Workpackage 2: Framework Building

Deliverable D2.11.b:

“Global Model Management Supporting Tool”
**Contract Number:** 034081  
**Project Acronym:** MODELPLEX  
**Title:** Modeling solution for comPLEX software systems

**Deliverable N°:** D2.11.b (third version of former deliverable D2.1.b)  
**Due Date:** 10/2009  
**Delivery Date:** 10/2009

**Short Description:**
This recently renamed report describes the version 2 of a global model management prototype that has been developed using and extending the existing INRIA toolset (available from the Eclipse-GMT AM3 project). It extends the second version of deliverable D2.1.b describing the v1 of this prototype.

**Lead Partner:** INRIA  
**Made available to:** PU (Public)

<table>
<thead>
<tr>
<th>Rev</th>
<th>Date</th>
<th>Author</th>
<th>Checked by</th>
<th>Internal Approval</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>07/01/08</td>
<td>Draft – Hugo Bruneliere, INRIA</td>
<td></td>
<td></td>
<td>Initial version</td>
</tr>
<tr>
<td>0.2</td>
<td>10/01/08</td>
<td>Hugo Bruneliere, INRIA</td>
<td>Mikael Barbero, INRIA</td>
<td></td>
<td>First complete version</td>
</tr>
<tr>
<td>0.3</td>
<td>24/01/08</td>
<td>Hugo Bruneliere, INRIA</td>
<td></td>
<td></td>
<td>Adding of section 7.3</td>
</tr>
</tbody>
</table>
| 0.4 | 05/02/08   | Contributions integrated from Hugo Bruneliere, INRIA | 1st Internal Reviewer: Brian Pickering, IBM  
2nd Internal Reviewer: Goran Olsen, SINTEF |     | Reviewers comments integration |
| 1.0 | 06/02/08   | Final deliverable v0 | QB          | EB                |                              |
| 1.1 | 05/09/08   | Guillaume Doux, INRIA | Hugo Bruneliere, INRIA |       | New sections 9 & 10          |
| 1.2 | 12/09/08   | Hugo Bruneliere, INRIA |            |                   | Various updates on the whole document |
| 1.3 | 24/09/08   | Contributions integrated from Hugo Bruneliere, INRIA | 1st Internal Reviewer: Brian Pickering, IBM  
2nd Internal Reviewer: Goran Olsen, SINTEF |     | Reviewers comments integration |
<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
<th>Reviewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>17/10/08</td>
<td>Contributions integrated from Hugo Bruneliere, INRIA</td>
<td>QB comments integration</td>
</tr>
<tr>
<td>2.0</td>
<td>21/10/08</td>
<td>Final deliverable v1</td>
<td>QB</td>
</tr>
<tr>
<td>2.1</td>
<td>11/09/09</td>
<td>Hugo Bruneliere, INRIA</td>
<td>Various upgrades all along the document Refactoring of section 7 Adding of new sections 11 &amp; 12</td>
</tr>
<tr>
<td>2.2</td>
<td>22/09/09</td>
<td>Hugo Bruneliere, INRIA</td>
<td>Various corrections and formatting updates all along the new version</td>
</tr>
<tr>
<td>2.3</td>
<td>16/10/09</td>
<td>Contributions integrated from Hugo Bruneliere, INRIA</td>
<td>1st Internal Reviewer: Brian Pickering, IBM 2nd Internal Reviewer: Goran Olsen, SINTEF Reviewers comments integration</td>
</tr>
<tr>
<td>3.0</td>
<td>19/10/09</td>
<td>Final deliverable v2</td>
<td>QB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EB</td>
</tr>
</tbody>
</table>
Table of Content

1. Executive Summary.................................................................6
2. Acronyms and Terminology ....................................................6
3. Introduction ............................................................................6
4. The Eclipse-GMT AM3 Project ...................................................7
5. Prototype Architecture ............................................................9
   5.1. Plug-ins ...........................................................................9
   5.2. Core Metamodel ...............................................................11
   5.3. Extension Mechanism .......................................................12
6. Core Environment .....................................................................13
   6.1. Megamodel Navigator ......................................................13
   6.2. Generic Editor Pages .......................................................15
      6.2.1. Entity Editor Page .....................................................15
      6.2.2. Relationship Editor Page ..........................................16
7. Global Model Management (GMM) Extension ..............................17
   7.1. GMM Metamodel Extension ..............................................17
   7.2. GMM-Specific Features ....................................................18
      7.2.1. Transformation Execution Extension Point ....................18
      7.2.2. Automated Generation of Traceability Links during Model Transformation ....19
   7.3. GMM-Specific Editor Pages ..............................................19
      7.3.1. Model Properties Page ............................................19
      7.3.2. Model Transformation Chain Page .............................20
      7.3.3. Transformation Execution Page .................................20
8. GMM4ATL Extension ...............................................................23
   8.1. GMM4ATL Metamodel Extension .......................................23
   8.2. ATL-Specific Features ....................................................24
      8.2.1. ATL Project Megamodel Discovery .............................24
      8.2.2. ATL Transformation Executor ....................................25
   8.3. ATL-Specific Editor Pages ..............................................26
      8.3.1. ATL Module Editor Page ..........................................26
      8.3.2. ATL Transformation Editor Page ...............................26
      8.3.3. ATL Transformation Launch Editor Page .....................28
      8.3.4. ATL Transformation Chain Configuration Editor Page ....28
   8.4. Demonstration on an SAP Material-Based Use Case .................30
9. GMM4TCS Extension ...............................................................32
   9.1. GMM4TCS Metamodel Extension .....................................32
   9.2. TCS-Specific Features ....................................................33
      9.2.1. TCS Injection Executor ...........................................33
      9.2.2. TCS Extraction Executor .........................................33
   9.3. TCS-Specific Editor Pages ..............................................33
      9.3.1. TCS Model Editor Page ...........................................33
      9.3.2. TCS Projector Editor Page .......................................34
10. GMM4CompositeTransformations (GMM4CT) Extension ..............35
    10.1. GMM4CT Metamodel Extension ....................................35
    10.2. CT-Specific Features ..................................................36
       10.2.1. Composite Transformation Executor .......................36
       10.2.2. Connector Execution Extension Point .......................37
       10.2.3. Simple Connector Executor ....................................37
    10.3. CT-Specific Editor Page for Composite Transformation ........37
    10.4. Demonstration Using a SAP Use Case ..............................39
11. AM3Navigability Extension ....................................................41
    11.1. AM3 Navigability Features ..........................................41
    11.2. Navigability-Specific Views .........................................42
List of Figures

Figure 1 Eclipse-GMT AM3 home page [4] ................................................................. 7
Figure 2 AM3 Plug-ins from the Eclipse SVN [7] ....................................................... 8
Figure 3 Prototype Plug-in Architecture ................................................................. 9
Figure 4 AM3Core: The Core Metamodel of Megamodel ....................................... 11
Figure 5 Using the Megamodel Navigator for Browsing the Available Metamodels ... 13
Figure 6 Using the Megamodel Navigator for Browsing the Available ATL Transformations ... 14
Figure 7 Using the Megamodel Navigator for Registering a New Terminal Model ... 14
Figure 8 Entity Editor Generic Page ...................................................................... 15
Figure 9 Relationship Editor Generic Page ............................................................. 16
Figure 10 GMM: The Global Model Management Specific Extension of AM3Core ... 17
Figure 11 Setting the Reference Model of a Model Using the Model Properties Page ... 19
Figure 12 Defining a Model Transformation Chain Specification Using the Appropriate Page ... 20
Figure 13 Executing a Transformation Using the Generic Execution Page .......... 21
Figure 14 Entering the Information on a Transformation Output Entity to Be Created ... 21
Figure 15 GMM4ATL: The ATL-Specific Extension of GMM .................................. 24
Figure 16 “Add Project to Metamodel” ATL Contextual Action ................................. 24
Figure 17 “Select Metamodels of Terminal Models” ATL Megamodel Discovery Dialogue ... 25
Figure 18 Defining ATL Parameters Using the ATL Module Specific Page ............ 26
Figure 19 Defining Metamodel Bindings Using the ATL Transformation Specific Page ... 27
Figure 20 Megamodel Element Selection Dialogue: ATL Module Example ............ 27
Figure 21 Defining Model Bindings Using the ATL Transformation Launch Specific Page ... 28
Figure 22 Defining an ATL Transformation Chain Configuration Using the Appropriate Page ... 29
Figure 23 SAP Use Case for the GMM4ATL Demonstration: Inputs and Output ... 30
Figure 24 SAP Use Case for the GMM4ATL Demonstration: Chain of ATL Transformations ... 31
Figure 25 The TCS-Specific Extension of GMM ....................................................... 32
Figure 26 The TCS Model Specific Editor Page ....................................................... 33
Figure 27 TCS Projection Editor Page (TCS Extraction Sample) ......................... 34
Figure 28 The Composite Transformation-Specific Extension of GMM .................... 36
Figure 29 Composing Transformations Using The Composite Transformation Editor Page ... 38
Figure 30 Parameters-Specific Selection Dialogue .............................................. 39
Figure 31 Transformation Chain Sample: GMM4CT Demo .................................... 40
Figure 32 Abstract view of our generic inter-model navigability approach ................ 41
Figure 33 Generic Model-Level Navigability View ............................................... 42
Figure 34 Generic Model Element-Level Navigability View .................................. 43
Figure 35 AM34LinuxPackages: The Linux Distribution Specific Extension of AM3Core ... 45
Figure 36 Overview of the System Cartography Specific Features ......................... 46
Figure 37 Sample Rendered Visualization of a Debian Distribution ....................... 47
1. Executive Summary

Global Model Management (GMM) aims to provide support for *modeling in the large* [1], i.e. managing global modeling resources in the field of MDE-oriented software development. These global resources (cf. [2]) are usually heterogeneous and distributed as they often come from different modeling tools, like those of the MDE frameworks mentioned in [3]. Thus, to use them without unintentionally increasing the complexity of MDE, we need to invent new ways of creating, storing, viewing, accessing, modifying and using the metadata associated with all these global modeling entities. This is the reason the concept of *megamodel* was introduced [5]. The goal of this prototyping work is to provide straightforward access to the MDE developers so that they may deal with complex problems using simple *Eclipse™* interfaces.

This deliverable focuses mainly on introducing and describing in detail the tool supporting GMM. The previous version of this document formerly named D2.1.b, delivered in month 26 (October 2008), presented the first version of the tool: *AM3 prototype V1* which is part of the current version of the MODELPLEX Platform. This final version of the document describes the second version of this same tool: *AM3 prototype V2*. This version is the last of the three planned for release within the context of *MODELPLEX* (V0 was due in January 2008 and V1 was due in October 2008). This prototype is currently available from the MODELPLEX SVN and will be part of the next MODELPLEX Platform release. It is also publically published as the latest version of the *Eclipse-GMT AM3* (AtlanMod MegaModel Management) project from its website [4].

2. Acronyms and Terminology

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM3</td>
<td>AtlanMod MegaModel Management</td>
</tr>
<tr>
<td>AMW</td>
<td>AtlanMod Model Weaver</td>
</tr>
<tr>
<td>ATL</td>
<td>AtlanMod Transformation Language</td>
</tr>
<tr>
<td>CVS</td>
<td>Concurrent Versions System</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain-Specific Language</td>
</tr>
<tr>
<td>EMF</td>
<td>Eclipse Modeling Framework</td>
</tr>
<tr>
<td>GMM</td>
<td>Global Model Management</td>
</tr>
<tr>
<td>GMT</td>
<td>Generative Modeling Technologies</td>
</tr>
<tr>
<td>HOT</td>
<td>Higher Order Transformations</td>
</tr>
<tr>
<td>M2M</td>
<td>Model-to-Model (Transformation)</td>
</tr>
<tr>
<td>M2T</td>
<td>Model-to-Text (Transformation)</td>
</tr>
<tr>
<td>MDE</td>
<td>Model Driven Engineering</td>
</tr>
<tr>
<td>SVN</td>
<td>Subversion</td>
</tr>
<tr>
<td>TCS</td>
<td>Textual Concrete Syntax</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>UTI</td>
<td>Uniform Type Identifier</td>
</tr>
</tbody>
</table>

3. Introduction

During the initial months of the *MODELPLEX* project, extensive research was carried out centered on Global Model Management problems and on how best to provide concrete solutions to them. The results of this investigation work, described in [11], concern theoretical and conceptual aspects but also more
practical ones. This work is being carried out within the context of Task 2.11 (formerly named Task 2.1) and continues to be developed; it is now made available in an upgraded form by the implementation of the second version of a prototype.

To begin with, the *Eclipse-GMT AM3* project, from which the prototype can be accessed, will be briefly presented. The architecture of the prototype will be detailed by emphasizing its two main parts: the core metamodel of megamodel and the extension mechanism. Then the prototype core environment will be described in the following section. More details of the various extensions which have been developed within MODELPLEX will be provided. Finally, some recent exploratory works using the prototype will be introduced, before conclusions are drawn.

## 4. The Eclipse-GMT AM3 Project

The current prototype supporting Global Model Management (referenced as v2 within MODELPLEX) is in fact the latest and “built from scratch” version of an already existing *Eclipse* project known as AM3 (AtlanMod MegaModel Management). This project is part of the GMT (Generative Modeling Technologies) incubation project which itself is part of the top-level *Eclipse/Modeling* project.

Figure 1 shows the *Eclipse-GMT AM3* project's home page:

![Eclipse-GMT AM3 home page](image)

The *Eclipse-GMT AM3* project already contains plug-ins offering facilities which
may be particularly useful when developing with ATL (such as model injection/extraction features via a specific navigator, ATL-specific Apache Ant tasks, etc). These older plug-ins have been kept as utilities provided by the AM3 project (cf. the AM3 SVN in a folder named “deprecated”). But from now on, they are not considered part of the new AM3 tool.

As an Eclipse project, the GMM prototype is fully open-source and thus freely available, via the AM3 project, from the Eclipse website and the Eclipse download servers.

For the time being, there is no binary version of the plug-ins available directly from Eclipse.org, but an update site will be put online in the coming months. Notwithstanding that provision of the update site, v2 of the plug-ins can already be found on the MODELPLEX SVN [6]. Note that the source versions of the v2 of the plug-ins can also be downloaded directly from the Eclipse SVN [7] using the location given in Figure 2:

![Figure 2 AM3 Plug-ins from the Eclipse SVN](image)

As shown in Figure 2, v2 of the plug-ins is available from the “plugins/trunk” directory stored in the AM3-dedicated Modeling Project SVN repository. This figure presents the core plug-ins as well as all the extensions described in the rest of the document.

In order to get the source files, you can follow the process detailed in the corresponding Wiki page [8].
5. Prototype Architecture

This section presents the concrete architecture of the prototype. It describes the internal structure of the prototype in terms of plug-ins, details its core metamodel and explains the extension mechanism provided.

5.1. Plug-ins

As an Eclipse component, the prototype is implemented by a set of Eclipse plug-ins, each of them having a specific role as well as in some cases dependencies on other components.
Within Figure 3, the different plug-ins which make up the prototype are presented with their main interdependencies.

The core plug-ins are the following:

- “am3” which implements the AM3 plug-in class;
- “am3.core” which provides all the generic interfaces for modeling, storage and projections;
- “am3.modelhandler.emf” which allows model handling using EMF;
- “am3.platform.runtime” which dynamically initializes the metamodel of megamodel according to the given extensions (note that it also contains the AM3Core metamodel, see section 5.2);
- “am3.platform.core” which allows the different megamodels to be handled (and more specifically the one currently loaded to be accessed);
- “am3.platform.ui” which implements all the AM3 generic UI parts (more details are given in section 6).

Several extension plug-ins are already available:

- “GlobalModelManagement” directly extends AM3Core. It specifies the GMM framework and provides corresponding basic facilities, as described in section 7;
- “GMM4ASM” extends GlobalModelManagement. It adds some concepts related to the ATL virtual machine;
- “GMM4AMW” extends GMM4ASM. It adds the model weaving-specific concepts (note that this will not be detailed here);
- “GMM4ATL” extends GMM4ASM. It adds the ATL-specific concepts and related features (see section 8);
- “GMM4TCS” extends GlobalModelManagement. It adds the TCS-specific concepts and related features (see section 9);
- “GMM4CompositeTransformations” extends GlobalModelManagement. It adds the concepts and facilities required for dealing with composite transformations (see section 10);
- “AM3Navigability” directly uses AM3Core. It provides generic inter-model navigation capabilities (see section 11);
- “AM34LinuxPackages” directly extends AM3Core. It adds the specific concepts which allow representing Linux packages and their interdependencies (see section 12.1).

Moreover, three utility plug-ins have already been made available. Two of them implement basic model-repository solutions which could be of use beyond the immediate context of the tool (note that none of them is actually used by the current version of the prototype). The remaining plug-in provides extraction facilities for Ant script models and Eclipse launch configuration models (i.e. “launchers”); it is already used (and required) by the GMM4ATL extension plug-in.

The prototype architecture, based on a set of AM3 core Eclipse plug-ins and possible other extension Eclipse plug-ins, is fully compatible with the MODELPLEX Platform guidelines proposed within [9]. Indeed, all the developed AM3 plug-ins are compliant with Eclipse 3.4 & EMF 2.4 and thus can be directly
integrated into the *Eclipse*-based modeling tools workbench, as it is described in the overview of the MODELPLEX Platform architecture from [9].

### 5.2. Core Metamodel

The core metamodel of megamodel is named **AM3Core**. It specifies the minimal sufficient set of concepts in order to be able to deal with any possible extensions users may wish to define. This core metamodel is presented in Figure 4 (with abstract concepts in light grey):

![AM3Core: The Core Metamodel of Megamodel](image)

A *megamodel* is basically defined as a set of *elements*, each of them possibly annotated with *metadata* (key/value pairs). Two main kinds of *element* are considered:

- **Identified elements** which are characterized by a unique *identifier* and which may be linked to other identified elements by different *relationships*;
- **Located elements** which are characterized by a specific *locator* indicating their physical location and by several type definitions (note that the *UTI*
and the TypeTagSpecification concepts allow the creation of such definitions).

There are two main sorts of IdentifiedElement which are the basis of a megamodel:

- **Entities** which are also LocatedElements (note that there is a special kind of entity called Container which allows entities containing several other entities to be created); and
- **Relationships** which associate different IdentifiedElements via a link (note that there is a special type of relationship called DirectedRelationship which explicitly distinguishes between source and target elements).

Two other kinds of IdentifiedElement are defined in order to allow special sets of elements to be specified:

- **Groups** which are simple sets of elements; and
- **Chains** which are sequences of directed relationships.

Note that most of the concepts in this metamodel are defined as being abstract. This means that they have to be extended explicitly by the AM3Core extensions.

### 5.3. Extension Mechanism

The prototype provides a dedicated extension mechanism which offers users the opportunity to define extensions to the tool for managing their own elements for their own specific needs. Thus, users can determine the possible actions that may be performed with/on a megamodel conforming to a given metamodel of megamodel.

Two extension points are indeed currently defined, each of them playing a specific role in the extension definition:

- **org.eclipse.gmt.am3.platform.am3Extension** simply allows extensions of the metamodel of megamodel to be specified by simply providing the new concepts in Ecore format and by indicating the metamodel of megamodel which is extended;
- **org.eclipse.gmt.am3.platform.ui.pages** allows specific editor pages for the concepts added in the extension to be defined by explicitly identifying the concept it has to be associated with for each page. In this way, the new pages are automatically made available when editing megamodel elements conforming to these concepts. For examples of how to implement these additional editor pages, refer to the source code of the pages (from the extension plug-ins) described in sections 7.3, 8.3, 9.3 and 10.3.

In addition to these two existing extension points, the extension mechanism may well be enhanced in the future with further extension points, for instance to be able to deal with specific kinds of locators, identifiers, and so forth.
6. Core Environment

This section describes the core generic parts added by the prototype to the basic Eclipse IDE. It provides screenshots and corresponding explanations about the Megamodel Navigator view and the generic editor pages linked to the main concepts of the AM3Core metamodel.

6.1. Megamodel Navigator

The Megamodel Navigator is an Eclipse view which is the core IDE part of the prototype. It is composed of two distinct (but nevertheless linked) parts, each of them playing a specific navigation role.

The one to the left allows navigation across the different elements of the metamodel of megamodel (representing the inheritance tree) considering all the dynamically loaded metamodel extensions.

The one to the right permits users to browse the currently loaded megamodelf (which conforms to the displayed metamodel of megamodel) and thus to display its different elements.

The two following screenshots show examples of the Megamodel Navigator usage in order to display the metamodels (Figure 5) and the ATL transformations (Figure 6) registered in the current megamodel.

![Megamodel Navigator](image)

Figure 5 Using the Megamodel Navigator for Browsing the Available Metamodels
Figure 6 Using the Megamodel Navigator for Browsing the Available ATL Transformations

In practice, when you click on an element of the metamodel of megamodel in the left hand tree view, you automatically see in the right hand view all the megamodel elements which conform to the selected metamodel (of megamodel) element. Then, each megamodel element can be edited by double-clicking on it in the right hand view.

Figure 7 Using the Megamodel Navigator for Registering a New Terminal Model
Further, using this navigator, it is possible to register a new element in the megamodel by right-clicking on the metamodel element this megamodel element must conform to (as shown in Figure 7). A new element of the selected type is then created in the megamodel. This new megamodel element may then be edited in order to complete the required fields.

6.2. Generic Editor Pages
For each megamodel element which is (by inheritance) of type Entity or Relationship, we provide a generic editor page displaying fields corresponding to the information defined in the AM3Core core metamodel of megamodel. These generic editor pages cannot be modified but the generic editor may of course be enhanced by adding extension-specific pages (such as those described in section 8.2 for the GMM4ATL extension).

Note that more advanced editor features, such as the automatic update of one editor when modifying information which may impact it in another, have yet to be implemented in the V1 prototype.

6.2.1. Entity Editor Page
A generic editor page has been defined to be displayed for each concept directly or indirectly extending the Entity one. Figure 8 is an example of such a page showing an example of a metamodel registered into the current megamodel:

![Figure 8 Entity Editor Generic Page](image)

Different information items are shown for the edited entity which can be specified using this generic editor page:
The Identifier part allows the identifier type which is under consideration to be selected and the string corresponding to the entity’s identifier to be entered;

In a similar way, the Locator part allows the locator type under consideration to be selected and the string indicating the entity’s locator to be specified;

The sourceOf part allows the DirectedRelationships that this entity is a source of to be selected;

The targetOf part allows the DirectedRelationships that this entity is a target of to be selected;

The relatedTo part permits the Relationships associated with this entity to be chosen.

6.2.2. Relationship Editor Page

A generic editor page has also been defined to be displayed for each concept directly or indirectly extending the Relationship one. Figure 9 is an example of such a page relating to an ATL transformation registered into the current megamodel:

![Figure 9 Relationship Editor Generic Page](image)

In addition to the relationship identifier (the Identifier part already described in section 6.2.1), the elements linked by this relationship may also be specified in the linkedElements part of the generic editor page as shown.
7. Global Model Management (GMM) Extension

This section describes the current state of the generic extension to Global Model Management. This extension, comprising a metamodel extension and additional editor pages, has been developed using the AM3 core APIs and the extension mechanism provided.

7.1. GMM Metamodel Extension

The Global Model Management metamodel (of megamodel) extension provides a generic MDE framework. Its aim is to allow any MDE artefacts and their possible relationships to be represented in a megamodel. Thus, this extension describes an abstract conceptual framework fully independent from any technology implementing MDE principles [9].

![Figure 10 GMM: The Global Model Management Specific Extension of AM3Core](image_url)
Within Global Model Management, models are first-class entities. There are two main kinds of models: terminal models and reference models, each of them conforming to a reference model (i.e. either a metamodel or a metametamodel). According to the MDE vision, a terminal model (M1-level) conforms to a metamodel (M2-level), a metamodel conforms to a metametamodel (M3-level) and a metametamodel conforms to itself. Transformation models, weaving models or megamodels are specific kinds of terminal models.

There are different kinds of relationships (see AM3Core definition in section 5.2): a model weaving is a simple relationship whereas a transformation is a directed relationship (i.e. with a distinction between source and target). Within this GMM extension, the focus is on all the types of transformation which may exist in MDE (and thus may be represented in a megamodel): external-to-external transformation (e.g. source code to source code), external-to-model transformation and model-to-model transformation (i.e. injection), model-to-model transformation and model-to-external transformation (i.e. extraction). All these relationship types act as abstract specifications of transformations. They may be linked with transformation models which provide concrete implementations of these specifications. They may also be associated with transformation records which represent real executions (and their results) of the specified transformations.

As an example, a sample megamodel can be found from section 4.2 of [23]. Considering ATL as the used model-to-model transformation technology, it shows a model-to-model transformation (KM32ATLCopier) specification, a transformation model (KM32ATLCopier-Module) implementing it and an associated transformation record (SQLCopierGeneration).

### 7.2. GMM-Specific Features

The GMM extension offers a set of features which are relevant in any Global Model Management case, i.e. in the context of GMM but also of any direct or indirect extension of GMM.

#### 7.2.1. Transformation Execution Extension Point

The generic execution of single transformations is supported straight from the GMM extension. To this aim, an extension point has been defined:

- org.eclipse.gmt.am3.platform.extension.globalmodelmanagement.
  TransformationExecutor. (see Figure 3)

For the transformations to be correctly launched and executed, it must be used for each transformation subtype (e.g. ATLTransformation). In order to do this, for each new transformation type, the ITransformationExecutor interface needs to be implemented. This interface contains two different methods to implement. The first one is the method createEntityClassFor which must create and return a transformation parameter (to be added in the megamodel). The second one is the execute method which takes the transformation and a map containing the parameters as inputs and must return a transformation record. Ultimately, the transformation record must store the trace of the execution. If a problem occurs during the transformation, this method must return null.
7.2.2. Automated Generation of Traceability Links during Model Transformation

When dealing with traceability in the context of model transformation, the first issue is to build the traceability links effectively. To this end, there are two options: 1) this can be done manually or 2) this can be performed automatically and systematically. In the second case being the ideal case, some automated traceability support is provided by the tool within the context of model transformation.

As detailed in section 5 of deliverable D3.2.d “Global Model Management Traceability Extension”, the approach which has been used consists in two successive steps:

1. The automated generation of an augmented transformation (providing the traceability support capability) from the original one by using a Higher-Order (model) Transformation or HOT;
2. The launching of this augmented transformation which produces, in addition to the original outputs, the traceability information.

Note that this mechanism has already been implemented for ATL transformations, but other transformation technologies may be supported in the future by applying a similar approach.

7.3. GMM-Specific Editor Pages

In order to be able to deal with the newly defined concepts presented above and to provide some GMM-specific features, several dedicated editor pages have been implemented. Note that these pages are automatically made available in the editor when editing a megamodel element of the corresponding type (a benefit provided by the extension mechanism described in section 5.3).

7.3.1. Model Properties Page

A specific editor page has been defined to be displayed for editing each Model registered into the megamodel. Figure 11 shows an example of how this page is used in connection with the “TraceSamples-Excel” model:

![Figure 11 Setting the Reference Model of a Model Using the Model Properties Page](image)

This page allows the reference model of the edited model to be specified. If the edited model is a terminal model, the Browse button allows one to be chosen from among all the registered metamodels. If it is a metamodel or a metametamodel, it allows the reference model to be selected from all the
registered metametamodels.

### 7.3.2. Model Transformation Chain Page

A specific editor page has been defined to be displayed for editing the specification of model transformation chain in the megamodel. Figure 12 shows an example of how this page is used in connection with the “XML2Metrics” model transformation chain:

![Model Transformation Chain Page](image)

Figure 12 Defining a Model Transformation Chain Specification Using the Appropriate Page

This page allows the single transformation specifications to be identified among all those registered and to select the ones which will define the transformation chain specification using the Add and Remove buttons. It also allows them to be reorganized (i.e. their order to be changed) in the chain using the Up and Down buttons.

### 7.3.3. Transformation Execution Page

A specific editor page has been defined for each concept directly or indirectly extending the Transformation page. Figure 13 is an example of such a page with an example ATL Transformation already registered in the current megamodel.

The “Transformation Info” section of this page displays all the parameters of the transformation including their direction (either “in”, “out” or both “in” and “out”) and their fully qualified type.

The “Execute this Transformation” section allows the transformation’s input and output entities, i.e. models or metamodels in the case of model-to-model transformation as with ATL, to be set as parameters for its effective launch. If an output of the transformation does not exist and has not been previously registered in the megamodel, the “Create” button allows the required information to be entered on this output entity so that it can then be correctly registered in the megamodel (as shown on Figure 14). The “Execute” button launches the
appropriate execution of the transformation using the corresponding data from the megamodel, in addition to the information set from the page. It calls the generic `ITransformationExecutor` interface, provided by the Transformation Execution extension point (section 7.2.1), independently of the kind of transformation (e.g. model-to-model, model-to-text, etc) and of the underlying technology which actually implements it.

Figure 13 Executing a Transformation Using the Generic Execution Page

Figure 14 Entering the Information on a Transformation Output Entity to Be Created
The “Previous Executions” section displays the history of the past executions for the given transformation by indicating both the inputs and outputs actually used. It also provides the status of each of these executions (either PASSED or FAILED) and allows their records to be removed from the megamodel if needed. Moreover, these records can be used in order to re-execute the transformation by maintaining some of the already specified inputs and outputs. In that case, the “Overwrite” checkbox is used to determine whether the existing outputs will be overridden or new similar outputs created with automatically generated identifier and locator.

Finally, the “Trace Management” section provides a single “Automated Trace Generation” checkbox which has to be selected if automated traceability support (section 7.2.2) needs to be activated during the next execution of the given transformation.
8. GMM4ATL Extension

This section describes the current state of the Global Model Management extension dedicated to ATL. This extension, composed of a metamodel extension and of additional editor pages, has been developed using the AM3 core APIs and the provided extension mechanism.

8.1. GMM4ATL Metamodel Extension

The GMM4ATL metamodel (of megamodel) extension adds a set of ATL-specific concepts to the global model management generic ones (Figure 15).

In this way, the ATL model type of transformation model is defined. There are two distinct kinds of ATL model mainly corresponding to the two different kinds of ATL programs (all of them conforming to the ATL metamodel):

- ATL module which represents an ATL module containing a set of transformation rules and possibly helpers. An ATL module is characterized by input and output parameters, each of them specifying the formal model name and corresponding formal reference model name used in the ATL module; and
- ATL library which represents a set of helpers designed to be used by other ATL models.

Note that each ATL model has a name and may reference a certain number of ATL libraries by name.

The ATL transformation type of model-to-model transformation relationship is also defined. An ATL transformation relationship has to be linked to an ATL module implementing it (using the transformationModel feature of the Model Transformation concept, see section 7). Thus, its main goal is to allow the source and target reference model names (specified in the ATL module parameters) to be bound with reference models registered in the megamodel. It represents a specification of an ATL transformation.

The ATL transformation launch type of directed relationship is also added by this extension. An ATL transformation launch refers to an ATL transformation relationship and in addition specifies the bindings between the source/target model names and models registered in the megamodel. It represents a real execution of an ATL transformation on given models.

Finally, the ATL transformation chain configuration kind of directed relationship chain is defined in order to allow complex chains of ATL transformation launches to be registered in the megamodel. It represents the real execution of an ATL transformation chain on given models.
8.2. ATL-Specific Features

The GMM4ATL extension provides some specific features which are relevant only within the context of ATL. The facilities detailed below may be particularly useful when megamodeling ATL projects and transformations.

8.2.1. ATL Project Megamodel Discovery

The GMM4ATL extension provides a navigator contextual action which allows an empty megamodel to be populated with the metadata retrieved from an ATL project stored within the current workspace, as shown on Figure 16.

![Figure 16 “Add Project to Megamodel” ATL Contextual Action](image)
This action is available only from an *Eclipse* project of type ATL. It parses the resources contained in the project using the following rules:

- All files with the ".ecore" extension are registered as metamodels (which conform to the Ecore metametamodel, thus automatically registered);
- All files with the ".atl" extension are registered as ATL modules (which conform to the ATL metamodel, thus automatically registered). The header of each ATL file is read and the corresponding ATL parameters are initialized accordingly;
- All files with the ".uml" extension are registered as terminal models which conform to the UML2 metamodel (thus automatically registered);
- All files with the ".xmi" extension are registered as terminal models. A dedicated dialogue allows the user to specify the metamodel for each terminal model by selecting it from those already registered (as shown on Figure 17);
- All ATL launch configuration files are parsed. ATL transformations and ATL transformation launches are registered and initialized using the information retrieved.

At the end of the process, the megamodel has been completely filled with the metadata associated with the ATL project: all the information about the metamodels, the terminal models and the ATL transformations is thus stored in the current megamodel.

![Figure 17 “Select Metamodels of Terminal Models” ATL Megamodel Discovery Dialogue](image)

Note that, for the megamodel discovery to work properly, the ATL project concerned must adhere strictly to the conditions of the rules set out above.

### 8.2.2. ATL Transformation Executor

The ATL Transformation Executor implements the *ITransformationExecutor* interface provided by the *Transformation Execution* extension point (section 7.2). To this end, the API from the *org.eclipse.m2m.engine* package of the ATL plug-ins is used to load, transform and save the models corresponding to the given parameters. Thus, ATL transformations correctly registered into the megamodel can be transparently executed from the transformation *Executor* generic page (section 7.3.3).
8.3. ATL-Specific Editor Pages

In order to be able to deal with the newly defined concepts presented above and to provide some additional ATL-specific features, several GMM4ATL dedicated editor pages have been implemented. Note that these pages automatically become available in the editor when editing a megamodel element of the corresponding type (a benefit provided by the extension mechanism described in section 5.3).

8.3.1. ATL Module Editor Page

A specific editor page has been defined for editing each ATL module registered into the megamodel. Figure 18 shows an example of how this page is used in connection with the “Trace2PerformanceMetrics” ATL module:

![Figure 18: Defining ATL Parameters Using the ATL Module Specific Page](image)

This page allows the input and output parameters of the module to be specified, each of them being a Forml Model Name/Formal Reference Model Name pair. These names are used in the module in order to point to the input/output models and the reference models they conform to. They will also be used later as keys in the explicit bindings to the models and reference models.

8.3.2. ATL Transformation Editor Page

A specific editor page has been defined to edit each ATL transformation registered into the megamodel. Figure 19 shows an example of this page based on the “Trace2PerformanceMetrics” ATL transformation:
This page allows the ATL module implementing the ATL transformation to be selected. For this, as with any megamodel element selection, the generic dialogue shown in Figure 20 is used. According to the selected ATL module, the corresponding bindings for all input and output reference models can be specified (i.e. those between the name of a reference model in the ATL module and its representation in the megamodel).
8.3.3. ATL Transformation Launch Editor Page
A specific editor page has been defined to edit each ATL transformation launch registered in the megamodel. Figure 21 shows an example of this page based on the “Trace2PerformanceMetrics” ATL transformation launch:

Figure 21 Defining Model Bindings Using the ATL Transformation Launch Specific Page
This page allows the ATL transformation corresponding to the given launch to be selected. Once again, this operation is performed using the generic dialogue shown in Figure 20. According to the ATL transformation selected (and thus the ATL module), the corresponding bindings for all input/output models can be specified (i.e. between the name of a model in the ATL module and its representation in the megamodel). In addition, the Eclipse launch configuration file can be generated from this page allowing the transformation execution to be triggered.

8.3.4. ATL Transformation Chain Configuration Editor Page
A specific editor page has been defined for editing each ATL transformation chain configuration registered into the megamodel. Figure 22 shows an example of this page based on the “XML2Metrics” ATL transformation chain configuration:
Figure 22 Defining an ATL Transformation Chain Configuration Using the Appropriate Page

This page allows the model transformation chain to be selected which corresponds to the given ATL transformation chain launch. This operation is performed again using the generic dialogue shown in Figure 20. According to the selected model transformation chain, it allows the different ATL transformation launches that compose the transformation chain configuration to be specified and ordered. In addition to this, the Ant script can be generated from this page, which allows the execution of the complete chain of transformations to be launched effectively.
8.4. Demonstration on an SAP Material-Based Use Case

In order to demonstrate the applicability of the GMM4ATL extension to a concrete use case, the prototype has been applied to a complex model transformation scenario (which was initially implemented using the ATL model-to-model transformation tool [10] and material provided by SAP in the context of MODELPLEX).

Figure 23 SAP Use Case for the GMM4ATL Demonstration: Inputs and Output
As shown in Figure 23, this use case takes as input:

- A UML2 state chart model;
- A simple “performance” profile;
- An Excel file containing execution traces.

It generates as output a UML2 state chart model (the content of which is identical to that of the input) to which a performance profile has been applied. In this way, the UML2 state chart model now contains information on performance metrics computed from the traces provided by the input Excel file.

The overall process has been implemented by a chain of ATL transformations, as depicted in Figure 24.

This chain of transformations will not be described in detail in the present document. However, a complete description and all the resources required for execution is available from the appropriate Eclipse.org webpage [12]. A demonstration of the full execution of this ATL use case is available from the MODELPLEX SVN [13].

In the context of our GMM prototype, the goal is to describe the overall process (and its corresponding MDE artefacts) of this use case in a megamodel using the GMM4ATL extension. For our demonstration, this has been done for the most part automatically by retrieving the information from the metadata on the corresponding Eclipse project. The megamodel produced is then used in order to generate ATL transformations launch configurations and Ant scripts automatically which allow the different steps of the generation process to be performed effectively.

The demonstration of the use of the GMM4ATL extension on the ATL use case is available in Flash format from the MODELPLEX SVN [14].
9. GMM4TCS Extension

This section describes the current state of the Global Model Management extension dedicated to TCS. This extension, composed of a metamodel extension and of additional editor pages, has been developed using the AM3 core APIs and the extension mechanism provided.

9.1. GMM4TCS Metamodel Extension

The GMM4TCS metamodel (of megamodel) extension adds a set of TCS-specific concepts to the Global Model Management generic ones (Figure 25).

Thus the TCS Model type of transformation model is defined. It represents a concrete TCS model (which conforms to the TCS metamodel) containing a set of templates. A TCS model is mainly characterized by its related metamodel (i.e. the metamodel associated with the textual notation it describes).

The TCS Injection type of external-to-model transformation relationship and the TCS Extraction type of model-to-external transformation are also defined. These two types of relationship have to be linked to their corresponding TCS Model (using the transformationModel feature of the Model Transformation concept, see section 7). TCS Injection and Extraction also allow the type of the source and target models or textual entities to be specified (using the properties inherited from the external-to-model and model-to-external transformation concepts). They also allow options to be specified and to be then used during the execution of the projections (such as specific string delimiters, for instance).

The Textual Entity type has been defined to permit the representation of textual entities (which are not models) into the megamodel. This is necessary because the TCS projections take text files either as input or as output. The Textual Entity concept inherits directly from the Entity abstract core concept (from AM3Core).

![Figure 25 The TCS-Specific Extension of GMM](image)
9.2. TCS-Specific Features

The GMM4TCS extension provides some specific features which are relevant only within the context of TCS. The facilities detailed below may be especially useful when megamodelling MDE projects including TCS projections (i.e. injections and/or extractions).

9.2.1. TCS Injection Executor

The TCS Injection Executor implements the ITransformationExecutor interface provided by the Transformation Execution extension point (section 7.2). To this end, the API of the org.eclipse.m2m.engine package of the M2M component is used to inject and save the models corresponding to the real textual entities. The entities provided as parameters are accessed in order to retrieve their identifiers and locations. The location of the required parser jar file (automatically generated by TCS) is thus derived from the location of the TCS model before launching the injection programmatically. Thus, TCS injections correctly registered in the megamodel can be transparently executed from the transformation Executor generic page (section 7.3.3).

9.2.2. TCS Extraction Executor

The TCS Extraction Executor is implemented using the same principles as for the TCS Injection Executor. The only difference is that the parser jar file is no longer necessary; only the TCS model is required in this case in order to perform the extraction effectively.

9.3. TCS-Specific Editor Pages

In order to be able to deal with the newly defined concepts presented above and to provide some TCS-specific features, several GMM4TCS dedicated editor pages have been implemented. Note that these pages are automatically made available in the editor when editing a megamodel element of the corresponding type (a benefit provided by the extension mechanism described in section 5.3).

9.3.1. TCS Model Editor Page

A specific editor page has been defined to be displayed for editing each TCS Model registered in the megamodel. Figure 26 shows an example of the use of this page based on the “DOT projection” (DOT is a language for graphical visualizations [19]):

![Figure 26 The TCS Model Specific Editor Page](image)

This page has been designed to allow the metamodel associated with a given TCS Model to be set. By using the Browse button, a dialogue is opened so that
this metamodel can be selected from the list of the metamodels registered in the megamodel.

9.3.2. TCS Projector Editor Page
A specific editor page has been defined for editing each TCS Projection (i.e.; Injection or Extraction) registered into the megamodel. Figure 27 shows an example of use of this page, still based on the “DOT projection”:

![Figure 27 TCS Projection Editor Page (TCS Extraction Sample)](image)

In this particular editor, the “TCS Model” section allows the TCS Model implementing the projection to be selected, as well as providing the type of the external parameter (a DOT textual file in the specific case of a DOT extraction) and the metamodel of the model parameter. Once again, these entities are chosen using dedicated selection dialogues. The “Options” section is used only if the projection requires some specific parameters (such as mentioned in section 9.1).
10. GMM4CompositeTransformations (GMM4CT) Extension

This section describes the current state of the Global Model Management extension dedicated to composite transformation management. This extension, composed of a metamodel extension and of additional editor pages, has been developed using the AM3 core APIs and the extension mechanism provided.

10.1. GMM4CT Metamodel Extension

The GMM4CT metamodel (of megamodel) extension adds a set of composite transformation-specific concepts to the Global Model Management generic ones (Figure 28).

The Composite Transformation type is defined by inheritance of the Transformation concept from the GMM extension. A Composite Transformation additionally contains composite parameters (CParameter), calls to sub-transformations and connectors.

The connectors between sub-transformations are represented by the Connector type. Each Connector has two ends (two Connector Ends). Three sub-types of connector have currently been defined: Simple Connector, Multi Connector and Direct Connector. The difference between a Simple Connector and a Multi Connector consists in the number of connector ends it can manage. A Direct Connector is used in order to directly reference an Entity.

The CParameter type (for Composite Parameter) defines the specific parameter kind for Composite Transformation. Thus, CParameter inherits from the Parameter concepts from the GMM extension and also from the ConnectorEnd concept.

The TransformationCall type is used to represent the call to a sub-transformation into a composite one. It refers to a Transformation and deals with ParamRef (Parameter References) whereas this Transformation deals with Parameters.

The ParamRef type has the same utility for the TransformationCall as the Parameter type has for the Transformation. It refers to a real Parameter of a Transformation and to a ParameterRefValue which corresponds to the ParameterValue of a concrete Parameter.

The CTransformationRecord type defines the specific TransformationRecord concept dedicated to Composite Transformations. It refers to sub-records which are the TransformationRecords appropriate for the given sub-transformations.
10.2. CT-Specific Features

The GMM4CT extension provides some specific features which are relevant only within the context of composite transformations. The facilities detailed below may be very useful when megamodelling complex chains of transformations (and more specifically complex chains of model transformations).

10.2.1. Composite Transformation Executor

The Composite Transformation Executor provides an implementation of the ITransformationExecutor interface provided by the Transformation Execution extension point (section 7.2). This implementation is target driven: for each output parameter, the composite asks for the parameter value for the corresponding binding. It then calls the executor of the connector which recursively queries the previous transformation and so on. Once all the transformation chain has been executed, the result obtained is finally retrieved and saved. Thus, composite transformations registered correctly in the megamodel can be transparently executed from the transformation Executor generic page (section 7.3.3).
10.2.2. Connector Execution Extension Point

The generic execution of composite transformations requires common support for different kinds of connector. To this end, an extension point has been defined:


This extension point specifies an interface for the execution of a connector. The effective “execution” of a given connector means a request for the value of its target(s) from its source(s). In order to allow this, the interface defines two methods to be implemented: canGetEnd which returns a boolean indicating if the other end of the connector is accessible, and getEnd which returns the value of the parameter requested.

10.2.3. Simple Connector Executor

This connector provides an implementation of the extension point described above (section 10.2.2) for the Simple Connector concept. It is relatively basic since a Simple Connector only has one source. As a consequence the implementation simply returns the value of the input parameter to all the output parameters.

10.3. CT-Specific Editor Page for Composite Transformation

In order to be able to deal with the newly defined concepts presented above and to provide some composite transformation-specific features, a GMM4CT dedicated editor page has been implemented. Note that this page is automatically made available in the editor when editing a megamodel element of the Composite Transformation type (a benefit provided by the extension mechanism described in section 5.3).

Figure 29 shows an example of the use of this page based on the “TIPM-to-AnyLogic transformation chain”: 
This page allows all the information required for specifying a complete composite transformation configuration to be set.

The “Parameters” section is used to determine the in/out parameters of a Composite Transformation. These parameters are added by selecting some of the parameters proposed by the corresponding sub-transformations.

The “Sub-Transformations” section allows new sub-transformations to be added (and then connected) to the composite: All the entities which inherit from Transformation can be potentially selected as sub-transformations (including other composite transformations). With this in mind, a simple selection dialogue is provided.

The “Contained Connectors” section allows the different links which exist between the specified sub-transformations to be displayed and set. Establishing these connections requires the selection (by the user) of the sub-transformation parameters to be linked to their two ends. A specific selection dialogue is thus displayed for each parameter selection action (Figure 30):
In order to demonstrate the applicability of the GMM4CT extension to a concrete use case, the prototype has been applied to a real model transformation scenario. The use case under consideration was initially implemented by SAP within the context of MODELPLEX. For this purpose, they have used their own material as well as the ATL model-to-model transformation tool [15] and the AMW model weaving tool [16].

The objective of this demonstration is to show how the SAP real-life complex transformation chain can be:
- Represented in a megamodel (i.e. megamodelled) using the GMM4CT extension; and
- Executed directly from the prototype.

This complex model transformation chain is summarized schematically in Figure 31. The transformation process takes as inputs a UML activity model as well as parameters and requirement annotation models. As final outputs, it produces a traceability model and an XML representation of the AnyLogic model obtained. Note that the transformation process, which is not the purpose of this deliverable, will not be described in detail in this document.

The demonstration of the use of the GMM4CT extension on the presented SAP use case is available in Flash format from the MODELPLEX SVN [17].
Figure 31 Transformation Chain Sample: GMM4CT Demo
11. AM3Navigability Extension

This section describes the current state of the AM3 Core extension aimed at inter-model navigability. This extension, comprising the navigation generic mechanism and corresponding two graphical views, has been developed using the AM3 core APIs. Note that this section is a summary of sections 6 & 7 of [18].

11.1. AM3 Navigability Features

The goal of this extension is to provide the users with some generic inter-model navigability capabilities considering any kinds of potential relationships (materialized by links) between models and their elements. In all cases, two distinct but related levels of navigation can be considered:

- Model-level; and
- Model element-level.

In order to be able to model these two levels, two complementary MDE techniques have to be combined: 1) Global Model Management for dealing with the relationships at the model-level and 2) Model Weaving for dealing with the relationships at the model element-level.

To summarize, the next figure shows an abstract view of this generic inter-model navigability approach (see [18] for more details).

The traceability information automatically generated using the corresponding tool support (sections 7.2.2 & 7.3.3) provides concrete examples of those links which need to be navigated frequently. Thus, in the previous schema, by replacing the...
“MMw” metamodel by a “Trace” metamodel, the approach is directly applicable to the problem of model transformation traceability navigation.

11.2. Navigability-Specific Views
For more user-friendly support to the generic inter-model navigability approach described above, two dedicated views have been implemented. They each allow navigation at model and model element levels respectively.

11.2.1. Model Level Navigability View
The “Model Level” navigation view (Figure 33) queries the megamodel for the automatic display of those models which can be reached, using any of the available model level links, from the model currently selected within the Megamodel Navigator.

![Figure 33 Generic Model-Level Navigability View](image)

The navigation is of course possible both forward and backward (like in a
standard browser), independently of the types of link involved.

11.2.2. Model Element Level Navigability View

The “Model Element-Level” navigation view (Figure 34) also uses the megamodel, but this time to retrieve the weaving models storing the correct model element-level links. By querying these weaving models, all model elements that can be reached from the one selected within the opened model editor are automatically displayed.

![Figure 34 Generic Model Element-Level Navigability View](image)

11.3. Demonstration on an SAP Use Case

In order to validate the applicability of the AM3Navigability extension to a concrete use case, the prototype has been applied to a real scenario of model transformation chain. The process is the following:

1. Enable the automated traceability support in order to generate the traceability information automatically during the execution of the various transformations composing the chain;

2. Use the trace data produced with the AM3Navigability extension to
navigate along the transformation chain between the different available models (initial, intermediate or final ones).

The use case considered was developed within the context of MODELPLEX by SAP; it is exactly as described in detail in section 8.4. The two points of the previously described process have been applied successively to this use case using the AM3 prototype, whose required plug-ins are available from the MODELPLEX SVN [6]. A complete run of this process may also be viewed in Flash format at the MODELPLEX SVN [20].
12. Exploratory Work Using the AM3 Approach

This last section introduces some recent experiments on disparate topics, performed using the AM3 approach and corresponding prototype. These represent preliminary attempts whose full content is yet to be made available completely. The first concerns the problem of system cartography (reverse-engineering), while the second is about providing formal typing for all the modeling artefacts involved in global model management.

12.1. AM34LinuxPackages Extension: System Cartography

The first experiment relates to the concrete application of global model management techniques in order to build a usable cartography of a given system or platform automatically (i.e. a system composed of several other systems). As an example, the case of the different packages which make up a Linux distribution (e.g. Debian [21]) has been considered. The idea is to (mega)model such a complex system with AM3 and to use the megamodel obtained to generate a visualization of the dependencies existing between the different system components. All the plug-ins and projects required for installing the extension and the provided use case are publically available from the AM3 website [22].

12.1.1. AM34LinuxPackages Metamodel Extension

To be able to build such a cartography, the AM34LinuxPackages metamodel (of megamodel) extension provides a set of package-specific (mega)modeling concepts. As shown on Figure 35, this metamodel is a direct extension of AM3Core.

![Figure 35 AM34LinuxPackages: The Linux Distribution Specific Extension of AM3Core](image)

A Distribution is a Group of Entity elements, each of them being of type Package. In order to identify the packages specifically, the PackageIdentifier concept has been introduced. Many of the different kinds of relationship which potentially exist between two packages can be represented using the concepts inherited from DirectedRelationship: Depend, Provide, Conflict, etc. For each of these
relationships, the source and target packages are explicitly specified.

12.1.2. System Cartography-Specific Features

The AM34LinuxPackages extension provides some specific features which are relevant in the context of megamodeling Linux distributions. Figure 36 gives an overview of these implemented features.

There are actually two main capabilities which are provided:

1. The automated discovery of the cartography, i.e. the megamodel, from the standard Linux package description files. This is implemented by chaining an injection with model transformations;
2. The transformation of the obtained megamodel into a graphical view of the available dependencies between packages. This is implemented by chaining model transformations with an extraction.

As a result, it is possible to generate many different views from the same discovered cartography automatically. Taking by way of example the cartography of a Debian distribution in terms of packages and dependencies between them, the dynamic graphical view shown statically (which it is not its initial goal) in Figure 37 can be directly obtained with GraphML [25].

From this view, the different packages composing the distribution can be displayed, with potentially several colors depending on their “status” (e.g. normal, in conflict, not used, etc). The relationships existing between these different packages may also be provided under various formats. Moreover, the visualization in general can be handled and thus reorganized by using an appropriated tool (i.e. a tool which fully supports the GraphML format and allows customizing the final rendering).

Note that the current version of this experiment targets GraphML as a visualization format. However, many other formats (either textual or graphical) may be also supported in the future by simply extending the described approach.
12.2. TypeSystem4GMM: Formally Typing Modeling Artefacts

This second experiment concerns work directed toward providing a formal way to type each of the modeling artefacts that can be represented in a megamodel. This mainly consists of first designing a type system for Global Model Management, and then seeing how this type system can be plugged into the AM3 megamodeling tool. This experiment is still in an early phase: neither the implemented plug-ins nor the corresponding projects are available. It is briefly presented within this section, though more details can be found in [23].

12.2.1. A Type System Dedicated to GMM

In order to deal with the explicit formal typing of GMM entities, a type system based on a type calculus called cGMM has been specified and implemented in Java. This calculus is a predicative typed λ-calculus with dependent types. This section does not provide the detailed specification of this calculus. However, to summarize, the type system developed mainly allows the formal typing of the following artefacts:

- Models in general (i.e. terminal models, metamodels and metametamodels);
• Transformation models by the precise typing of their sources and targets;
• Weaving models (note that full support for them will probably only be part of the next version of the type system).

Using such a type system, the main objective is to be able to infer as much as possible the typing information from the modeling artefacts which are produced during any given MDE process, for instance as a result of a model transformation.

One concrete example is the typing of Higher-Order Transformations (HOTs), and more particularly of the transformation models that can be generated by these HOTs. In this case, the idea is to be able to infer automatically the source and target reference models of the transformation produced by using the typing information and rules available from the type system [23].

12.2.2. First Experiment on Plugging the Type System to the AM3 Tool

The AM3 GMM tool uses a megamodel which can be considered as a semi-formal typing system. Indeed, it already provides precise typing information on each of the modeling artefacts which are correctly registered in it. However, the available metadata are not always sufficient when trying to address the issue of systematically typing some automatically produced modeling artefacts (cf. the example presented at the end of last section 12.2.1).

As a consequence, it was decided to attempt some experiments with the common use of the AM3 tool and the GMM-dedicated typing system, and to work on the way they might interact with each other appropriately. In order to achieve this, the approach which has been considered is to create a generic type system interface that the AM3 tool can call as needed. This will allow both to send information to a type system and to retrieve some results from it. Of course, the first implementation of this interface corresponds to the current version of the GMM-dedicated type system.

Considering the case introduced in the last section, i.e. infer the full type of a transformation model generated by the execution of a HOT, the process followed by the AM3 tool is:

1. All the available typing information on the HOT and its sources/targets is retrieved from the megamodel and sent by the AM3 tool to the type system thanks to the generic type system interface;
2. The type system internally uses this information in order to infer the actual full type of the generated transformation model (including its sources and targets);
3. The AM3 tool, still using the generic type system interface, collects these additional metadata from the type system and properly completes the megamodel with reference to it.

Note that this process of integrating the type system into the AM3 tool is still in progress. Some initial concrete results of its application to different use cases are expected within the coming months.
13. Conclusion

Within the different sections of this document, the v2 prototype of the tool supporting Global Model Management has been described in some detail. The Eclipse-GMT AM3 project, from which the prototype is available, was briefly introduced at the start. Then, the prototype architecture and the currently provided core environment have been described. The other sections set out a description of the GlobalModelManagement (GMM), GMM4ATL, GMM4TCS, GMM4CompositeTransformations (GMM4CT) and AM3Navigability extensions along with concrete examples of their use on MODELPLEX Work Package 1 partner case studies (SAP scenarios). Finally, the last section has introduced some recent exploratory works also made with the prototype.

The experiments which have already been performed with the prototype show interesting results in terms of representation, storage and reusability of the metadata on MDE artefacts and processes. As the tool allows users to specify all this information explicitly as a megamodel (which is a model conforming to a given metamodel), it is thereby much easier to navigate, analyze and process it. The automated generation of Ant scripts, Eclipse launch configurations or DOT visualizations, the automated launching of transformations (with the generation of the related traceability information), the navigability of inter-model links or the cartography of existing systems are only a few examples of the many different possible applications of our prototype.

As described as the result of the opening months of work on Task 2.11, previously named Task 2.1 (cf. deliverable [11]), our objective was to design and implement a generic open-source Global Model Management tool capable of dealing with any kind of modeling artifact and the various relationships between them. This is what has been achieved with our current version of this open-source prototype based on a Global Model Management abstract conceptual framework, on an extension mechanism and on a generic but customizable Eclipse IDE.

Up to its conclusion in February 2010, the following steps within the context of the MODELPLEX project are:

- Continue to improve the core of the prototype and the generic IDE (Megamodel Navigator and Generic Editors);
- Fix bugs progressively on the extensions already developed in order to upgrade them;
- Upgrade the available extensions, and more especially GMM4ATL, so that they will be fully compatible with ATL 3.0 (current compatibility is ensured with ATL 2.1);
- Continue the various exploratory works begun recently on system cartography and formal typing (cf. section 12).
14. References


[2] MODELPLEX Deliverable: D2.4.a “Model Configuration Management”


[6] AM3 plug-ins v2 from a specific update site on MODELPLEX SVN:

```
work_area\WP2\T2.1 Global Model Management\deliverables\ModelPlex_ATL-AM3_UpdateSite.zip
```


[8] How to get AM3 sources:

```
```

[9] MODELPLEX Deliverable: D2.8 “MODELPLEX Architecture Orientations”


```
http://www.eclipse.org/gmt/modisco/useCases/PerformanceAnnotatedUmlStateCharts/
```

[13] SAP Use Case ATL demo: work_area\WP2\T3.1 Composition and refinement\D3.1d Model Transformation\ATL Demo - Palaiseau Jan 08

[14] SAP Use Case GMM4ATL demo: work_area\WP2\T2.1 Global Model Management\demos\GMM4ATL Demo - September 2008


[17] SAP Use Case GMM4CT demo: work_area\WP2\T2.1 Global Model Management\demos\GMM4CT Demo - September 2008


[20] SAP Use Case AM3Navigability demo: work_area\WP2\T2.1 Global Model Management\demos\GMM4ATL and Navigability Demo - June 2009


