Broadening the use of MDA

© Copyright SOFTEAM 2005

Philippe Desfray (*see XI – About the Author*)
May 2006
Version 1.3

I. Presentation

Most OMG standards related to MDA are now available, such as UML2.0 (Unified Modeling Language), MOF2 (Meta Object Facility), QVT (Queries Views and Transformations), OCL (Object Constraint Language) or XMI1.2 (XML Model Interchange). MDA is a technology that is already supported by many tools, and that has been used within a large number of organizations. Many people from the software community know what the MDA technology is all about.

However, the use and recognition of MDA is still not widespread in software application developments.

What are the reasons for this? We are still at the early stage of the technology dissemination curve for MDA, but this fact alone does not explain the current status. We are convinced[1] that there is a decisive advantage in the use of an MDA-based approach in most software application developments: MDA represents an important step forward in terms of software quality and productivity. Nobody would contradict this statement, but the adoption curve is still low.

So what is missing? This white paper will present the expected features and facilities that should facilitate the dissemination of the MDA technology. It is based on work carried out on the MODELWARE European research project, and on the internal research conducted within SOFTEAM on a new breed of tool called "MDA Modeler".

[1] To quote the OMG MDA Guide: "Imagine if the construction worker could take his blueprint, crank it through a machine, and have the foundation of the building simply appear. We can take models, defined in standards like OMG's own UML, MOF and CWM, and automate the construction of data storage and application foundations. Even better, when we need to connect these "buildings" to each other we can automate the generation of bridges and translators based on the defining models; and when a new, more fire-resistant type of steel is invented, we can regenerate for the new infrastructure." This image lets us realize that we still haven't yet obtained the full benefits that we can expect from a generalized MDA usage.

II. The Modelware project

MODELWARE is a project co-funded by the European Commission under the "Information Society Technologies" Sixth Framework Program (2002-2006). The overall objective of MODELWARE is to improve productivity in software development. This objective will be pursued by contributing to the realization of the vision of model-driven development (MDD). MODELWARE participates in the realization of this vision by pursuing three main goals:

1. Improve MDA technologies through new modeling technologies and tools, and through generalized interoperability techniques between tools.
2. Define the appropriate software development processes that take advantage of the MDA evolution. MDA maturity levels are defined within this task.
3. Use these technologies and processes on large software application developments, in order to measure the ROI and to prove the added value of MDA with the new techniques and processes.

MODELWARE is a large research project (IP: Integrated Project) with 19 partners. It has already increased the knowledge and maturity of its participants. Lessons can already be learned from this project and its experiments.

MODELWARE contributes to realizing the vision of MDD by taking up three main challenges: (1) new development processes must be elaborated, with better and more mature tools to support them; (2) know-how must be reusable across development projects; (3) tool chains must be open to allow interoperability and substitutability between tools, thereby avoiding tool vendor lock-in and allowing existing models to be capitalized. In the MODELWARE MDD whitepaper (Modelware/I6.1), these challenges are described in detail, and the way MODELWARE addresses these challenges is explained.

In this whitepaper, we will show new tool support for MDA (related to challenge 1) and the notion of MDA Tool Component (MDATC) that relates to challenges 2 and 3. The current whitepaper mainly addresses issues related to the accessibility of MDA techniques, and the way in which these techniques can be made easier to use, so as to lower the MDA adoption threshold.

© SOFTEAM 2005

III. MDA as a technique to continuously increase an organization's software development maturity

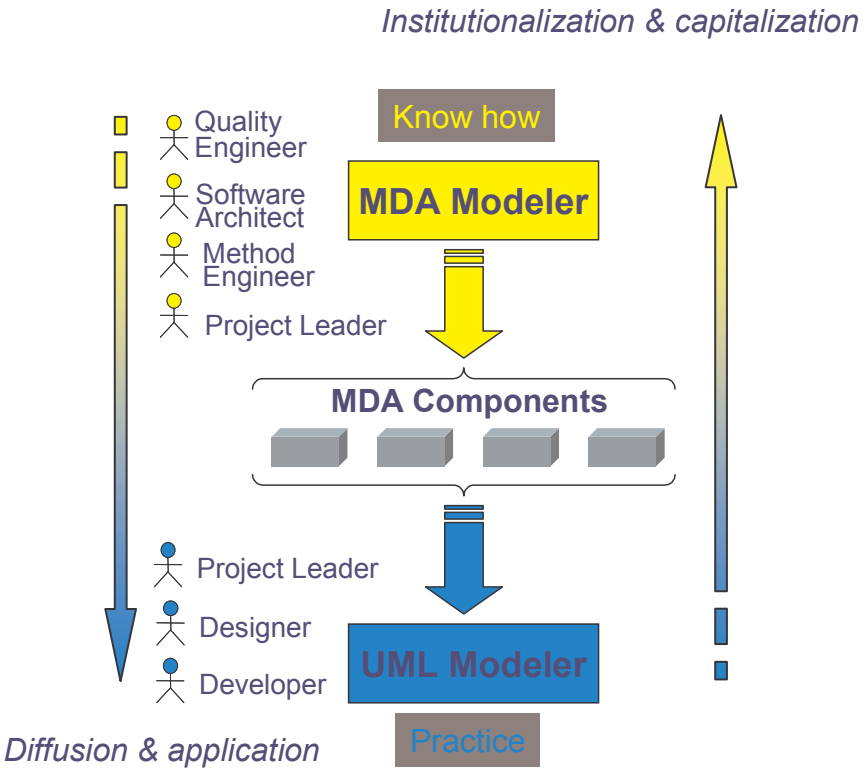

Figure 1 – *Using MDA, experts formalize their expertise and disseminate it to modeling tools on projects (illustration from the Objecteering model-driven development tool suite)*

Expertise is a key element in the success of software developments. This expertise ranges from process related expertise, methodology expertise and domain specific expertise, to architecture, platform and development expertise. Currently, some organizations manage this expertise through orthogonal structures that are responsible for its gathering, diffusion and correct usage. These orthogonal structures are for example Q&A departments, methodology departments or architecture teams. The issues with this approach are as follows:

- It is mainly empirical, based on documents, on the goodwill of developers and on good communication between teams.
- It is expensive to maintain such structures. The ROI is not easy to evaluate, and investment is only envisageable when a large number of development projects are repeatedly conducted.
- There are always communication issues between the "experts" and the developers. Developers are gaining new expertise, may not find the "expert rules" well formalized, practical or adapted to their context, and need to negotiate, in order to introduce new rules. The empirical aspect of the expertise leads to interpretation and consequently to confusion that is always a source of conflict.

The MDA technology provides a means of formalizing the expertise of an organization. Every viewpoint expressed during the development of a system can be supported by an adapted modeling language and specific support through model transformations and specific services. This expertise, once formalized using MDA techniques, then becomes a tool that

© SOFTEAM 2005

supports the appropriate modeling languages, guides and checks the construction of models, automates the production of other more specific models or artifacts, and provides additional functionalities related to the models.

Organizations are moving from an empirical set of rules to a toolset formalizing these rules and supporting developers in their application. Instead of constraints, developers will see these rules as a valuable help.

Expertise is growing through practice: every new project may add new expertise knowledge, for example by improving the development process or by providing new design patterns or optimized mappings from a model to the architecture. Participants in projects may use MDA technologies in order to implement new expertise that can then be gathered by the expertise structure, in order to enhance the expertise thesaurus. Expertise effectively becomes an asset of the organization, materialized in the form of MDA Components.

In effect, the MDA technology is a means of formalizing and managing the expertise of an organization and therefore of enhancing an organizations's maturity. It helps to constantly improve the development process, as recommended by the CMMI model (see *www.sei.cmu.edu/cmmi*). Reifying expertise as a toolset facilitates its dissemination, automates its application, guarantees its proper usage and diminishes the assistance and control work to be carried out by dedicated teams within an organization.

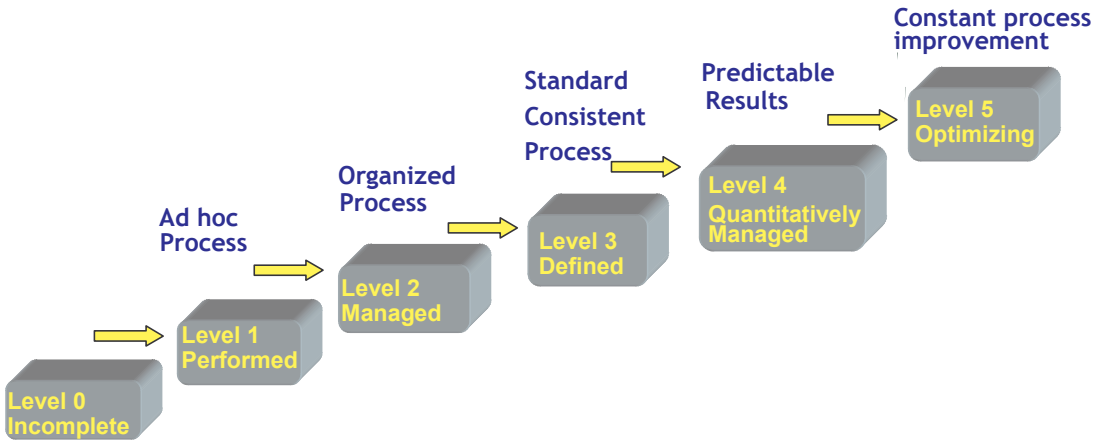

Figure 2 –*CMMI maturity levels*

© SOFTEAM 2005

IV. MDA implies development activity for its application

The effort required to establish a bespoke MDA approach is perceived as the major drawback of the MDA technology: MDA will help you have a dedicated modeling and software production environment … but will be costly due to the significant specific development required. This induces several negative aspects that prevent a global adoption of MDA:

- The preparation of the proper MDA environment requires a significant investment. Tools have to be acquired in order to set up the environment, and specific developments made in order to customize the environment. Studies on the best modeling languages to use and the most adapted transformations to employ must be conducted. So the message that managers often perceive is "pay first, get your ROI later".
- Customization development has to be carried out using proprietary or esoteric languages.
- The MDA approach is supported by a very complex standard, a global vision of which is not easy to get.

The first activity is to define the methodology that will be followed. One of its main aspects is to identify the viewpoints that are needed for the bespoke methodology. These viewpoints will correspond to CIMs (Computation Independent Models), PIMs (Platform Independent Models) and PSMs (Platform Specific Models). They will be materialized by the definition of dedicated modeling languages. These languages can be defined as specific dedicated languages, such as Domain Specific Languages (DSLs), or as an adaptation of the UML. For the former, a specific metamodel has to be defined, while for the latter, UML Profiles are designed. Defining a language is a complex task: semantics, constraints, notation, model organization and management rules, model exploitation services, wizards, traceability rules, methodological guidelines, consistency with the other modeling languages have to be specified.

Customization development then has to be conducted. The first job is to model the metamodel or profile of the modeling language, a task for which dedicated tools exist. A metamodel can be modeled using a CASE tool, and implemented using infrastructures such as the EMF open source framework, or commercial meta-CASE tools. Several UML CASE tools support profile modeling and implementation. A profile implementation is directly obtained from its model and thus does not require any further development work.

The metamodel or profile needs specific development to implement the modeling language constraints, wizards, transformations and all the functionalities that are required. Development techniques vary greatly from tool to tool, ranging from ergonomic tailoring wizards, to proprietary languages, Java programming or the use of standard dedicated languages such as OCL, QVT or MOF2text. In practice, only preliminary implementations of the standard languages exist, OCL being more mature but not used as an implementation language for that purpose.

This implementation work is indeed a development similar to traditional software developments, as it has to follow a development process itself, with the final test, validation, delivery and maintenance phases.

© SOFTEAM 2005

V. MDA Components: turning the expert's know how into an asset

How can the work accomplished when tailoring an MDA approach to a specific context be thesaurized, shared and disseminated? This is a major issue that has been addressed within the MODELWARE project. Currently, this tailoring work is based on techniques and languages that depend largely on the modeling tools. Bearing in mind that tools change with each new version, and that tool chains combining several tools tend to be specific and unique to each organization or project, the definition of specific modeling languages is very tool specific and not interchangeable. In addition, there are several types of artifacts to develop and package, in order to implement a complete tailored modeling language: Metamodels or profiles, model transformation rules, GUI, consistency checks, on line help, guidance wizards, generators and external resources such as icons or libraries are examples of what needs to be packaged.

For these reasons, the notion of MDA Tool Component has been defined within the MODELWARE project, and proposed for standardization by the OMG. An MDA Tool Component contains the material used to customize a modeling environment, in order to apply MDA to a specific domain or context. The MDA Tool Component support will unify tool tailoring techniques by providing a common tailoring and packaging unit. When the notion of MDA Tool Component is standardized, the connection between MDA Components and their host tools will be uniformly defined, thus allowing the customer to consider his MDA Tool Components as tool independent assets supporting his MDA specific approach. In addition, vendors will be able to market such components for industry-wide best practices. The goal of the MDA Tool Component technique is that:

- The constituents necessary to the tailoring and packaging of a specific MDA approach will be standardized, and therefore will be able to be used as a reference by those implementing an MDA approach.
- MDA Tool Component interoperability will be facilitated, thus making it easier to port a tool-tailoring development to another tool.
- MDATCs and their constituents will become more interchangeable: tools will be able to exchange MDA Tool Components and/or exploit parts of the MDA Tool Component's standard constituents.
- MDATCs will have the potential to be run on several different platforms, based on a standardized abstract architecture, which will provide standardized APIs.

Expertise will therefore become an asset that is tool independent and easy to deploy as a single packaging unit. MDA Tool Components will be a tool to share and reuse MDA approaches, and to combine them within different usage contexts. This will facilitate the dissemination of MDA, and decrease the entry cost by providing a wide set of "off-the-shelf" MDA tailored solutions.

© SOFTEAM 2005

VI. MDA has a much wider scope than code generation

Most of the time, MDA is illustrated and explained through code generation or more rarely design pattern automation examples. However, MDA is a technique that can provide support for the entire software development lifecycle, in particular to support the software process and methodology. MDA is a means of formalizing the viewpoints used during a development, as well as their supporting formalism with their consistency and methodological rules, their exploitation in terms of deliverables, and their orchestration and combination during the software development lifecycle.

With MDA, different supports can be attached to the selected modeling languages. For example, wizards can help at different modeling stages, model consistency rules can check the correctness of models, and methodological rules can state whether the model is complete for a specific purpose, well documented , not too complex or correctly traced to requirements or goals.

For a given modeling language, MDA can help state what the appropriate diagrams are, and what shall be presented within these diagrams. For example, a methodological approach can state that a package dependency diagram is required, showing only packages and their dependencies, that a general class diagram is required per package, showing only the package's classes and their interdependencies (associations, generalizations, etc.), and that a detailed class diagram is required per class, showing all the properties of the current class and its dependencies to other classes. Depending on the phase (analysis, design), these diagrams may not show the same level of detail. For example, at the analysis stage, visibilities are not presented and detailed operation signatures are not shown. Once these rules are spelled out, diagram construction can be automated, thus sparing the developer the burden of drawing squares and lines, thereby uniformizing the presentation of models per viewpoint.

Very often, software developers are not good at presenting and documenting their models. Documentation is a key methodological element that can be automated and that can greatly benefit from a uniformized diagram presentation.

The use of these methodological supports significantly enhances model quality, which is a big help to modelers. It is pointless to automate design patterns and code generation based on low quality models.

© SOFTEAM 2005

VII. Who is addressed by MDA?

The Modelware project has identified the following major roles involved within MDA activities:

- The *Platform Expert* has a detailed knowledge of the platform and can guide the proper usage of platform features for good quality software production. The platform can be a middleware, or a set of middleware libraries and patterns selected for a specific application.
- The *Language Engineer* is capable of creating bespoke modeling languages. He must be an expert in his domain or business.
- The *Transformation Specifier* defines the relationship between different view points and their modeling languages. He defines and automates part of the model-driven development process.
- The *Method Engineer* has to identify and orchestrate the activities necessary for the MDD software development project.

These roles correspond (with different names) to those presented in Figure 1, and highlight the fact that MDA addresses experts such as quality engineers, project managers or architects as the primary users, and that these experts need to be specialized in MDA related techniques.

Indeed, the main expertise should be technological, methodological or domain related, with some knowledge on how MDA can help disseminate the expertise. The main difficulty is that these precious experts are balking at using MDA techniques, given the frightening amount of standard documentation and technologies to absorb.

It is therefore absolutely essential for MDA adoption that the supporting tools provided hide this complexity, and reduce as far as possible the necessity of handling metamodels and programming program rules or transformations.

Developers, modelers and other participants in software development are also interested in MDA, but to them, MDA appears as a background technology that enables powerful model-driven tools to help them. They simply take advantage of MDA, as they would from any software tool, given that their practice has to be model-driven.

VIII. Techniques for easily and efficiently setting up an MDA approach

The most obvious advice for setting up an MDA approach is to reuse standards or existing modeling supports as widely as possible. Defining a new modeling language is a huge undertaking, which very often is not adequate in its first versions. UML is the primary candidate. UML can be easily adapted to specific contexts by defining profiles. This has the important advantage of re-using UML notation and structuring rules and of defining an extension to a widely known model, thus reducing training and evangelization work.

© SOFTEAM 2005

In addition, the OMG provides a growing number of existing profiles for specific domains, that can themselves also be extended by another profile for more focused applications. UML CASE tool vendors also provide profiles, which typically address widely used platforms, such as programming languages, RDBs or application servers.

Nevertheless, off-the-shelf solutions still need to be tailored to the context of each organization and application. The techniques described above can be used to facilitate this tailoring activity. Most of them are implemented by the innovative Objecteering MDA Modeler tool, while others are provided through open source solutions.

Modeling profiles

Modeling profiles requires only the knowledge of UML class diagrams in their simplest version. A profile is a kind of package, and a stereotype a kind of class. Defining an extension to a UML kind of element is obtained by modeling a stereotype that "extends" (a sort of generalization) that UML kind of element (called a metaclass). For example (see Figure 3), in order to define the notion of a "persistent" class, the "Class" metaclass from the UML metamodel will be referenced, and a "persistent" stereotype will extend it. Icons can be attached to that stereotype in order to adapt the notation. The profile is completed for example to define the notion of the "identifier" attribute. This simple model (the "persistency" profile) is immediately operational, and the persistency profile applied to UML models. Designers can then model persistent classes and specify which attributes are identifiers.

The UML metamodel is provided as a "readonly" reference model whose metaclasses are referenced for extension. Obviously, more complex cases require a deeper knowledge of the UML metamodel, and the level of required expertise may vary greatly.

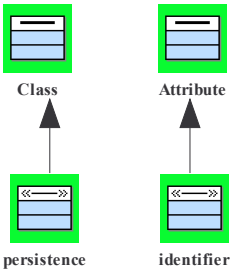

Figure 3 – *Simple Profile example*

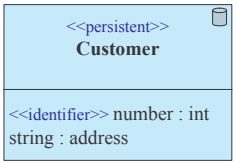

Figure 4 – *Example of model based on the "persistency" profile*

The "template" technique

"*I want to have a document chapter for every package, an overview chapter with a global presentation of the diagram, an overall description, a summary description for each class, and a subchapter for each class with detailed descriptions for each property and a detailed class diagram*."

© SOFTEAM 2005

This natural language specification for a kind of document expresses "metamodel elements and navigation". When UML users handle their models, they use the metamodel transparently. In most cases, knowledge of how to use UML is sufficient for handling the metamodel without knowing it, provided that wizards facilitate the expression of navigation.

The most natural way to specify a document is by defining a detailed table of contents in a generic way, using wizard facilities to designate the kind of content to be inserted. This is called the "template" technique that provides an intuitive way of formalizing how to obtain a specific result from a model.

Figure 4 shows an example of a document template. It expresses the target in terms of the kind of element to show, such as title, text, bullets or diagram, the content of the element, such as the detailed class diagram or the description text associated to a package, and the navigation to be carried out in order to access this kind of element, such as the classes of the current package. The use of this graphical tool does not require any specific knowledge of MDA and metamodel principles. Once the document template is defined, it can immediately be applied to a model part, in order to generate documentation.

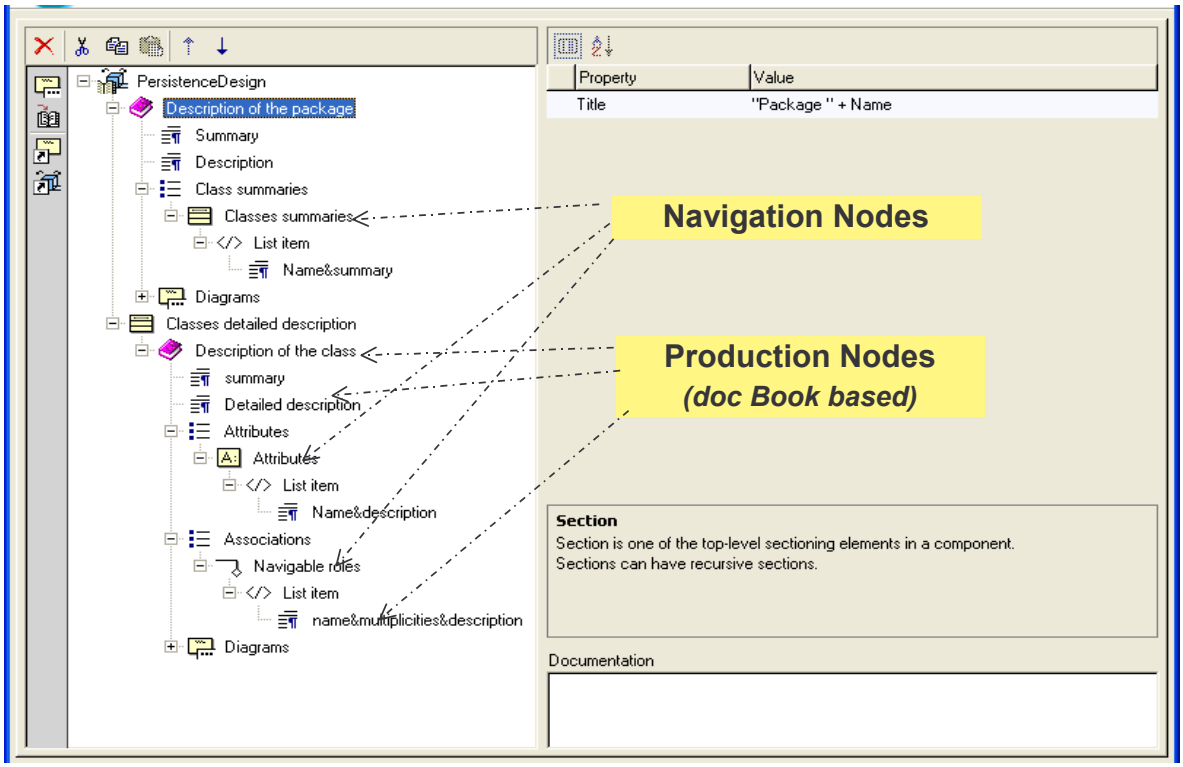

Figure 4 – *Document template example*

Code generation follows a similar principle. It is quite straightforward to imagine the tree structure of the targeted language. If we take for example the Java language, we can structure the code as a tree, with global comments, preliminary declarations (typically import), class declarations that embed attribute declarations, operation declarations that embed parameter declarations, and so on. The target structure is thus represented as a tree.

This technique has been reinforced using the traditional code "skeleton" technique. The code is directly entered within text areas, where escape areas express a string value to be substituted by contextual values. Merging both techniques makes them more powerful, and

© SOFTEAM 2005

avoids for example the insertion of structure expressions such as "while" or "if" within the skeleton. Template nodes can be referenced from skeletons as values to substitute.

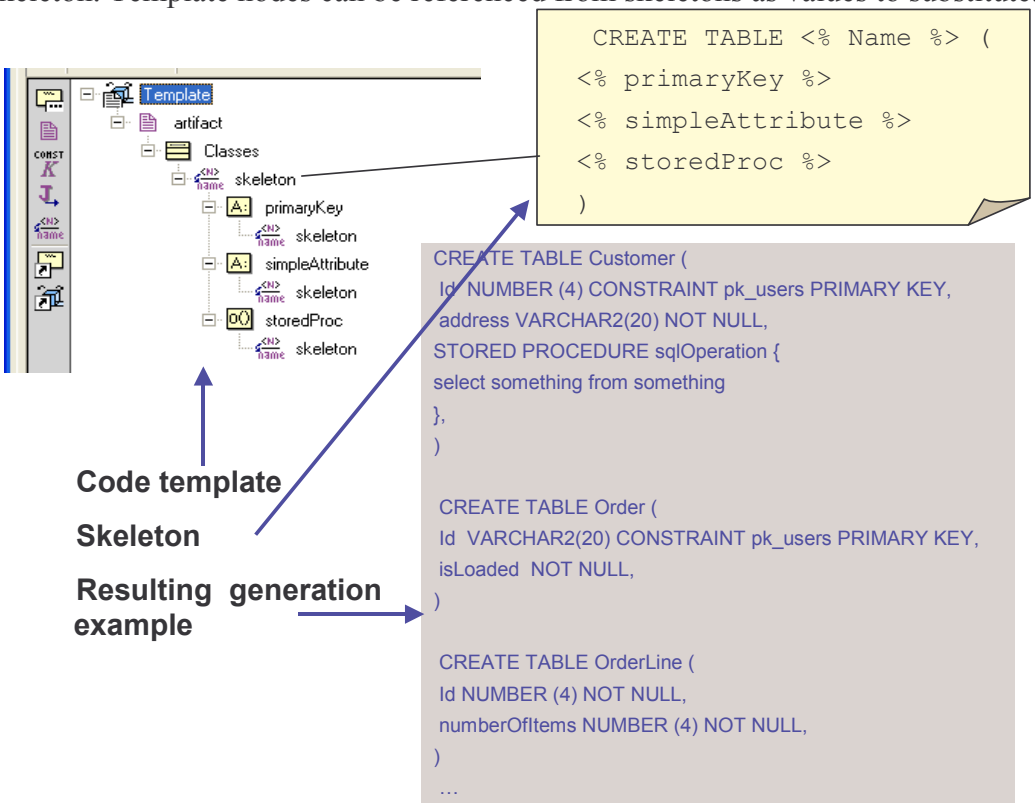

Figure 5 – *Code template for a trivial SQL generation example*

In a less intuitive way, but in an even more practical approach, diagram construction can also be expressed using the template technique. In this case, the tree expresses navigation nodes that define the hierarchical structure of the diagram (attributes are inside classes, which are in turn inside packages, and so on). A display node expresses that a model element has to be displayed, and its graphical attributes (color, presentation options) are specified.

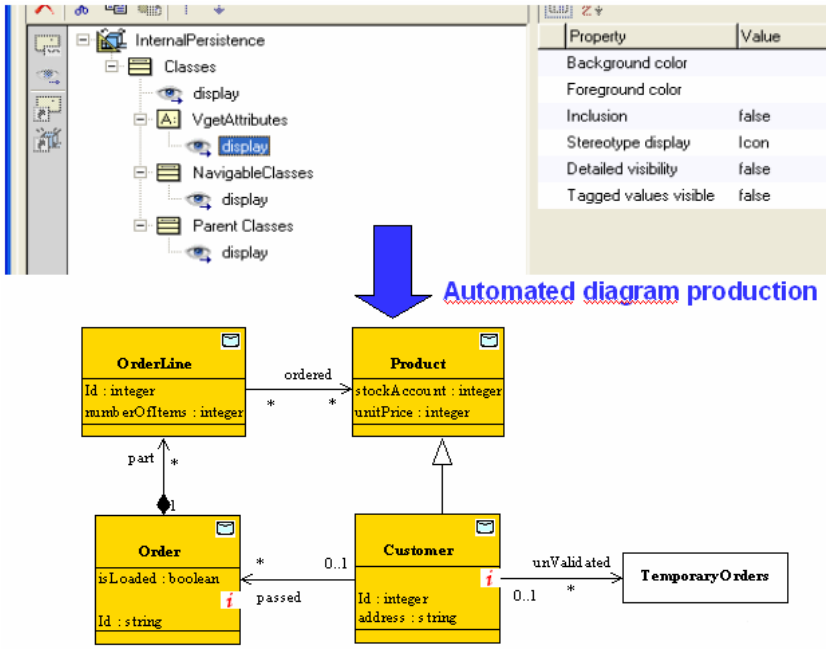

Figure 6 – *Diagram template example*

© SOFTEAM 2005

The MOF2text standard and its preliminary implementation

MOF2Text is a standard currently under construction within the OMG. Model to text transformations are tailored transformations that produce text output from models based on metamodels. The MOFScript language (one of the proposed responses to the standard, developed within MODELWARE - http://www.omg.org/docs/ad/05-05-04.pdf) has been designed with the aim of reusing QVT with a limited set of new concepts. Many concepts of QVT are not essential for model to text transformations. On the other hand, there are additional requirements in model to text transformations that are not covered by QVT, such as file output. The main aim of MOFScript is to provide a set of facilities for end users writing text transformations from models, including text manipulation utilities and simple control mechanisms. The MOFScript tool is an implementation of the MOFScript model to text transformation language. It is developed as an Eclipse plug-in, and supports parsing, checking and execution of MOFScript code. A binary download and a user guide are available at http://www.modelbased.net/mofscript/.

More details on MOFScript and MOF Model to Text can be found in the paper "Toward Standardised Model to Text Transformations"[2].

The package merge technique

The package merge concept was introduced in the UML 2.0 standard. This technique allows pieces of models to be defined and subsequently assembled at will, with different configurations. Models within different packages are assembled into a merger package that reflects the assembly of the selected packages. Features defined in different packages are aggregated in the merger package. Each merged package can be considered as a specific facet of the model that can be reused in a different merge configuration providing different models. This technique is useful for defining variation points within families of software. It has also an important usage field for defining and automating design patterns. A parametric package that represents the defined pattern is then instantiated in accordance with parameter values and merged with existing packages. Defining a pattern using this technique requires therefore only that a class diagram or a classic UML model be modeled, before annotating what is intended to act as parameters. This pattern definition then becomes a model transformation specification, by simply applying the "package merge" mechanism.

Using this technique, expressing model transformation rules only requires the usual modeling techniques, without metamodel concepts or model transformation languages having to be addressed. This technique is not as complete as a general model transformation language, but has a wide range of coverage that includes the design pattern automation area.

[2] Jon Oldevik, Tor Neple, Roy Grønmo, Jan Aagedal, Arne-J. Berre, Toward Standardised Model to Text Transformations, Lecture Notes in Computer Science, Volume 3748, Oct 2005, Pages 239 - 253

© SOFTEAM 2005

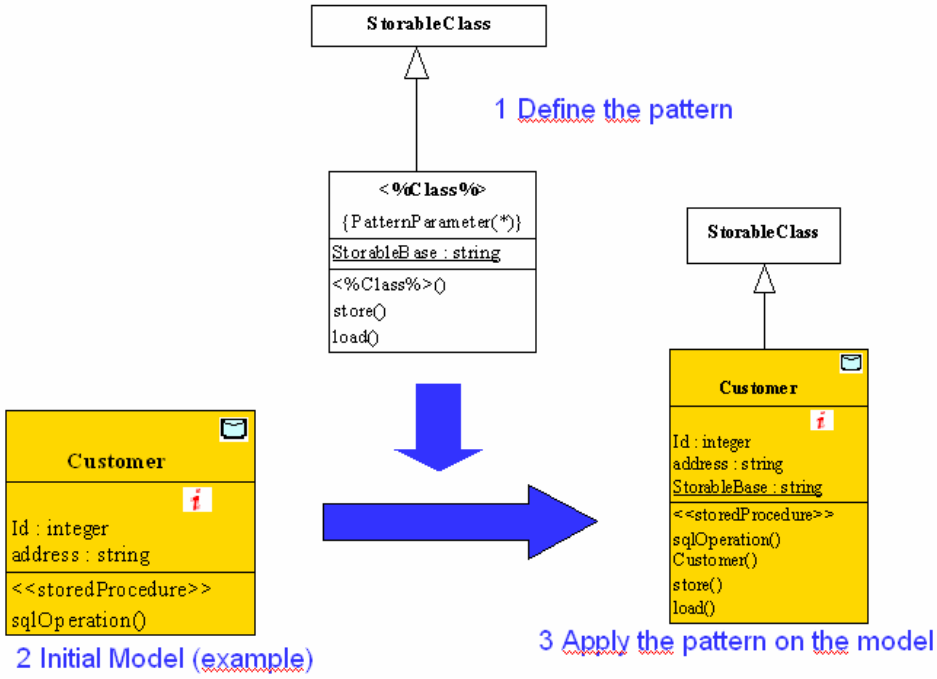

Figure 7 – *Example of pattern definition and application*

IX. Will MDA be solely based on UML?

Broadening the use of MDA requires the availability of existing off-the-shelf solutions. The standardization and uniformization of modeling languages is therefore a very important prerequisite. UML is the standardized and most widely-recognized modeling language. It is known to a very wide community of software engineers and taught to most software engineering students. In addition, its tailoring mechanism (Profile) allows it to be adapted to a very wide range of contexts and domains. Numerous dedicated profiles already exist, many of which are standardized. For new tailoring needs, many CASE tools on the market allow the definition of new profiles, which consequently inherit from all the modeling capacities from existing toolsets. Using UML profiles for MDD modeling means capitalizing on the considerable experience provided in the development of this language.

With its new version 2.0, UML has proven to be durable and has extended its modeling capacities to major modeling fields.

In addition, the profile-based extension mechanism is flexible and combinable. During a model lifecycle, new profiles can be applied, other profiles can be withdrawn and all profiles can be combined. This allows specific extensions to be combined, making it possible to share models between environments and groups that do not use the same extensions. This flexibility allows most organizations to develop multiple kinds of software, using a common modeling approach, with appropriate customizations on a case by case principle.

In this situation, only a compelling necessity can justify not using the UML language. However, UML does not aim to cover the entire universe of modeling needs. For example, specific models exist for mechanical, chemical or other domain specific problems that cannot easily be expressed with UML. For this reason, the MOF technology allows other specific

© SOFTEAM 2005

modeling languages, called Domain Specific Languages (DSL), to be defined. DSLs exist due to domain-specific expression needs, but also due to cultural usages or specific platform targets. As an important example, Microsoft is releasing specific DSLs on top of its platforms or languages such as C# or Biztalk in its latest Visual Studio version. Microsoft could easily have chosen a UML profile for that purpose. This choice is not affordable for most organizations.

In summary, the MDA approach does not specifically rely on UML and can be applied to every domain-specific field, but the straightforward default choice is to reuse the wide UML expertise and tooling when setting up an MDA approach.

X. References

- MDA Components: A Flexible Way for Implementing the MDA Approach - Reda Bendraou, Philippe Desfray and Marie-Pierre Gervais, in ECMDA, Nurember Nov 2005
- Blanc X., Gervais M.P., and Sriplakich P. "Model Bus: Towards the Interoperability of Modelling Tools", in Proc. of the MDAFA'04, Linköping University, Sweden, 2004.
- Desfray P. “Techniques for the early definition of MDA artifacts in a UML based development” Enterprise UML & MDA, London May 12 and 13 at: http://www.enterpriseconferences.co.uk/programme.pdf
- MDA Guide. "Model Driven Architecture (MDA)", OMG TC document ormsc/2001-07-
- 01, July 2001, at http://www.omg.org .
- MODELWARE Project, at http://www.modelware-ist.org
- the MODELWARE MDD whitepaper (Modelware/I6.1)
- What Senior Managempent Needs to know about the value of MDA – Louis J. Eyermann III, (http://www.omg.org/news/whitepapers/OMG-MDA-Final-Ariticle_June-2004v6.pdf)
- Roles in the MDA Process: MDA will make developers more productive, not redundant : Stephen J. Mellor, Andrew Watson. (http://www.omg.org/registration/registration-roles_mda.htm)

XI. About the author

Philippe DESFRAY, co-founder and technical director of SOFTEAM, is an internationally renowned expert on models and methods, notably those based on UML. Creator of the "Classe Relation" object-oriented method in the 1990s, Philippe DESFRAY has published three books, in particular "Object Engineering - The fourth dimension", published by ADDISON WESLEY in 1994, and has driven the development of the UML tool suite "Objecteering" . Forerunner of "MDA[3]" technology, Objecteering introduced support of this approach in 1994. Since 1994, Philippe Desfray has represented SOFTEAM within the OMG, where he has actively participated in the development of several standards, most notably

[3] "Model Driven Architecture": approach and technology at the base of new OMG standards.

© SOFTEAM 2005

UML. His continuing work on model-driven development has led to him to participate in the definition of the UML standard from the very beginning, by directing the definition of new notions such as "UML profiles", and to develop support of these notions within Objecteering. At the head of extensive R&D activity within SOFTEAM, and in partnership with large European organizations, Philippe Desfray is committed to directly applying results across the entire range of SOFTEAM activities: consulting, training and support through the Objecteering model-driven development tool suite.

© SOFTEAM 2005