Etat de l'art sur le développement logiciel basé sur les transformations de modèles

Samba Diaw* — Redouane Lbath* — Bernard Coulette*

** **Université de Toulouse***
Laboratoire IRIT-UTM, Université de Toulouse 2-Le Mirail
*5, allées A. Machado 31058 **TOULOUSE** CEDEX 9*
{diaw, lbath, coulette}@univ-tlse2.fr

RÉSUMÉ. L'Ingénierie Dirigée par les Modèles (IDM) est une discipline récente du génie logiciel qui met les modèles au premier plan au sein du processus du développement logiciel. Elle a apporté plusieurs améliorations significatives dans le développement des systèmes logiciels complexes en fournissant des moyens permettant de passer d'un niveau d'abstraction à un autre ou d'un espace technologique à un autre. Cependant, la gestion des modèles peut s'avérer lourde et coûteuse. Pour pouvoir mieux répondre aux attentes des utilisateurs, il est nécessaire de fournir des outils flexibles et fiables pour la gestion automatique des modèles ainsi que des langages dédiés pour leurs transformations. Dans cet article, nous proposons un tour d'horizon sur les travaux récents de l'IDM en mettant l'accent sur la transformation de modèles qui constitue le thème central de cette discipline.

MOTS-CLÉS : *Ingénierie dirigée par les modèles (IDM), Architecture dirigée par les modèles (MDA), Transformation de modèle*

ABSTRACT. MDE (Model-Driven Engineering) is a recent software engineering discipline that focuses on models within the software development process. It has allowed several significant improvements in the development of complex software systems by providing the means that enable to switch from one abstraction level to another or from one modelling space to another. However, models management may be tedious and costly. Thus, it is necessary to provide some flexible and reliable tools for automatic management of models and some specific languages for their transformations in order to live up to users' expectations. In this paper, we present the state of the art of recent works in the MDE domain by focusing on models transformation which constitutes the central topic of this discipline.

KEYWORDS: *Model-Driven Engineering (MDE), Model-Driven Architecture (MDA), model transformation*

1. Introduction

L'Ingénierie Dirigée par les Modèles (IDM) est une discipline récente du génie logiciel qui promeut les modèles en entités de première classe dans le développement logiciel (Bézivin, 2004). Elle fait l'objet d'un grand intérêt aussi bien de la part des équipes de recherche académiques (Action IDM-website) que des laboratoires industriels (Compuware-website, 2008), (Softeam, 2008), (AndroMDA, 2008), (OAW, 2008), (Xactium-website), etc.

Persuadés que le modèle est devenu le paradigme majeur par lequel l'industrie du logiciel pourra lever le verrou de l'automatisation du développement, des organismes tels que l'OMG (Object Management Group) et les chercheurs en génie logiciel, cherchent à enrichir les modèles qui sont utilisés dans la conception d'applications ou à en définir de nouveaux, à faciliter la création de nouveaux espaces technologiques plus adaptés aux besoins des utilisateurs, et à faciliter les différentes étapes de modélisation nécessaires à l'élaboration d'un produit fini. Ainsi, pour obtenir un produit fini qui réponde aux attentes des utilisateurs, il est nécessaire de pouvoir transformer des modèles d'un niveau d'abstraction à un autre ou d'un espace technologique à un autre.

L'IDM est donc une forme d'ingénierie générative par laquelle tout ou partie d'une application informatique est générée à partir de modèles. Dans cette nouvelle perspective, les modèles occupent une place de premier plan parmi les artéfacts de développement des systèmes et doivent en contrepartie être suffisamment précis et riches afin de pouvoir être interprétés ou transformés par des machines. Le processus de développement des systèmes peut alors être vu comme une séquence de transformations de modèles partiellement ordonnée, chaque transformation prenant un ou des modèles en entrée et produisant un ou des modèles en sortie, jusqu'à l'obtention d'artéfacts exécutables.

Cette transformation des modèles n'est bien sûr pas une tâche aisée. Il est donc nécessaire de disposer d'outils flexibles pour la gestion des modèles et de langages dédiés pour leurs transformations et leur manipulation tout au long de leur cycle de vie. Pour donner aux modèles cette dimension opérationnelle, il est essentiel de spécifier leur sémantique et donc aussi de décrire de manière précise les langages utilisés pour les représenter. On parle alors de métamodélisation. L'intérêt pour l'IDM a été fortement amplifié, en novembre 2000, lorsque l'OMG a rendu publique son initiative MDA (Model-Driven Architecture) (Soley et al., 2000) qui vise la définition d'un cadre normatif pour l'IDM.

Le travail présenté dans cet article est le fruit d'une étude bibliographique sur l'IDM. Dans cette présentation, nous avons choisi de mettre l'accent sur la transformation de modèles et de ne pas aborder certains aspects néanmoins importants tels que la validation/test (Mottu et al., 2008), la vérification ou la traçabilité (Amar et al, 2008). Ce choix est motivé par le fait que nous avons cherché

à focaliser notre attention sur le thème central du développement IDM qu'est la transformation de modèles, et qu'à lui seul ce thème suffit à justifier un article de synthèse. D'autres travaux de notre équipe IRIT-MACAO (Anwar et al, 2007), (Combemale, 2008), (Ober et al, 2008), (Tran et al., 2007b), (Amar et al, 2008), portent sur d'autres aspects de l'IDM.

Nous précisons que cet article ne s'adresse pas en priorité à des spécialistes de l'IDM mais à des jeunes chercheurs, des enseignants, des industriels,… souhaitant avoir une vue d'ensemble sur l'IDM en général et sur la notion de transformation en particulier. Ce domaine étant particulièrement vaste et évolutif, nous avons bien conscience d'en donner une vision limitée à un instant donné et de ne pas prendre forcément en considération les tout derniers résultats de recherche.

Cet article est organisé comme suit. Dans la section 2, nous définissons les notions clés de l'IDM et présentons les travaux ayant contribué à sa genèse (MIC, Software Factories). Dans la section 3, nous présentons les bases de l'IDM (principalement les standards de l'OMG et l'approche MDA). La section 4 est dédiée à la transformation de modèles avec un panorama d'outils et de langages dédiés. Nous concluons par la section 5 qui reprend les points marquants de l'article et ouvre des perspectives de recherche en IDM.

2. Genèse de l'IDM

L'ingénierie logicielle fournit sans cesse de nouvelles technologies facilitant la mise en œuvre des systèmes informatiques (Blanc, 2005). Quelle que soit l'approche utilisée (fonctionnel, objet, composants, Middleware ou services, etc.), le but est toujours le même : l'accroissement de la qualité des systèmes informatiques tout en facilitant leur mise en œuvre sur une plate-forme d'exécution donnée. L'inconvénient majeur de ces plates-formes réside dans leur complexité croissante et leur rapide évolutivité. Face à cette situation, les chercheurs et les industriels se sont mis d'accord sur le fait que la solution à ce problème devait se traduire par une montée en puissance des modèles, et une séparation plus nette entre le métier et la technologie. Cette décision a permis de faire passer les modèles de leur stade **contemplatif** qui se réduisait à la représentation et à la documentation des systèmes informatiques, à un stade plus **productif** qui envisage l'utilisation des modèles au cœur du cycle de développement de ces systèmes.

Avec cette approche, le code final exécutable n'est plus considéré comme l'élément central dans le processus de développement mais comme un élément – naturellement important – qui résulte d'une transformation de modèles. Dans (Bézivin, 2004) cette transformation de modèles est définie comme la génération d'un ou de plusieurs modèles cibles à partir d'un ou de plusieurs modèles sources.

Dans l'IDM, un *modèle* peut être défini simplement comme une représentation ou une abstraction d'un ou d'une partie d'un système. Par analogie avec le monde de

la programmation, un programme exécutable représente le système alors que le code source de ce programme représente le modèle. De cette première définition découle une nouvelle relation appelée *represents* (Bézivin, 2004) reliant le modèle et le système modélisé. En poursuivant l'analogie avec le monde de la programmation, un programme est écrit ou est exprimé dans un langage de programmation, un modèle est écrit ou est exprimé dans un langage de modélisation. Dans le contexte de l'IDM, la définition d'un langage de modélisation a pris la forme d'un modèle, appelé *métamodèle* (Combemale, 2008) : On dit aussi que le métamodèle représente ou modélise le langage. Il résulte de cette deuxième définition une nouvelle relation appelée *conforms-to* qui lie le modèle et son métamodèle. A titre d'exemple, le métamodèle d'UML offre des concepts permettant de décrire les différents modèles (diagramme de classes, diagramme de cas d'utilisation, …) d'un système.

Toujours par analogie avec le monde de la programmation, on dira qu'un langage de programmation respecte la grammaire BNF. Il résulte de cette définition une troisième relation, également appelée *conforms-to*, qui lie le métamodèle à son *méta-métamodèle*. Le méta-métamodèle est un langage d'expression de métamodèles (par exemple le méta-métamodèle de MOF (OMG-MOF, 2006)). La Figure 1 ci-après résume ces notions de base dans l'IDM.

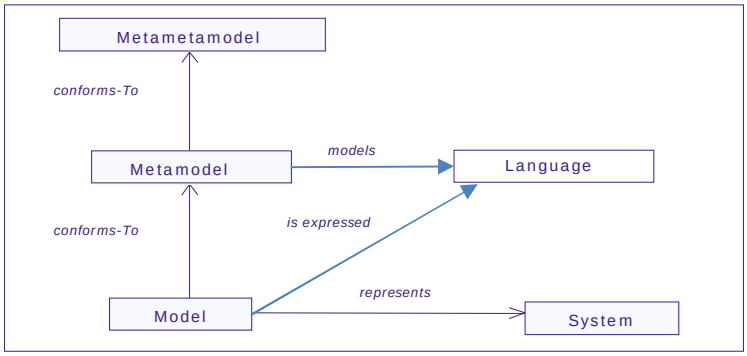

Figure 1. *Notions de base en ingénierie des modèles*

La métamodélisation : l'activité qui consiste à représenter les langages de modélisation par le biais des modèles, a permis l'émergence de plusieurs DSMLs (Domain Specific Modeling Languages, littéralement langages de modélisation spécifiques à un domaine). A l'image des langages de programmation, un DSML est défini par :

- *sa syntaxe abstraite* qui décrit les constructions du langage et leurs relations ;
- *sa syntaxe concrète* qui décrit le formalisme permettant à l'utilisateur de manipuler les constructions du langage ;
- *sa sémantique* qui décrit le sens des constructions du langage.

Plusieurs DSML on été standardisés par l'OMG : CWM (Common Warehouse Metamodel) (OMG-CMW, 2003), le standard pour les techniques liées aux entrepôts de données, *le profil UML pour CORBA* (Common Object Request Broker Architecture) (OMG-CORBA, 2002) qui est un moyen standardisé pour exprimer la sémantique du langage de définition d'interface CORBA (CORBA IDL) en utilisant la notation UML. CORBA est une architecture logicielle, pour le développement de composants et d'Object Request Broker ou ORB. Ces composants, qui sont assemblés afin de construire des applications complètes, peuvent être écrits dans des langages de programmation distincts, être exécutés dans des processus séparés, voire être déployés sur des machines distinctes. IDL (Interface Definition Language) est un langage standardisé et destiné à la définition d'une interface de composants logiciels qui permet de faire communiquer des modules implémentés dans des langages différents (Wikipedia-website).

Pour mieux comprendre la philosophie de l'IDM, nous présentons dans cette section quelques approches à base de modèles qui ont fortement contribué à l'émergence de l'IDM : le Model-Integrated Computing (MIC) (MIC-website) (Sztipanovits et al, 1995) et les usines logicielles de Microsoft (Software factories) (Greenfield, 2004).

2.1. Model-Integrated Computing (MIC)

Les travaux autour du Model-Integrated Computing (MIC) (MIC-website) (Sztipanovits et al., 1995), proposent, dès le milieu des années 90, une vision du génie logiciel dans laquelle les artéfacts de base sont des modèles spécifiques au domaine d'application considéré. Initialement, le MIC est conçu pour le développement des systèmes embarqués complexes mais aujourd'hui il est utilisé aussi dans une large gamme de systèmes logiciels.

Le MIC met particulièrement l'accent sur la représentation formelle, la composition, l'analyse, et la manipulation des modèles durant le processus de conception. Il place les modèles au centre du cycle de vie des systèmes incluant la spécification, la conception, le développement, la vérification, l'intégration, et la maintenance. Le MIC facilite le développement logiciel basé sur les modèles en fournissant une technologie pour la spécification et l'utilisation des DSMLs (Domain-specific modeling languages), des outils de méta-programmation, des techniques de vérification et de transformations de modèles.

Le MIC repose en fait sur une architecture à trois niveaux :

- Le niveau *Meta* qui fournit des langages de métamodélisation, des métamodèles, des environnements de métamodélisation et des méta-générateurs pour créer des outils spécifiques à un domaine qui seront utilisés dans le niveau MIPS (*Model-Integrated Program Synthesis*).

- Le niveau *Modèle* (MIPS) qui est constitué de langages de modélisation spécifiques à un domaine, et de chaînes d'outils pour la construction, l'analyse de modèles et la synthèse d'applications.
- Le niveau *Application* qui représente les applications logicielles adaptables. Dans ce niveau les programmes exécutables sont spécifiés en terme de composition de plates-formes (CORBA, etc.).

2.2. Les usines logicielles (Software Factories)

Les usines logicielles (Software Factories) (Greenfield, 2004) sont la vision par Microsoft de l'ingénierie des modèles. Le terme d'usine logicielle fait référence à une industrialisation du développement logiciel et s'explique par une analogie entre le processus de développement proposé et une chaîne de montage que l'on peut caractériser par les quatre points suivants :

- Si l'on considère comme exemple l'industrie automobile, une chaîne de montage ne fabrique généralement qu'un seul type de voiture avec différentes combinaisons d'options.
- Les ouvriers sont spécialisés. Il n'est pas rare de trouver un ouvrier ayant des compétences sur plusieurs postes de la chaîne de montage mais il est très rare qu'un ouvrier ait toutes les compétences depuis l'assemblage à la peinture des véhicules.
- Les outils utilisés sont très spécialisés et fortement automatisés. Les outils utilisés dans une chaîne de montage sont conçus uniquement pour cette chaîne de montage, ce qui permet d'atteindre des degrés d'automatisation importants.
- Toutes les pièces assemblées ne sont pas toutes fabriquées sur place. Une chaîne de montage automobile ne fait généralement qu'un assemblage de pièces normalisées ou préfabriquées à un autre endroit.

L'idée des usines logicielles est d'adapter ces caractéristiques au développement de logiciels. Les deux premiers points correspondent à la spécialisation des éditeurs de logiciels et des développeurs à l'intérieur des équipes de développement. Le troisième correspond à l'utilisation d'outils de génie logiciel spécifiques au domaine d'application, c'est-à-dire de langages, d'assistants et de transformateurs spécifiques. Le dernier point correspond à la réutilisation de composants logiciels sur étagères. La plate-forme IDM Microsoft DSL Tools est conçue autour de ces idées. Elle a pour vocation de créer des éditeurs graphiques et des générateurs de code personnalisés pour Visual Studio .Net 2005 (Microsoft, 2005), permettant de manipuler des modèles exprimés dans un langage proche des experts métiers, et d'en déduire le code source.

2.3. Synthèse

Les approches classiques à base de modèles ont permis l'émergence de l'IDM dans la mesure où elles font ressortir les idées clés de l'IDM. Le Model-Integrated Computing a fait émerger la notion de générateur d'éditeurs de modèles pour un domaine. Les usines logicielles de Microsoft ont mis en exergue la notion de chaîne de transformation d'artéfacts. Aujourd'hui le terme d'usine logicielle est très utilisé dans le domaine de l'IDM comme en témoigne le projet éponyme au sein du pôle SYSTEM@TIC (SYSTEM@TIC-website).

3. Les bases de l'IDM

L'IDM a pour principal objectif de relever un certain nombre de défis du génie logiciel (qualité, productivité, sûreté, séparation des préoccupations, coût, etc.) en suivant une approche à base de modèles dite *générative*. En focalisant le raisonnement sur les modèles, l'IDM permet de travailler à un niveau d'abstraction supérieur et de vérifier sur une *maquette numérique* (ensemble de modèles qui traitent divers aspects d'un système) un ensemble de propriétés que l'on devait vérifier auparavant sur le système final. L'un des apports supplémentaires du travail sur maquette numérique en lieu et place du traditionnel *code* logiciel est par ailleurs de fournir un référentiel central permettant à plusieurs acteurs de s'intéresser aux différents aspects du système (sécurité, performance, consommation, etc.). Avec cette approche, l'IDM est en passe de lever le verrou qui consistait à ne pouvoir tester un système que tardivement dans le cycle de vie.

Dans la suite de cette section, nous présentons tout d'abord les principaux standards de l'OMG préconisés pour l'IDM. Par la suite, nous décrivons l'approche MDA de l'OMG qui, par sa proposition *historique* d'architecture à quatre niveaux d'abstraction, constitue une base indéniable de l'IDM même si cette architecture n'est pas l'unique manière d'aborder le développement dirigé par les modèles (Favre et al., 2006).

3.1. Standards de l'OMG

L'OMG est un consortium regroupant des industriels, des fournisseurs d'outils et des académiques dont l'objectif principal est d'établir des standards pour résoudre les problèmes d'interopérabilité entre les systèmes d'information. Ces standards sur lesquels repose l'IDM sont centrés sur les notions de métamodèles et de méta-métamodèles.

Dans cette section, nous présentons essentiellement les standards de l'OMG utilisés pour les techniques générales de métamodélisation : MOF (Meta-Object Facility), le standard pour établir de nouveaux langages de modélisation ; UML

(Unified Modeling Language), le standard de modélisation des systèmes ; OCL (Object Constraint Language), le standard pour exprimer la sémantique statique des modèles et des métamodèles ; XMI (XML Metadata Interchange) qui permet de représenter les modèles sous forme de documents XML pour des besoins d'interopérabilité ; et enfin DI (Diagram Interchange) qui permet la représentation au format XML des parties graphiques d'un modèle.

3.1.1. Meta-Object Facility (MOF)

MOF (OMG-MOF 2.0, 2006) se situe au sommet dans l'architecture à quatre niveaux de l'OMG (voir figure 2 ci-après). MOF est un méta-formalisme c'est-à-dire un formalisme pour établir des langages de modélisation permettant eux-mêmes d'exprimer des modèles. Dans sa version 2.0 le métamodèle MOF est constitué de deux parties : EMOF (Essential MOF), pour l'élaboration des métamodèles sans association, et CMOF (Complete MOF) pour les métamodèles avec associations. Il faut souligner que MOF s'auto-décrit pour pouvoir limiter l'architecture pyramidale de l'OMG à quatre niveaux (Figure ci-dessous). Dans cette architecture, le monde réel est représenté à la base de la pyramide (niveau M0). Les modèles représentant cette réalité constituent le niveau M1. Les métamodèles permettant la définition de ces modèles (UML, SPEM, etc.) constituent le niveau M2. Enfin, le méta-métamodèle (MOF), unique et métacirculaire, est représenté au sommet de la pyramide (niveau M3).

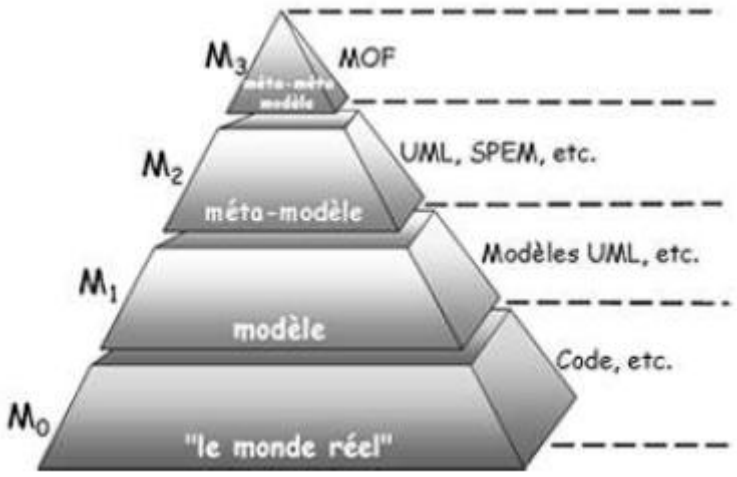

Figure 2. *Pyramide de modélisation de l'OMG (Bézivin, 2003)*

3.1.2. Unified Modeling Language (UML)

UML est le standard de facto en matière de modélisation des systèmes logiciels, notamment dans les domaines d'application tels que l'aéronautique, le spatial ou

l'automobile. Dans sa deuxième version (OMG-UML 2.2, 2009), le métamodèle UML est composé de deux paquetages : UML 2.2 Superstructure qui devient le standard en matière de modélisation orientée objet, et UML 2.2 Infrastructure qui décrit le noyau d'UML commun avec le MOF.

UML Infrastructure a été conçu de façon modulaire pour pouvoir être réutilisé par MOF et UML Superstructure. Il est important de noter qu'il est sans niveau fixe ; il peut appartenir au niveau M3 s'il est intégré dans MOF ou au niveau M2 s'il est intégré dans UML Superstructure.

UML 2.2 Infrastructure est constitué des paquetages *Core ::Abstraction* (permettant la réutilisation des paquetages lors de la création de métamodèles), *Core ::Basic* (permettant l'élaboration des diagrammes de classes sans associations), *Core ::Constucts* (permettant l'élaboration des diagrammes de classes avec associations), *Core ::Primitive Types* (contenant des types primitifs prédéfinis (entier, booléen, chaînes de caractères, etc.) et qui sont utilisés lors de la définition de la syntaxe abstraite des métamodèles), et enfin *Core ::Profiles* (permettant l'élaboration des profils UML).

3.1.3. Object Constraint Language (OCL)

Le but du langage OCL (OMG-OCL, 2005) est de permettre l'ajout de contraintes pour exprimer la sémantique statique des modèles et métamodèles. Un métamodèle n'a pas toujours l'expressivité suffisante pour décrire toutes les relations entre les méta-éléments qu'il contient. Dans le contexte des normes de l'OMG, OCL est utilisé pour décrire des contraintes non capturées dans le métamodèle, sous forme d'invariants. L'ajout de ces contraintes est fondamental pour obtenir des métamodèles clairement définis et donc outillables. En outre, OCL définit des pré et postconditions sur les opérations pour que le système modélisé reste dans un état cohérent quand on exécute ces opérations. Les constructions d'OCL ne permettent pas de créer, de détruire ou modifier les objets d'un modèle : la vérification des contraintes se fait sans effet de bord. En plus de la définition de contraintes, OCL peut être utilisé pour spécifier *de manière déclarative* le comportement de propriétés dérivées et d'opérations, sans toutefois pouvoir exprimer impérativement des modifications de modèles. OCL fournit donc une bonne solution pour exprimer des contraintes sur les modèles mais pas pour décrire le comportement ou la sémantique des modèles. Cette limitation d'OCL a amené l'OMG à définir le Standard AS (Action Semantics) qui permet de modifier l'état d'un modèle. Cependant, il n'y a pas à ce jour de syntaxe concrète standardisée pour AS (Blanc, 2005). Depuis la deuxième version d'UML, le standard AS est intégré dans UML Superstructure. Cette limitation d'OCL ne remet pas en cause ses vertus, d'autant plus qu'il peut être appliqué sur tout type de modèle (MOF, Ecore, UML …) et est utilisé dans plusieurs langages ou normes de transformation (QVT, ATL, Kermeta …).

3.1.4. XML Metadata Interchange (XMI)

Les modèles étant des entités abstraites au niveau conceptuel, l'OMG a décidé de standardiser XMI (Blanc, 2005), (OMG-XMI, 2007) qui offre une représentation concrète des modèles sous forme de documents XML. Cette représentation concrète des modèles se fait par des mécanismes appelés sérialisation et génération. La génération consiste à transformer un métamodèle en un DTD (Document Type Definition) alors que la sérialisation permet de représenter les modèles sous forme de document XML (voir figure 3 ci-dessous).

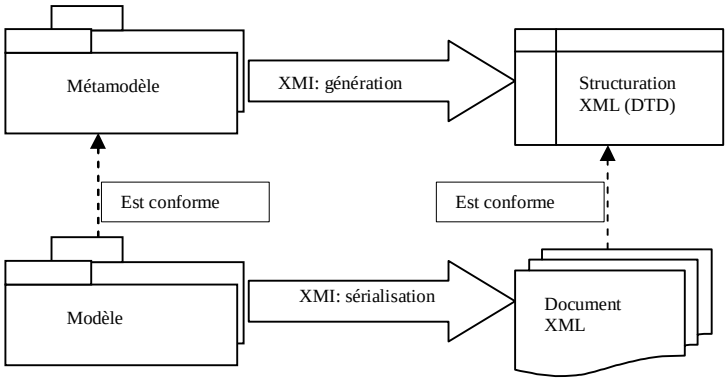

Figure 3. *XMI et la structuration de balises XML (BLANC, 2005)*

3.1.5. Diagram Interchange (DI)

Dans la sérialisation des modèles en documents XML, nous pouvons remarquer que la partie graphique des modèles n'est pas représentée. Cela est dû au fait que XMI ne permet de représenter sous forme de documents XML que les informations dont la structure est définie dans un métamodèle (voir figure 3 ci-dessus). Pour faire face à cette difficulté, l'OMG a décidé de définir un standard spécifique sous le nom de DI (*Diagram Interchange*) (OMG-DI, 2005). L'idée de ce standard est de définir un nouveau métamodèle *DI* lié à UML et représentant sous forme de métaclasses les idiomes graphiques nécessaires et suffisants à la représentation des parties graphiques des modèles UML. En plus de cela, DI définit une transformation de modèles permettant de générer automatiquement des documents *SVG* (W3C-SVG, 2003) à partir des documents XML, ceci dans le but de rendre visibles les parties graphiques des modèles UML dans n'importe quel outil supportant le format SVG.

3.2. L'approche MDA (Model-Driven Architecture)

MDA (Model-Driven Architecture) (Soley et al., 2000) (OMG-MDA, 2001) est une initiative de l'OMG rendue publique en 2000. C'est à la fois la proposition d'une architecture et d'une démarche de développement. L'idée de base de MDA est de séparer les spécifications fonctionnelles d'un système des détails de son implémentation sur une plate-forme donnée. Pour cela, MDA définit une architecture de spécification structurée en plusieurs types de modèles : modèles indépendants de l'informatisation appelés CIM (Computational Independent Model), modèles indépendants des plates-formes appelés PIM (Platform Independent Model), et modèles spécifiques à une plate-forme appelés PSM (Platform Specific Model), générés à partir des PIM en se basant sur les PDM (Platform Description Model). Un CIM modélise les exigences d'un système, son but étant d'aider à la compréhension du problème mais aussi de fixer un vocabulaire commun pour un domaine. Dans UML, le diagramme des cas d'utilisation est un bon candidat pour représenter un CIM. Le PIM, connu aussi sous le nom de modèle d'analyse et de conception, est un modèle abstrait indépendant de toute plate-forme d'exécution. Le PIM a pour but de décrire le savoir-faire ou la connaissance métier d'une organisation. Ayant isolé le savoir-faire métier dans des PIMs, on a besoin soit de transformer ces modèles en d'autres PIMs pour des besoins d'interopérabilité, soit de produire des modèles PSM ciblant une plate-forme d'exécution spécifique en se basant sur les PDMs pour améliorer la portabilité et augmenter la productivité. Le PDM modélise la plate-forme sur laquelle le système va être exécuté (ex. modèles de composants à différents niveaux d'abstraction : PHP, EJB, etc.). Plus précisément, il définit les différentes fonctionnalités de la plate-forme et précise comment les utiliser.

L'initiative MDA de l'OMG a donné lieu à une standardisation des approches pour la modélisation sous la forme d'une structure en 4 niveaux de modélisation (dit encore *Pile de modélisation* (Figure 2) présentée dans la section précédente dédiée à MOF. La proposition initiale était d'utiliser le langage UML et ses différentes vues comme unique langage de modélisation. Cependant, il a fallu rapidement ajouter la possibilité d'étendre le langage UML, par exemple en créant des profils, afin d'exprimer de nouveaux concepts relatifs à des domaines d'application spécifiques. Ces extensions devenant de plus en plus importantes, la communauté MDA a élargi son point de vue en considérant les langages de modélisation spécifiques à un domaine (DSML). La figure 4 ci-après donne une vue générale d'un processus MDA appelé couramment cycle de développement en Y en faisant apparaître les différents niveaux d'abstraction associés aux modèles.

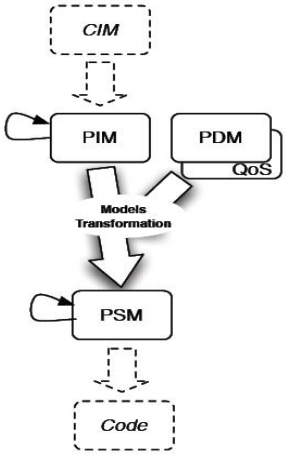

Figure 4. *Principes du Processus MDA*

Aujourd'hui beaucoup d'outils adhèrent à la mouvance MDA. Parmi les plus récents nous pouvons citer :

- AndroMDA (AndroMDA, 2008) qui est une plate-forme de génération de code extensible qui transforme les modèles UML en composants déployables sur une plate-forme donnée (J2EE, Spring, .NET, etc.).
- OAW (OAW, 2008), l'acronyme de OpenArchitectureWare. C'est une plate-forme modulaire de génération développée en Java, qui supporte les modèles EMF, UML2, XML ou des modèles exprimés en JavaBeans, etc.

4. Transformations de modèles

La définition de transformations a pour objectif de rendre les modèles opérationnels dans une approche IDM, ce qui augmente considérablement la productivité des applications. La transformation de modèles est une opération très importante dans toute approche orientée modèle. En effet, les transformations assurent les opérations de passage d'un ou plusieurs modèles d'un niveau d'abstraction donné vers un ou plusieurs autres modèles du même niveau (transformation horizontale) ou d'un niveau différent (transformation verticale). On peut citer comme exemple de transformation verticale, la transformation PIM vers PSM dans l'approche MDA. Le modèle transformé est appelé *modèle source* et le modèle résultant de la transformation est appelé *modèle cible*. Cette section introduit le standard QVT, les principes de la transformation de modèles, et les outils IDM utilisés dans la transformation de modèles.

4.1. La norme QVT (Query, Views, Transformation)

Les transformations de modèles étant au cœur de l'IDM, un standard dénommé QVT (Query, Views, Transformation) (OMG-QVT 2.0, 2008) a été établi pour modéliser ces transformations. Ce standard définit le métamodèle permettant l'élaboration des modèles de transformation. L'appel à proposition de l'OMG d'avril 2002 pour la norme QVT (OMG-QVT, 2002) visait à atteindre les objectifs suivants :

- normaliser un moyen d'exprimer des correspondances (transformations) entre langages définis avec MOF ;
- exprimer des requêtes (Query) pour filtrer et sélectionner des éléments d'un modèle (notamment sélectionner les éléments source d'une transformation) ;
- proposer un mécanisme pour créer des vues (Views) qui sont des modèles déduits d'un autre pour en révéler des aspects spécifiques ;
- formaliser une manière de décrire des transformations (Transformations).

La spécification finale de MOF QVT (OMG-QVT, 2005) fut adoptée par l'OMG en novembre 2005. Les grandes lignes retenues dans cette spécification étaient les suivantes :

- La syntaxe abstraite d'un métamodèle QVT sera décrite en MOF et sa syntaxe concrète sera textuelle ou graphique ;
- Le langage de requête devra s'appuyer sur le langage OCL ;
- La gestion des liens de traçabilité devra être automatique ;
- MOF QVT devra permettre plusieurs scenarii d'exécution (transformations uni-directionnelles, multi-directionnelles, incrémentales, etc.).

La dernière version du standard QVT (OMG-QVT 2.0, 2008) (Jouault et al., 2006) présente un caractère hybride dans le sens qu'elle est composée de trois langages de transformation différents (Figure 5 ci-dessous). La partie déclarative de QVT est définie par les langages Relations et Core ayant des niveaux d'abstraction différents. Relations est un langage orienté utilisateur permettant de définir des transformations à un niveau d'abstraction élevé. Il a une syntaxe textuelle et graphique. Le langage Core forme l'infrastructure de base pour la partie déclarative ; c'est un langage technique de bas niveau défini par une syntaxe textuelle. Il sert à spécifier la sémantique du langage Relations, sous la forme d'une transformation Relations2Core. La vision déclarative passe par une association de patterns, côté source et cible pour exprimer la transformation, en laissant l'outil se débrouiller seul. Manifestement, elle permet une expression plus simple des transformations de type mappings.

La composante impérative de QVT est supportée par le langage Operational Mappings. La vision impérative impose une navigation explicite et une création explicite des éléments du modèle cible. Le langage Operational Mappings étend les deux langages déclaratifs de QVT en ajoutant des constructions impératives

(séquence, sélection, répétition, etc.) ainsi que des constructions OCL à effet de bord. Les langages de style impératif sont mieux adaptés pour des transformations complexes qui comprennent une composante algorithmique importante. Par rapport au style déclaratif, ils ont l'avantage de gérer les cas optionnels dans une transformation.

Enfin, QVT propose un deuxième mécanisme d'extension pour spécifier des transformations, en permettant d'invoquer des fonctionnalités de transformations implémentées dans un langage externe (*Black Box*).

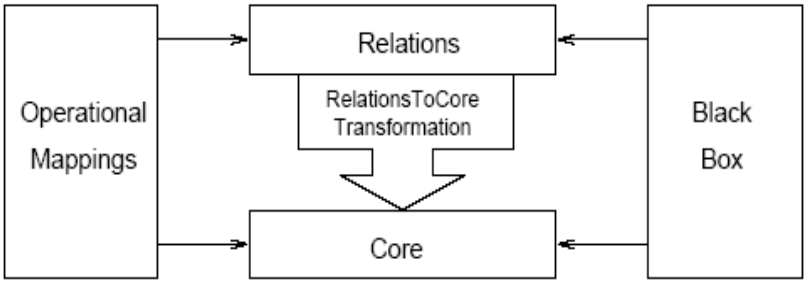

Figure 5. *Architecture du standard QVT (OMG-QVT 2.0, 2008)*

Dans la littérature, beaucoup d'outils implémentent le standard QVT. Parmi eux, nous pouvons citer : SmartQVT (Alizon et al., 2008), MediniQVT (Medini QVT-website), etc.

- SmartQVT est une implémentation du langage standardisé QVT-Operational qui permet la transformation de modèles. Cet outil compile des transformations de modèles exprimées en QVT pour produire du code Java exécutables. Cet outil est fourni sous forme de plug-in Eclipse basé sur le framework de métamodélisation EMF. Comme il implémente un langage impératif, il est bien adapté pour décrire des transformations complexes.
- MediniQVT est une implémentation du langage standardisé QVT-Relations qui permet la transformation de modèles à modèles. Cet outil inclut un éditeur pour spécifier de manière déclarative les transformations et un débogueur graphique. Comme il implémente un langage déclaratif, cet outil n'est pas adapté pour des transformations faisant intervenir beaucoup de modèles.

4.2. Principales approches de transformation de modèles

Dans (Blanc, 2005) trois approches de transformations de modèles sont identifiées : l'approche par programmation, l'approche par template et l'approche par modélisation.

L'approche par *programmation* consiste à utiliser les langages de programmation en général, et plus particulièrement les langages orientés objet. Dans cette approche, la transformation est décrite sous forme d'un programme à l'image de n'importe quelle application informatique. Cette approche reste très utilisée car elle réutilise l'expérience accumulée et l'outillage des langages existants.

L'approche par *template* consiste à définir des canevas pour les modèles cibles souhaités. Ces canevas sont des *modèles cibles paramétrés* ou *modèles templates.* L'exécution d'une transformation consiste à prendre un modèle template et à remplacer ses paramètres par les valeurs d'un modèle source. Cette approche par template est implémentée par exemple dans *Softeam MDA Modeler*.

L'approche par *modélisation* consiste, quant-à-elle, à appliquer les concepts de l'ingénierie des modèles aux transformations des modèles elles-mêmes. L'objectif est de modéliser les transformations de modèles et de rendre les modèles de transformation pérennes et productifs, en exprimant leur indépendance vis-à-vis des plates-formes d'exécution. Le standard MOF 2.0 QVT de l'OMG a été élaboré dans ce cadre et a pour but de définir un métamodèle permettant l'élaboration des modèles de transformation de modèles. A titre d'exemple, cette approche a été choisie par le groupe ATLAS pour le langage de transformation de modèles ATL qui est détaillé dans la section 4.9. Dans l'approche par modélisation, la transformation se fait par l'intermédiaire de règles de transformations qui décrivent la correspondance entre les entités du modèle source et celles du modèle cible. En réalité, la transformation se situe au niveau des métamodèles source et cible qui décrivent la structure des modèles cible et source. La figure 6 ci-dessous détaille le schéma de base d'une transformation dans une approche IDM. Le moteur de transformation de modèles prend en entrée un ou plusieurs modèles sources et crée en sortie un ou plusieurs modèles cibles. Une transformation des entités du modèle source met en jeu deux étapes.

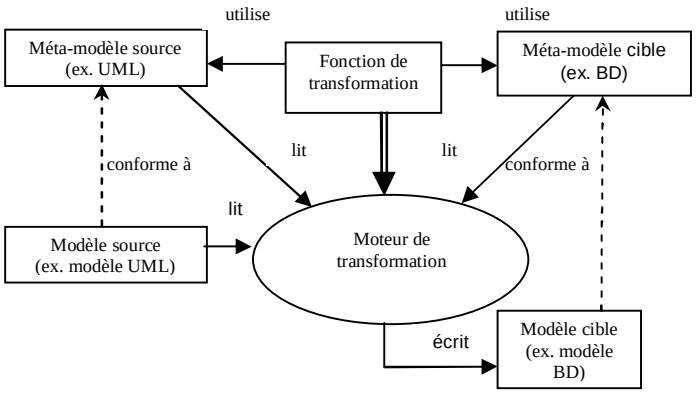

Figure 6. *Schéma de base d'une transformation (TOPCASED-WP5, 2008)*

La première étape permet d'identifier les correspondances entre les concepts des modèles source et cible au niveau de leurs métamodèles, ce qui induit l'existence d'une fonction de transformation applicable à tous les modèles conformes au métamodèle source. La seconde étape consiste à appliquer la transformation du modèle source afin de générer automatiquement le modèle cible par un programme appelé moteur de transformation ou d'exécution.

En résumé, nous pouvons dire que ces différentes approches ont pour finalité commune la transformation de modèles ; ce qui les différencie, c'est la façon dont les règles de transformations sont implémentées. L'approche par programmation est sans doute la plus facile à utiliser, car elle ne nécessite que peu d'apprentissage pour ceux qui maîtrisent les langages de programmation orientés objet. Mais cette approche a des limites au niveau de la lecture des modèles stockés sous forme de fichiers XMI. En effet, même en utilisant un analyseur XML standard, il n'est pas simple de lire un fichier XMI, d'autant plus que selon l'outil utilisé, le XMI généré peut différer dans sa forme pour un même contenu. L'approche par template implémentée dans MDA Modeler n'est pas difficile à mettre en œuvre pourvu qu'on dispose d'un langage de définition de template. L'approche par modélisation, même si elle est la plus complexe à mettre en œuvre, reste la plus prometteuse, car elle offre une solution favorisant la pérennité des transformations et facilite donc leur réutilisation.

4.3. Types de transformation

Une transformation de modèles met en correspondance des éléments des modèles cible et source. Dans (TOPCASED_WP5, 2008), on distingue les types de transformation suivants :

- une *transformation simple* (1 vers 1) qui associe à tout élément du modèle source au plus un élément du modèle cible. Un exemple typique de cette transformation est la transformation d'une classe UML munie de ses attributs en une table d'une base de données relationnelle.
- une *transformation multiple* (M vers N) prend en entrée un ensemble d'éléments du modèle source et produit un ensemble d'éléments du modèle cible. Les transformations de décomposition de modèles (1 vers N) et de fusion de modèles (N vers 1) sont des cas particuliers de transformations multiples.
- une *transformation de mise à jour*, encore appelée transformation sur place, consiste à modifier un modèle par ajout, modification ou suppression d'une partie de ses éléments. Dans ce type de transformation, les modèles source et cible sont confondus. Une telle transformation agit directement sur le modèle source sans créer de modèle cible. Un exemple typique est la restructuration de modèles (Model Refactoring) qui consiste à réorganiser les éléments du modèle source afin d'en améliorer sa structure ou sa lisibilité.

4.4. Axes de transformation

Selon la classification de (TOPCASED_WP5, 2008) la transformation de modèles est caractérisée par trois axes principaux : l'axe processus, l'axe métamodèle et l'axe paramétrage.

- L'axe *processus* permet de positionner fonctionnellement les transformations par rapport au processus global d'ingénierie. Il est composé de deux sous-axes : le sous-axe *vertical* qui consiste à faire une transformation de modèles en changeant de niveau d'abstraction (par exemple : raffinement d'un modèle, passage de PIM vers PSM), et le sous-axe *horizontal* qui consiste à faire une transformation de modèles en restant au même niveau d'abstraction (par exemple : Restructuration de modèle (Model Refactoring), Fusion de modèles).
- L'axe *métamodèle* permet de caractériser l'importance des métamodèles mis en jeu dans une transformation. En programmation classique, un algorithme vérifie la conformité des types des variables mis en jeu dans une opération. Nous pouvons faire le parallélisme avec un algorithme de transformation de modèles qui permet de faire des opérations entre modèles (différence, composition, fusion, etc.). Dans le cas de la transformation, les métaclasses des métamodèles correspondent aux types des variables, d'où l'influence des métamodèles dans le code d'une transformation de modèles. Comme exemple de transformation utilisant l'axe métamodèle, on peut citer UMLtoSysML (SysML-website).
- L'axe *paramétrage* permet de caractériser le degré d'automatisation des transformations avec la présence de données pour paramétrer la transformation. Le passage des informations à une transformation de modèles peut être soit interne (par exemple, des valeurs ou des relations fixées dans des règles), soit externe (informations transmises au moment de l'exécution de la transformation). Nous parlerons de *transformation automati*que si tous les paramètres sont mis à la disposition de la transformation avant son exécution et de *transformation semi-automatique* si certains paramètres ne sont renseignés que lors de l'exécution de la transformation.

4.5. Taxonomie des transformations

Partant de la nature des métamodèles source et cible, on distingue selon la classification adoptée dans (TOPCASED_WP5, 2008) les transformations dites *endogènes* et *exogènes* combinées à des transformations dites verticales et horizontales. Une transformation est dite endogène si les modèles cible et source sont conformes au même métamodèle, exogène dans le cas contraire. Une transformation simple ou multiple peut être exogène ou endogène selon la nature des métamodèles source et cible impliqués dans la transformation. Par contre une transformation sur place implique un même métamodèle donc elle est endogène.

Une démarche de transformation peut aussi induire un changement de niveau d'abstraction. Une transformation est dite *verticale* si elle met en jeu différents niveaux d'abstraction dans la transformation. Le passage de PIM vers PSM ou rétro-conception est une transformation exogène et verticale alors que le raffinement est une transformation endogène et verticale. Une transformation est dite *horizontale* lorsque les modèles source et cible impliqués dans la transformation sont au même niveau d'abstraction.

La restructuration, la normalisation et l'intégration de patrons sont des exemples de transformation endogène et horizontale ; tandis que la migration de plates-formes et la fusion de modèles sont des exemples de transformation exogène et horizontale. Il est important de noter que les modèles source et cible peuvent appartenir à des espaces technologiques différents. Un exemple typique est la transformation d'un modèle de classes persistantes en un schéma de base de données relationnelle qui fait intervenir respectivement l'espace technologique de la modélisation, en général UML, et celui des bases de données. La figure 7 ci-après résume les combinaisons possibles entre transformations de modèles.

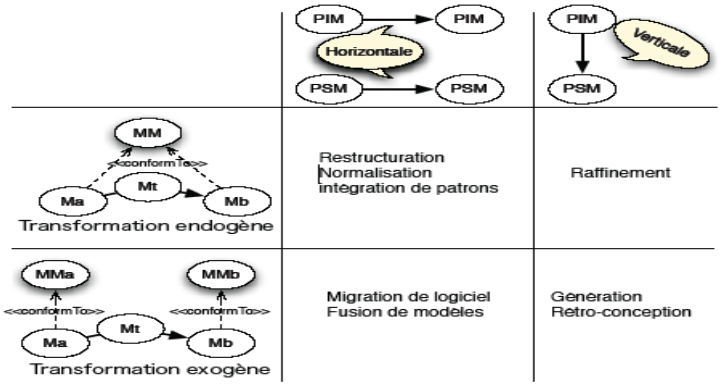

Figure 7. *Taxonomie des transformations de modèles (Combemale, 2007)*

4.6. Propriétés des transformations

Les principales propriétés (TOPCASED-WP5, 2008) qui caractérisent les transformations de modèles sont : la réversibilité, la traçabilité, la réutilisabilité, l'ordonnancement et la modularité.

- *Réversibilité* : une transformation est dite réversible si elle se fait dans les deux sens. Exemple : Model to Text et Text to Model.
- *Traçabilité* : la traçabilité permet de garder des informations sur le devenir des éléments des modèles au cours des différentes transformations qu'ils subissent. Dans un contexte d'ingénierie dirigée par les modèles, il est normal que l'information relative à la traçabilité soit considérée comme un modèle. Un modèle

est donc associé à chaque exécution d'une transformation tracée. La définition d'un métamodèle de traces permet de structurer les traces qui seront générées par la plate-forme de traçabilité et ainsi de mieux les manipuler.

- *Réutilisabilité* : la réutilisabilité permet de réutiliser des règles de transformation dans d'autres transformations de modèles.
- *Ordonnancement* : l'ordonnancement consiste à représenter les niveaux d'imbrication des règles de transformation. En effet, les règles de transformations peuvent déclencher d'autres règles.
- *Modularité* : une transformation modulaire permet de mieux modéliser les règles de transformation en faisant un découpage du problème. Un langage de transformation de modèles supportant la modularité facilite la réutilisation des règles de transformation.

4.7. Les outils génériques

Dans cette catégorie on retrouve les outils de la famille XML (XLST (W3C-XLST, 1999), Xquery (W3C-Xquery, 2007)) et les outils de transformation de graphes AGG (AGG, 2007), UMLX (Willink, 2003), GreAT (Agrawal et al., 2005), etc. qui sont principalement utilisés dans le monde académique. Les outils de la famille XML ont atteint un certain degré de maturité du fait d'une large diffusion dans le monde XML. En revanche, des retours d'expérience montrent que le langage XLST est assez mal adapté pour des transformations de modèles complexes. En effet, selon (Muller, 2006) les transformations XLST ne permettent pas de travailler au niveau de la sémantique des modèles manipulés mais seulement à celui d'un arbre couvrant le graphe de la syntaxe abstraite d'un modèle, ce qui rend ce type de transformation fastidieuse dans le cas des modèles complexes.

4.8. Outils intégrés aux AGL

Dans cette catégorie d'outils nous retrouvons les outils commerciaux (Objecteering 6/MDA Modeler, IBM Rational Software Modeler) (Blanc, 2005) et certains outils du monde académique parmi lesquels OptimalJ (Compuware-website), FUJABA (Burmester et al, 2004), etc.

4.8.1. Objecteering MDA Modeler

Objecteering MDA Modeler (Blanc, 2005) est un outil puissant commercialisé par Objecteering Software. Il permet aux utilisateurs d'élaborer des modèles et d'appliquer des opérations de production sur ceux-ci. Les utilisateurs sont généralement des ingénieurs qualité, des architectes, des ingénieurs méthode ou des chefs de projet. Leur objectif est de capitaliser un savoir-faire identifié d'une entreprise à travers des composants MDA afin que ces derniers puissent être

exploités directement par d'autres outils tels qu'UML Modeler. Dans (Softeam, 2008), un composant MDA est un ensemble fonctionnel autonome apportant des services et des extensions à un modeleur UML pour en étendre ses capacités selon le domaine couvert par ce composant. Un composant MDA est constitué des éléments suivants : un ensemble de *profil*s, des *éléments IHM*, un *modèle Java*, un projet de *test* et un projet *First-Steps.*

MDA Modeler utilise une approche par programmation pour définir des opérations de transformation de modèles. Dans sa dernière version, MDA Modeler est accessible depuis Eclipse ce qui permet de programmer en Java des transformations de modèles par le biais des classes d'implémentation associées à chaque élément d'UML. L'architecture de MDA Modeler ne supporte que des transformations de modèles endogènes (modèle UML vers modèle UML).

La création d'une transformation de modèles se fait dans MDA Modeler par le biais de la création de profils UML. MDA Modeler offre un support graphique de modélisation des profils, représentant les métaclasses référencées, les stéréotypes utilisés et leurs propriétés. La définition d'une transformation de modèles nécessite de référencer toutes les métaclasses UML concernées par la transformation. Ces références contiennent le code de transformation qui sera en langage J ou Java selon le cas. En résumé nous pouvons dire que MDA Modeler offre des mécanismes permettant de définir des opérations sur les modèles et ainsi de les rendre productifs. La figure 8 ci-après résume les principes des outils MDA Modeler et UML Modeler.

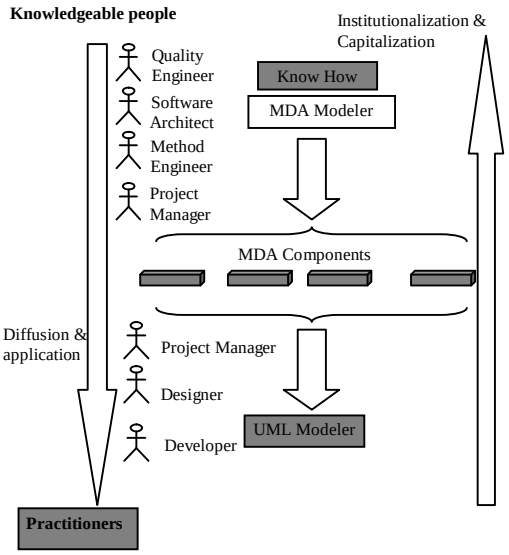

Figure 8. *Principes de MDA Modeler et UML Modeler (Softeam, 2008).*

4.8.2. IBM Rational Software Modeler (RSM)

RSM (Blanc, 2005) permet à ses utilisateurs de définir des transformations de modèles. L'architecture de RSM permet de définir n'importe quel métamodèle. Il est donc théoriquement possible de définir des transformations exogènes. Après la création d'un projet de transformation de modèles, la définition d'une transformation de modèles passe par la spécification de l'ensemble des règles qui composent la transformation. Chaque règle porte sur un unique élément d'UML – par exemple une classe, un paquetage – et dispose de sa propre classe d'implémentation où seront codées en Java les transformations effectuées sur les modèles. Après avoir défini intégralement la transformation par le biais de règles utilisant des classes d'implémentation, RSM peut *packager* le projet de transformation sous forme de plug-in afin de le rendre utilisable dans n'importe quel projet de modélisation. En résumé nous pouvons dire qu'à l'image de MDA Modeler, RSM propose aussi plusieurs mécanismes de définition de transformation de modèles.

4.8.3. OptimalJ

Optimal J (Compuware-website, 2003) est le fruit d'un long travail au sein de la société Compuware qui utilise le langage TPL (Template Pattern Language) comme langage de transformation. L'outil OptimalJ représente un exemple typique de l'approche dite *guidée par la structure* qui est une approche de transformation de modèles à modèles. L'outil propose l'ajout de patrons de transformation au processus métier (PIM). Les approches dites guidées par la structure sont caractérisées par un processus de transformation de modèles composé de deux étapes. La première étape permet de créer la structure hiérarchique du modèle cible (PSM), alors que durant la seconde étape, les attributs et les références sont mis en place afin de compléter le modèle produit. Dans cette approche, l'environnement de transformation définit l'ordonnancement et la stratégie d'application des règles, l'utilisateur ne fournissant que les règles de transformation. La version 3 de l'outil OptimalJ est un exemple d'implémentation du langage Core de QVT.

4.8.4. FUJABA

FUJABA (Burmester et al, 2004), l'acronyme de *From UML to Java and Back Again*, a pour but de fournir un environnement de génération de code Java et de rétro-conception. Il utilise UML comme langage de modélisation visuel. Durant la dernière décennie, l'environnement de FUJABA est devenu une base pour beaucoup d'activités de recherche notamment dans le domaine des applications distribuées, les systèmes de bases de données ainsi que dans le domaine de la modélisation et de la simulation des systèmes mécaniques et électriques. Ainsi, l'environnement de FUJABA est devenu un projet open-source qui intègre les mécanismes de l'IDM.

4.8.5. Synthèse

L'intérêt de cette catégorie d'outils essentiellement proposés par des vendeurs d'atelier de génie logiciel est d'une part leur maturité et d'autre part leur excellente intégration dans l'atelier qui les héberge. L'inconvénient en est que la majeure partie d'entre eux intègre des langages de transformation propriétaires qui s'adaptent difficilement à l'évolution de l'IDM. Ces outils montrent aussi leur limite lorsque les transformations de modèles deviennent complexes et qu'il faut les gérer, les faire évoluer et les maintenir sur une longue période. MDA Modeler ne peut faire que des transformations de modèles endogènes. RSM est limité s'il s'agit de faire une transformation multiple de modèles (M vers N). En outre FUJABA, même s'il présente des idées novatrices, a besoin d'être renforcé notamment dans le cadre du reverse-engineering qui constitue un thème de recherche prometteur de l'IDM. En résumé, nous pouvons dire que les outils intégrés aux AGL ont atteint un bon niveau de maturité (RSM et MDA Modeler) mais présentent certaines limitations.

4.9. Les langages/outils dédiés à la transformation de modèles

Dans cette catégorie, on retrouve les outils et langages conçus spécifiquement pour faire de la transformation de modèles et prévus généralement pour être intégrables dans les environnements de développement standard (Eclipse par exemple). Parmi ces outils, nous pouvons citer Mia-Transformation (MiaSoftware-website) de Mia-Software (outil qui exécute des transformations de modèles prenant en charge différents formats d'entrée et de sortie (XMI, API, etc.)), et PathMate de PathFinders Solution (PathFinders Solution-website) (outil configurable qui s'intègre dans les AGL implémentant UML).

Dans le monde académique nous retrouvons AndroMDA (AndroMDA, 2008), BOTL (Bidirectional Object oriented Transformation Language) (Marschall et al., 2003), Coral (Alanen et al., 2004) (outil pour créer, éditer, et transformer des modèles à l'exécution), ModTransf (ModTransf-website) (un langage de transformation basé sur les règles d'XML), QVTEclipse (QVTEclipse-website) (une implantation préliminaire du standard QVT dans Eclipse), UMT-QVT (UMT-QVT-website) (UML Model Transformation Tool) et le plug-in Eclipse ADT qui implémente le langage ATL(ATLAS, 2005) (Bézivin et al., 2005) du groupe ATLAS de l'INRIA-LINA. Dans la suite de cette section, nous présentons essentiellement le langage ATL qui est représentatif de cette catégorie des langages dédiés aux transformations, et fait l'objet de nombreuses expérimentations dans la communauté IDM.

ATL est l'acronyme d'ATLAS *Transformation Language* ; c'est un langage à vocation déclarative, mais qui est en réalité hybride et permet de faire des transformations de modèles aussi bien endogènes qu'exogènes. Les outils de transformation liés à ATL sont intégrés sous forme de plug-in ADT (ATL Development Tool) pour l'environnement de développement Eclipse. Un modèle de

transformation ATL se base sur des définitions de métamodèles au format XMI. Sachant qu'il existe des dialectes d'XMI, ADT est adapté pour interpréter des métamodèles décrits à l'aide d'EMF (Eclipse) ou MDR (NetBeans). Afin d'assurer son indépendance par rapport aux autres outils de modélisation, ADT met à disposition le langage KM3 (Kernel MetaMetaModel) qui est une forme textuelle simplifiée d'EMOF permettant de décrire des métamodèles et des modèles. En effet, ADT supporte les modèles décrits dans différentes dialectes d'XMI, qui peuvent par exemple être décrits au format KM3 et transformés au format XMI voulu. Une fois les métamodèles décrits en KM3, ils sont injectés dans le métamodéle Ecore.

ATL est défini par un modèle MOF pour sa syntaxe abstraite et possède une syntaxe concrète textuelle. Pour accéder aux éléments d'un modèle, ATL utilise des requêtes sous forme d'expressions OCL. Une requête permet de naviguer entre les éléments d'un modèle et d'appeler des opérations sur ceux-ci. Une règle déclarative d'ATL, appelée M*atched Rule*, est spécifiée par un nom, un ensemble de patrons sources *(InPattern*) mappés avec les éléments sources, et un ensemble de patrons cibles (*OutPattern*) représentant les éléments créés dans le modèle cible. Depuis la version 2006 d'ATL, de nouvelles fonctionnalités ont été ajoutées telles que l'héritage entre les règles et le multiple *pattern matching* (plusieurs modèles en entrée). Le style impératif d'ATL est supporté par deux constructions différentes. En effet, on peut utiliser soit des règles impératives appelées C*alled Rule*, soit un bloc d'instructions impératives (*ActionBlock*) utilisé avec les deux types de règles. Une *Called Rule* est appelée explicitement en utilisant son nom et en initialisant ses paramètres. La figure 9 ci-dessous présente la syntaxe abstraite d'une règle de transformation ATL.

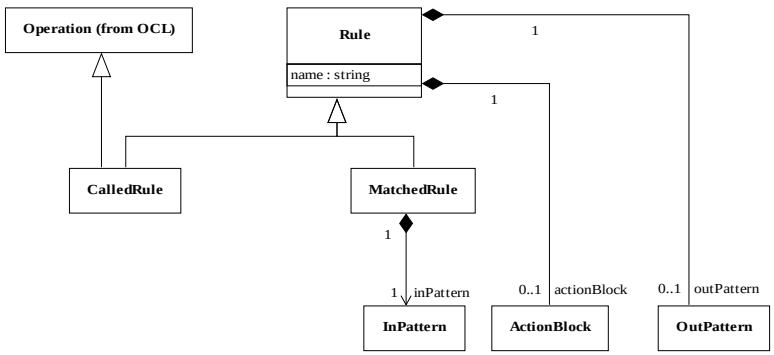

Figure 9. S*yntaxe abstraite d'une règle de transformation ATL (extrait du métamodèle ATL)*

ATL supporte deux modes d'exécution différents : le mode standard et le mode par raffinement. Dans le mode standard, les éléments sont créés seulement quand les patterns sources définis dans les règles déclaratives ont été reconnus dans le modèle ; le système instancie alors les éléments du pattern cible. Une fois l'étape d'instanciation passée, un lien de traçabilité est créé, et associe chaque élément reconnu du modèle source à un élément du modèle cible. Finalement, le système évalue ensuite ces liens de traçabilité afin de déterminer les valeurs des propriétés des éléments instanciés. Dans le mode par raffinement, les éléments dont les patterns sources non pas été *matchés* par les règles sont automatiquement copiés dans le modèle cible par le moteur d'exécution. Ceci réduit considérablement le développement de transformations destinées à modifier une petite partie d'un modèle en gardant le reste inchangé.

4.10. Outils de métamodélisation

La dernière catégorie des outils de transformation de modèles est celle des outils de méta-modélisation dans lesquels la transformation de modèles revient à l'exécution d'un méta-programme. Parmi ces outils nous pouvons citer Kermeta (Triskell, 2005) (Jézéquel et al, 2005) (Muller et al, 2005) de l'IRISA-INRIA Rennes, MetaEdit+ (Tolvanen et al, 2003) de MetaCase (outil qui permet de définir explicitement un métamodèle et au même niveau de programmer tous les outils nécessaires, allant des éditeurs graphiques aux générateurs de code en passant par des transformateurs de modèles), XMF-Mosaic (Xactium-website) de la société Xactium, EMF/Ecore (Eclipse-EMF-website) de la fondation Eclipse, l'outil TOPCASED (TOPCASED_WP5, 2008) (TOPCASED-website) (atelier de développement open source), OpenEmbedded (une plate-forme IDM). Dans la suite de cette section, nous présentons essentiellement Kermeta, XMF-Mosaic, EMF/Ecore et TOPCASED qui sont représentatifs de cette catégorie d'outils de métamodélisation, et qui font l'objet de nombreuses expérimentations dans la communauté IDM.

4.10.1. Kermeta

Dans l'approche par programmation classique, on dit qu'un programme est composé d'un ensemble de structures de données combinées à des algorithmes. C'est sur la base de cette assertion que le langage de métamodélisation Kermeta (Muller et al, 2005) a été élaboré. Une description en Kermeta est assimilable à un programme issu de la fusion d'un ensemble de métadonnées (EMOF) et du métamodèle d'action AS (Action Semantics) qui est maintenant intégré dans UML Superstructure. Le langage Kermeta est donc une sorte de dénominateur commun des langages qui coexistent actuellement dans le paysage de l'IDM. Ces langages sont les langages de métadonnées (EMOF, ECORE,…), de transformation de modèles (QVT, ATL,…), de contraintes et de requêtes (OCL), et d'actions (Action Semantics, Xion).

Kermeta est un véritable langage de métamodélisation exécutable. En effet, le métamodèle de Kermeta est composé de deux packages, Core et Behavior correspondant respectivement à EMOF et à une hiérarchie de métaclasses représentant des expressions impératives. Ces expressions seront utilisées pour programmer la sémantique comportementale des métamodèles. En effet, chaque concept d'un métamodèle Kermeta peut contenir des opérations, le corps de ces opérations étant composé de ces expressions impératives (package Behavior). On peut ainsi définir la structure aussi bien que la sémantique opérationnelle d'un méta-modèle, rendant ainsi, exécutables les modèles qui s'y conforment. Kermeta est intégré à l'IDE Eclipse sous la forme d'un plug-in et, grâce aux outils de génération Ecore2Kermeta, on peut traduire un métamodèle Ecore en un squelette de code Kermeta. Ce squelette servira de base pour le codage des méta-opérations.

Comme UML 2.2 Infrastructure, il intervient à deux niveaux dans l'architecture de métamodélisation de l'OMG. D'une part il intervient comme un langage de niveau M3 c'est-à-dire que tous les métamodèles lui sont conformes, mais également comme une bibliothèque de base pour construire les métamodèles de niveau M2.

4.10.2. XMF-Mosaic

XMF-Mosaic (Xactium-website) est un outil destiné à développer des langages de modélisation et à déployer les outils nécessaires au travail avec ces langages incluant éditeurs, analyseurs de code et transformateurs. Dans XMF-Mosaic, les métamodèles sont considérés comme des langages spécifiques à un domaine et sont appelés modèles de domaine. XMF-Mosaic est régi par un langage appelé *XOCL* (eXtensible Object Constraint Language) qui est un langage basé sur XML pour représenter les contraintes OCL dans les modèles UML. Parallèlement, le logiciel fournit un ensemble d'outils graphiques qui permettent de créer des métamodèles, de construire des modèles correspondants et de décrire des transformations. Pour la transformation de modèles, XMF-Mosaic propose le langage *Xmap*. A l'aide de ce langage, on peut définir un ensemble de règles de transformation ou *mappings* définis chacun entre un ou plusieurs éléments d'un métamodèle source et un élément d'arrivée du métamodèle cible. Le logiciel dispose d'un outil graphique pour définir le squelette d'un *mapping* qui permet d'indiquer les métaclasses de départ, la méta-classe d'arrivée et de définir des dépendances entre *mappings*, lorsque l'un d'entre eux peut être amené à faire appel à une autre règle de transformation.

4.10.3. EMF/Ecore

EMF (Eclipse-EMF-website), qui signifie *Eclipse Modeling Framework*, est un environnement de modélisation et de génération de code qui facilite la construction d'outils et d'applications basées sur des modèles de données structurées. Il est à la base de nombreux outils IDM. Son métamodèle Ecore sert de pivot et permet donc l'interopérabilité entre outils. EMF permet aussi le développement rapide et

l'intégration de nouveaux plug-ins Eclipse. EMF est composé d'un ensemble de briques appelées plug-ins. Parmi ces plug-ins nous citons :

- Le métamodèle *Ecore* qui est un canevas de classes pour décrire les modèles *EMF* et manipuler les référentiels de modèles.
- *EMF.Edit* est un canevas de classes pour le développement d'éditeurs de modèles EMF.
- Le Modèle de génération *GenModel* qui permet de personnaliser la génération Java.
- *JavaEmitterTemplate* qui est un moteur de template générique.
- *JavaMerge* qui est un outil de fusion de code Java.

La rencontre de l'IDM et du phénomène Eclipse a donné lieu à de nombreux projets tels que : EMP (Eclipse Modeling Project) qui met l'accent sur l'évolution et la promotion des technologies de développement basées sur les modèles dans la communauté Eclipse, MoDisco qui fournit une plate-forme extensible pour développer des outils IDM, etc.

EMP est un package qui regroupe plusieurs projets qui traitent de la modélisation afin d'unifier les visions et d'améliorer l'interopérabilité. Ces projets sont : EMF, GMF (Graphical Modeling Framework) qui permet de développer des générateurs d'éditeurs de modèles, UML 2, et GMT (Generative Modeling Tools).

4.10.4. TOPCASED

Le projet TOPCASED (TOPCASED_WP5, 2008) (TOPCASED-website) a pour but de fournir un atelier basé sur l'IDM pour le développement des systèmes logiciels et matériels embarqués. Les autorités de certification pour les domaines d'application de TOPCASED (aéronautique, espace, automobile, etc.), imposent des contraintes de qualification fortes pour lesquelles les approches formelles (analyse statique, vérification du modèle, preuve) sont nécessaires. L'outillage TOPCASED a pour principal objectif de simplifier la définition de nouveaux DSLs (Domain Specific Languages) ou de langages de modélisation en fournissant des technologies de niveau méta telles que des générateurs d'éditeurs syntaxiques (textuels et graphiques), des outils de validation statique et d'exécution de modèle, ce qui lui confère les atouts d'un véritable outil de méta-programmation.

4.10.5. Synthèse

Les outils de métamodélisation proposent des technologies qui permettent de faire des transformations impératives (cas de Kermeta), qui engendrent la syntaxe concrète d'un langage à partir de sa syntaxe abstraite (c'est le cas de TOPCASED et de l'outil OpenEmbedded), et qui permettent l'interopérabilité entre les outils IDM (cas de l'environnement de modélisation EMF). Malheureusement, ces outils sont limités s'il s'agit de transformer beaucoup de modèles ou des modèles de taille importante. C'est ainsi que des recherches s'engagent sur le passage à l'échelle de

l'IDM, la notion de méga-modèles et de modèles infinis, et la gestion collaborative de modèles (Sriplakich, 2007).

5. Conclusion

Le développement logiciel dirigé par les modèles a pour principal objectif de concevoir des applications en séparant les préoccupations et en plaçant les notions de modèles, métamodèles et transformations de modèles au centre du processus de développement. Nous avons présenté dans cet article un état de l'art sur le développement logiciel basé sur les transformations de modèles. Comme nous l'avons dit dans l'introduction, nous avons plus particulièrement mis l'accent sur les concepts, langages et outils associés à la transformation de modèles – paradigme central de l'IDM. Cette focalisation sur la notion de transformation ne doit pas faire oublier qu'une transformation étant in fine un programme exécutable, toutes les problématiques liées au développement du logiciel (test, vérification, traçabilité, …) peuvent s'appliquer à elle. De même, les modèles étant représentables par des graphes (généralement enrichis), les résultats de recherche théoriques sur les graphes sont de plus en plus utilisés pour modéliser et prouver des propriétés sur les modèles. Pour des contraintes d'espace, cet aspect de l'IDM n'a pas été traité dans l'article.

Cette étude a montré la rapide évolution de cette jeune discipline qu'est l'IDM, car plus de la moitié des ouvrages, thèses et articles référencés est postérieure à l'année 2005. L'IDM reste fortement marquée par les propositions de l'OMG (standards, initiative MDA) même si chacun reconnaît aujourd'hui que le développement à base de modèles ne peut pas se cantonner à une vision *orientée UML*. De fait, nous constatons aujourd'hui que l'IDM à langage unique (UML) perd sa position centrale au profit de l'IDM à langage multiple (approche DSL). Cette situation a pour conséquence le développement de nombreux langages spécifiques (par exemple pour l'expression des contraintes, les actions, les transformations, la sérialisation, etc.). Il en résulte par contre une délicate problématique d'intégration de ces langages qui tendent à disperser les concepts de manipulation de modèles. Il faut donc doter l'IDM de moyens pour contrôler la définition et la fusion de ces langages et créer une synergie entre les différents groupes de recherche dans ce domaine. L'action IDM (Action IDM-website) et plusieurs conférences (telles que MODELS, EC-MFA, IDM, etc.) ont été ainsi créées pour offrir un cadre d'échange à la communauté IDM.

Enfin, l'un des défis majeurs de l'IDM dans les prochaines années est sans aucun doute le passage à l'échelle et son industrialisation. Il s'agit entre autres de développer des plates-formes supportant des processus de développement IDM collaboratifs et certifiés pour des logiciels éventuellement critiques, tels que les systèmes embarqués. Parmi les axes dans lesquels notre équipe est d'ores et déjà impliquée, nous pouvons citer : le passage à l'échelle de l'IDM avec des modèles

nombreux et de grande taille (prise en compte des points de vue des utilisateurs), la modélisation et la mise en œuvre automatisée de processus IDM collaboratifs, la traçabilité des processus de développement IDM, la définition et l'application de patrons orientés IDM, la génération par transformations de composants exécutables multi-cibles, etc.

6. Bibliographie

Action IDM, http://www.actionidm.org.

AGG, The Attributed Graph Grammar system (AGG), http://tfs.cs.tu-berlin.de/agg/, 2007

Agrawal A., Karsai G., Kalmar Z., Neema S., Shi F., Vizhanyo A., "The Design of a Language for Model Transformations", *Software and Systems Modeling*, vol. 5, n° 3, 2006, p. 261-288.

Alanen M., Porres I., "Coral: A Metamodel Kernel for Transformation Engines", *Proceedings of the Second European Workshop on Model-Driven Architecture (MDA)*, Canterbury, September 7th-8th 2004, Canterbury, University of kent, p. 165-170.

Alizon F., Belaunde M., Dupre G., Nicolas B., Poivre S., Simonin J., «Les modèles dans l'action à France Télécom avec SmartQVT», *Génie Logiciel, source : Journées Neptune n° 5*, 08/04/2008, p. 35-42.

AndroMDA (2008), available at: http://www.andromda.org/.

Anwar A., Ebersold S., Coulette B., Nassar M., Kriouile A., «Vers une approche à base de règles pour la composition de modèles : Application au profil VUML», *RSTI -L'OBJET*, numéro spécial IDM, vol. 13, n° 4, 2007, p. 73-103.

Amar B., Leblanc H., Dhaussy P., Coulette B., «La description d'un modèle de traçabilité pour la mise en œuvre d'une technique de validation formelle de modèles», *Génie Logiciel, source : Journées Neptune n°5*, 08/04/2008, p. 20-24.

ATLAS group, KM3: Kernel MetaMetaModel, Technical Report version 0.3, August 2005, LINA&INRIA.

Bendraou R., UML4SMP : Un langage de modélisation de procédé de développement de logiciel exécutable et orienté modèle, Thèse de doctorat, Université de Paris VI, 06 Septembre 2007.

Bézivin J., Blanc X., «Vers un important changement de paradigme en génie logiciel» *Journal Développeur Référence* - http://www.devreference.net/, Juillet 2002, p. 1-9.

Bézivin J., Blanc X., «Promesses et Interrogations de l'approche MDA», *Journal Développeur Référence* - http://www.devreference.net/, Septembre 2002, p. 1-14.

Bézivin J., «La transformation de modèles», *Ecole d'Eté d'Informatique CEA EDF INRIA : cours #6*, INRIA-ATLAS & Université de Nantes, 2003.

Bézivin J., «Sur les principes de base de l'ingénierie des modèles » *RTSI-L'OBJET*, vol.10, n° 4, 2004, p. 145-157.

Bézivin J., Jouault F., «Using ATL for Checking Models», *In Proceedings of the International Workshop on Graph and Model Transformation* (*GraMoT): Session 1: Different views on model transformation*, Tallinn, Estonia, 2005, p.1-12.

Blanc X., *MDA en action Ingénierie logicielle guidée par les modèles*, Paris, Eyrolles, 2005.

Burmester S., Giese H., Niere J., Tichy M., Wadsack J., Wagner R., Wendehals L., Zundorf A., "Tool Integration at the Meta-Model Level within the FUJABA Tool Suite", *International Journal on Software Tools for Technology Transfer (STTT*), vol. 6, n° 3, 2004, p. 203-218.

Combemale B., Cregut X., Rougemaille S., Migeon F., Pantel M., Maurel C., «Sémantique dans la méta-modélisation», Actes des *Secondes Journées sur l'Ingénierie Dirigée par les Modèles (IDM'06)*, Lille, 26-28 Juin 2006, Lille, Université de Lille, p. 17-33.

Combemale B., Cregut X., Pantel M., Transformations de modèles : Principes, Standards et Exemples, Rapport de recherche, 05 Novembre 2007, IRIT&CNRS.

Combemale B.., Approche de métamodélisation pour la simulation et la vérification de modèle -- Application à l'ingénierie des procédés, Thèse de doctorat, IRIT&ENSEEIHT, 11 Juillet 2008.

Compuware-website, OptimalJ - Model-driven development for Java, Electronic Source: Compuware, http://www.compuware.com/products/optimalj/, 2003.

Eclipse-EMF, http://www.eclipsetotale.com, http://www.eclipse.org/modeling/emf/.

Favre J., Estublier J., Blay-Fornarino M., *L'ingénierie Dirigée par les Modèles. Au delà du MDA*, Cachan, Hermes-Lavoisier, 2006.

Greenfield J., "Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools", *LNCS*, n° 3286, 2004, p. 403-482.

James D., "GME: the generic modeling environment", *In Proceedings of the 18th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages 94/102 and Applications (OOPSLA '03)*, Anaheim, California, USA, October 2003, New York, ACM Press, p. 82-83.

Jézéquel J., Fleurey F., Drey Z., Muller P., Pantel M., Maurel C., « Kermeta : Un langage de méta-modélisation exécutable compatible avec EMOF/ECORE, et son environnement de développement sous Eclipse », *Actes des Premières Journées sur l'Ingénierie Dirigée par les Modèles IDM'05, session Démos&Posters*, Paris, 30 Juin- 01 Juillet 2005, Paris, Université Pierre & Marie Curie, p. 1-4.

Jouault J., Kurtev I., «On the Architectural Alignment of ATL and QVT», *In Proceedings of the 2006 ACM symposium on Applied computing, session Model transformation*, Dijon, 2006, New York, ACM Press, p. 1188-1195.

Marschall F., Braun P., "Model Transformations for the MDA with BOTL", *In Proceedings of the Workshop on Model-Driven Architecture: Foundations and Applications*, Enschede, the Netherlands, June 26th-27th 2003, Enschede, University of Twente, p. 25-36.

Medini QVT, available at: http://projects.ikv.de/qvt/.

Meylan S, Tatibouet B., Etude et comparaison d'outils de transformation de modèles. Rapport de recherche, Janvier 2006, LIFC –Université de FRANCHE-COMTE.

MiaSoftware-website, Mia –Transformation: e-Source: http://www.mia-software.com/.

MIC-website, MIC: http://www.isis.vanderbilt.edu/research/MIC.

Microsoft, Visual Studio .Net, available at: www.microsoft.com/visualstudio, DSL Tools available at: http://msdn.microsoft.com/vstudio/dsltools/default.aspx, 2005.

MODELS, Electronic source: http://www.modelsconference.org/.

ModTransf, Electronic Source: http://www.lifl.fr/west/modtransf/.

Mottu J. M., Baudry B., Le Traon Y., «Test de Transformation de Modèles : Expression d'Oracles», *Actes des 4èmes Journées sur l'Ingénierie Dirigée par les Modèles IDM '08*, Mulhouse, 5-6 juin 2008, Mulhouse, Université de Haute Alsace, p. 1-16.

Muller P. A., Fleurey F., Jézéquel J. M., «Weaving Executability into Object-Oriented Meta-Languages», *MODELS/UML'2005*, *LNCS 3713*, Montego Bay, Jamaica, October 2005, Berlin/ Heidelberg, Springer, p. 264-278.

Muller Pierre Alain, De la modélisation objet des logiciels à la métamodélisation des langages informatiques, HDR, Université de Rennes 1, 20 Novembre 2006.

Nano O., Blay M., «Une approche MDA pour l'intégration de services dans les plates-formes à composants », *Journées du groupe de travail OCM 2003*, Vannes, Février 2003, p. 1-6.

OAW (2008), available at: http://www.openarchitectureware.org/.

Obeo, L'approche MDA pour accélérer les développements JEE : mythe ou réalité, Rapport Technique, 23 novembre 2006, Société Obeo.

Ober I., Coulette B., Lakhrissi Y., "*In Proceedings of ACM/IEEE International Conference Model-Driven Engineering Languages and Systems (MODELS 2008)*, Toulouse, October 1-3 2008, Berlin/Heidelberg, Springer, p. 219-233.

OMG, CWM, http://www.omg.org/cwm, 2003.

OMG, UML profile for CORBA, http://www.omg.org/cgi-bin/doc?formal/02-04-01, 2002.

OMG, DI 2.0, http://www.omg.org/cgi-bin/doc?ptc/2003-09-01, 2003.

OMG, MDA, http://www.omg.org/mda/specs.htm, 2001.

OMG, MOF2.0, http://www.omg.org/cgi-bin/doc?formal/06-01-01, 2006.

OMG Object Constraint Language (OCL), OMG Available Specification, Version 2.0, source: http://www.omg.org/spec/OCL/2.0/, 2005.

OMG, QVT 2.0 RFP, http://www.omg.org/docs/ad/02-04-10, Avril 2002.

OMG, QVT Final Adopted spec., http://www.omg.org/docs/ptc/05-11-01, November 2005

OMG, QVT 2.0 Transformation Spec., http://www.omg.org/spec/QVT/1.0/PDF/, Avril 2008.

OMG, UML 2.2, http://www.omg.org/spec/UML/2.2/, 2009.

OMG, XMI, http://www.omg.org/spec/XMI/2.1.1/PDF/index.htm, Décembre 2007.

PathfinderSolutions, source: http://www.pathfindermda.com.

Perez-Medina J., Marsal-Layat S., Favre J-M., «Transformation et vérification de cohérence entre modèles du génie logiciel et modèles de l'interface homme-machine», *Actes du Congrès INFORSID'07,* Perros-Guirec, France, May 2007, IRISA/BADINS& ENSSAT, p. 382-397.

QVTEclipse, Electronic Source: http://qvtp.org/downloads/qvtp-eclipse/.

Softeam, support de formation Objecteering 6/MDA Modeler version 2.0, Janvier 2008.

Soley et al., MDA (Model-Driven Architecture), White Paper, Draft 3.2, November 27-th, 2000, OMG Staff Strategy Group

Sottet J., Calvary G., Favre J. M., « Ingénierie de l'interaction homme-machine dirigée par les modèles.», *Actes des Premières Journées sur l'Ingénierie Dirigée par les Modèles IDM'05,* Paris, 30 Juin- 01 Juillet 2005, Paris, l'université Pierre & Marie Curie, p. 1-16.

Sriplakich P., ModelBus : un environnement réparti et ouvert pour l'ingénierie des modèles, Thèse de doctorat, Université de Paris VI, 05 Septembre 2007.

SYSTEM@TIC PARIS REGION, http://www.usine-logicielle.org/.

SysML-website, http://www.sysml.org/.

Sztipanovits J., Karsai G., Biegl C., Bapty T., Ldeczi K., Misra A., "MULTIGRAPH: architecture for model-integrated computing", *In proceedings of the First IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'95),* pp. 361-368, Southern Florida,USA, November 6-10 1995, Washington DC, IEEE Press, p. 361-368.

Tolvanen J.-P., Rossi M., "MetaEdit+: defining and using domain-specific modeling languages and code generators*", In Proceedings of the conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '03),* Anaheim, California, October 26-30 2003, New York, ACM Press, p. 92-93.

TOPCASED-WP5, Guide méthodologique pour les transformations de modèles. Rapport de recherche, version 0.1, 18 Novembre 2008, IRIT/MACAO.

TOPCASED-website, http://www.topcased.org.

Tran H.N., Coulette B., Dong Thi B. T., "Broadening the Use of Process Patterns for Modeling Processes*", In Proceeding of the International Conference on Software Engineering and Knowledge Engineering (SEKE'07),* Boston, July 9-11 2007, Skokie, Illinois, USA, Knowledge Systems Institute, p. 57-62.

Tran H.N., Coulette B., Dong Thi B. T., "Modelling Process Patterns and their Application", *In Proceeding of International Conference on Software Engineering Advances (ICSEA'07),* Cap Estérel, France, August 25-31 2007, Washington, DC, IEEE Press, p. 15-20.

Triskell, Kermeta, IRISA, Rennes, http://www.irisa.fr/triskell, http://www.kermeta.org, 2005.

UMT-QVT, UMT, Electronic Source: http://umt-qvt.sourceforge.net/.

Wikipédia, available at: http://fr.wikipedia.org.

Willink E.D., “UMLX: A graphical transformation language for MDA”, *In Proceedings of the Workshop on Model-Driven Architecture: Foundations and Applications*, Enschede, The Netherlands, June 26th-27th 2003, Enschede, University of Twente, p. 13-24.

W3C, XLST, http://xmlfr.org/w3c/TR/xslt/, Novembre 1999.

W3C, http://www.w3.org/TR/SVG/, Janvier 2003.

W3C Xquery, http://www.w3.org/TR/xquery/, Janvier 2007.

Xactium-website, XMF-Mosaic, Electronic Source: http://www.xactium.com.

7. Biographie

Samba Diaw *termine un doctorat en informatique en cotutelle entre l'Université de Ziguinchor au Sénégal et l'Université de Toulouse II - Le Mirail. Membre de l'équipe MACAO de l'IRIT, et Enseignant-Chercheur à l'Université de Ziguinchor, ses travaux portent sur la modélisation et la mise en œuvre des processus de développement logiciel à base de modèles. Il s'intéresse plus particulièrement à la modélisation des procédés logiciels IDM et à leur mise en œuvre assistée afin de guider les développeurs dans l'élaboration et la génération des modèles.*

Redouane Lbath *est Maître de Conférences en informatique à l'université de Toulouse 1 Capitole et membre permanent de l'équipe MACAO de l'IRIT. Ses premiers travaux de recherche ont porté sur l'Intelligence Artificielle appliquée aux domaines techniques. Depuis 2003, ses travaux se focalisent sur l'application des techniques IA à la modélisation et mise en œuvre du processus de développement logiciel et à l'Ingénierie Dirigée par les Modèles.*

Bernard Coulette *est Professeur en Informatique depuis 2000 à l'Université de Toulouse II - Le Mirail et membre de l'Equipe MACAO de l'IRIT. Ses travaux portent d'une part sur l'analyse/conception par points de vue (profil VUML) et la composition de modèles, d'autre part sur la modélisation et la mise en œuvre des processus de développement IDM. Son activité de recherche et d'encadrement de thèses se déroule en particulier dans le contexte de collaborations internationales (Vietnam, Maroc).*