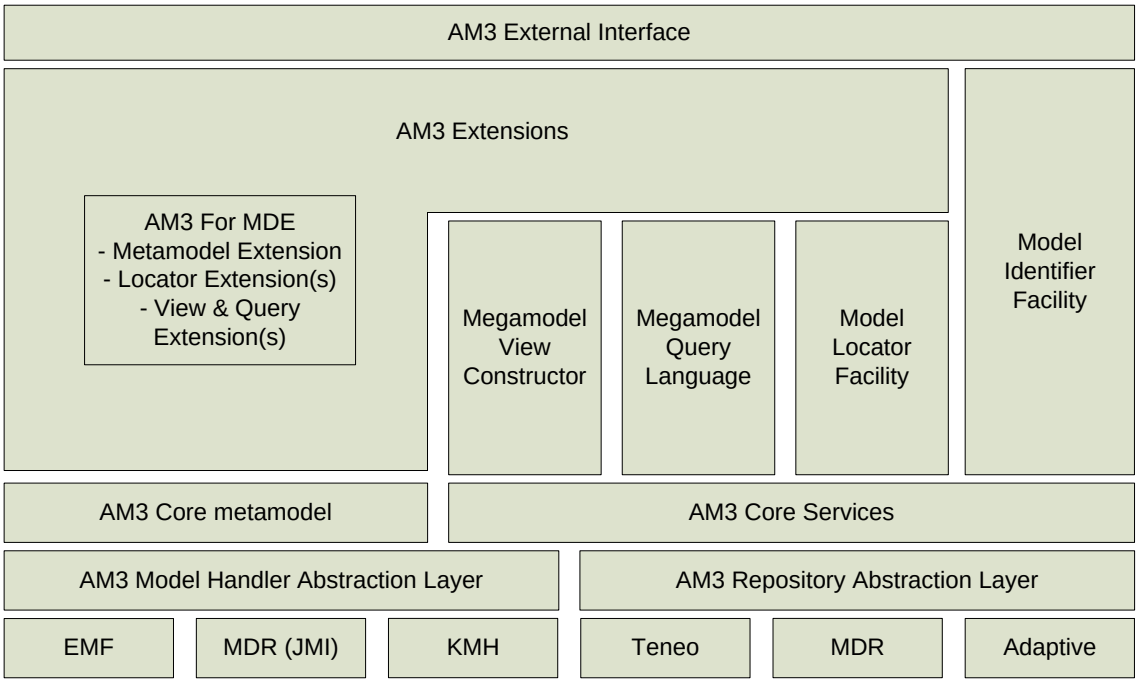

Figure 10 Overview of the AM3 GMM Support Tool architecture

At the bottom of the presented construct are the already existing model manipulation tools like EMF, MDR, etc. There are two distinct categories of underlying tools: model handlers and model repositories. On top of these two kinds of tools, we must have an abstraction layer being able to communicate with them in a transparent way independently from the context. For instance, a same model handler implementation must be able to communicate with different repository implementations and vice-versa.

The model handler abstraction layer is mainly about doing specific dirty work of the underlying model handler and providing a way for manipulating megamodel elements.
The aim of the model repository abstraction layer is to define adding, retrieving, deleting and browsing generic facilities on repositories. Some repositories also provide a mean for making some request onto them or for managing model versioning. This also has to be abstracted.

On top of those two abstraction layers, there is two other core AM3 (our GMM tool support) facilities:

- The first one is the core metamodel which is defined only on top of the model handler abstraction layer. It is implemented by only one model handler but it is manipulated by the abstraction layer.
- The second one is the core services wrapper which is built atop the two underlying layer. It is just an adaptive one for unifying the way the services have to communicate with model handlers and repositories.

The core services are of four different kinds:

- View constructor is a generic service for building views of a megamodel. It provides display facilities and some editing functions.
- Query language provides a simple default language for querying megamodel. This can be OCL with a set of helpers as a very first start.
- Model locator is an abstract facility for dereferencing a model locator. We will provide some default implementation (Eclipse File System locator, EMF registry locator and Adaptive, Teneo and MDR repositories locators)
- Model Identifier is a common facility for identifying model. It mainly relies on URI generic syntax. It does not have to be dereferencable. A document about models URI is currently on the go: “Cool URIs for megamodeling” describing best practice for URI construction of models.

Finally, upon the core metamodel and extensible facilities, it is possible to define what we called some AM3 extensions. These extensions will provide a metamodel extension and some specific implementations of model locators, of extensions of the query language, etc.

8. Requirements for Distribution

As we have already explained it within this deliverable, the use of MDE methodologies for developing complex software solutions implies the management of a huge number of modeling artifacts. That is the reason why there is an imperative need for a GMM support tool.

These modeling artifacts are resources that are not systematically locally stored. They may be physically and/or geographically distributed. Moreover, a GMM process often involves many different modeling tools or components that may also be distributed. This is particularly true if we consider real-life practical use cases like those described by WP1 partners.

As a consequence, an efficient GMM environment must be able to manage a large number of distributed modeling resources and tools. It has to implement facilities for handling local components but also for dealing with external ones. The Eclipse platform seems to be a suitable environment for addressing this specific problem. Indeed, it already provides tools and mechanisms for allowing not only simple distribution but also large-scale interoperability and extension.

In our specific implementation of GMM, distribution of artifacts will be managed with a set of proxies. Models that are referenced in a megamodel with an unknown identifier will be created as models proxy. When it will be necessary to get all metadata on the model, we will request this model proxy to get every data from other megamodels.

9. Repository Support for GMM

Because of the potential large quantity of data handled in a GMM process, having an efficient repository is an essential requirement for a GMM support tool. However, depending on the volume of data that has to be processed, we can consider three main different kinds of repository for three different kinds of use: a “lightweight” one, a “database” one and a “heavyweight” one. All of them have to be directly pluggable into the GMM support tool by using a generic interface. This common repository interface, as well as the different types of repository we plan to provide, are presented and described within this section.

9.1. Common Repository Interface

Since the need of having different kinds of repository for different kinds of use has been highlighted, the first step is to design and specify a common interface for the different repositories that may be plugged into the GMM support tool.

The common repository interface will provide some generic operations such as:

- Adding a model,
- Removing a model,
- Retrieving a model,
- Making some basic queries

Queries on repository may or may not be different from megamodel queries. It may be necessary, in some cases, to have megamodel queries directly linked to repository queries.

9.2. Lightweight Repository

In order to be able to test or experiment on basic use cases without having to deploy a heavy solution for storage, it is important to provide with our GMM support tool a simple lightweight repository. This subsection is about presenting the “lightweight” file system-based repository we have implemented.

This simple repository will be based on a fixed folder containment hierarchy. This containment will represent the “conforms to” relationship between models and metamodels. Models will be retrieved by a unique serial number and a mnemonic name. There will be no query facilities on this repository.

In a first time, this kind of file based repository will be only provided for the EMF model handler implementation.

9.3. Database Repository

A solution for dealing with practical, but not too complex, use cases is to have a repository which is built upon a database. This kind of repository may be considered as a “middleweight” one. This subsection is about briefly presenting the repository we plan to implement using the TENEO solution [10] for mapping efficiently (with the Hibernate framework) the content of EMF models into a relational database.

Model locator in a database may be as simple as the primary key of the root container. It may also be a set of keys.
A mapping between the megamodel query language and possible SQL requests may be implemented.

9.4. Heavyweight Repository

In order to be able to deploy the GMM support tool in complete and complex industrial use cases involving huge amounts of data (like those proposed by WP1 partners), it is essential to provide a large-scale “heavyweight” repository. This sub-section is about presenting the already existing Adaptive repository [11] which is a practical and concrete example of such a repository.

Below are the Adaptive Repository Services, further support is being developed for the Model Handler and Model Location services, through interfaces based upon MOF Facility and Versioning, which are exposed via web services so that the repository can be remote from the GMM support tooling.

There needs to be an implementation of the megamodel OCL query language into something that can be used to query the megamodels from the support tool.

- *MOF Metamodeling and supporting Global Model Management*
  - Support for UML tools for authoring MOF™ metamodels, such as IBM RSM and MagicDraw. Also Adaptive Visio stencil for creating MOF metamodels
  - Support for standard metamodels, particularly from the OMG
  - Dynamic in-place MOF metamodel changes through MOF XMI differences import
- *Model Partitioning* – ability to define logical partitions of the repository for the purposes of model access control and versioning.
- *Model Versioning*
  - Support for versions and branches
  - Support for labeling versions and branches
  - Efficient ‘delta’ storage
  - Version freezing from modification, including successive version and branch
  - Merging of versions (with specification of which branch has merge conflict precedence)
  - Basis for the OMG MOF 2.0 Versioning specification, co-authored by Adaptive and IBM
- *Extended Data Types*
  - Additional data types to those defined by MOF but in a consistent manner. For example Date, Timestamp and BLOB for storage (and versioning) of sources or formats that are not parsed into structured models.
- *XMI Import and Export*
  - XMI import and export are fully supported for XMI levels 1.x and 2.1, including delete but not other aspects of the XMI differences section. There are sophisticated pre and post processing capabilities which can take advantage of the Transformation Architecture described in a following section
  - Instances of multiple metamodels may be included in a single XMI file
  - XMI supports remote references into the repository
  - Export can also be based upon an Adaptive Object View
- *MOF Reflective Interface* – that calls through the specific generated interface to ensure any hooks or OCL4 constraints are executed.

- *Database Connection Pooling* – JDBC connections from which repository requests are serviced; if connections are stale, they are automatically reconnected.
- *Aspect-Oriented Metamodeling* - Adaptive has implemented an implicit common super class ‘Ref(lective) Object’ that allows independent metamodels to be developed to cover different aspects of behavior, by establishing associations to RefObject without requiring modification to the ‘content’ metamodels. This permits, for example:
  - Support for cross metamodel linkage
  - Views that have metamodel merges
  - Namespace Support – Allows for unique naming of objects within a defined scope which is distinct from the internal UUID. This is used for import reconciliation and aliases for objects that can be referred to differently in external systems when maintaining interoperability with the repository

10. GMM for End-Users: WP1 Requirements First Analysis

Within this section, we are going to describe different possible applications of the previously presented GMM principles and corresponding tool in the context of the MODELPLEX project. We want to more specifically emphasize on how GMM may fit some of the requirements expressed by the WP1 partners in their case studies and gathered into the deliverable D1.2.a “Initial Requirements Specifications”.

In a general way, GMM may be applied to each of the MDE scenarios which are parts of a software development or management process. This is particularly true when considering complex software systems where these kinds of process are often complicated and involve a lot of modeling resources (see section 4).

If we read the deliverable presenting the four case studies provided by the four WP1 partners (namely SAP, THALES (TIS), Telefonica (TID) and Western Geco (WGO)), we immediately see the need for GMM. Indeed, each case study specifies a set of MDE scenarios in the context of complex systems. For instance in the THALES use case, the presented MDE scenarios (see section 2.1.4.3 of D1.1.a “Case Study Scenario Definitions”) will oblige to be able to deal with many models (in the broad sense) concerning different contexts and for different purposes. A similar situation will be encountered within the SAP use case, for example when considering the MDE artifacts produced and/or used during the definition and development phases of the SAP’s Product Innovation Lifecycle (or PIL, see section 2.2.2 of D1.1.a “Case Study Scenario Definitions”). In other words, a large number of various and varied modeling artifacts are involved in such processes. These artifacts are often characterized by the different kinds of relationship that may exist between them. Thus, there is in such cases (and in the MODELPLEX project in general) an important need for being able to deal with numerous and heterogeneous MDE artifacts as well as their related metadata. The aim of GMM is to provide an overall solution to this specific problem.

By starting studying deeper the Task 2.1 related requirements explicitly expressed by the WP1 partners, we see that the GMM solution presented within this deliverable brings answers to many of them. Thus, we now present the first results of this initial analysis.
First, we are going to consider the requirements which have been directly assigned to Task 2.1, and to show how the described GMM approach and tool provide solutions to them.

Id	Title/Stakeholders/Description	Answer
30	**Schemas evolvability (TIS)** Persistency function is in charge of managing model storage. Different kinds of element may be managed so the repository shall be able to define different schemas.	The GMM tool will be based on an extensible core metamodel of megamodel. So it is possible to defined different metamodels of megamodels for dealing with the different kinds of modeling artifacts.

Id	Title/Stakeholders/Description	Answer
	Persistency function shall be able to upgrade schemas and to provide "migration" means of existing data.	"Migration" feature may be implemented by a model-to-model transformation (from a megamodel to another one). The building of this transformation may be, in some cases, semi-automated.
32	**Models data distribution (TID, TIS)** Persistency function must be able to support data distribution (across platforms and across location).	We introduce the separation between the concepts of "model locator" and "model identifier" in order to be able to support data distribution. Thus, for each stored model it will be possible to access its location(s) that may be on various platforms.
34	**Persistency management function (TID, TIS)** • create, read, update, delete element • perform hierarchical update and delete • perform advance query (SQL or OCL like) • support ACID properties	1) The "create", "read", etc features will be part of the tool Core services. 2) We will have to discuss more precisely about what the partners exactly mean by "hierarchical updates & delete". 3) We will provide a megamodel query language which is a DSL for querying megamodels. 4) We have to study more in details the support we can provide for these four properties.
84	**Model libraries (TID, TIS)** Model libraries: need for facilities to define reference models of entities with known and documented properties that can be retrieved and used in an architecture.	The concept of library corresponds to what we call models "zoos" which are specific megamodels. We already provide within the Eclipse/GMT AM3 component several libraries of metamodels, transformations, etc. We may also provide in the future libraries of terminal models such as UML models for instance.
121	**Transparent model and code repository (WGO, TID)** A tool should have transparent access to required data, models and information produced from other tools.	As indicated by its name, the GMM tool is dedicated to model management so the access to the models and their metadata (various information such as the related source code file, etc) will be transparent.

Table 7 Analysis of the requirements which are assigned to Task 2.1

We are now going to take a look at the requirements that implicate Task 2.1 but that have not been directly assigned to it.

Id	Title/ Stakeholders/Description	Answer
28	**Model transformation function dependencies towards other model management functions (TIS)** Interoperate with: • persistency management function, • version control function, • multi-user function, • user profiling function.	In our sense, we consider model transformations as terminal models (that conform to model transformation metamodels), so they will benefit of the entire GMM tool provided features.
36	**Persistency function dependencies towards other model management functions (TIS)** Interoperate with: • version control function, • multi-user function • user profiling function.	This depends more on the repository available features (Task 2.4)
49	**User profile management dependencies towards other model management functions (TIS)** Interoperate with: • persistency management function • version control management function • multi user function • model to model transformations • etc.	This depends more on the repository available features (Task 2.4)
52	**Version control branching mechanism (TID, TIS)** • retrieve any version of a versionable element, • branching mechanism, • apply branching mechanism at versionable element granularity • apply branching hierarchically or not grant or not branching rights to users and to users profiles	This depends more on the repository available features (Task 2.4)
53	**Version Control diff mechanism (TID, TIS)** • provide diff feature for any versionable element,	This depends more on the repository available features (Task 2.4)

Id	Title/ Stakeholders/Description	Answer
	• perform diff between two different version of a versionable element, • perform hierarchical diff, • provide understandable diff information	
54	**Version control merge mechanism (TID, TIS)** • provide merge feature for any versionable element • merge at least two different version of a versionable element • perform hierarchical merge • provide understandable merge information • grant or not merge rights to users and to users profiles	This depends more on the repository available features (Task 2.4)
55	**Version Control baselining mechanism (TID, TIS)** • manage baselining mechanism • apply baselining mechanism at versionable element granularity • apply hierarchical baselining mechanism • mount/dismount any baseline • grant or not baselining rights to users and to user's profile	This depends more on the repository available features (Task 2.4)
56	**Version control user workspace management (TIS, WGO)** • manage different user workspace • filter user workspace data according to user profile	Each user can manage different megamodels. The elements referenced by these megamodels may be different. Thus, the elements of a given “workspace” may be referenced in a same megamodel or in several ones. We may also define different megamodels for the different user profiles (and thus allow selecting the data available to each user).
64	**Meta modelling functions relationships management (TIS)** • manage different Meta Model relationship types (dependencies, import, composition, refinement, inheritance)	It is possible to create megamodels referencing several metamodels and managing the different relationships that may exist between them. You may also define different extensions of the metamodel of megamodel for

Id	Title/ Stakeholders/Description	Answer
	• load, edit, modify, delete Meta Model relationship types • load, edit, modify, delete Meta Model relationships (instances of relationship types)	dealing with some more specific relationships that may exist between metamodels. This is true for metamodels but also for all models in general at every different level of abstraction.
66	**Meta modelling function dependencies towards other model management functions (TIS)** Interoperate with: • persistency management function, • version control function, • multi-user function, • user profiling function.	It depends on megamodel's metamodel extension (for example if the metamodel takes into account multi-user aspects).

Table 8 Analysis of the requirements which implicates Task 2.1

This section presented the first results of the initial analysis of the requirements that have been expressed by the WP1 partners and determined as being relevant (directly or indirectly) to Task 2.1.
Most of these requirements have been expressed by TIS, but TID and WGO also seem to be interested at some point in the global model management problem. Note that there are no requirements expressed by SAP which have been directly or indirectly mapped to Task 2.1. However, we strongly believe that the use of a global model management solution may be relevant for all partners, at least concerning some delimited parts of their respective MDE processes.

Within the coming months and in parallel with the GMM supporting tool V1 development, we will continue working on these requirements. We will also follow their consolidation in order to try to bring solutions to most of them with our GMM approach and corresponding prototype.

11. Initial Calendar: First Version of the GMM Prototype

The objective of this last section is to provide an initial calendar covering the coming months until the end of October 2007 and the release of the first version of the GMM supporting tool prototype.
This simple calendar indicates the different elements of the GMM supporting tool architecture (see section 7.3) that will be made available and when they will be available. It is presented in the following table which summarizes the next months' important development steps.

ID	*Provided Elements*	*Date*	*Q2 07*	*Q3 07*			*Q4 07*
			June	*July*	*Aug.*	*Sept.*	*Oct.*
1	Model handler generic interface	22/06/2007	◆				
2	EMF model handler implementation	29/06/2007		◆			
3	Repository generic interface	13/07/2007			◆		
4	Lightweight repository implementation	27/07/2007			◆		
5	Core metamodel of megamodel + metamodel extension mechanism	31/08/2007				◆	
6	GMM core services interface	14/09/2007				◆	
7	First extension (first implementation of basic GMM features)	05/10/2007					◆

Figure 11 Calendar for the release of the GMM prototype first version

Concerning the requirement management, we plan that the first version of the prototype (which is due in October 2007) will fulfil partially all the requirements directly assigned to Task 2.1 (see Table 7). This means that we will provide the GMM core components and a first extension for being able to create and manage several megamodels referencing various kinds of models in the broad sense.

Note that the plans and calendar for the development of the next versions of the prototype (from November 2007 to the end of the project) will be discussed in the document that will be provided with the prototype first version (i.e. deliverable D2.1.b "Model Management Supporting Tool").

12. Conclusion

In its simplest and most common form, MDE is used for applications such as UML to Java transformations. Such operations can be handled by simple model to text transformations. However, many real life MDE applications are often much more complex. For example they may contain long chains of transformation going from XML-based data to programs (e.g. grammar based data). These complex situations may involve an important number of MDE artifacts like terminal models, metamodels, and transformations. Furthermore these artifacts are sometimes obtained from large libraries of open–source components and adapted to the work at hand. As a consequence there is need to create and maintain large volume of MDE artifacts with metadata associated to these artifacts and complex semantic relations holding between them. This report has discussed the need for this and provided some hints on how to organize these collections of related data. It has also described an initial implementation (mainly supported by the AM3 Eclipse prototype) which could allow experimenting with such situations. The AM3 component contains for example a significant number of transformations, terminal models, or metamodels.

This report has set up the stage for the work that is going on now to build an extended MODELPLEX prototype (V1), due for month 14 (see section 11). Starting from the initial V0 prototype (i.e. the described AM3 component whose development has been started before MODELPLEX), the prototype (that will be developed within the context of the project) will provide the different features that have been discussed in this report.

13. References

[1] Boehm, B. Sherlis, W. **MegaProgramming**, in Proc. of the DARPA Software Technology Conference, (Arlington, Va, April 1992.).

[2] Bézivin, J., Jouault, F., and Valduriez, P., **On the Need for Megamodels**. In: Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications.

[3] Bézivin, J., Jouault, F., **KM3: a DSL for Metamodel Specification**. In: Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, Bologna, Italy.

[4] A Object and Reference Model Subcommittee (ORMSC) of the OMG Architecture Board: **A Proposal for an MDA Foundation Model**, white paper OMG-ORMSC/05-08-01, http://www.omg.org/cgi-bin/doc?ormsc/05-08-01

[5] 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.

[6] **ATL "ATLAS Transformation Language"** official site (**Eclipse/M2M** component): http://www.eclipse.org/m2m/atl/

[7] Bézivin, J., Jouault, F., Rosenthal, P., and Valduriez, P., **Modeling in the Large and Modeling in the Small**. In: Proceedings of the European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, LNCS 3599, edited by Uwe Aßmann, Mehmet Aksit, Arend Rensink. Springer-Verlag GmbH, pages 33-46.

[8] **AM3 "ATLAS MegaModel Management"** official site (**Eclipse/GMT** component): http://www.eclipse.org/gmt/am3/

[9] **AMW "ATLAS Model Weaver"** official site (**Eclipse/GMT** component): http://www.eclipse.org/gmt/amw/

[10] **Teneo Project** official site (**Eclipse/EMFT** project) http://www.eclipse.org/emft/projects/teneo/#teneo

[11] **Adaptive Repository** official site: http://www.adaptive.com/index.html

[12] Allilaire, F, Bézivin, J, Brunelière, H, and Jouault, F, **Global Model Management In Eclipse GMT/AM3**. In: Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference, Nantes, France. 2006.

[13] The MODELWARE Project: http://www.modelware-ist.org