

Profils UML et langage J : Contrôlez totalement le développement d'applications avec UML

White Paper

© Softeam 1999



Le profile UML est conçu pour structurer et assembler les extensions standards du modèle UML. Très bientôt, nous aurons à notre disposition des modèles dérivés de UML pour traiter des besoins spécifiques, tels que les applications distribuées (EDOC), les EJB (Enterprise Java Beans) ou encore le temps réel. Dans le même temps, le profile UML est voué à un dessein stratégique : formaliser et supporter le processus de développement d'applications avec UML. SOFTEAM met à profit son expérience de six années dans ce domaine, pour mettre à disposition l'atelier dédié : OBJECTEERING/UML Profile Builder dans sa version 4.3.

Présentation

Il ne suffit pas d'avoir des diagrammes UML, pour prétendre maîtriser le développement d'une application. Le « savoir faire », c'est à dire les compétences et l'expérience des intervenants, les procédures et les connaissances initiales existant dans l'entreprise ou dans le projet, demeure la clé du succès. Le « *Profile UML* » présenté dans ce « white paper » constitue un support actif de ce précieux « savoir faire ».

UML 1.3, prochainement standardisée par l'OMG (Object Management Group - octobre 99) a introduit la notion de Profile UML. SOFTEAM bénéficie d'une longue expérience en la matière. A ce titre, SOFTEAM dirige le groupe de travail devant consolider cette notion dans UML1.4.

Le Profile UML permet en particulier de définir et maîtriser le processus de développement logiciel avec UML, ce qui représente un bénéfice considérable pour tous les développeurs.

Le présent « white paper » fournit les informations utiles sur le mécanisme des profils et sur leur utilité. La première partie est centrée sur la nature des besoins satisfaits, et sur l'emploi des profils. Cette partie s'adresse particulièrement aux gestionnaires de projet, ingénieurs qualité, ingénieurs processus et chefs de projets.

La seconde partie, à vocation technique, décrit l'outillage adapté : UML Profile Builder. UML Profile Builder permet de définir des profils UML, et de piloter la modélisation UML par des règles et annotations spécifiques. Toute personne voulant automatiser un savoir faire lié à UML peut réaliser un module avec cet atelier, qui peut ensuite être diffusé sur les sites Objecteering.

Partie 1 – Les profils UML au service du processus de développement

Maîtriser le processus de développement avec UML

Modèle universel, utilisé par un nombre croissant de développements logiciel, UML a désormais une importance stratégique pour les développements logiciels. Chaque organisme réalisant des développements logiciel doit maintenant maîtriser son processus de développement logiciel avec UML, pour son domaine d'application, et pour les techniques de développement qu'il met en œuvre. La maîtrise du processus de développement nécessite un effort important comme par exemple :

- Définir le processus,
- Fournir des guides pour chaque étape de développement, telle que les phases d'analyse de conception ou de codage,
- Fournir des guides liés aux techniques employées, tels que les règles de programmation, les règles de modélisation de bases de données, les règles de rédaction de document,
- Introduire des techniques spécifiques, telles que des architectures dédiées, des environnements de développement spécifiques, des techniques d'analyse des besoins adaptées,
- Former les équipes de développement, aux procédures, aux techniques,
- Gérer les évolutions, ou fusions de culture liées aux changements très fréquents dans les organisations, ou encore dans les techniques employées,
- Vérifier l'application de ces guides par les développeurs.

On remarque alors que définir, maintenir et appliquer un processus de développement est une tâche lourde, nécessitant un investissement et un suivi difficiles à assumer. L'outillage, dédié à UML, permettant de supporter et d'automatiser ces actions devient ainsi une nécessité.

Les profils UML : L'outil support de votre processus

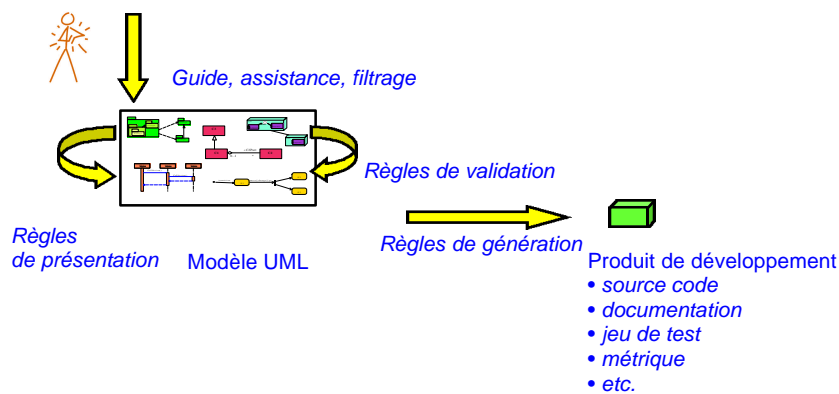


Figure 1 - Profils UML : Guider, Contrôler, Automatiser le développement UML

Les profils UML apportent un mécanisme permettant de spécialiser UML pour chaque contexte de travail comme par exemple l'analyse, la conception technique, le codage, etc. Ils introduisent des notions plus adaptées au type de travail courant, des règles de modélisation spécifiques, des règles de production de livrables, et des modes de présentation des modèles adaptés.

Par exemple, définissons les guides et techniques devant être appliqués pour faire de la conception détaillée pour C++ avec UML, et générer le code. Un profil UML « C++ » va définir les extensions nécessaires pour réaliser une conception détaillée UML pour C++ ; vérifier que le modèle UML spécialisé respecte des contraintes de modélisation spécifiques pour C++ ; présenter des diagrammes UML à l'attention des programmeurs C++ ou

concepteurs UML ; et produire un code C++ conforme aux règles de qualité de programmation en C++, appliquant les règles de traduction modèle/Code recommandées par les guides de programmation.

On peut aussi développer des profils UML pour la spécification, pour la représentation d'architectures, pour la modélisation de bases de données, ou pour toute autre phase de développement ou contexte de travail.

Sous l'atelier Objecteering/UML modeler, il suffit alors de sélectionner les profils correspondant au type de travail voulu, pour bénéficier des guides, assistances, vérifications et automatisations adaptées. L'ensemble de ces services complémentaires structurés en profils, garantissent que la modélisation UML est effectuée conformément aux règles et procédures préconisées.

Les profils UML, supportés par des outils, sont le garant d'un haut niveau de qualité des développements logiciels.

Lors du développement d'une application, les développeurs sélectionnent les profils correspondant au contexte de leur activité courante. Lorsqu'ils changent de type d'activité, les développeurs appliquent ensuite d'autres sélections de profils sur le modèle UML qu'ils enrichissent progressivement. A chaque étape, de nouvelles extensions UML sont ainsi mises à disposition, et les développeurs sont guidés et assistés en fonction de leur travail courant. Dans la Figure 2, un modèle UML d'analyse est ainsi élaboré sous le contrôle et l'aide du profil « Analyse avec UML », puis ce modèle est enrichi en conception sous les profils « Conception UML » et « Modélisation SGBD UML », pour enfin être détaillé en réalisation et complété par des ajouts complémentaires de code C++ et ORACLE sous les profils « UML/C++ » et « UML/ORACLE ». Ces profils comportent les règles de génération de code nécessaires pour produire le schéma de base de données et le code final.

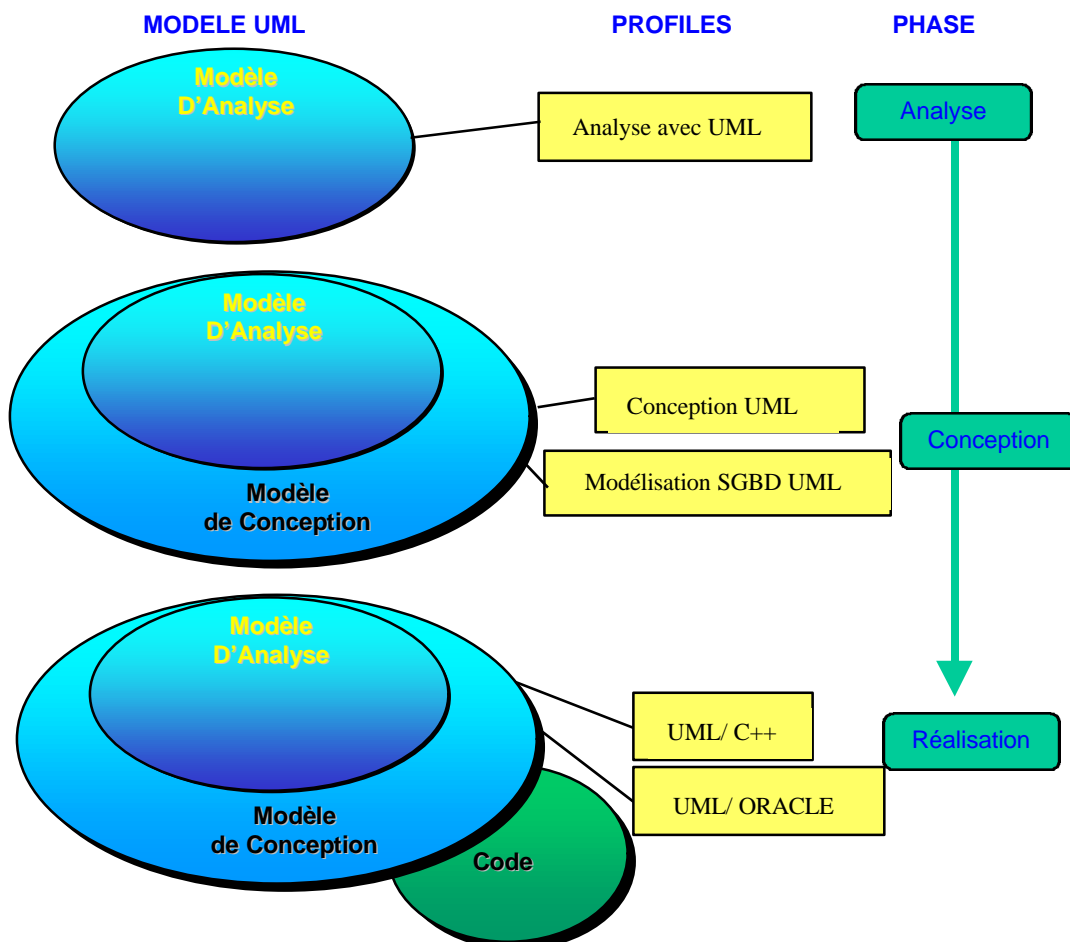


Figure 2 – Les profils assistent la modélisation UML tout au long du développement

A chaque étape, l'atelier UML va s'adapter, pour fournir un mode d'exploitation de UML approprié. Grâce à ses mécanismes d'extensibilité, UML sera alors spécialisé pour chaque type d'utilisation. Des services spécifiques permettront de réaliser les automatisations ou assistances nécessaires : produire une documentation de

présentation d'analyse; générer un code C++ d'après le modèle, conforme aux règles qualités internes ; transformer automatiquement un modèle pour appliquer des « design pattern », véritables savoir faire architecturaux; mettre en configuration un modèle selon les procédures de l'entreprise. D'autres services assureront des assistances lors de la modélisation, comme par exemple créer automatiquement un diagramme appliquant certaines règles de présentation, ou compléter automatiquement un modèle pour un certain objectif (analyse, ou génération Java, etc.). Enfin, certains services viendront assurer des mesures et des vérifications du modèle, tels que la vérification de la cohérence et de la complétude d'un modèle d'analyse, ou d'un modèle de bases de données, ou encore d'un modèle produisant du code C++.

Qu'est ce qu'un profile UML

Les profiles dans le standard UML

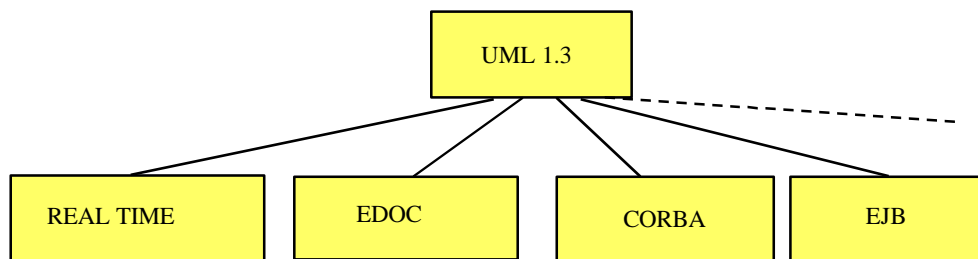


Figure 3 – Standardisation des différents domaines sous forme de Profiles UML

La notion de Profile UML est apparue dans le standard UML 1.3, comme un moyen permettant de structurer les extensions UML (tagged values, stereotypes et constraints). UML est un langage de modélisation à destination d'un grand nombre de domaines d'application : tous types d'applications logicielles. Cependant, chaque domaine a des notions particulières, des besoins particuliers, que UML peut supporter par le biais de ses extensions, regroupées en « Profiles UML ». On voit ainsi émerger des standards (en cours de développement à l'OMG ou au sein du consortium Java) tels que « EDOC » (Enterprise distributed object computing », ou UML pour CORBA, ou UML pour le temps réel, ou encore UML pour les EJB (Enterprise Java Beans) [Figure 3]. Chacun de ces standards est en fait un profile UML spécifique d'un domaine d'application ou d'un environnement technique.

Un profile UML est une spécialisation du modèle UML pour un domaine d'utilisation particulier. Il regroupe de manière cohérente des extensions du modèle UML, comme par exemple en introduisant la notion de « EJB », et définit leurs règles de cohérence. Les profiles UML peuvent hériter d'autres profiles, avoir des dépendances entre eux, ou encore être regroupés. Un modèle UML est construit sous un profile particulier, c'est à dire qu'il est élaboré relativement à un contexte qui lui apporte une sémantique spécifique.

La notion de profile UML sera renforcée dans UML 1.4 (attendu à l'OMG pour Juin 2000). Dans cette perspective, le profile UML est surtout perçu comme étant un mécanisme « statique » organisant les extensions UML, pour structurer UML en domaines d'applications spécifiques. SOFTEAM dirige le groupe devant effectuer ce travail au sein de l'OMG.

Les profiles : pilotes du processus de développement UML

Mais SOFTEAM va plus loin dans la définition et le support des profile en ajoutant la notion de *règles* associées à un profile. Dans un profile, on peut définir des règles associées, typiquement pour introduire et automatiser un savoir faire sur UML. Le langage *J* (qui a une syntaxe à la Java, et qui est adapté à la structure des profiles et à l'exploitation des modèles UML) permet de réaliser toutes formes d'exploitation d'un modèle UML comme par exemple des requêtes, des règles de validations, des générations de code, ou des transformations de modèle. De cette manière, un profile spécialise UML pour également apporter des assistances à la modélisation, des automatisations du développement et des contrôles adaptés au domaine d'utilisation.

Les profiles constituent le référentiel du savoir faire que l'on peut appliquer sur UML. Ils constituent un outil puissant pour spécifier le processus de développement et pour le guider. A chaque étape de développement, les profiles permettent d'exprimer comment il faut utiliser UML, quels sont les produits de développement attendus, et quelles sont les règles que le modèle doit respecter.

Les Profiles UML permettent ainsi d'exprimer :

- *Les éléments UML utilisés* : tous les éléments de UML ne sont pas pertinents pour toutes sortes de travaux, par exemple les Use Case ne sont souvent pas utilisés en phase de programmation C++, ou d'autres n'utilisent pas les composants en phase d'analyse
- *Les extensions UML ajoutées* : Le modèle UML peut être adapté pour des domaines d'utilisation, ou pour des étapes de développement particulières. Par exemple, la modélisation de bases de données nécessite d'apporter les notions de persistance et d'identification, qui sont alors des extensions à UML.
- *Les règles de validation* : Ces règles permettent de vérifier des critères de cohérence sur un modèle pour un profilé donné. Ainsi, on pourra vérifier en fin d'analyse que tous les acteurs interviennent dans des Use Case, ou que tous les Use Case ont des acteurs associés. Une mesure qualitative dépendant des étapes de développement peut être conduite.
- *Les règles de présentation* : Un bon diagramme d'analyse UML doit présenter certaines informations et en cacher d'autres. Il en va de même pour la conception technique, ou pour une vision proche de C++. Chaque étape a ainsi des règles de présentation UML, permettant de filtrer des diagrammes ou même de les créer automatiquement.
- *Les règles de transformation* : La définition des produits de développements, les règles de génération de code et les design patterns permettent d'assister ou d'automatiser le développement spécifiquement pour chaque type d'activité.

Tableau 1 : Exemple de contenu d'un Profilé UML d'analyse

Eléments	Package, Classe, Use Case
Extensions	Stereotype: <<Business_Object>> (classe). Tagged Value: {analysis}
Règles de Validation	Métriques: 10 classes max par packages ; 10 opérations max par classes. Tous les acteurs doivent coopérer avec au moins un Use Case Détection des objets sans classes, des messages sans opérations
Règles de Présentation	Diagrammes de classes et de Use Case. Seules les opérations publiques sont affichées. Visualisation particulière des classes <i>Business_Object</i> .
Règles de Transformation	Plan type de documentation. Complétion automatique des diagrammes

Modéliser son processus en utilisant les profils UML

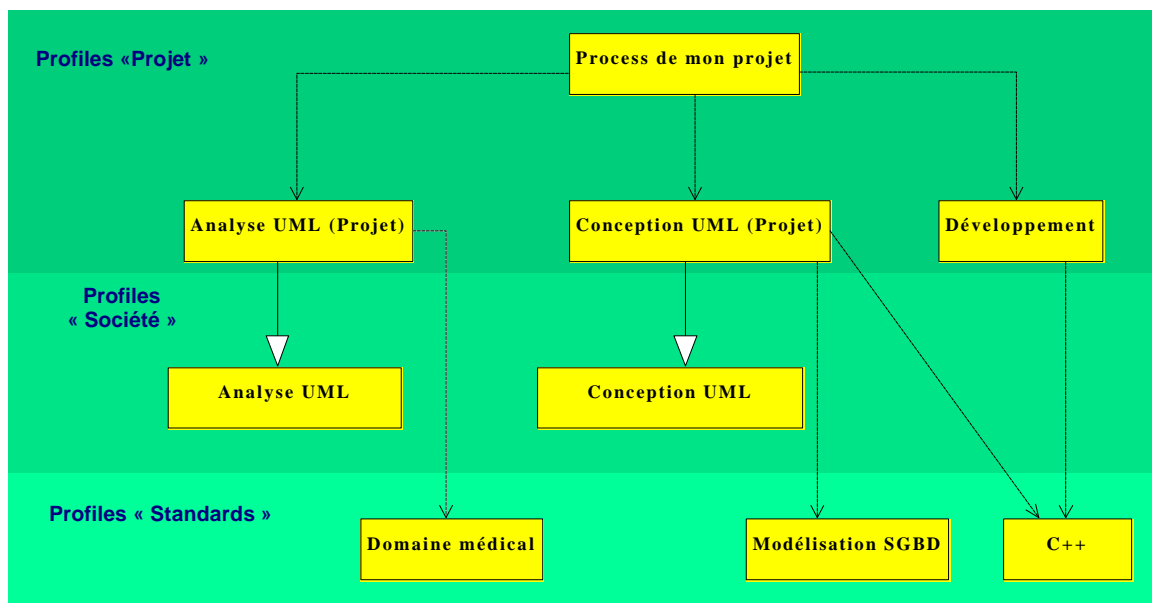


Figure 3 – Modèle de profils pour le processus d'un projet

Il existe des domaines d'intérêt général, qui sont des domaines fonctionnels comme par exemple le domaine médical, ou des domaines techniques comme par exemple « développer en C++ avec UML ». Des profils dédiés « sur étagère » se répandent sur ces domaines. Certains profils sont déjà en cours de standardisation,

comme par exemple « UML pour CORBA », cependant que d'autres sont diffusés de manière générale. Ainsi SOFTEAM diffuse un ensemble de profils liés aux techniques (C++, Java, etc.) mais également liés aux étapes de modélisation (analyse, conception).

Chaque entreprise peut ensuite définir des profils relatifs à son processus. Les étapes d'analyse, de conception, mais aussi de test, d'intégration, peuvent typiquement être définies à ce stade.

Enfin, au niveau de chaque projet, une sélection des profils utiles doit être faite, et des adaptations spécifiques peuvent être introduites. Il est par exemple facile de spécialiser le profil UML/C++ fourni par SOFTEAM, pour introduire ses propres règles de programmation C++.

La modélisation des profils d'un projet relève de l'ingénierie du processus de développement. Le modèle des profils montre exactement les techniques employées et la façon de travailler sur un projet.

Partie 2 – Support des profils : UML Profile Builder et Langage J

Une nouvelle race d'ateliers : UML Profile builder

Les deux référentiels d'une application

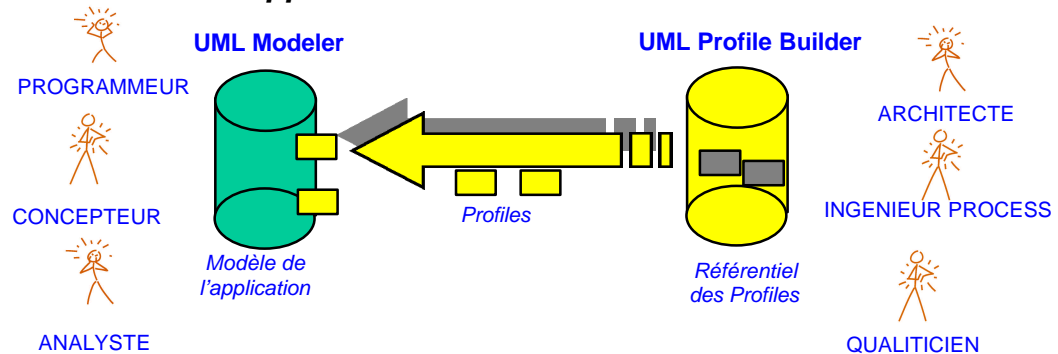


Figure 4 – UML Modeler et UML Profile Builder adressent des utilisateurs différents dans des référentiels différents

« Objecteering/UML Modeler » est un outil dédié aux développeurs logiciels tels que les *analystes*, les *concepteurs*, et les *programmeurs*.

« Objecteering/UML Profile Builder » exploite un référentiel de profils indépendant de celui de « Objecteering/UML Modeler » [Figure 4]. Il s'adresse à des *ingénieurs qualité*, *méthodologistes*, *ingénieurs processus*, *architectes techniques*, et à toute sorte d'utilisateur ayant un savoir faire à définir et à faire appliquer lors d'un développement d'application UML. « Objecteering /UML Profile Builder » permet de modéliser des profils, d'élaborer des règles en langage J, et de les tester dynamiquement sur un projet de test associé. Ensuite, les profils sont « packagés », et peuvent être déployés sur les projets. Chaque projet sélectionne les profils qu'il veut employer, et l'atelier « Objecteering/UML Modeler » se spécialise en fonction des profils sélectionnés.

La définition et l'exploitation de profil avec les outils Objecteering se résume donc de la manière suivante [Figure 4] :

1. Avec *UML Profile Builder*, définissez un ensemble de profils, « packagez » les pour pouvoir les diffuser
2. Diffusez ces profils sur vos sites Objecteering
3. Utilisez ces profils sous *UML Modeler* : guidez la modélisation UML

Sélectionner un profil sur un projet UML : L'adaptation de l'outil « UML modeler »

Sans aucun profil sélectionné, l'atelier « UML Modeler » est simplement un éditeur de modèles UML, assurant la cohérence des modèles saisis. La *sélection de profil* [Figure 4] va apporter tous les services à haute valeur ajoutée, spécifiques d'un domaine, tels que des extensions spécifiques du modèle UML, des générations de code adaptées, des transformations de modèle automatisant des design patterns, etc. Objecteering/UML modeler va s'enrichir de :

- *Nouvelles Tagged values* : les profils UML déterminent quelles sont les tagged values autorisées. Celles-ci seront proposées dans des listes d'aide appropriées
- *Nouveaux stéréotypes* : Les stéréotypes déclarés au sein d'un profil seront alors proposés dans une liste d'aide.

- *Notes* : Les profils définissent quels types de textes peuvent être associés aux éléments UML. Par exemple, une documentation peut demander un résumé et une description détaillée, ou encore une Note peut contenir du code C++, etc.
- *Commandes* : les menus contextuels sur les éléments de modèle UML vont apporter des services dédiés au profil, tels que des contrôles de cohérence, des design patterns automatisés, des commandes de génération,
- *Produits de développement* : Ces produits représentent ce que l'atelier Objectteering doit générer. Ce peut être une documentation, un source code, un Makefile, un schéma de bases de données, un jeu de test exécutable, un binaire, une applet Java, etc. Ils garantissent la liaison et la cohérence entre des éléments externes au modèle (bien souvent des fichiers) et la partie de modèle qui les a produit. Ils garantissent aussi la forme et le mode de production permettant de les générer. Par exemple, un produit « documentation d'analyse » référence le plan type qui a permis de le fabriquer.
- *Editeurs spécialisés* : Selon la cible, des éditeurs seront associés, permettant notamment de gérer en temps réel la cohérence modèle UML/produit généré.

Objectteering/UML Modeler devient alors un outil spécialisé à un domaine particulier. Il exploite les profils sélectionnés pour gérer la cohérence, et automatiser la production des produits de développement qui resteront toujours conformes aux règles de développement et cohérents avec le modèle [Figure 5].

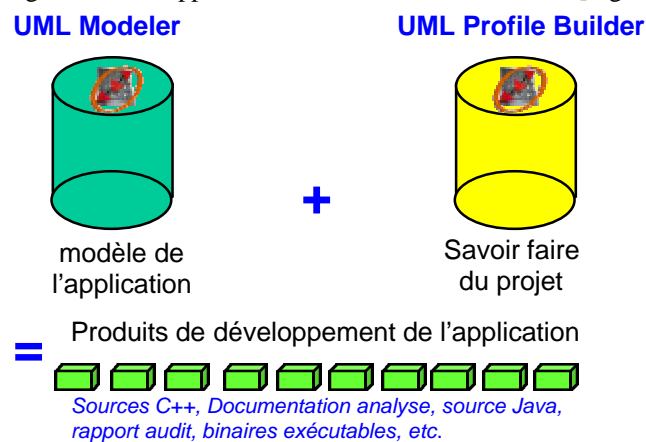


Figure 5 – L'utilisation des profils permet de rendre UML directement exploitable pour tous les contextes des projets

Les services de UML Profile builder

UML Profile Builder est un atelier de modélisation de profils, de programmation et d'exécution J, permettant de structurer et définir les extensions UML, ainsi que les règles J apportant des traitements sur les modèles. De manière interactive, l'utilisateur peut définir de nouvelles extensions, des commandes (qui seront des entrées de menus dans « UML Modeler »), des traitements, des produits de développement, et les tester sur un modèle UML immédiatement.

UML Profile Builder présente le *métamodèle Objectteering* (conforme au métamodèle UML 1.3) à l'utilisateur, qui va l'utiliser pour ajouter ses extensions, et insérer des méthodes J [Figure 6].

UML Profile Builder offre également un *éditeur de plans types*, qui permet de définir des générateurs en décrivant la structure de la cible. C'est typiquement le cas pour la génération de documentation, mais aussi pour les générateurs de code, comme par exemple Java [Figure 7]. Les plans types permettent de réduire considérablement le code J nécessaire, assurent une documentation et un paramétrage très simple des générateurs.

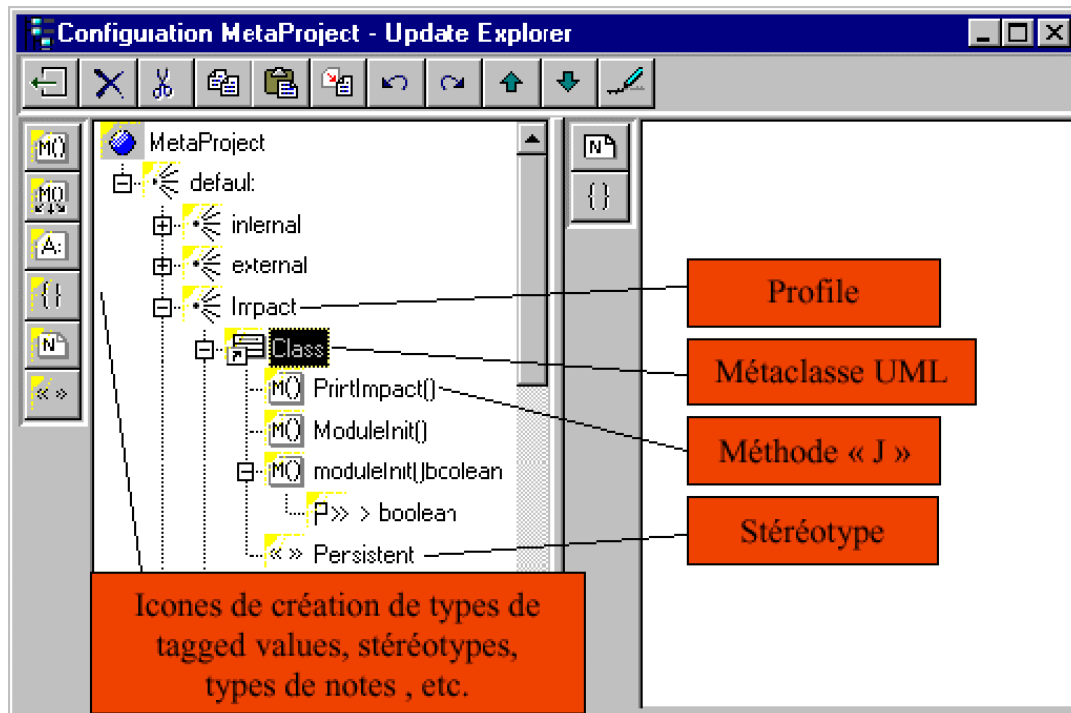


Figure 6 – L'outil central de UML Profile Builder est un explorateur de Profiles

Enfin, UML Profile Builder offre des services de packaging des modules, afin que ceux ci puissent être mis à l'état de produits, livrables sous tout environnement « UML Modeler ».

Tous les modules Objecteering (Documentation, Metrics, Wizards UML, générateurs C++, Java, ILD, SQL, Design Patterns, XMI, etc.) sont réalisés avec UML Profile Builder.

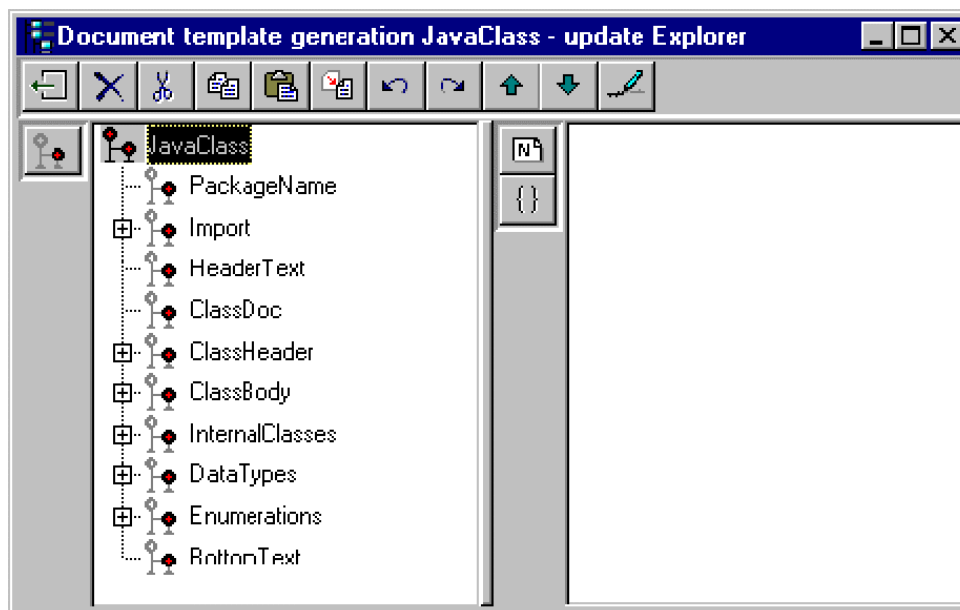


Figure 7 – Plan type de génération de code Java

Le langage J : dédié « métamodèle » et « profile »

Simplicité et puissance de la manipulation de modèles UML

Un des constituants essentiels de la puissance de UML Profile Builder est le langage J. Sa syntaxe « Java like » le rend accessible immédiatement, permettant à toute personne connaissant Java de manipuler les modèles UML sans apprentissage.

Le métamodèle *Objecteering* (conforme au métamodèle de UML 1.3 ; voir extrait [Figure 8]) est très aisément manipulé par ce langage qui navigue dans le modèle, et qui traite les ensembles de manière immédiate sans avoir à utiliser des « itérateurs », ou des boucles « while » et « for » comme dans les langages traditionnels. Par exemple, si P est un package, « P.OwnedElementClass » est un terme que l'on lit en suivant les associations, les noms de rôle et les classes que l'on veut manipuler : ici, ce terme désigne l'ensemble des « Class » jouant le rôle « OwnedElement » pour le Package P, c'est à dire l'ensemble des classes du package P.

Le mécanisme de « diffusion » (symbole « .< ») de J permet d'écrire l'instruction suivante pour activer la méthode J «generate» sur toutes les classes du package P :

```
« MyPackage.OwnedElementClass.<generate() »
```

Une aide en ligne puissante permet de naviguer simplement dans ce métamodèle, et d'en connaître toutes les propriétés.

J permet de gérer sous forme de *transaction* unitaire les *sessions de modification* d'un modèle, offrant ainsi la modification de modèle dynamiquement (pendant une édition de modèle sous UML modeler) et contrôlée (vérifiée par les contrôles de cohérence de UML Modeler), ainsi que l'annulation et le retour en arrière (undo ou « rollback ») sur modification de modèle.

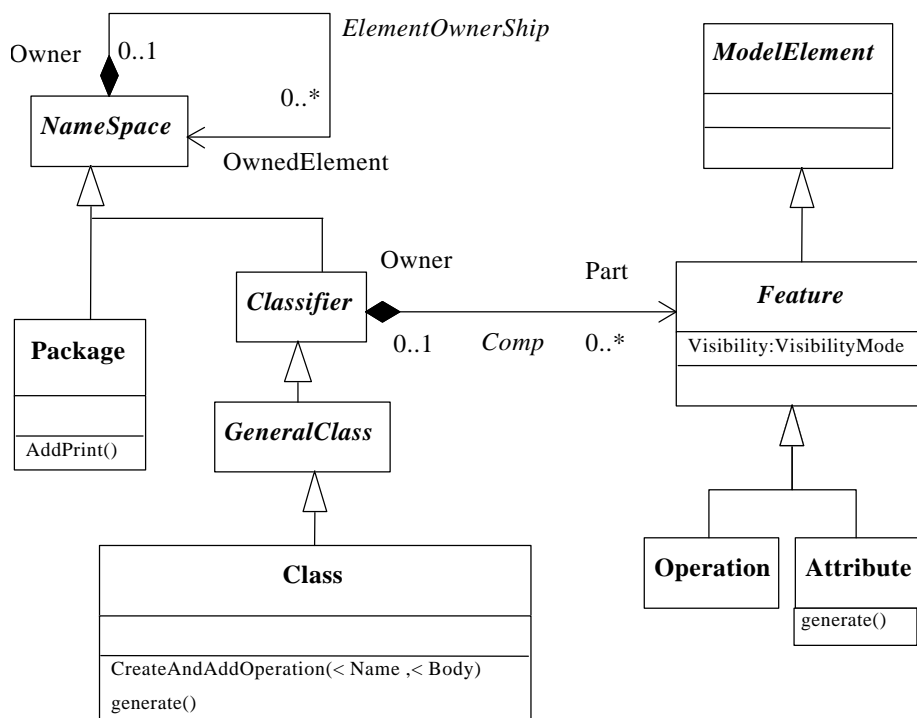


Figure 8 – Fragment de métamodèle *Objecteering*

J permet aussi de manipuler les *diagrammes UML* par programmation. Ainsi, la création automatique de diagramme, le positionnement, le filtrage, l'application de règles de présentation sont supportés.

Le programme « J » ci-dessous, est un exemple qui va ajouter à toute les classes d'un package, la méthode « Print » dont le code sera : « pour tous les attributs A de la classe courante, insérer l'instruction « cout << Nom attribut ; » (impression des attributs en C++). C'est un exemple simple de transformation de modèle.

```

#Example#Package::AddPrint () -- déclaration de méthode sur un Package sous le
                                -- profile "Example"
{
    string Body ;
    OwnedElementClass           -- ensemble des classes du package
    {                           -- Traitement portant sur chacune des classes
        PartAttribute           -- ensemble des attributs de la classe
        {                       -- Traitement sur chaque attribut
            Body= "cout" + "<< " + Name);
        }
    Body.concat("; ");
    CreateAndAddMethod("Print",Body);--Ajout de la méthode "Print" à la classe
    }
}

```

UML Profile Builder fournit enfin un ensemble *de bibliothèques prédéfinies J*, qui fournissent des briques de base pour les exploitations du modèle UML augmentant encore la productivité. Par exemple, ces bibliothèques fournissent des services de pilotage de l'atelier UML Modeler ; des primitives de création d'éléments de modèle de haut niveau ; des bibliothèques d'accès au système d'exploitation (fichiers, processus, etc.); des bibliothèques de gestion des produits de développement ; des services d'inspection. Enfin, J est un langage interprété, qui peut évaluer dynamiquement un fragment de programme J.

J permet de faire aisément par programmation tout ce qu'un utilisateur peut faire avec UML Modeler. Là résident ses ressources illimitées d'exploitation des modèles.

J offre des facultés de paramétrage des « profiles » uniques sur le marché

Les méthodes du langage J sont structurées en même temps par les classes (métaclasses) et les profiles. Ainsi, une méthode « generate() » sera définie sur la métaclasse « Class » sous le profile « C++ ». Ce mécanisme original permet de surcharger des profiles par des profiles héritiers en J. Il confère à Objecteering une capacité de paramétrage unique sur le marché. Ainsi, lorsque l'on veut adapter un profile, comme typiquement un profile d'un module standard Objecteering tel que « C++ », il suffit de créer un profile « C++ Spécifique » héritier du profile « C++ » [Figure 9], et de redéfinir les méthodes « J » dont on veut adapter le comportement. J est un langage possédant un mécanisme de double « lookup », pour gérer l'héritage des classes en même temps que celui des profiles.

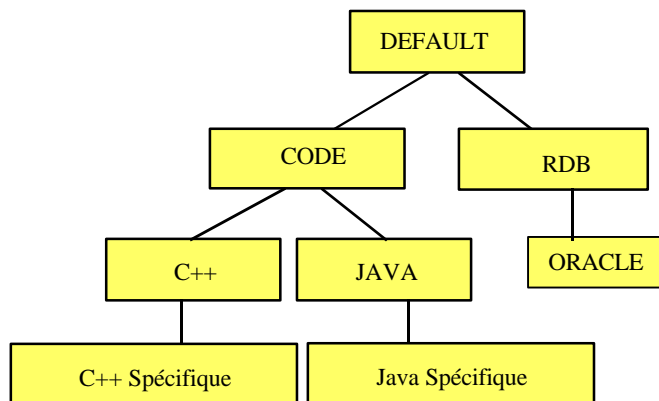


Figure 9 – Héritage de Profiles

Exemple d'applications : tirer le maximum de bénéfices de UML

Génération de code

Les générateurs de code Objecteering sont tous réalisés avec l'atelier UML Profile Builder. Les générations pour les langages C++, Java, IDL, SQL, sont notamment réalisées avec cet atelier. De nombreux autres exemples existent comme les bases de données objet, ou encore des bibliothèques spécifiques de composants. En quelques jours, on peut construire un générateur avec UML Profile Builder pour n'importe quel langage. UML Profile

Builder apporte des facilités uniques pour cette construction, comme par exemple gérer la cohérence permanente modèle UML/code généré, ou encore automatiser des « design patterns » très utiles pour un langage spécifique. UML Profile Builder apporte enfin naturellement la possibilité de paramétrer un générateur, offrant ouverture et extensibilité aux utilisateurs des modules réalisés.

La technique des *plans types* renforce encore ces facultés, en permettant une programmation visuelle, orientée sur la structure du langage cible, auto-documentée, et encore plus facile à paramétrer.

La *génération de documentation* est un cas particulier de génération de code ayant des services spécifiques associés (outils de formatage, génération de liens hypertextes, insertion de vues graphiques, insertion de textes typés) qui assurent une documentation professionnelle et précise, répondant à tous types d'objectifs comme par exemple documentation d'analyse, mais aussi documentation d'audit, ou rapport de tests.

Ces générateurs peuvent ensuite être déployés sur les environnements « UML Modeler ».

Paramétrage des générateurs

Bien souvent, UML Profile Builder est utilisé pour paramétrer des générateurs fournis avec Objectteering. Les facultés d'héritage de profile, de redéfinitions de plans types, de surcharge de méthodes J font de UML Profile Builder l'outil le plus puissant en la matière.

Par ailleurs, certains générateurs comme C++ ou Java ou SQL permettent de paramétrer la génération des bibliothèques de base indépendamment de celle du générateur. De cette façon, on peut changer simplement une bibliothèque de base C++ par une autre, pour fournir d'autres types de base et changer les mode de gestion des ensembles (containers, etc.). Ainsi, sur un même principe de génération C++, on peut inter-changer des bibliothèques comme par exemple STL et MFC de manière indépendantes.

Analyse ou audit d'un modèle

Les facultés uniques de J de navigation dans un modèle, couplés aux facultés de génération de documentation, et à la possibilité de créer des rapports interactifs permettent de réaliser toutes sortes d'analyses sur le modèle et d'éditer des rapports.

Ceci s'applique à la rédaction de nouvelles règles de cohérence, mais aussi à celle de modules comme « Metrics » mesurant la qualité d'un modèle, ou à celle d'outils de mesures d'impact, ou à celle d'outil de recherche d'éléments selon certains critères .

Transformation de modèles

La transformation de modèle est un des services les plus impressionnants pouvant être faits avec UML Profile Builder. De cette façon, on crée des assistances à la modélisation comme par exemple des design patterns automatisés, ou encore des services complétant automatiquement un modèle pour un objectif donné (ajout d'annotations automatiquement, création automatique de diagrammes, etc.).

Les design patterns C++ et Java, ainsi que les idiomes Java en sont quelques exemples.

SOFTEAM, un savoir faire reconnu depuis 1993

Les techniques présentées dans ce « white paper ne sont pas une nouveauté pour SOFTEAM, même si elles demeurent innovante sur le marché. En 1993, SOFTEAM a édité Objectteering version 3.4, supportant une technique appelée « Hypergénéricité ». Des livres ont été publiés sur ce sujet¹, et des centaines de projet l'ont utilisé pour réaliser leurs développements.

Dans sa dernière version (Objectteering 4.3), cette technique a été appliquée aux profiles, en apportant toute la maturité issue de sept années d'expériences, et de six versions majeures de cet atelier.

¹ Object Engineering – The fourth dimension. Philippe Desfray;

Addison Wesley 1994