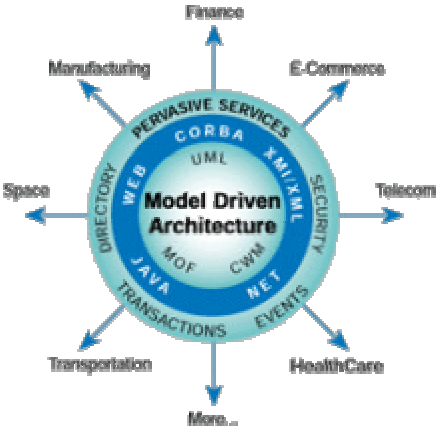
MDA – When a major software industry trend meets our toolset, implemented since 1994.

By Philippe Desfray – VP for R&D

http://www.objecteering.com

http://www.softeam.org



Presentation

MDA (Model Driven Architecture), defined and supported by the OMG (Object Management Group), defines an approach to IT system specification that separates the specification of system functionalities from the specification of the implementation of these functionalities on a particular technological platform.

This approach places the emphasis on models, provides a higher level of abstraction during development, and enables significant decoupling between platform-independent models (PIMs) and platform-specific models (PSMs).

Since its creation in 1989, SOFTEAM has been conducting research & development into model driven engineering, and has provided a tool supporting this approach since 1991. A specific technology named "hypergenericity", created in 1994 and applied to UML since 1996, has since evolved to integrate and enhance UML profile technology. UML profiles are now part of the OMG UML standard, and will be improved in the UML 2.0 standard. These improvements will be close to the UML profile improvements that SOFTEAM has been implementing in the Objecteering/UML CASE tool since 1994. Since this date, SOFTEAM has gained in experience, constantly providing an ever-improving toolset and range of consultancy services, designed to supply the customer with dedicated MDA approaches and implementations.

Based on UML, UML profiles and the Objecteering/UML Profile Builder toolset, this technology is now mature and has been applied to more than one hundred projects over seven years.

This white paper will present how MDA can be successfully supported by a proven toolset in a highly efficient way, based on the UML profile technology extended through advanced features such as model transformation techniques and traceability management. Certain pragmatic issues regarding how to establish an MDA approach will also be addressed. All these leading to true, fully deployable and reusable “MDA components”.

Benefits of the MDA approach

MDA necessitates the formalization of knowledge involved in software development. This leads to a thorough understanding of the techniques, architectures and methodologies applied, and thus provides better control of the organization's know-how. In particular, it guarantees that no knowledge is lost, and that a large organization is able to apply proven practices in a regular and systematic manner. This is the key to software quality and development productivity.

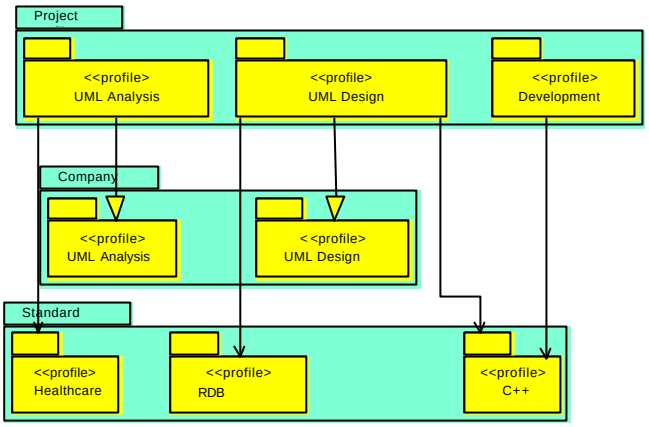


Figure 1 – *Model of the UML profiles used in an organization's projects*

A well established MDA-based organization can provide a model of the development know-how involved in the organization, and can maintain and automate a thesaurus of know-how for all upcoming developments. Figure 1 shows an example of a model of UML profiles supporting a specific MDA approach. The UML profiles used in a specific project are based on the UML profiles defined at company level, which are in turn based on standard UML profiles. In addition to this global picture, model checking, presentation and transformation rules can be attached to every UML profile, helping to guide and automate the developer's work. As presented in Figure 2, an organization can respect the typical structure of a "know-how" team, which maintains and improves architectural, methodological and software process knowledge through the selection, provision, improvement and support of UML profiles, and a development team, whose development tasks are automated, supported and checked by the UML profiles of the "know-how" team. This approach factorizes and catalogs the organization's software development skills.

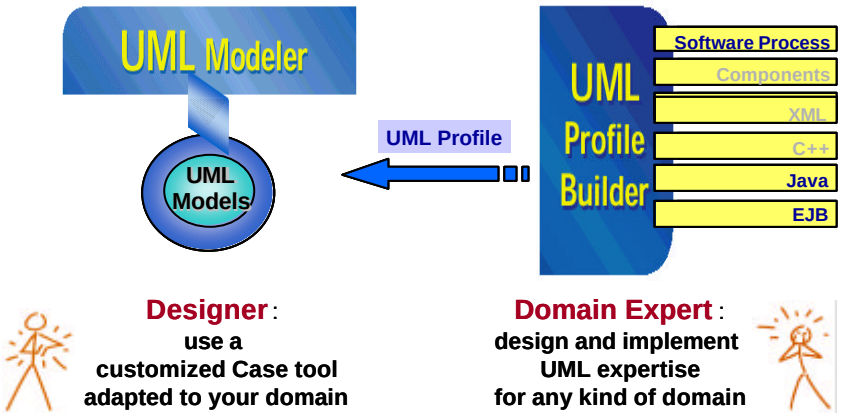


Figure 2 – *Example of an MDA based organization structure*

Obviously, the ROI (return on investment) is high when an organization develops families of applications related to similar domains or contexts.

Pragmatic issues related to MDA

The establishment of an MDA-based approach in an organization requires a high degree of formalization of the different kinds of models, platform-specific middleware and architecture and methodology being used in the organization. From this knowledge, the different types of important model are defined, mapping between models and towards different targets is formalized, and guidelines are automated, in order to improve efficiency and guarantee that the process is properly followed.
Standards, such as the UML standard and UML profiles for different targets or domains, help in this definition. However, these standards are not sufficient in themselves and need further work, so as to specialize and closely adapt them to the organization's context.

Once this important step has been completed, there remains a significant amount of implementation work. All this formalization customization and tool automation work has to be put in place and used by all those involved in application development. More than just simple training, this work needs to be evangelized. People have to be truly convinced and the ROI has to be proven to all.
Some aspects such as *practicability* have to be worked out in detail. How "comfortable" is it to use the approach and the toolset for every phase? Can the approach be used at every stage, even during the final stages of development, such as testing, deploying and debugging?
Cultural aspects should not be neglected. For example, "black box" generation from a model to a target is not often easily accepted by developers. This involves a well established model/target mapping customization process, including model transformations that show transformation steps that are not "obvious" (obvious meaning here "widely accepted") to the general community.

It is therefore crucially important to have an efficient tool, capable of rapidly implementing a robust and ergonomic MDA-dedicated support toolset, which end users will easily adopt.
It is also important to have a pragmatic approach to the introduction of MDA into an organization. MDA should be put in place iteratively, along with approaches and tools that provide an easily-perceived ROI and fast results. Once an MDA approach has been initiated, improvements made to the approach are continuously introduced, in order to minimize development efforts, increase consistency check levels, accelerate the learning curve for newcomers to the team or improve the efficiency of model or code transformation.

Basic principle: Using UML-based models for PIMs and PSMs

The definition of a specific model is a difficult and lengthy task. Model evangelization is even harder. Additional efforts, such as connecting the model to third party models, teaching it to developers or communicating it to external players, must also be carefully considered.
This is why it is so important to stick to existing standards as far as possible, and to introduce only specificities which add a proven value to the business and development process.

We consider that all PIMs and PSMs should be based on UML. In particular, existing UML profile standards published by the OMG or by other standardization organizations should, as far as possible, be seen as a foundation for an organization's PIMs and PSMs. Table 1 presents existing or upcoming standard UML profiles. It is important to reuse on-the-shelf UML profiles which already implement standard UML profiles, or which propose other extensions for domains not covered by existing standards. Table 2 presents a set of UML profiles supported by Objecteering/UML as "on-the-shelf products". However, there still remain highly important areas which are poorly covered (or not covered at all) by standards. For example, UML/C++ mapping or UML/Relational Database mapping are not specified by standards.

Name	Current state	Description	Organization
UML Profile for CORBA	Available	Describes how to model CORBA with UML.	OMG[1]
SPEM (Software Process Engineering Management)	Available	Provides a language for modeling software processes.	OMG
EDOC: UML Profile for Enterprise Distributed Object Computing	Voted, ftf[3] in progress	Targets component based modeling.	OMG
UML Profile for Scheduling Performance and time	Voted, ftf in progress	Used for "real time" applications.	OMG
UML profile for EAI	Voted, ftf in progress	Enterprise Application Intergration is the key to large IT systems.	OMG
UML profile for QOS and fault tolerance	Responses awaited		OMG
UML profile for EJB	Available	JCP standard. Expresses how to model EJB with UML.	JCP[2]
UML testing profile RFP	Vote expected in June 2003	Profile for modeling tests at UML level.	OMG

Table 1 – Standard UML profiles already or soon to be proposed

Every time UML has to be used "for something specific", a UML profile can be defined for that specific purpose. This has a wider scope than simply targeting a specific kind of model. For example, a specific configuration or version management policy can be realized through a dedicated UML profile, or indeed utilities can be realized, such as the underlying "Macro" UML profile. Objecteering/UML makes extensive use of this capacity to manage any kind of specific variation to the central UML Modeler tool.

1 OMG : Object Management Group
2 JCP : Java Community Process
3 ftf: finalization task force

Name	Current state	Description
UML Profile for C++	Available	Provides complete design patterns and C++ code generation.
SPEM (Software Process Engineering Management)	Soon to be available	Supports software process modeling.
EDOC: UML Profile for Enterprise Distributed Object Computing	Will to be covered with UML2.0	Supports EDOC modeling.
UML Profile for EJB with extension subprofiles for Websphere, Weblogic, iPortal	Available	Transforms a model into EJBs, maintains consistency between the classes constituting the EJB, and generates all the code, including deployment information. Manages reverse engineering and permanent consistency code/model.
UML profile for CMS, with extension subprofiles for ClearCase, CM Synergy , CVS and SCC (which targets VSS, PVCS and other CM tools)	Available	Enterprise Application Integration is the key to large IT systems.
UML profile for CORBA	Available	Includes a standard CORBA UML profile, with a complete CORBA/idl code generation support.
UML profile for tests with the "test for Java" sub profile.	Available	Test modeling is supported by this UML profile. From a UML model, this profile generates a test model framework and assists in modeling additional tests. The "Tests for Java" sub profile generates all the Java code of the test application, using the "JUnit" open framework.
UML Profile for design patterns	Available	Automates many of the E. Gamma & all design patterns, through model transformation techniques.
UML Profile for SQL	Available	Transforms a logical UML model into a physical RDB model, maintains consistency between these two models, and generates all the final schema definition SQL from the physical model.
UML Profile for document generation	Available	Generates documentation for MS Word or HTML, and proposes two standard templates: Analysis and Design. Includes a customization tool: a documentation template editor used to define new templates.
UML Profile for Java	Available	Generates all the Java code, carries out reverse engineering, maintains consistency between the generated code and the model, automates the Java specific pattern (idioms).
Macros	Available	Utility that allows you to write Macros in J using the UML Modeler tool, in order to make certain requests to the model or even certain model transformation.
Metrics	Available	Measures the metrics of a model, according to well known OO metrics, and provides a detailed report.
Reverse COM	Available	Reverses COM interfaces into UML.
VB	Available	Generates VB code from the model and maintains model/VB code consistency.
UML Profile for XMI	Available	Even XMI, the standard UML exchange format, is generated or reversed using Objecteering /UML Profile technology.
Xtreme programming	Available	This profile supports the Xtreme programming approach, by providing CRC card and story support, a consistent model and application naming support. Its Xtreme programming support is highly efficient when used in conjunction with the "Tests for Java" sub profile.
Process Wizard	Available	This profile drives UML modeling, by transforming the model throughout the lifecycle, providing hints and examples, and checking the completeness of a model.
UML Profile for requirement analysis	Available	Requirement analysis consistent with UML, with efficient traceability management support .

Table 2 – Profiles supported by Objecteering/UML, and their connection to commercial "MDA components"

Model transformation levels and techniques

Literature on MDA often mentions model transformation or model mapping, both of which are central to MDA. Once the different models have been specified inside an MDA approach, model mapping and transformation provide the dynamics of a development lifecycle based on this MDA approach. Model transformation needs to be supported by a specific technique, powerful and flexible enough for a wide range of situations. We will characterize different kinds of transformations, and detail why they are required.
Objecteering/UML Profile Builder provides a language called "J". "J" is a simplified form of Java, very flexible and dedicated to model transformation, in which model navigation principles close to those of the OCL (Object Constraint Language which is part of the UML standard) are provided. "J" is the language used to add behavior to UML profiles.

1 Model/code transformations

This is a frequent situation. The model is close to the technical target, which frequently has a syntax-based language, and is directly transformed into the syntax of the target. In this case, the model is a PSM (platform-specific model). Each feature present in the model has a connotation specific to the target. For example, the model uses generalization in a way that can be implemented in the target. Notions can be present (such as "UML interfaces" for Java or CORBA) or hidden (such as "UML interfaces" for C++). The UML model has been extended by a UML profile, which maps the model more directly to the target. For example, the notion of the "virtual" C++ method will be supported by a tagged value or a stereotype extending the UML notion of "operation".
There is frequently no metamodel of the target, the equivalent to the metamodel being the "BNF" (Bachus Naur Form), which describes syntax. (One can argue that a BNF is equivalent to a metamodel, but practicability is not straightforward, due to the necessity of producing a syntax form). Translation is very frequently formulated by rules which express how each model construct is translated into a specific syntactic construct.
In this case, transformation techniques cannot be easily defined as a metamodel to metamodel transformation, and need a specific language such as "J" for Objecteering/UML to express the translation rules.

2 Model/Model transient transformations

Transient transformations are carried out for PSMs, where transformations need no human interaction, and correspond to well known and well accepted patterns. They implement low level patterns, which are frequently language axioms. For example, transforming an association into an accessor operation is a common practice for implementing associations in an OO language. Java, for example, has many such axioms described in the reference documents. This kind of transformation frequently occurs transparently just before code generation. For example, associations are transformed into accessor operations, which are in turn transformed into code. These transformations are called "transient", because there remains no storage of the transformed model. The transformed model is never presented to the developer.

3 Internal model transformation (enhancement of a model)

Once the transformation has been applied to the model, the model itself is enhanced. This is not the creation of a new model from an older one, but rather the enhancement of the same model. Generally, this kind of transformation happens in the same kind of model (PIM or PSM). In most cases, it concerns PSM models, where, for example technical patterns are applied, in order to automate systematic model constructions. This kind of transformation is typically applied when there are incomplete transformations, which necessitate human intervention to add information that cannot be automatically deduced. The state pattern enhances the properties of a class as shown in Figure 3, but requires further specification of the implementor.

Transformations can be specified by formalizing the mapping between two metamodels, or in our approach between two different UML profiles. Dedicated languages can support this, as "J" typically does. The language has to formalize the notion of the "transformation transaction", in order to be able to "do and undo" the transformation. Furthermore, traceability often needs to be supported. This will be developed later.

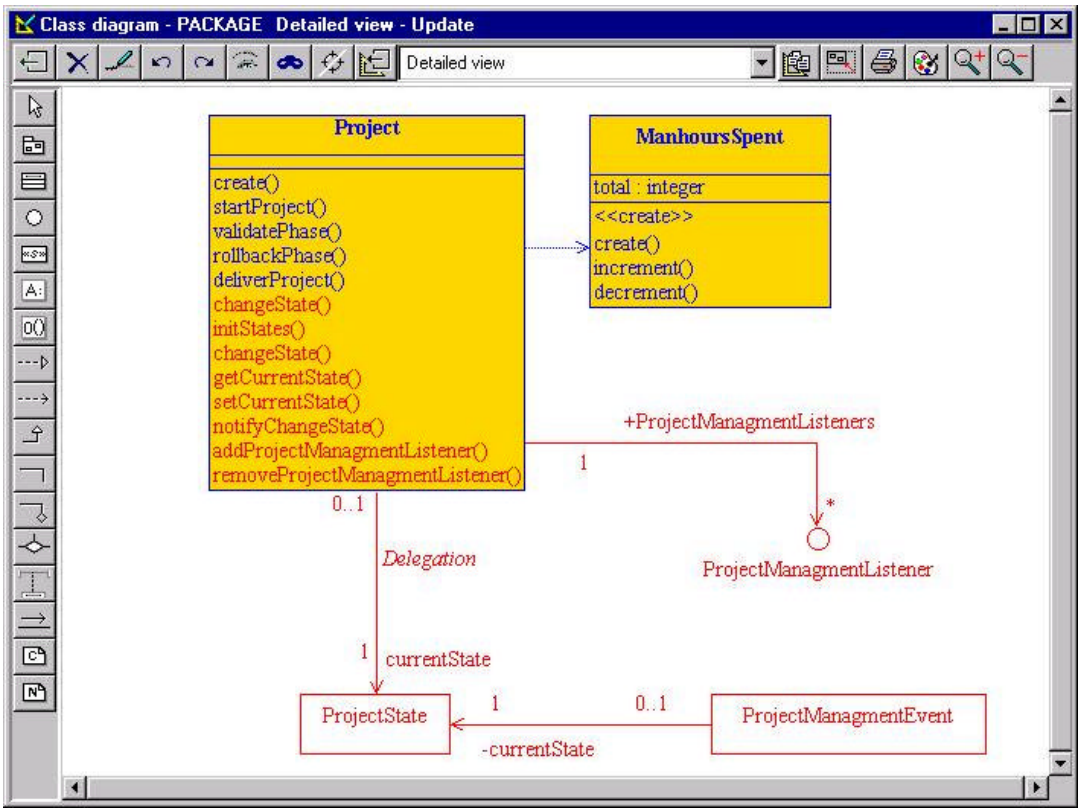


Figure 3 – *The State pattern, transformations (in red in the picture) enhancing a class associated to newly generated classes*

4 Transformations between two different models (generation of a new model)

This case is close to the previous one, and uses the same techniques. However, new independent model elements are created in a separate model. This happens frequently during a PIM to PSM transformation, where from a model specifying a high level of abstraction that does not depend on a platform, a new model specific to a platform and to recommended patterns for implementation is created. In this case, traceability management becomes crucial, in order to be able to maintain consistency between changes that may occur at both levels of abstraction. It is preferable to use the "internal model transformation" approach as much as possible, due to the more complex traceability management of the "transformation between two different models" approach.

An example of this is the transformation of a logical class model into a relational database physical model, as illustrated in Figure 4. In this example, traceability links exist between the two models.

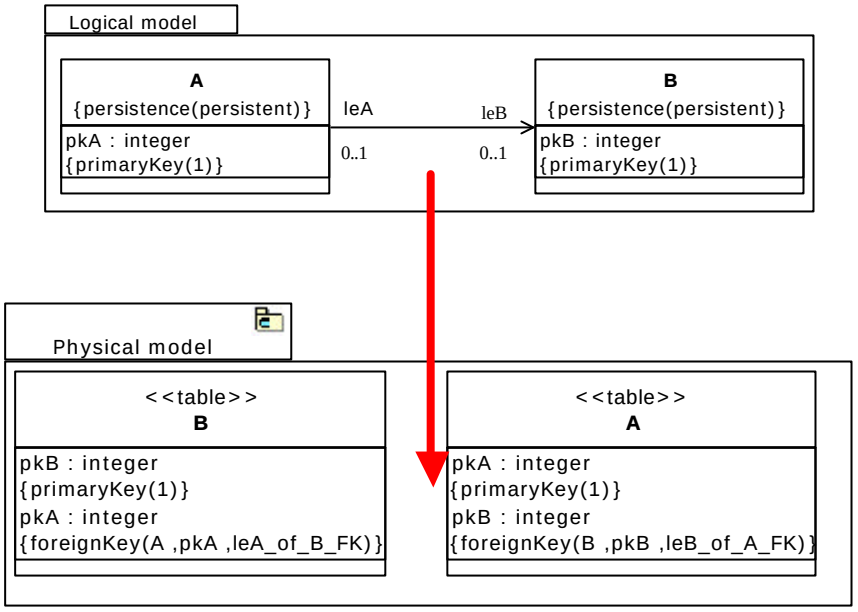


Figure 4 – *Transforming a class model into a RDB physical model*

5 Transformations as guidelines or methodological supports

This is not a new kind of transformation, but instead another important use of the model transformation technique. Until now, model transformation was only mentioned as a tool for automating software development, for example, generating code or implementing design patterns.

Model transformation can also be used to automatically generate diagrams from a model. A diagram has its own metamodel, and this case corresponds to transformation mode number three. Another example is the support of naming conventions: the use of uppercases, specific separators or plural mode in some cases can be enforced by checking and transforming every name in a model. This is an illustration of transformation mode number four.

As illustrated by these two examples, methodological modeling recommendations can be automated, enforced or guided, using the model transformation technique. Naming rules or diagramming rules such as "*for each package in a model, a diagram showing external dependencies and a diagram showing the elements belonging to the package should be provided*" can be automated, allowing developers to avoid having to carry out the tedious drawing of numerous squares and lines. This helps to obtain "normative diagrams" leading to "normative documentation".

Traceability, a key issue in MDA - two types of traceability

The MDA approach recognizes the necessity of having several kinds of model, from early requirements right through to implementation. This raises the even more compelling issue of how to manage traceability between all these different models, as well as implementation. Traceability management is absolutely essential in an MDA-based approach. If it is not carried out in an automated fashion, models and implementation will inevitably become inconsistent. At this stage, the consequence is that models have no value (they are simply not true), and that the only valuable information is implementation. Welcome back to the good old days!

1 Identifying elements

Supporting traceability necessitates the identification of the elements to be traced. Identification has to be based on the existence of an element, and never on its properties. The properties of an element are subject to change and cannot be used as the foundation for traceability. Typically, using the name of an element as an identifier (as too many CASE tools do) is a spurious approach.

For example, Objecteering/UML (in both the Project Edition and the Enterprise Edition) provides a universally unique identifier for each model element, and manages import and merge features based on this identifier. This unique identification is the key to successful traceability management.

2 Intrinsic model traceability

UML and UML profiles specify links between related model elements that provide a natural and transparent traceability to manage. For example, a message can be related to an operation, or a link to an association. Tools should transparently maintain consistency between these links and the information of the related element. If this very basic service is not provided, then an MDA approach cannot be seriously envisaged.

3 Traceability inside or between models

There exist numerous cases where traceability links should be explicitly specified between different model elements. For example, traceability links can exist between a Use Case and the classes which implement it, or between a requirement and the model elements related to it.

Some traceability links, as in the two examples below, are defined manually by modelers. However, in an MDA approach, many traceability links will be automatically defined and managed. Their role is to maintain the correspondence and consistency between the different views related to the different PIM and PSM models.

Objecteering/UML provides a general purpose traceability link, as specified by UML, and automatically manages this link and its consistency in several generators, such as the RDB physical model generator or the EJB generator. In a future requirement analysis UML profile, Objecteering/UML will allow modelers to specify their own traceability links and generate reports on them.

4 Traceability between the model and the implementation or other external elements

When inputs to the model or outputs from the model are not model elements but external artifacts, such as documents, source code, relational tables or XML files, traceability becomes more difficult to manage, since one part of the information is not controlled inside the modeling tool. At this point, it is extremely important to have the universal identifier of model elements that can be attached to pieces of the external artifact, in order to maintain traceability, whatever evolutions may occur in the traced elements.

For example, Objecteering/UML manages the consistency between a model and generated source code using this technique, which guarantees complete consistency between the traced elements throughout the entire development lifecycle. This technique should be distinguished from the reverse engineering technique, which creates a physical model from source code, and is obligatorily based on element names. Objecteering/UML supports the two techniques but for different purposes.

Parameterizing the MDA global infrastructure

It is also important to find "on-the-shelf" MDA components, which are already acceptable for the organization context and can be customized for accurate adaptation to it.

Customizability of the toolset is the key to the continuous improvement of the overall approach. Being able to customize on-the-shelf MDA components provides the possibility of buying on-the-shelf know how, easily introducing it into an organization and adapting it to the organization's context. Families of organization contexts constituting different customizations of a given MDA component can even be established.

On-the-shelf MDA components exist for widely used software targets, such as C++, CORBA, Java, EJB, RDB or XML (see Table 2). There are also MDA components supporting different software disciplines, such as requirements analysis or application tests.

Parameterizing a UML profile is supported by the UML profile generalization mechanism. The Objecteering/UML "J" language supports UML profile generalization and the usual (meta) class generalization through a dedicated double lookup mechanism. This provides a very straightforward way of parameterizing UML profiles, by creating sub-profiles without impacting the original code.

Practicability - filtering and editing a specific PIM or PSM

Each UML profile which supports a PIM or a PSM should provide a graphical user interface dedicated to this PIM or PSM. For example, a Java designer needs to see a Java specific interface (as in Figure 5), rather then a generic UML interface, for which he has to fetch and select the appropriate extensions. He should be guided in his platform-specific model, in order to build only platform-specific legal models and be assisted when automation can occur.

Each PIM- or PSM-specific UML profile should be considered by the user as being a tool targeted to specific modeling aspects, and not as a generic UML tool which has some extensions to help with mapping.
For this reason, Objecteering/UML provides the possibility of customizing the GUI, in order to filter out inappropriate UML notions and provide target-specific editors.

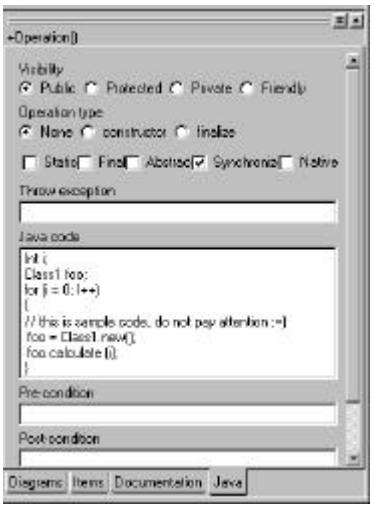


Figure 5 – *The Java specific property box provided by the Objecteering/UML to Java profile*

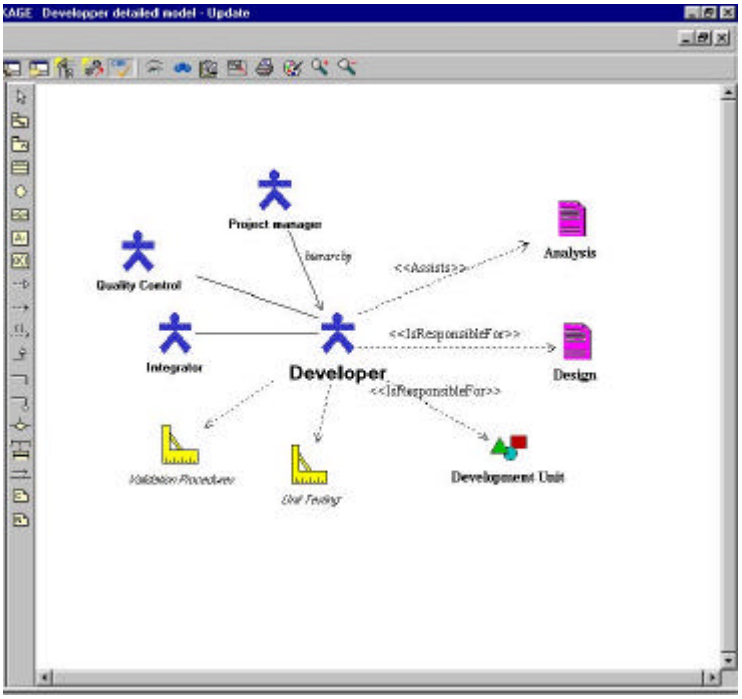


Figure 6 – *SPEM model presenting "Roles and responsibilities" in a specific software process*

Industrializing UML profiles - the MDA component

UML profiles cannot just be a group of extensions (tagged values and stereotypes), in order to be an industrial solution supporting a specific PIM or PSM, guaranteeing that the models built are consistent, and providing mapping to technical platforms or other PIMs or PSMs. A UML profile must be supported by a dedicated GUI, specific services (wizards, model transformations, code generation, etc), dedicated consistency checks and other features, such as on-line help, installation procedures, and so on. All these elements should be packaged into a deployable MDA component. UML profiles constitute the model of the MDA component, and the MDA component itself is the deployable unit which can be delivered to "users".

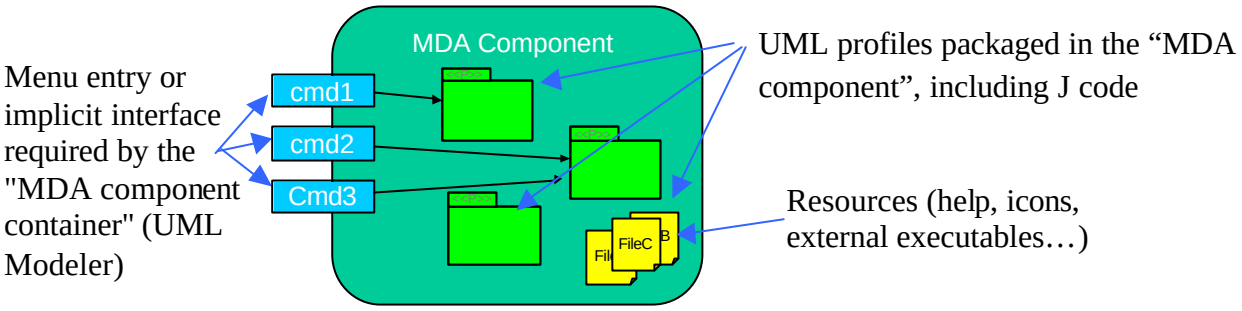


Figure 7 – *Building "Profile components" using the UML Profile Builder tool*

The Objecteering/UML Profile Builder tool makes this possible. UML profiles are packaged into "MDA components", accompanied by behaviors, GUI elements and software resources, in order to be easily deployable in the MDA component container which is Objecteering/UML Modeler. When an “MDA component” is loaded into a UML Modeler tool, “UML modeler” is transformed into a tool dedicated to a specific target or purpose, through specific extensions, a specific GUI and specific behaviors. An "MDA Component" can also be deployed in the "Personal Edition" of UML Modeler, which is a freeware tool. MDA components can be published in an open source mode, on the “http://www.umlopenedition.com” website.

MDA components conform to true component-based definition: their deployment means and user manual are included, they provide entry points used to access them and they have to implement interfaces required by the Objecteering/UML Modeler tool, in order to provide various sets of services.

Once deployed in Objecteering/UML Modeler, Objecteering/UML recognizes the MDA component and adapts itself as instructed by the component in question.

The MDA component as provided by Objecteering/UML profile builder supports versioning management: once an MDA component has been deployed on the Objecteering/UML modeler tool, a new version can be deployed again, and replace the previous one for all end user models extended by the contained profiles. This feature, based on the universal identification mechanism is strategic for end-user who never has to “re-annotate” their existing extended models.

Objecteering/UML MDA components are platform independent. Objecteering/UML modeler can load the same MDA component on all its platforms : Linux, other Unix platforms (Sun OS, etc.), and Windows.

Objecteering supports extensibility mechanisms, based on profile generalization. End users can therefore extend existing MDA components on the shelf, in order to customize improved implementations for their specific MDA approach.

In conclusion, a strong industrial and open implementation of MDA components is supported by Objecteering/UML: Objecteering/UML profile builder is the MDA component development environment, and Objecteering/UML modeler is the platform independent component container, which exists in a free form and in a commercial form.

A repository-centric approach

Having a set of consistent PIMs, PSMs, source code and other artifacts necessitates the use of a tool based on a repository. The repository of the tool must be accessible to multiple users, and must be coupled with version management tools. It must also have a centralized access point and administration tools. It is just not sufficient to use files spread over several file systems.

Objecteering/UML provides such a repository, based on a dedicated optimized database management system. This repository provides facilities for importing parts of models and for managing group work, as well as for distributed development.

Conclusion

UML profiles provide a flexible and very useful technology for supporting the MDA approach. The Objecteering/UML toolset provides the necessary extensions to UML profiles, in particular the J language, used to add semantics and services. Using UML profiles and J, PIMs and PSM can be supported by extending UML, by providing model transformation for model mapping or development automation, code generation, model checking, filtering and GUI customization.
UML profile specialization, combined with "J" polymorphism, supports the customization of UML profiles, thus allowing on-the-shelf or standard UML profiles to be adapted to specific contexts.

In order to be able to deploy a UML profile on UML Modelers, the notion of "MDA component" has been supported by Objecteering/UML. An MDA component is made up of one or several UML profiles ready to be employed by end users.

Supporting an entire MDA approach for a company is only a matter of selecting, customizing and providing new MDA components which is a notion supported by Objecteering/UML.

UML 2.0 will improve the notion of UML profiles, and will augment the expressiveness of the UML model and its code generation facility. SOFTEAM, as a member of the UML 2.0 response team, is working for complete MDA support, supported by a highly flexible toolset, a proper set of adaptable on-the-shelf MDA components and a close respect of standard specifications.

References

1 Design Patterns Automation, Concepts Tools and Practices; Philippe Desfray : Distributed Computing – November 1998
2 Object Engineering, The fourth dimension; Philippe Desfray : Addison Wesley 1994

Trademarks