INRIA-ATLAS Response to the MDA Tool Capabilities OMG RFI

Jean Bezivin, Hugo Brunelière, Freddy Allilaire, Mikaël Barbero, Marcos Didonet del Fabro, Frédéric Jouault

Object Management Group

140 Kendrick Street
Building A Suite 300
Needham, MA 02494

Telephone: +1-781-444-0404
Facsimile: +1-781-444-0320

MDA Tool Capabilities

Request For Information

OMG Document: mda-user/2006-09-01

Responses due: March 5, 2007

Response submitted by:

Jean Bezivin, Hugo Bruneliere, Freddy Allilaire, Mikaël Barbero, Marcos Didonet Del Fabro and Frédéric Jouault

INRIA (ATLAS Group) – Nantes, France

1.0 Cover letter

In the past years, the INRIA ATLAS Group has been building an MDA tool bench named AMMA (ATLAS Model Management Architecture). The present discusses the main characteristics and overall vision of this platform in the context of the OMG MDA Tool Capabilities RFI. In the following pages, we will provide an overall description of what MDA tool capabilities means for the ATLAS Group. We will show, within this response, how our overall Model-Driven Engineering (MDE) vision and implemented platform bring answers to the different RFI questions. We will also highlight the various MDA tool-specific needs and requirements we have already identified, even though some are not yet fully addressed by the current version of our platform.

In a more organizational point of view, we have tried to follow as much as possible the logical sequence of the RFI proposed questions; however in many cases we have answered several questions at once. Our goal is not to answer exhaustively all the questions but more to cover all the different requirement areas.

In order to provide some additional information about our MDE approach, platform and corresponding tools, we are attaching the *AMMA_References.doc* document that contains references and pointers to the different tools and some of the papers published by the ATLAS Group.

More information on our research group and its MDE contributions is publicly available on the AMMA Platform official website http://www.sciences.univ-nantes.fr/lina/atl/.

We welcome any questions and/or remarks about our response and look forward to the opportunity of discussing the issues touched by this RFI with the OMG ADTF group members and affiliates.

Kind Regards

Hugo Bruneliere - R&D Engineer
E-mail: Hugo.Bruneliere@univ-nantes.fr
Phone: +33 2 51 12 58 10

ATLAS Group (INRIA & LINA) - University of Nantes
2, rue de la Houssiniere
44322 Nantes Cedex 3 - France

2.0 Response

2.1 Summary of this RFI

This RFI requests information on what capabilities (functionalities, methodology definition and process guidance) of MDA tools the MDA user community currently use for their projects, and which capabilities they would like to have. Responses should distinguish between capabilities they use now and capabilities they would like to have. Since different projects and applications have different needs, we also request how and why the capabilities are and/or would be useful. Responders are asked to distinguish between capabilities that are required (their company or project will not purchase a tool without it) and those that are preferred in an MDA tool.

Information obtained may be used to develop standards for compliance levels for MDA tools, or to identify new standards that are required by the MDA user community. This RFI also gives MDA users a method to communicate their needs to other users and to the MDA tool vendor community. The information will help tool suppliers decide which features to keep, which features to add, which features to enhance, and which features to drop.

2.2 Detail

Within this section, we provide the core of our response, i.e. our set of answers to the different RFI questions according to our MDE global approach and to the corresponding implemented AMMA Platform. Note that we sometimes use within our answers a specific terminology which is explained in the glossary (see section 3.0).

2.2.1 Models

1. *What kinds of models do you use to apply MDA processes?*
2. *What kinds of models do you use to model your system/software for MDA?*

In our MDA general processes, we use the three different types of models which are terminal models (M1-level), metamodels (M2-level) and metametamodels (M3-level): these kinds of models correspond to the three main levels promoted by the OMG in its proposed MDA vision.
For example, we may have a UML 1.4 for (terminal) model of the OMG organization (task forces, interest groups, staff, members, etc). We may also consider the UML 2.1 metamodel or the MOF 1.4 metamodel. Note that according to the latest versions of the MDA guide, all these three artifacts may be considered as (abstract) models. This means that they share common properties and behaviors. For example, all these artifacts may be stored, retrieved, transformed, etc.

More generally, each model (whatever its type) conforms to its reference model: a terminal model conforms to a metamodel, a metamodel conforms to the metametamodel and the metametamodel conforms to itself (and is unique within a particular technical space, see the glossary of section 3.0). The terminal models are the direct representations of real-world systems.
We summarize this overall vision in the following conceptual schema:

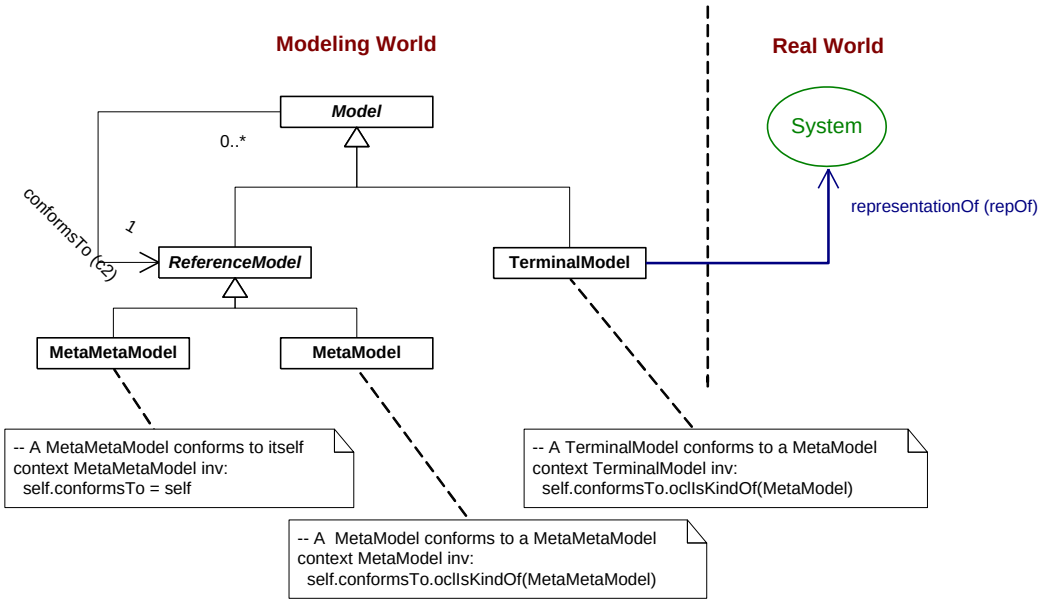

Figure 1: General Modeling Framework

On the left of the dotted line, Figure 1 shows the modeling world which is of main interest to us here. On the right of this dotted line, there is the real world. We make a strong difference between a system (drawn as an oval) and a model (drawn as a rectangle). For example, we could consider the real OMG organization as a system S and separately an UML model M of this organization: we then say that M is a representation of S (in short *repOf*).
Any MDA approach is based on a precise and unique metametamodel (M3). Unfortunately, there are several possible choices like MOF 1.4, MOF 2.0, Ecore, etc. This is the reason why we have defined a minimal M3 pivot named KM3 (Kernel MetaMetaModel). The KM3 pivot comes with operational mappings to most common proposals. These mappings are implemented in ATL (ATLAS Transformation Language), which is dedicated to model-to-model transformation, and are available as open-source components.
Platform independence is one of the key initial ideas of MDA. If we cannot achieve this, a solution is not MDA compliant. Independence from the M3 has been found of paramount importance in AMMA. Initially, the ATL virtual machine was built on top of Netbeans/MDR (based on MOF 1.4). Then, we had to rapidly cope with Eclipse/EMF (Eclipse Modeling Framework, based on Ecore). Through this experience, we learned the value of being M3-independent. In the future, AMMA may use new versions of the MOF or other different

platforms like the Microsoft DSL Tools. This is possible thanks to the KM3 pivot that makes AMMA truly platform independent and thus MDA-compliant.
KM3 is a textual DSL allowing easily defining a metamodel. This metamodel may immediately be transformed in another notation like Ecore. Of course, there are plenty of other ways to generate a metamodel like using a UML CASE tool to draw a class diagram and transforming the resulting XMI into a XMI for a metamodel (this last operation is called promotion because it takes as input an M1-entity and produces as output an M2-entity).
Metamodels are the central artifacts in any MDA activity. There are different sources for such metamodels. The primary one is of course the set of standardized OMG MOF metamodels (UML, SPEM, KDM, etc). Another source of metamodels may be the open-source communities like Eclipse that is providing Ecore metamodels. But we have found in practicing MDA with AMMA that there is also an important need for locally defined metamodels. For example, in a chain of M2M transformations, there may be a need for ten specific metamodels. Sometimes these metamodels may be shared: this brings new important issues about how to organize libraries of metamodels. There is also an issue about metamodel reusability, which brings to attention the need to have a precisely defined mechanism for metamodel extension.
A metamodel represents the abstract syntax of a DSL. Since there are a lot of such DSLs, there will be a lot of metamodels too (see the answer to question 3). In addition of a metamodel, a DSL may be associated to some concrete syntaxes and some semantics as well. Projects like Eclipse GMF (Graphical Modeling Framework) or Microsoft DSL Tools allow weaving a graphical concrete syntax to the metamodels related to a DSL.
The same models may be expressed in different formats. Since the AMMA platform is currently built upon Eclipse, we often use the XMI 2.0 variant of EMF to serialize our models. However, we also provide several bridges, implemented by ATL transformations, between the Eclipse Modeling Framework, and other modeling environments or formats (see the different libraries or "zoos" of metamodels).

3. Which modeling languages should an MDA tool support, and why?

According to the nature of the MDA process to perform, and also to the kinds of systems to model, we design different domain-specific metamodels (by using the KM3 language). These metamodels define the abstract syntaxes of Domain-Specific Languages (DSLs), and are tuned to domain-specific purposes. The systems to model (it can be software but not necessarily) are represented by terminal models that conform to these domain-specific metamodels.
Our MDA approach is thus metamodel-independent (i.e., not based on a limited set of modeling languages). We believe that domain-specific languages are more useful to capture relevant information in models than one or a few general-purpose languages are. As a consequence, we recommend that MDA tools should be able to support the definition of DSLs for different purposes, and should not be limited to a few languages only. This is a significant difference with the approach offered by most CASE tools, which are based on a single general-purpose metamodel: UML.

For example, within the AMMA platform, it is possible to define different modeling languages for domain-specific purposes. To this intent, we specify their abstract syntaxes as different metamodels (by using KM3). When their concrete syntax is textual, we define it by using TCS (see the glossary of section 3.0). Thus, the users (e.g., domain experts) can define and use different DSLs depending on the kind of systems they want to model and also on the point of view from which they want to observe them.

4. *How important is it for you to see both graphical and textual views of your models as you build them?*

As it is mentioned in the question, we are considering different views (graphical and/or textual) on the same models. By experimenting with our AMMA platform, we noticed that having a textual representation of models is most of the times sufficient. For instance, the KM3 language (which is a textual one) seems to be really adapted for defining metamodels even though it does not directly provide a graphical view of the produced metamodels.
By using this kind of DSLs, we do not feel the particular need for always having graphical views. This type of view may be user-friendly but often becomes difficult to handle when dealing with large models (several hundreds entities). However, combining both views (textual and graphical) may be a useful additional feature when building models (i.e., metamodels and/or terminal models).
Once again, we believe that a model development toolkit should allow easy definition of DSLs:

1) By creation and modification of abstract syntaxes represented by metamodels.
2) By definition of textual concrete syntaxes.
3) By definition of graphical/visual concrete syntaxes.
4) By definition of semantics.

Our proposal for 1) is KM3 but many other solutions are also possible in AMMA (e.g., MOF, Ecore).
Our proposal for 2) is TCS for mapping metamodels and grammars.
We do not offer a specific solution for 3) because we are using the Eclipse GMF (Graphical Modeling Framework) project.
Our proposal for 4) is ATL that may be used to map unknown to known domains. For instance, a Workflow DSL can be mapped to a Petri net (i.e., a well-known and precisely defined domain) metamodel by an ATL transformation.

5. *What system architectural viewpoints do you/would you like to model: Enterprise, Information, Computation, Engineering, Distribution, Other?*

According to our vision of MDA, any viewpoint on any system may potentially be modeled by creating or extending the appropriate DSLs. Our approach does not have any restriction concerning the different system architectural viewpoints that may be modeled. We believe that, for a given problem, every relevant

viewpoint should be considered. The tools should not prevent this by limiting what modeling languages are available.
Within our approach, we want to consider the problem of viewpoints on systems in the broad sense. The different possible architectural visions of a system are only a specific subset of the overall problem. Again here, any viewpoint on a system or on a model may be defined by an adequate DSL. With our platform, we plan to experiment on this general topic, define the main related concepts and provide some corresponding tooling.

6. *Which elements of UML and its extensions (e.g., SysML) do you need an MDA tool to support?*

We need the MDA to address most parts of UML, SysML but also many other metamodels. MDA is not restricted to UML and SysML and can be potentially applied independently of them.
We currently consider that one of the most important risks and challenges of the OMG MDA is the lack of modularity in the expression of most standard metamodels. UML profiles are not a scalable solution to these problems. Most users want to use only well-defined parts of UML, SysML, SPEM, KDM, etc, and it is currently very difficult to delimit precise boundaries within metamodels such as UML and SysML. There is a growing demand for much simpler and clearer modularity features that will allow user delimiting precise scope and tailoring the metamodels to the exact needs of a company or organization.

7. *What capabilities should an MDA tool have for describing actions?*
8. *How should actions for states, transitions, and operations be modeled? For example, should actions be described by activity diagrams, sequence diagrams, action language, and/or code?*
9. *When the actions are described with action language or code, how should they be integrated with the model?*

After experimenting in a previous project with executable models and metamodels, we do not believe anymore in this solution.
We have been using Smalltalk blocks (similar to Lisp closures) attached to model and metamodel elements. In the initial period, this seemed interesting since these “semantic actions” written as Smalltalk blocks were using a precise API to access model and metamodels elements. But after some experiments, this approach does not seem to scale up. There will be difficulties in converging on a standard action annotation language while Eclipse is already using Java and MS DSL Tools are using C#.
Our position is to avoid polluting the metamodels and models with such information. Instead, we propose a clear separation of concern based on the notion of model weaving (like in the AMW or ATLAS Model Weaver, see next answer): the actions should be kept separated in a decoration model.
This decoration model may conform to any appropriate action metamodel, depending on the problem at hand. This metamodel may define activity diagrams, sequence diagrams, action language, code, or more precise formalisms like Petri net.

In other words, we do not believe it is possible to standardize one unique action language. There have been a lot of experiments in this field and there are currently a lot of proposals. Again, our position here may be summarized as: "do not pollute your metamodels". Instead, as we have shown in the AMW Eclipse component (see the AMW use cases), it is possible to separately decorate metamodels with a number of different action languages (or even programming languages) by the well-established technique of model weaving.

10. What restrictions should an MDA tool impose on actions?

As we previously mentioned it, our approach is metamodel-driven and based on the concept of DSL. By following these principles (and in order to be able to describe actions), an MDA tool can provide several DSLs for expressing actions. These DSLs could be generic with extension capabilities, so that it is possible to define new actions DSL (for specific actions-related purposes) as extensions of the generic DSLs.
Actions should be represented (i.e., described) by models that conform to "actions" metamodels. As a consequence, the kinds of actions that may be expressed are not restricted: their possible formats or potential options should be determined by the corresponding "actions" metamodels. For instance, the UML metamodel provides some ways to describe actions but we may define a lot of other ones.
The produced "action" models may be woven with other models, by using the ATLAS Model Weaver (AMW) solution for example. The behaviors described by the actions may be applied to the corresponding modeling elements, and thus produce other models, etc.

11. What types of marks on the model do you want supported by the tools?

Model marking is a very important issue. The previous answers on how to describe actions by annotating model elements will be repeated here in a more general context.
The problem may be abstractly defined as follows:

a) We have an original model M_a and we want to decorate this model with some data in order to get a decorated model M_g.
b) The question of which metamodels M_a and M_g do conform to is left unanswered at this time.
c) The decoration information should ideally constitute a model by itself if possible.
d) Decorations should not destroy nor pollute the original model.
e) It should be possible to apply multiple independent decorations to a same model.
f) It should be possible to decorate an already decorated model.
g) It should be possible to transform a decoration model into another model (M2M transformation).

We believe that MDA processes explicitly or implicitly assume the existence of facilities for marking or decorating models. The MDA guide is full of references

to this. Nevertheless, in practice we are for having tool support for all the previously expressed requirements.
What we claim is that any model toolkit should provide model marking or model decoration facilities implementing these requirements. The AMW component of the AMMA platform has proven that implementing these facilities at a low cost is already possible.

12. How should the marks on the model be expressed?

MDA tools should be able to support any kind of marks. But within our MDA approach, we generally prefer to talk about "marking models" instead of "marks". Following the same principles as those previously presented for expressing the actions, we represent marks on models by other models that conform to specific marks metamodels. Because in this case a given element of a system may be referenced in several different models, we recommend using a weaving mechanism in order to establish some correspondence links between these models.
Once again, the goal is to not pollute the models with a lot of additional data (instructions, marks, etc). As a much more modular alternative, we suggest using model weaving as implemented by AMW.

2.2.2 Data

13. What work products should an MDA tool produce?

Since in a MDA approach models are considered as first-class entities, the main artifacts produced should be models (i.e., terminal models and/or metamodels). These models may then be used to perform MDE operations (mainly derived from transformations and/or weavings) and for producing different visualizations or outputs based on their content (in graphical formats, textual formats or both of them).
Typically, the management of all these models and their metadata is one of the goal of our global model management (or GMM, see the glossary of section 3.0) approach, which is based on the core concept of *megamodel*.
The following conceptual schema (derived from Figure 1) summarizes our overall GMM approach:

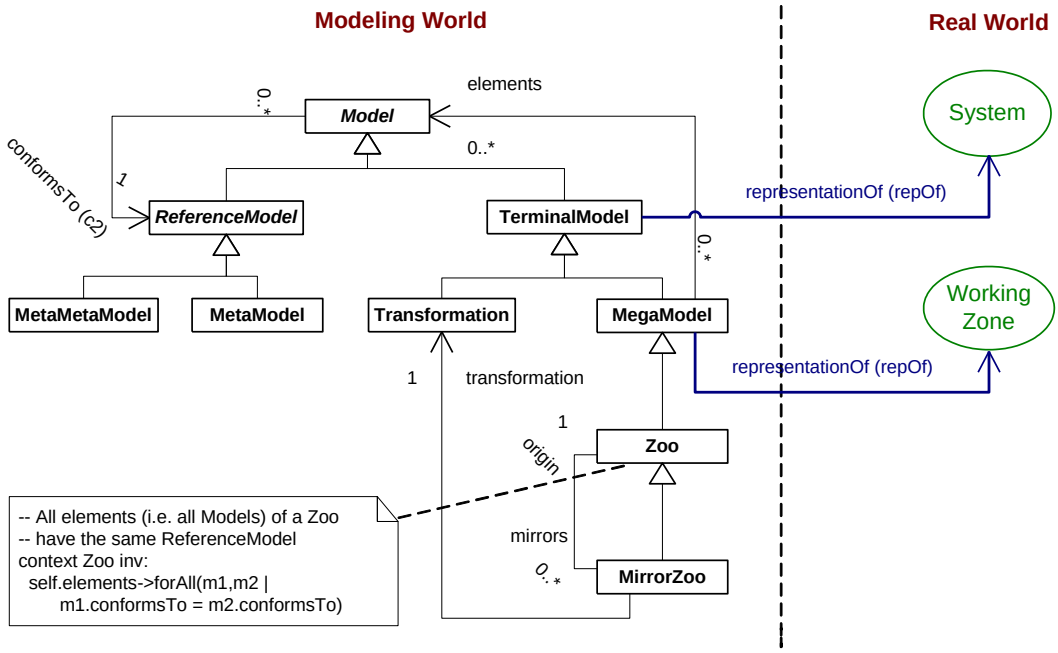

Figure 2: GMM Conceptual Framework

A megamodel is a terminal model in which elements are representing models, relationships between them and more generally metadata on them. By extension of this concept, we define the concepts of "zoo" and "mirror zoo" that correspond to megamodels in which all elements (i.e., all models) conform to a single metamodel. For instance, in a zoo of metamodels, all elements conform to a single metametamodel (e.g., MOF).

14. *What are the inputs (documents, tables, data) to your MDA process and MDA tools?*
15. *Which data formats should be supported for input/output data?*

The inputs/outputs of an MDA process or tool are strongly linked to the type of system you want to model and to the viewpoint from which you want to observe it. Indeed, the formats of the inputs/outputs may differ depending on the systems which are considered.
We strongly believe in the notion of "projection" between different technical spaces, and especially from/to the "modeling" technical space. It consists in switching from the heterogeneous world of the systems to the homogeneous world of the models (injection) and vice-versa (extraction).
According to this vision, the more projectors a MDA process or tool provides and/or supports the more efficient and valuable it is.

16. *At what level should model elements and MDA artifacts be version controlled?*

Because in a MDA approach most of the artifacts (if not all) are models, we believe that we should focus on the control of the different versions of a model.

The main problem is about how to deal with different versions of a same model without loosing any information. Once again, this is one of the main goals reached by our global model management (GMM) approach and the metadata it handles (see the answer to question 13).
One possible solution for implementing this version control is to use AMW: the differences between two different versions of a model may themselves be represented as a model (a weaving model). The weaving model conforms to an extension of a core weaving metamodel. The core weaving metamodel defines basic link management semantics. It is extended with different kinds of links depending on the application scenario. In the case of version control, "difference" links can be defined to capture differences between models. The weaving model should explicitly define the relationships between the model elements. A complete use case implementing such an approach and using AMW is available here.

2.2.3 Interoperability

17. *What interchange capabilities do you require from MDA tools? For example, XMI interchange, diagram interchange.*

Interoperability between different MDA tools and produced artifacts is for us a main issue in the MDA global approach. Indeed, one of the most important properties of models is their portability between several platforms and/or environments. As a consequence, an MDA tool really needs model import/export facilities.
According to our different experiments with the AMMA Platform, XMI seems to be at the current time the best and most commonly used format for exchanging models. For instance within the Eclipse platform, the EMF XMI-based format is widely used and reliable.

18. *Between which types of MDA tools do you need interoperability? Please refer to the types of tools in the taxonomy, if possible.*

The main idea of MDA (and MDE in general) is to gather the problems in the homogeneous world of the models so that it breaks the complexity which can exist in the heterogeneous worlds of the systems.
As a consequence, there should not be any restriction in the possible exchanges of models between different types of MDA tool. All the MDA tools should be interoperable because all the exchanged artifacts should be models expressed in well-known formats.

19. *Which OMG standards should an MDA tool support?*
20. *What standards could the OMG develop to make MDA easier for your company, e.g. standards for transforming UML to code, packaging transformation patterns?*

In our opinion, MDA tools should support all of the OMG standards. In addition to the already existing ones, we provide below a set of possible MDA standards that may be interesting to develop:

- Workflowing operations on models
- Global Model Management
- Model weaving or composition
- Mappings (or projections) to external spaces (XML, EBNF, etc)

2.2.4 Transformations

21. Which transformations do you require? For example, Model → Code, Model → Documentation. Please describe the types of transformations such as inputs, outputs, and markings as specifically as possibly.

We consider two main families of transformations: model-to-model and model-to-text (or text-to-model).
Within the AMMA Platform, we already provide an important library of model-to-model (M2M) transformations, expressed in ATL, covering a large set of possible domains of application.
The model-to-text and text-to-model transformations respectively correspond to possible implementations of the concepts of "extractors" and "injectors" (both are "projectors", as already explained in the answer to question 14 and 15). To this intent, the AMMA Platform provides a tool named TCS that allows defining and building such "projectors".

22. Which transformations should be reversible?

There are no general rules for specifying the kinds of transformations that should be reversible. It depends on the kind of MDA process the transformation is designed for.
We believe that there is no need for a real reversibility mechanism in a transformation language. While developing a transformation (and only if necessary), the corresponding reverse transformation may be also manually implemented and/or (at least partially) generated. We have followed this principle within the AMMA Platform for the development of ATL.
However, the reversibility of model transformations can be supported by defining relationships between elements at a higher abstraction level, by the means of weaving models. The weaving models define declarative links that are used as specification for producing transformations. The kinds of links are defined as extensions to a core weaving metamodel (see the answer to question 16). We already experimented within the AMMA the generation of ATL transformations (by using what we name High Order Transformations or HOT, see the answer to questions 24-25) in order to transform UML Profiles into DSLs, and vice-versa. As main result, we were capable to create a single weaving model that was used to generate ATL transformations in both directions. This weaving model is the only user-created artifact, and may be considered as a bidirectional transformation.

23. Which transformations should be traceable?

We provide here the same kind of answer as for the previous question about reversibility: the need for traceability depends on which kind of model you want to transform and/or produce.
However, it may be useful to have a traceability mechanism provided with the language (but not necessarily directly integrated into the language). As a consequence, we are currently experimenting within the AMMA Platform in order to provide a simple traceability mechanism based on a traceability metamodel.
Actually, the traceability metamodel is a weaving metamodel, because it keeps links between several models (source and target). This weaving metamodel consists in a traceability extension to the core weaving metamodel. The traceability extension is available in the AMW metamodel Zoo. In our experiments, we used higher-order transformations (HOT, see the answer to questions 24-25) that take ATL transformations as input and that produce another ATL transformation as output. This output transformation creates an additional weaving model to store the traceability information.

24. Between which pairs of viewpoints do you/would you like to use automated transformations?
25. Do you need to customize or build your own transformations?

We believe that an MDA tool should permit both: a user may want to build its own transformations from scratch or may also want to customize some already provided basic transformations.
Another solution, which is a very promising approach, is to use what we name High Order Transformations or HOT. A higher-order transformation is a transformation T_{HOT}: $T_{IN} \rightarrow T_{OUT}$, such that the input and/or the output models are transformation models. Higher-order transformations either take a transformation model as input, either produce a transformation model as output, or both.
We learned that there are many possible uses for HOTs by conducting several experiments that use them:

- A HOT can be used to modify existing transformations, such that the resulting transformation produces traceability information (see the answer to question 23).
- We made several experiments with HOTs that take weaving models as input and that produce ATL transformations as output. For instance, one weaving model with bidirectional links was used to produce two ATL transformations (see the answer to question 22).
- We have developed extensions of weaving metamodels that are used as high-level specifications for interoperability operations. The higher-order transformations take these weaving models as input and produce executable transformations in ATL (the AMW use cases provides several examples that use this approach).

26. What kinds of characteristics should transformations support? For example, configuration management of transformations? Deployment, composition, graphical interface, command line interface?

In real MDA processes, we consider complex chains of transformations more than simple elementary transformations. As a consequence, one of the main characteristics of the transformations is that they can be composed. Thus, transformation tools builders should provide facilities for chaining transformations.
Within the AMMA Platform, we have implemented ATL-specific ANT tasks to this intent. They allow building ANT scripts for automating the execution of complex chains of ATL transformations involving several models (i.e., terminal models and metamodels).
However, we believe that transformation chaining should be provided by a workflow DSL, and that such a language should be standardized by the OMG (see answer to question 20).

2.2.5 Process

27. How is MDA used in your development process?

Because our activity is focused on MDA and on its different applications, we try to use MDA principles as much as possible in our development processes but also in all our processes in general. We concentrate most of our efforts in the development of new MDA tools and technologies so that MDA is always our main interest.
We apply this principle within the AMMA Platform: most of the tools we provide are based and/or use ATL, which is our transformation language, and its corresponding virtual machine and engine. Due to this architecture, AMMA fully meets the goals of platform independence as stated in the original MDA guidelines.
Moreover, because the DSLs offered by AMMA (e.g., ATL, KM3, TCS, AMW core) provide means to define new DSLs, they can also be used to define themselves. We follow this principle because we believe that MDA should also be used to develop MDA tools. For instance, the abstract syntax of KM3 is specified in KM3, its concrete syntax in TCS, and its semantics is represented by ATL mappings to MOF 1.4, Ecore, etc.

28. What types of tools do you need an MDA tool to integrate with? Please refer to the types of tools in the taxonomy, if possible.

An MDA tool should first interface with any other MDA tool and also with non-MDA tools. Let us consider these two categories separately.

A) Interface with another MDA tool.

According to Wikipedia, an MDA tool is a tool used to develop, interpret, compare, align, measure, verify, transform, etc. models or metamodels. In the following section "model" is interpreted as meaning any kind of terminal model (e.g. a UML model) or metamodel (e.g. the CWM metamodel). In any MDA approach we have essentially two kinds of models: initial models are created manually by human agents while derived models are created automatically by programs. For example an analyst may create a UML initial model from its observation of some loose business situation while a Java model may be automatically derived from this UML model by a Model transformation operation. An MDA tool may be one or more of the following types:

- Creation Tool: A tool used to elicit initial models and/or edit derived models. Some of these tools may be used in collaborative environments (e.g. Jazz-based)
- Analysis Tool: A tool used to check models for completeness, inconsistencies, or error and warning conditions. Also used to calculate metrics for the model. Note that all these kinds of operations may be implemented by using model-to-model transformations, for instance in ATL.
- Transformation Tool: A tool used to transform models into other models or into code and documentation. The ATLAS Transformation Language (ATL) is such a tool.
- Storage and retrieval tool: a tool that allow to store models and to retrieve them from some repository. The Teneo Eclipse project is such a tool. Another example is the Adaptive repository.
- Composition Tool: A tool used to compose (i.e. to merge according to a given composition semantics) several source models, preferably conforming to the same metamodel, but not always. The ATLAS Model Weaver (AMW) can be used in this scope.
- Matching Tool: A matching tool is used to find links between elements of different models. The links can be used in many application scenarios, for instance as the input for a Compare and Diff Tool, or to easy the task of producing transformations. The ATLAS Model Weaver (AMW) provides generic matching mechanisms.
- Compare and Diff Tool: A tool able to take for example two models as input and to create a model representing their difference. This difference is a model and could later be used by a composition tool. Note that all these models may not conform to the same metamodel. The ATLAS Model Weaver (AMW) can also be used in this scope.
- Test Tool: A tool used to "test" models. Testing a model means verifying that the model has some particular properties. Should not be confused with a model of the test related to some code generated from another model.
- Simulation Tool: A tool used to simulate the execution of a system represented by a given model. This is related to the subject of model execution. It may work if a model is somewhat decorated or annotated with actions written in a given language like Java or C# or any proprietary or standard language.

- Metadata Management Tool (Global Model Management Tool): A tool intended to handle the general relations between different models, including the metadata on each model (e.g. author, date of creation or modification, method of creation (which tool? which transformation? etc.)) and the mutual relations between these models (i.e. one metamodel is a version of another one, one model has been derived from another one by a transformation, etc.). The Eclipse/GMT AM3 (ATLAS MegaModel Management) component goal is to intent to address this Global Model Management problem.
- Reverse Engineering Tool: A tool intended to transform particular legacy or information artifact portfolios into full-fledged models. Some tools perform more than one of the functions listed above. For example, some creation tools may also have transformation and test capabilities. There are other tools that are solely for creation, solely for graphical presentation, solely for transformation, etc. The Eclipse/GMT MoDisco (Model Discovery) component goal is to intent to provide such a reverse-engineering framework.

One of the characteristics of MDA tools is that they mainly take models (e.g. MOF models or metamodels) as input and generate models as output. In some cases however the parameters may be taken outside the MDA space like in model to text or text to model transformation tools.
The Eclipse top level modeling project (EMP) is currently developing a set of open source tools of various profiles (EMF, GMF, M2M, GMT, etc.).
So any two MDA tools should integrate in a smooth way without any problem. The only aspect that needs being taken care of is the global model management (GMM). Several GMM facilities should take care of any modeling artifacts created, deleted, updated or simply available to a given tool. A given MDA tool should bear a signature stating which kinds of models it is supposed to use and which kinds of models it is supposed to produce.

B) Interface with a non-MDA tool.
A non MDA tool does not use models or modeling standards like XMI. As a consequence its collaboration with MDA-tools should be handled via projectors (i.e., injectors and/or extractors). These projectors are transformations between an external space and the MDA space. Some may be generic (i.e., metamodel-independent), whereas others may be specific (i.e., metamodel dependent). There should be libraries of such projectors for the main tools and the main technical spaces. As an example, it should be possible to define a chain of MDA transformations with, inside this chain, some other operation performed by an external tool.

29. What test capabilities do you need an MDA tool to provide?

An MDA tool should be enhanced with model measurement tooling in general (and not only test capabilities: see the answer to question 36).
However, concerning more particularly test capabilities, it should be interesting to have a tool for making some constraints validation on metamodels (such as

checking if each abstract class of a metamodel has at least one subclass, etc). Constraint-based model construction would also be a useful facility.

30. When in the lifecycle do you wish to simulate and/or execute a model?

We already have implemented a model execution engine dedicated to the execution of model-to-model transformations (named the ATL virtual machine): it allows performing ATL transformations, which are models that conform to the ATL metamodel. This is the main kind of executable models we consider within our MDA approach.
Because transformations may be involved in each step of the lifecycle, there is no particular moments for "executing" transformation models. However, an MDA tool should keep track of model versions in a GMM, so that it can either automatically re-execute transformations when the source models have changed or suggest that they should be re-executed.
Note that many models are not directly executable but may be transformed into executable ones. This is a widely used technique within the AMMA platform.

2.2.6 Miscellaneous

31. Aside from customization of transformations (see question 25), what kinds of user customizations and user preferences should an MDA tool support?

According to our vision of MDE, the main customizations that an MDA tool should support concern the Domain-Specific Languages (or DSLs). Indeed, it should permit to create, design, add and manage new DSLs into the environment by using the providing DSL-specific facilities. In an ideal situation, the domains of application of the added DSLs should not be restricted by the provided facilities.
Another more advanced but very useful user customization should be to allow the connection of external tools directly into the environment by connecting to some provided "generic" interfaces. Thus, a user may customize its own modeling environment by adding some specific tools that fix its requirements and needs.

32. What capabilities do you require for MDA tools to support non-functional requirements in specialized systems? For example, embedded systems, real-time systems, fault-tolerance, high availability, etc.
33. What capabilities do you require for matching a PIM to an existing architecture?

In order to do that, we absolutely need two distinct items:

- A model of the platform (i.e. a Platform Description Model or PDM).
- A facility to weave the PIM to the PDM in order to later generate a PSM. Weaving PIM and PDM may be achieved with the help of a tool like AMW.

34. *What types of tools are part of your MDA toolchain? Please refer to the types of tools in the taxonomy, if possible.*

Within the AMMA Platform, several tools are provided for different purposes. Most of them are fully integrated, as plug-ins, into the Eclipse environment.
The KM3 language and the corresponding Eclipse textual graphical editor (TGE) can be considered as a modeling tool dedicated to the design of metamodels.
Although we do not provide a specific analysis "tool", a library of ATL transformations producing metrics (which are represented as models) on models is currently being developed. It will be used, in the future, as the analysis tool of the AMMA Platform.
The ATL component, including an Eclipse IDE with a specific view, editor, outline and debugger, is the AMMA transformation tool.
The AMW component, also including a complete Eclipse IDE, is designed for model weaving and in this way can be considered as a composition tool.
The AM3 component is our metadata management tool (Global Model Management or GMM tool in the context of the AMMA Platform). In our future vision of our platform, it would be the core component of the environment because it allows globally managing and providing modeling resources (such as metamodels, transformations, etc). Version control facilities should also be provided by this component.
The TCS component allows building "projectors" so that it can be considered as a transformation tool. Since injectors (which are projectors) allow creating models from already existing data in different grammar-based formats, TCS can also be considered as a reverse-engineering tool.
Concerning reverse-engineering, there is no single dedicated component directly integrated into the AMMA Platform. However, the MoDisco component (which has been recently created and is currently being developed in parallel with the AMMA Platform) will provide a set of tools for performing model-driven reverse-engineering.

35. *Do we need these to be in a suite of tools vs. one tool. Perhaps tooling environment.*

We do not believe in the idea of having one single tool for performing all the possible MDA operations and following all the step of a MDA process.
Our approach is based on the fact of considering a simple general environment (for the time being, we use Eclipse) in which we can connect the set of the tools required for our MDA processes. The tools which are integrated into this environment may come from tool suites (but not necessarily, it may also be single tools) and may be provided by different vendors or tool builders.

36. *What kinds of analysis do you need an MDA tool to support?*

Because in our approach of MDA most of the handled artifacts are models in the broad sense (i.e., terminal models or metamodels), we think that the main kind of analysis a MDA tool should concern the calculation (and of course

visualization) of metrics on models. An MDA tool should provide a set of tools for generating, analyzing and visualizing several kinds of metrics on the different models it produces and/or handles. The semantic of these metrics will depend on the kinds of model that are considered and also on their domain of application. This specific problem has already been mentioned in our answer to question 34.

3.0 Glossary

This last section is defines the specific terminology we use within this response. We focus more particularly on the mapping of this terminology to the OMG standard terminology, whenever possible.

Term	Definition
Technical Space	A model management framework, belonging to the "modeling world", with a set of tools that operate on the models defined within the framework. MDA is an example of Technical Space. XML and Grammarware are two other examples, based on different technical solutions.
System	A delimited part of the world (the "real world") considered as a set of elements in interaction. It can be represented by 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.
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).
Megamodel	A terminal model such that all its elements are 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.
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" 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…).
Weaving model	A model that contains relationships between elements of different models. It conforms to a weaving metamodel.
Core weaving metamodel	A core weaving metamodel defines elements that support basic link management, i.e., N:N relationships between model elements.
AM3	ATLAS MegaModel Management
AMW	ATLAS Model Weaver
ATL	ATLAS Transformation Language
KM3	KM3 (Kernel Meta Meta Model) is a neutral language to write metamodels and to define Domain Specific Languages (DSLs).
MoDisco	Model Discovery component dedicated to model-driven reverse-engineering (MDRE)
TCS	TCS (Textual Concrete Syntax) is a DSL for the specification of textual concrete syntaxes.