UML Profiles and the J language : Totally control your application development using UML

White Paper

The UML profile was designed to structure and group together standard extensions to the UML model. In the very near future, we will be able to use models derived from UML to deal with specific needs, such as distributed applications (EDOC), EJBs (Enterprise Java Beans) or real time. At the same time, the UML profile is at the heart of a more strategic plan : to formalize and handle the application development process using UML. SOFTEAM has taken its six years' experience in this field and created the dedicated OBJECTEERING/UML Profile Builder (version 4.3) CASE tool.

Presentation

Simply having UML diagrams is not enough to justify the claim of having mastered the development of an application. "Know-how", that is to say the skill and experience of those involved, as well as the procedures and the initial knowledge which exist in the company or in the project, remains the veritable key to success. The "*UML Profile*" presented in this "white paper" is a valuable back-up to this precious "know-how".
UML 1.3, soon to be standardized by the OMG (Object Management Group - October 1999) first introduced the notion of the UML profile, a field in which SOFTEAM has extensive experience. For this reason, SOFTEAM is co-leading the work group which has the task of consolidating this notion in UML 1.4.
The UML Profile is notably used to define and master the software development process using UML, which represents a considerable advantage for all developers.
This "white paper" provides useful information on profile mechanisms and on their use. The first part concentrates on the nature of the requirements satisfied and on the use of UML profiles, and is aimed at project managers, software quality engineers, software process engineers and project leaders.
The second part, which contains more technical information, describes the adapted tool, *UML Profile Builder*. *UML Profile Builder* is used to define UML profiles and to drive UML modeling, using specific rules and notes. Anyone wishing to automate know-how related to UML can build a module (packaged profile) using *UML Profile Builder*, which can then be distributed on Objecteering sites.

Part 1 – UML profiles and the development process

Master the development process using UML

UML, the universal model used by an ever-growing number of software developments, has become strategic in the field of software development. Any organization carrying out software developments must now manage its software development process using UML for its application domain and for the development techniques it implements. In order to master the development process, significant efforts must be made to :

- Define the process
- Provide guides for each development stage, such as the analysis, design or coding phases
- Provide guides linked to the techniques used, such as programming rules, database modeling rules and document composition rules
- Introduce specific techniques, such as dedicated architecture, specific development environments or adapted requirements analysis techniques
- Train development teams in procedures and techniques
- Manage evolution and culture mergers linked to the frequent changes in companies or in techniques used
- Check the application of these guides by developers

The task of defining, maintaining and applying a development process is a heavy one, necessitating a level of investment and follow-up which can be difficult to maintain. For these reasons, tools dedicated to UML and which allow the handling and the automation of these operations have become an absolute necessity.

UML Profiles : your process support tool

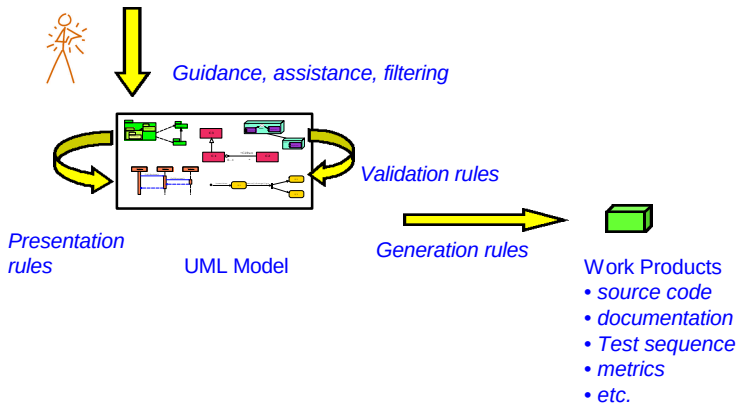

Figure 1 - *UML Profiles : Guide, check and automate UML development*

UML profiles provide a mechanism which is used to specialize UML for every work context, such as analysis, technical design and coding. They introduce notions adapted to the current type of work, to specific modeling rules, to deliverable production rules and to the presentation modes of the adapted models.
For example, let us define guides and techniques which should be applied to carry out the detailed design for C++ with UML and to generate the code. A "C++" UML profile will define the necessary extensions, in order to carry out detailed UML design for C++ and check that the specialized UML model respects the specific UML modeling constraints for C++. It will also present UML diagrams for the attention of C++ programmers or UML designers and produce C++ code which conforms to C++ programming quality rules, by applying model/code mapping rules recommended by programming guides.
UML Profiles can also be developed for specification, architectural representation, database modeling or for any other development phase or work context.

Using the Objecteering/*UML Modeler* tool, the user simply selects those profiles which correspond to the type of work in question, in order to take advantage of guides, help, checks and adapted automation. All these complementary services structured into profiles guarantee that UML modeling is carried out in accordance with recommended rules and procedures.

UML profiles, handled by the tools, are the guarantee of a high level of quality in software development.

During application development, developers select profiles which correspond to the context of their current activity. When they change activity type, developers then apply other profile selections to the UML model, which is progressively enriched. At each stage, new UML extensions are thus made available, and developers are guided and assisted with regard to their current work. In figure 2, a UML analysis model is developed and checked using the "Analysis with UML" profile. This model is then enriched in design using the "UML Design" and "UML DBMS Modeling", before being detailed in realization and completed using complementary C++ and ORACLE code additions under the "UML/C++" and "UML/ORACLE" profiles. These profiles include code generation rules necessary for the production of the database schema and the final code.

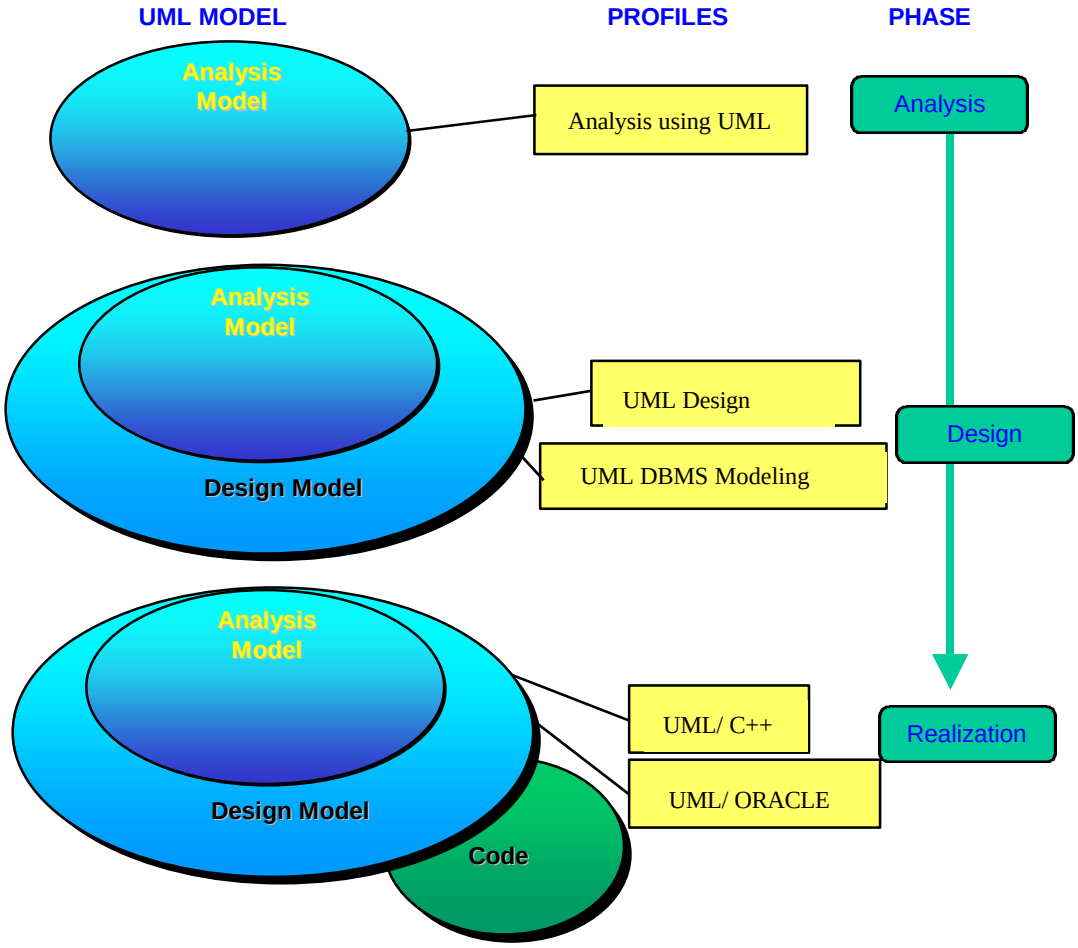

Figure 2 – *Profiles assist in UML modeling throughout the development process*

At each stage, the UML CASE tool will adapt itself, in order to provide the appropriate UML operating mode. Using its extensibility mechanisms, UML can then be specialized for each use type. Specific services will be used to carry out the necessary automation or assistance, such as : producing analysis presentation documentation; generating C++ code from the model in accordance with internal quality rules; automatically transforming a model in order to apply "design patterns", which represent real architectural know-how; and configuring a model according to company procedures. Other services guarantee assistance during modeling, such as the automatic creation of a diagram through the application of certain presentation rules, or the

automatic completion of a model for a certain objective (analysis or Java generation, etc.). Finally, certain services guarantee model measures and checks, such as consistency checks and measures on how complete an analysis model, a database model or a C++ code producing model is.

What is a UML profile?

The place of UML profiles in the UML standard

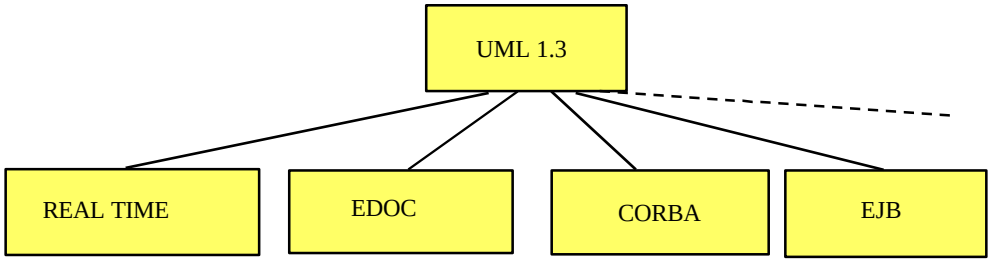

Figure 3 – *The standardization of different domains in the form of UML profiles*

The notion of the UML profile appeared in the UML 1.3 standard as a means of structuring UML extensions (tagged values, stereotypes and constraints). UML is a modeling language used in a large number of application domains and all types of software applications. However, each domain has specific notions and particular needs, which are handled by UML through extensions which are grouped into "UML Profiles". In this way, standards (which are being developed at the OMG or within the Java consortium) emerge, such as "EDOC" (Enterprise Distributed Object Computing), UML for CORBA, UML for real time or UML for EJBs (Enterprise Java Beans) [Figure 3]. Each of these standards represents a specific domain application or technical environment UML profile.
A UML profile specializes the UML model for a particular usage domain. It is used to group together UML model extensions in a coherent way, for example by introducing the notion of "EJBs" and defining their consistency rules. UML profiles can specialize other profiles, have dependencies between themselves or indeed be grouped together. A UML model is built using a particular profile; in other words, it is developed with regard to a context which implies specific semantics.
The notion of the UML Profile will be further strengthened in version 1.4 of UML (expected at the OMG in June 2000). In this light, the UML profile is above all perceived as being a "static" mechanism used to organize UML extensions and to structure UML into specific application domains. SOFTEAM co-leads the group responsible for carrying out this work within the OMG.

Profiles : driving the UML development process

SOFTEAM goes even further in the definition and handling of profiles, by adding the notion of *rules* associated with a profile. In a profile, associated rules can be defined, typically to introduce and automate UML know-how. The *J language* (which has a Java-like syntax and which is adapted to the structure of profiles and the use of UML models) is used to realize all forms of UML model use, for example, requests, validation rules, code generation or model transformation. In this way, a profile specializes UML in order to provide valuable help in modeling, in automating development and in automating checks adapted to the usage domain.
Profiles represent the repository of know-how which can be applied to UML, and constitute a powerful tool used to specify and drive the development process. At each development stage, profiles are used to express how UML should be used, what the expected development work products are and what rules the model must respect.
UML Profiles are thus used to express :

- *UML elements used* : not all UML elements are relevant to all types of work. For example, Use Cases are often unused during the C++ programming phase, whilst others do not use components during the analysis phase.
- *UML extensions added* : the UML model can be adapted to particular usage domains or development stages. For example, database modeling necessitates the presence of persistence and identification notions, which are UML extensions.

- *Validation rules* : These rules are used to check consistency criteria on a model for a given profile. In this way, it is possible to check at the end of analysis that all actors intervene in the Use Cases or that all Use Cases have actors associated to them. A qualitative measure which depends on the development stages can be carried out.
- *Presentation rules* : A good UML analysis diagram should present certain information and mask other information. The same is true for the technical design or for a vision close to C++. Each stage has, therefore, its own UML presentation rules, which are used to filter diagrams or even to create them automatically.
- *Transformation rules* : The definition of development work products, code generation rules and design patterns is used to help or automate development specifically for each type of activity.

Table 1 : *Example of the contents of a UML analysis profile*

Elements	Package, Class, Use Case
Extensions	Stereotype : <<*Business_Object*>> (class) Tagged Value : {analysis}
Validation rules	Metrics : maximum of 10 classes per package and maximum of 10 operations per class All actors must cooperate with at least one Use Case Detection of objects without classes and of messages without operations
Presentation rules	Class diagrams and Use Case diagrams. Only public operations are displayed. Special visualization of *Business_Object* classes.
Transformation rules	Document template. Automatic completion of diagrams

Modeling your process using UML profiles

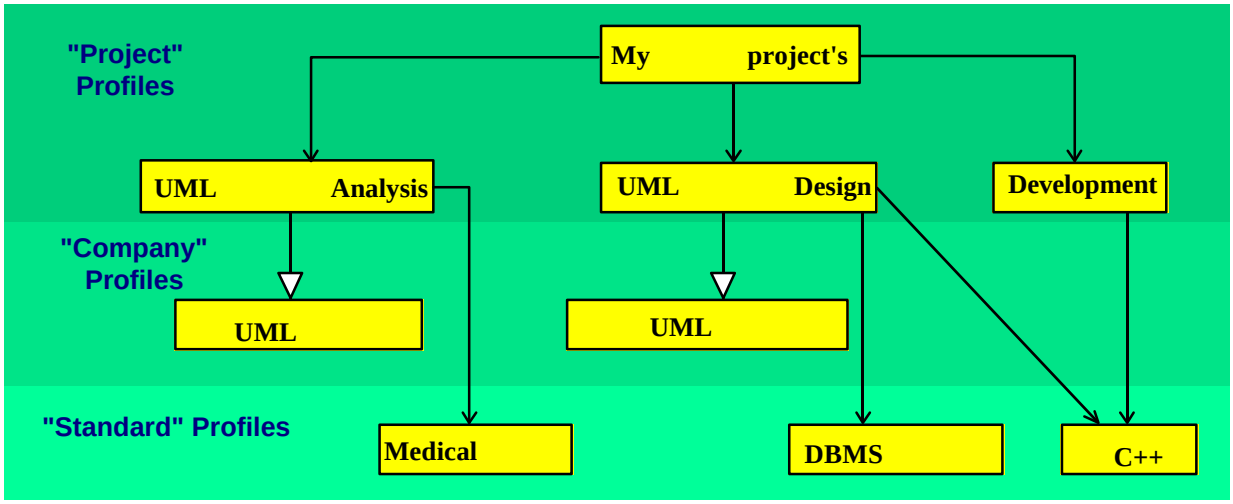

Figure 4 –*Profiles model for a project's process*

Certain general interest, functional domains exist, for example, the medical domain, or technical domains such as "develop in C++ using UML". Dedicated "on the shelf" profiles are spread over these domains. Certain profiles are already in the process of being standardized, such as "UML for CORBA", whilst others are being generally distributed. SOFTEAM is thus distributing a set of profiles linked to techniques (C++, Java, etc.) but also linked to modeling stages (analysis, design, etc.).
All companies can define profiles relevant to their processes. The analysis and design phases, as well as the test and integration phases, can typically be defined at this stage.

Finally, at the level of each project, useful profiles may be selected and specific adaptations can be introduced. It is, for example, easy to specialize the UML/C++ profile provided by SOFTEAM, in order to introduce your own C++ programming rules.

The modeling of a project's profiles is a matter for software process engineering. The model of the profiles shows exactly which techniques are used, as well as the way of working on a project.

Part 2 – Handling profiles : UML Profile Builder and the J Language

A new breed of CASE tool : UML Profile Builder

The two application repositories

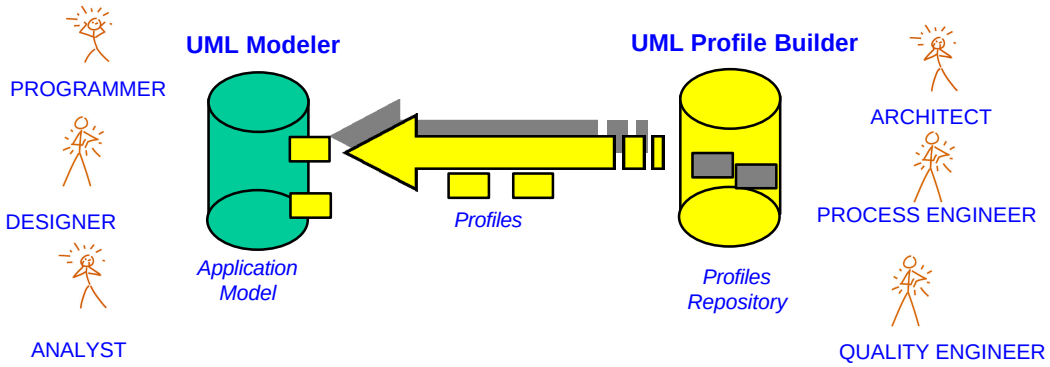

Figure 5 – *UML Modeler and UML Profile Builder address different users in different repositories*

"*Objecteering/UML Modeler*" is a tool dedicated to software developers, such as *software analysts*, *software designers* and *programmers.*
"*Objecteering/UML Profile Builder*" uses a repository of profiles independent of that of "Objecteering/*UML Modeler*" [Figure 5]. It is addressed to *software quality engineers*, *methodologists*, *technical software architects*, and to all users looking to define and apply know-how during the development of a UML application. "*Objecteering/UML Profile Builder*" is used to model profiles, to elaborate rules in the J language and to test them dynamically on an associated test project. Profiles are then "packaged" and can be deployed in projects. Each project selects the profiles it wishes to use, and the "Objecteering/*UML Modeler*" tool adapts itself according to which profiles have been selected.
Profile definition and usage through Objecteering tools can be summed up in the following manner (as shown in Figure 5) :

1. With "*UML Profile Builder*", define a set of profiles and then "package" them in order to be able to distribute them.
2. Distribute these profiles on your Objecteering sites.
3. Use these profiles in "*UML Modeler*" : drive UML modeling.

Select a profile on a UML project : how to adapt the "UML Modeler" tool

Until a profile is selected, the *UML Modeler* tool is simply a UML model editor, which guarantees the consistency of the models built. The act of *selecting a profile* [as shown in Figure 5] provides all high added value services, specific to a particular domain, such as specific UML model extensions, adapted code generation, model transformations used to automate design patterns, etc. *Objecteering/UML Modeler* is enriched with :

- *New Tagged values* : UML profiles determine which tagged values are allowed. These tagged values will be proposed in the appropriate help lists.
- *New stereotypes* : Stereotypes declared within a profile will be proposed in a help list.
- *Notes* : Profiles define which text types may be associated to UML elements. For example, documentation can require a summary and a detailed description. A Note can even contain C++ code, etc.
- *Commands* : Context menus on UML model elements provide services dedicated to the profile, such as consistency checks, automated design patterns, generation commands, etc.

- *Development work products* : These work products represent what the Objecteering CASE tool must generate. This can be documentation, source code, a Makefile, a database diagram, an executable thread, a binary, a Java applet, etc. They guarantee the liaison and consistency between elements external to the model (often files) and the part of the model which has produced them. They also guarantee the form and production mode used to generate them. For example, an "analysis documentation" work product references the document template which was used to create it.
- *Specialized editors* : According to the target in question, editors will be associated. These editors will be used notably to manage consistency between the UML model and the generated work product in real time.

In this way, *Objecteering/UML Modeler* becomes a specialized tool for a particular domain. It uses the selected profiles to manage consistency and to automate the production of development work products, which will constantly remain in accordance with the development rules and consistent with the model [Figure 6].

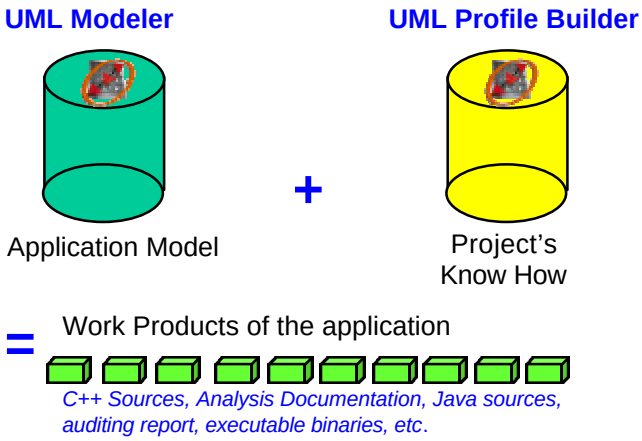

Figure 6 – *Profiles are used to make UML directly usable in all project contexts*

UML Profile Builder Services

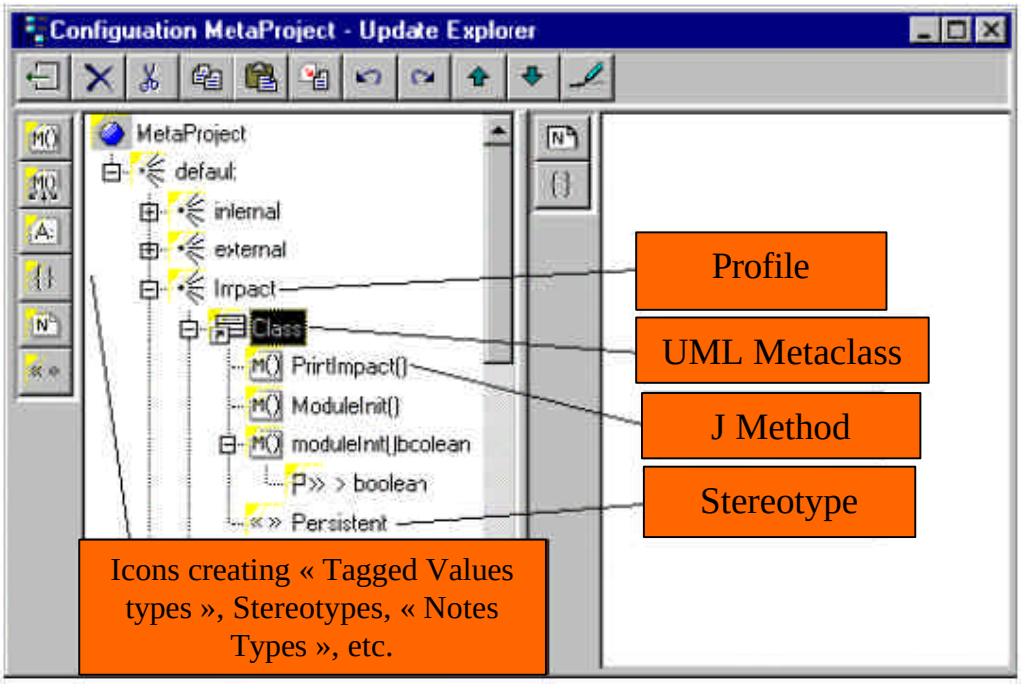

Figure 7 – *The central tool in UML Profile Builder is the Profiles explorer*

UML Profile Builder is a profile modeling, programming and J execution tool, used to structure and define UML extensions, as well as J rules used to process the model. The user can interactively define new extensions

and commands (which will then appear in the "*UML Modeler*" menus), as well as processing and development work products, before immediately testing them on a UML model.
UML Profile Builder presents the user with the *Objecteering metamodel* (which conforms to the UML 1.3 metamodel). The user can take advantage of this tool to add extensions and insert J methods. [Figure 7]
UML Profile Builder also provides a document template editor which can be used to define generators, by describing the target structure. This is typically the case for documentation generation, but also for code generators, such as, for example, Java [Figure 8]. Document templates allow you to considerably reduce the J code necessary, and to ensure the simple parameterization and documentation of the generators.
UML Profile Builder also provides a document template editor which can be used to define generators, by describing the target structure. This is typically the case for documentation generation, but also for code generators, such as, for example, Java [Figure 8]. Document templates allow you to considerably reduce the J code necessary, and to ensure the simple parameterization and documentation of the generators.

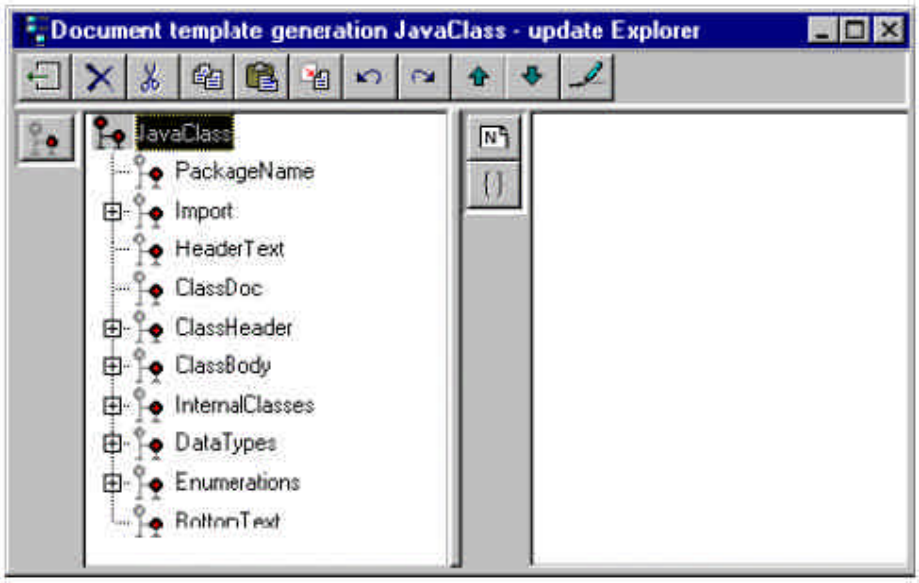

Figure 8 – *Java code generation document template*

The J language : dedicated to the "metamodel" and the "profile"

Simplicity and power for handling UML models

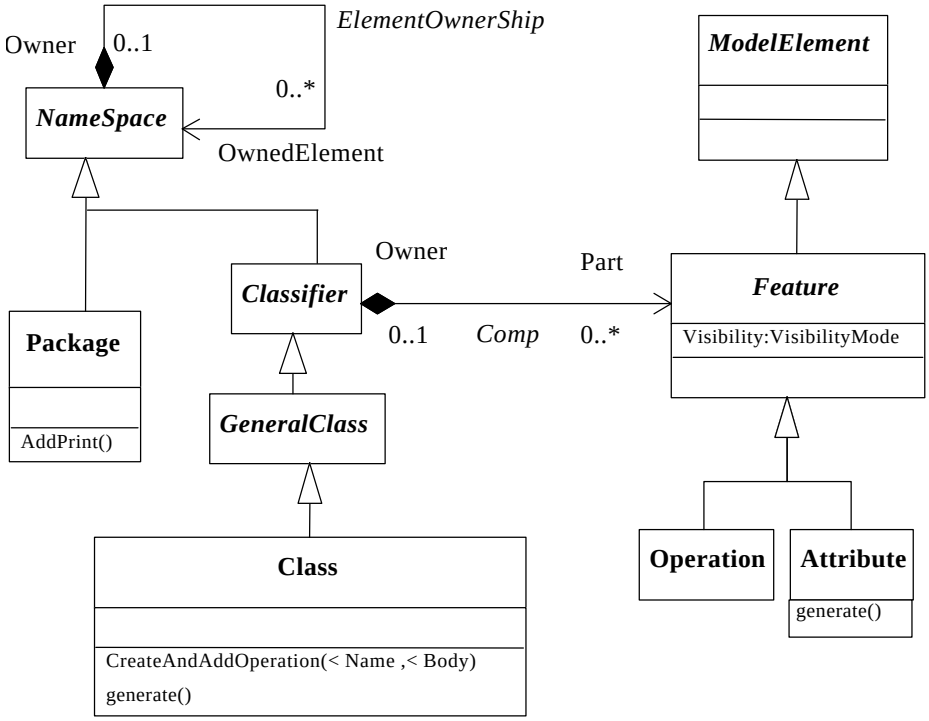

Figure 9 – *Extract from the Objecteering metamodel*

The J language is one of the central elements of the *UML Profile Builder* tool. Its "Java-like" syntax makes this language immediately accessible, allowing any person with experience of Java to handle UML models without undertaking specific training.
The *Objecteering metamodel* (in accordance with the UML 1.3 metamodel : see extract [Figure 9]) is very easily handled using this language, which browses and immediately processes the model, without having to use "iterators", or "while" and "for" loops as in traditional languages. For example, if "P" is a package, " `P.OwnedElementClass` " is a term we read by following the associations, role names and classes we wish to handle : here, this term designates the set of "Classes" which play the "OwnedElement" role for the Package P, in other words, "all Package P's classes".
The " diffusion " mechanism (whose symbol is " .< ") in J is used to write the following instruction, in order to activate the "generate" J method on all package P's classes :

" `MyPackage.OwnedElementClass.<generate()` "

Powerful on-line help allows you to easily browse this metamodel and thus get to know all its properties.
J is used to manage model *modification sessions* in the form of unitary *transactions*, thereby providing dynamic (during the editing of the model in *UML Modeler*) and controlled (checked by the *UML Modeler* consistency checks) modification of the model, as well as the cancellation and undo, or "rollback", of model modification.
J can also be used to handle *UML diagrams* through programming. In this way, automatic diagram creation, positioning and filtering, as well as the application of presentation rules, is handled.
The "J" program below is an example which adds the "Print" method, whose code will be "*for all attributes A of the current class, insert the " cout << Attribute name ;*" (printing of C++ attributes), to all a package's classes. This is a simple example of model transformation.

```
#Example#Package::AddPrint () – method declaration on a Package in the
                              -- "Example" profile
{
      string Body ;
      OwnedElementClass        -- all a package's classes
      {                       -- Processing impacting each of the classes
        PartAttribute          -- all a class' attributes
          {                    -- Processing on each attribute
           Body= "cout" + "<< " + Name);
          }
        Body.concat("; ");
        CreateAndAddMethod("Print",Body);--Addition of the "Print" method to the
                                         class
      }
}
```

UML Profile Builder provides a set of predefined J libraries which supply the building blocks for use of the UML model, thus further increasing productivity. For example, these libraries provide *UML Modeler* CASE tool driving services, high level model element creation primitives, operating system access libraries (files, processes, etc.), development work product management libraries, diagram handling and introspection services. J is an interpreted language which can dynamically evaluate a fragment of a J program.

Thanks to J, everything which can be achieved interactively by an end-user through UML Modeler can be easily achieved through programming. This is the real key to the J language's unlimited model handling capacities.

J provides profile parameterization which is unique in the market place

J language methods are structured simultaneously by classes (metaclasses) and profiles. In this way, a method "generate()" will be defined on the "Class" metaclass under the "C++" profile. This original mechanism is used to overload profiles with profiles which specialized in J. It provides Objecteering with parameterization abilities which are unique in the market place. In this way, when one wishes to adapt a profile, a typical example being a profile from a standard Objecteering module such as "C++", all you need to do is create a "C++ Specific" profile which specializes the "C++" profile [Figure 9], and then redefine the "J" methods whose behavior you

wish to adapt. J is a language which has a "double lookup" mechanism, used to manage class generalization in accordance with profile generalization.

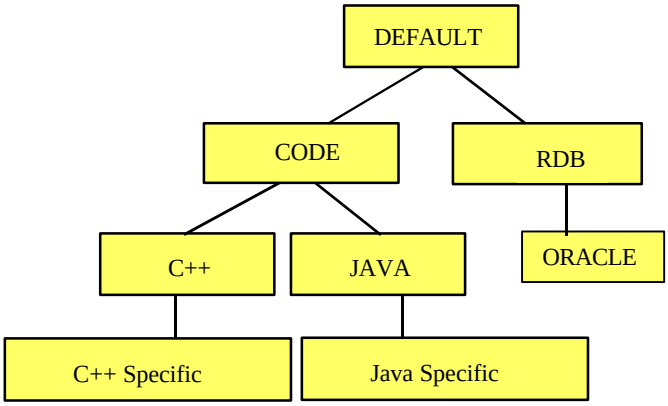

Figure 9 –*Specialization of Profiles*

Application examples : make the most of UML

Code generation

Objecteering code generators are all built using the *UML Profile Builder* tool. C++, Java, IDL and SQL languages are notably realized using this CASE tool. Numerous other examples exist, such as object databases or specific component libraries. In only a few days, it is possible to build a generator for any language using *UML Profile Builder*. *UML Profile Builder* provides unique facilities for this construction, such as, for example, managing permanent UML model/generated code consistency or automating useful "design patterns" for a specific language. *UML Profile Builder* naturally provides the possibility of parameterizing a generator, by providing the opening and extensibility of modules realized to users.
The *document template* technique strengthens these abilities, by allowing visual programming which is oriented towards the structure of the target language, and which is self-documenting and even easier to parameterize.
Documentation generation is a particular form of code generation with specific services (formatting tools, generation of hypertext links, insertion of images, insertion of typed text) associated to it. These services guarantee professional and precise documentation, which satisfies all kinds of objectives such as analysis documentation, auditing documentation or test reports.
These generators can then be deployed on "*UML Modeler*" environments.

Parameterizing generators

UML Profile Builder is often used to parameterize the generators provided with Objecteering. Indeed, this tool's capabilities in profile generalization, document template redefinition and J method overloading make *UML Profile Builder* the most powerful tool in this respect.

Furthermore, certain generators, such as C++, Java or SQL, are used to parameterize the generation of base libraries independently of the generator. In this way, it is possible to easily change a C++ base library for another, to supply other base types and to change set management modes (containers, etc.). In this way, for a single C++ generation principle, it is possible to independently interchange such libraries as STL and MFC.

Analyzing or auditing a model

J's unique model browsing capacities, coupled with its documentation generation abilities and the possibility of creating interactive reports, allow you to realize all sorts of analyses and to print reports.
This applies to the definition of new consistency rules, but also to that of modules, such as "Metrics", which measure the quality of a model. The same is true for the compiling of impact measuring and element research tools, according to certain criteria.

Model transformation

Model transformation is one of the most impressive services provided by *UML Profile Builder*. Using this tool, modeling assistants, such as automated design patterns or the automatic completion of a model for a given objective (automatic addition of notes, automatic creation of diagrams) are created.
C++ and Java Design Patterns and Java idioms are some examples of this.

SOFTEAM know-how recognized since 1993

The techniques presented in this "white paper" are not a new development by SOFTEAM, but they remain innovative as far as the market is concerned. In 1993, SOFTEAM produced Objecteering version 3.4, a tool capable of handling a technique called "Hypergenericity". Books have been published on this subject, which has been used by hundreds of projects to realize their developments.
In the latest version of the tool (Objecteering 4.3), this technique is applied to profiles, with SOFTEAM using all the experience gained from seven years in the domain and six major versions of this CASE tool.

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