	AM3 USE CASE EXAMPLE “LINUX PACKAGES DEPENDENCIES”	Guillaume Doux Guillaume.doux@inria.fr
	Use Case Description		Date : 2009/04/27

1. General Description & Objectives

This use case describes how AM3 [7] can be used to manage packages dependencies in distributions of Debian [8] Linux based systems. This use case can be divided in three parts:

- Discovery part, consisting in injecting the distribution package list into a model using TCS [4].
- Transformation to AM3 part, transforming the injected model into an AM3 megamodel. For this part, we need to define a specific extension to the AM3Core metamodel.
- Visualization part, consisting in extracting the content of the megamodel to a GraphML [5] model and then generate the visualization for this graph.

An overview of this use case is presented in Figure 1.

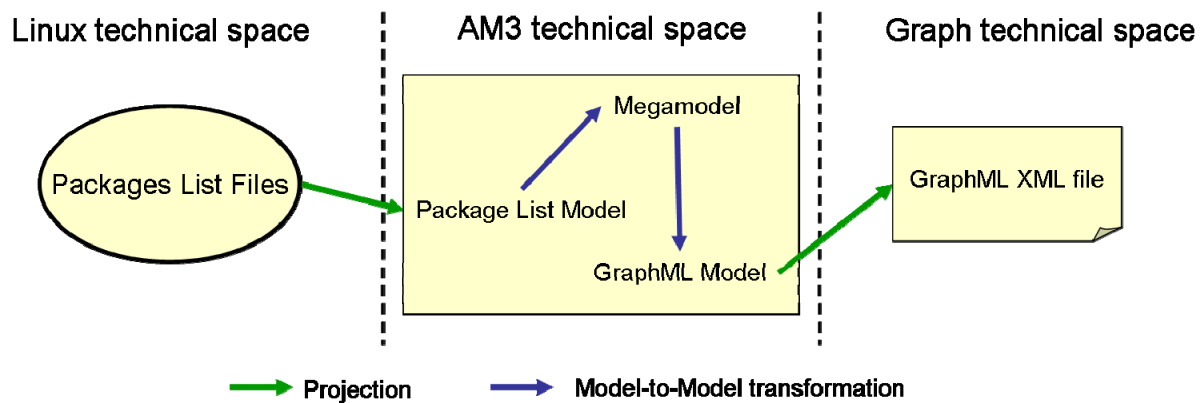


Figure 1 - Linux Packages Dependencies Use Case Overview

The main reason for using AM3 in this kind of use cases is that it allows managing them in a homogeneous way. With this approach we can use AM3 base features (e.g.: navigation between links), and also some specific features like visualization generation that can then be used for other use cases.

The development of this use case, realized by AtlanMod **Erreur ! Source du renvoi introuvable.**, has been supported by the French ANR TLOG IDM++ project (Model Driven Engineering ++) and by the IST European MODELPLEX project (MODELing solution for comPLEX software systems, FP6-IP 34081) [9].



2. Debian Package List Discoverer

The aim of this discoverer is to allow the injection of a “Debian” based distribution packages lists into a model. A sample package list can be found at [1]. To reach this goal, we first define in KM3 a metamodel allowing the package metadata representation and a TCS [4] definition to create the injector. This metamodel is shown in Figure 2.

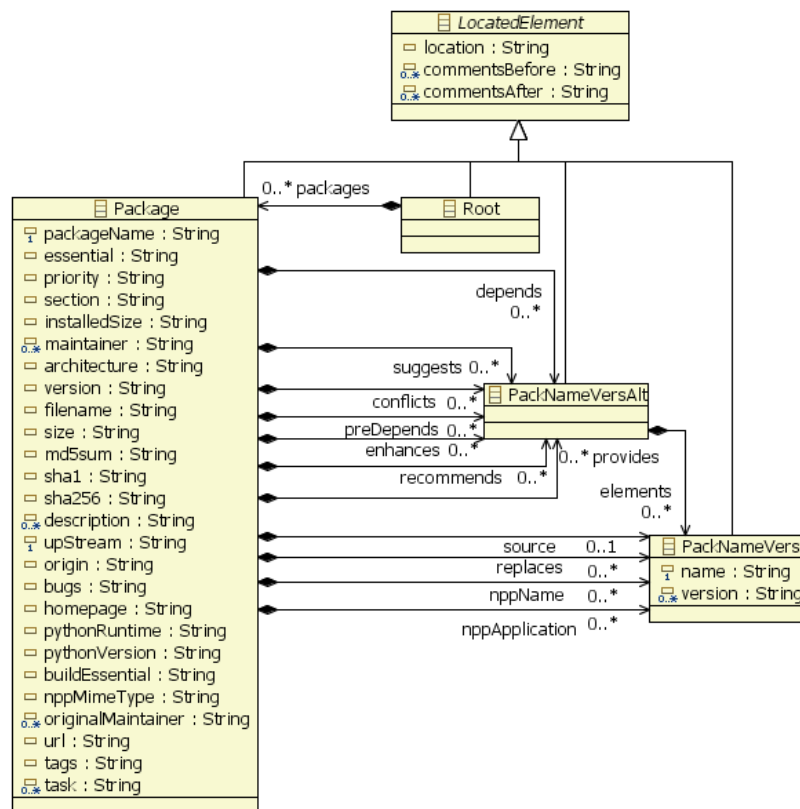


Figure 2 - Package List Metamodel

In this metamodel, a package is represented by all its metadata. Most of the fields are textual so there are stored as String attribute in the metamodel. The dependencies are stored as records containing the name of the dependence and the corresponding version.

The complete discovery overview from the package list file to the AM3 megamodel is presented in Figure 3.

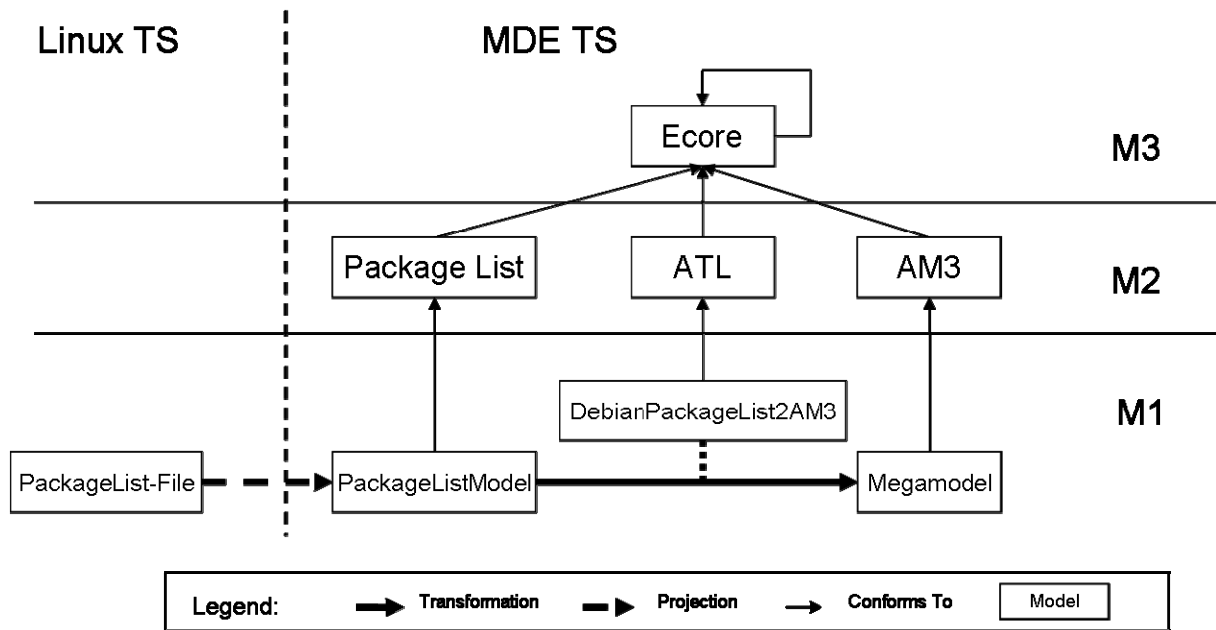


Figure 3 - Discovery and Transformation to AM3 Overview



3. AM3 Metamodel Extension & Transformation to AM3

3.1. AM3 Metamodel Extension

To be able to manage the specific complexity of the package dependencies, we need to add some concepts to the metamodel. These concepts have been added in an extension to the AM3Core metamodel. This metamodel is shown in Figure 4.

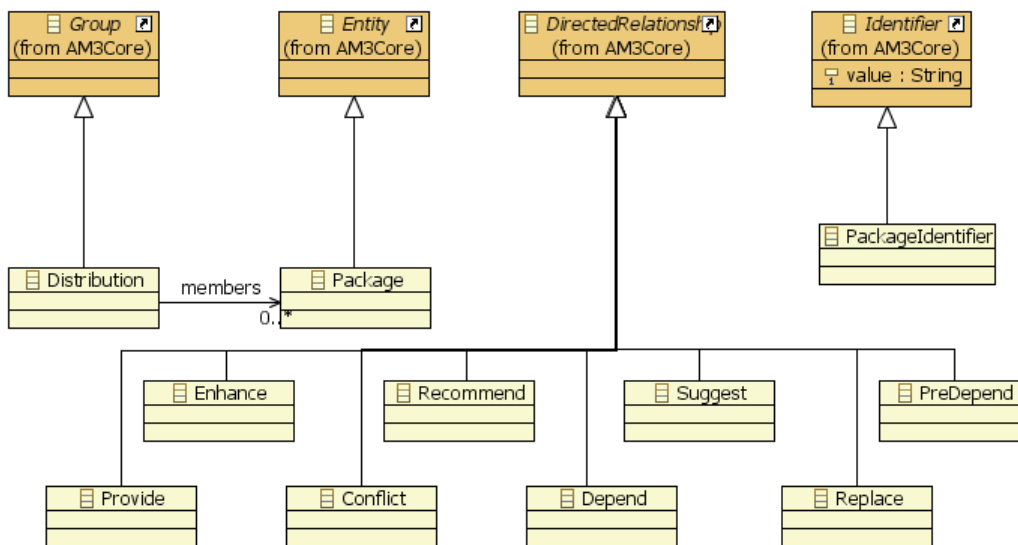


Figure 4 - Extension to the AM3Core Metamodel



In this figure, the orange meta-classes come from the AM3Core metamodel and the yellow one from the Linux Package Extension.

In this extension, we have defined a set of concrete meta-classes that extends some abstract ones of **AM3Core**. A distribution (i.e.: a set of packages) is represented by extending a **Group**. A package is represented by extending **Entity**. The different dependence relations are represented by extending the **DirectedRelationship** concept. We create also a *PackageIdentifier* in order to have a concrete identifier for the elements of the megamodel.

3.2. Transformation to AM3

This transformation takes as source the model injected from the package list and produces a megamodel conforms to Linux package extension metamodel. This transformation is implemented in *DebianPackageList2AM3.atl*.



The rule implemented in this transformation is presented in Figure 5.

		<p align="center">AM3 USE CASE EXAMPLE “LINUX PACKAGES DEPENDENCIES”</p>	<p align="right">Guillaume Doux Guillaume.doux@inria.fr</p>
	<p align="center">Use Case Description</p>		<p align="right">Date : 2009/04/27</p>

Source Element	Target Produced
Root	Distribution containing as members the packages of the root
Package	Package
	For each attribute concerning a dependency link (e.g.: depends, conflicts, recommends...), the corresponding relation is created. This relation have the current package as source and the target is found by a (name, version) resolution.

Figure 5 - Package List To AM3 Transformation Rules

Once this transformation, the produced megamodel can be imported in AM3. It can then be navigated and edited using the AM3 prototype.

		AM3 USE CASE EXAMPLE “LINUX PACKAGES DEPENDENCIES”	Guillaume Doux Guillaume.doux@inria.fr
	Use Case Description		Date : 2009/04/27

4. Visualization Generation

The aim of this part is to allow the visualization of the package dependencies graph. So, we choose to project the contents of the megamodel into a **GraphML** [5] file. Then, this file can be used for rendering by a tool like **Prefuse** [6].

To reach this aim, we use a generic transformation chain that can be used with any AM3 megamodel. This transformation chain allows the transformation of any megamodel containing entities and directed relationships into a graph. This process is shown in Figure 6.

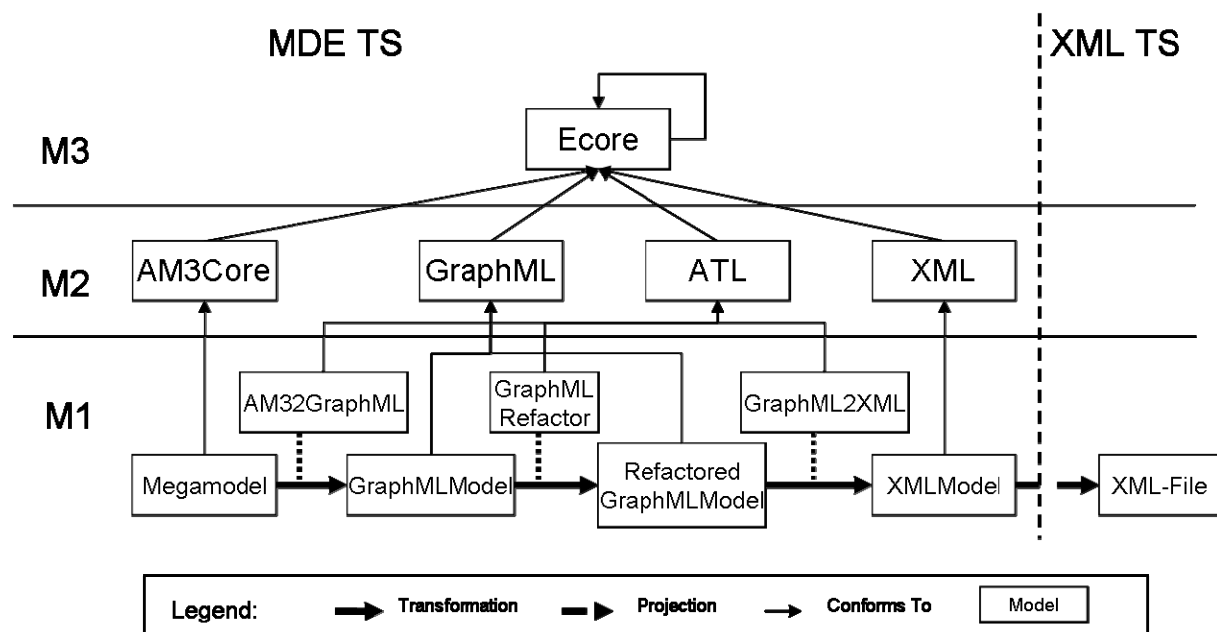




Figure 6 - Visualisation Generation Overview

It contains three transformations:

- AM32GraphML,
- GraphMLRefactor,
- GraphML2XML.

The last operation to do is the extraction of the final XML file from the obtained XML model.

These three transformations are presented in the following subsections.

		AM3 USE CASE EXAMPLE “LINUX PACKAGES DEPENDENCIES”	Guillaume Doux Guillaume.doux@inria.fr
	Use Case Description		Date : 2009/04/27

4.1. AM32GraphML

The rules used in this transformation are detailed in Figure 7.

AM3 Element	Corresponding GraphML Element
Group	Root of the GraphML document
	Main Graph of the document
Entity	Node, the identifier of the entity become the identifier of the node
DirectedRelationship	For each target element of the relationship, an Edge is created. The source of the relationship become the source of the edge

Figure 7 - AM3 To GraphML

This transformation is implemented in AM32GraphML.atl.

4.2. GraphMLRefactor

This is a refactoring step on the GraphML model obtained from AM32GraphML. It allows the suppression of the incomplete information that can exist if the megamodel source is incomplete. This transformation deletes the edges that refer to a not existing node.

4.3. GraphML2XML

The GraphML to XML transformation implements a direct mapping between GraphML model elements and XML model elements. As the GraphML metamodel has been designed especially for this task, the transformation is simple but quite verbose. For example, a graph in the GraphML model is translated into a node which has as name “graph” and contains XML elements for nodes, edges and attributes.

This transformation is implemented in GraphML2XML.atl.

4.4. Sample

In Figure 8, we see the result of the rendering of a megamodel containing dependencies between packages. The data used in this sample can be obtained at [1].





Use Case Description

Date : 2009/04/27



Figure 8 - Sample Rendered Graph

		AM3 USE CASE EXAMPLE “LINUX PACKAGES DEPENDENCIES”	Guillaume Doux Guillaume.doux@inria.fr
	Use Case Description		Date : 2009/04/27

References

- [1] An Ubuntu package list (in packages.gz -> Packages-2 file):
<http://mirror.ovh.net/ftp.ubuntu.com/ubuntu/dists/dapper/multiverse/binary-i386/>
- [2] The Kernel MetaMetaModel (KM3) Manual:
<http://www.eclipse.org/gmt/am3/km3/doc/KernelMetaMetaModel%5Bv00.06%5D.pdf>
- [3] The **AtlanMod** Team: http://www.emn.fr/x-info/atlanmod/index.php/Main_Page
- [4] TCS: <http://www.eclipse.org/gmt/tcs/>
- [5] GraphML: <http://graphml.graphdrawing.org>
- [6] Prefuse: <http://prefuse.org/>
- [7] The Eclipse/GMT **AM3** Component: <http://www.eclipse.org/gmt/am3/>
- [8] Debian distribution web site: <http://www.debian.org/>
- [9] The **MODELPLEX** IST European Project: <http://www.modelplex-ist.org>