Etat de l'art sur le développement logiciel dirigé par les modèles.

Samba Diaw* — Rédouane Lbath* — Bernard Coulette*

* ***Université de Toulouse***
Laboratoire IRIT
Université de Toulouse 2-Le Mirail
5, allées A. Machado 31058 ***TOULOUSE*** *Cédex 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 de modélisation 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), Architectures dirigées par les modèles, Transformations de modèles*

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 into another or from one modelling space into 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), models transformations*

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). L'intérêt pour l'IDM était apparu en novembre 2000, lorsque l'OMG (Object Management Group) avait rendu publique son initiative MDA (OMG-MDA, 2001) qui visait à la définition d'un cadre normatif pour l'IDM. Elle fait l'objet depuis d'un intérêt croissant aussi bien de la part des équipes de recherche académiques (cf. par exemple (ATLAS, 2005), (TOPCASED, 2008), (Triskell, 2005)), 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 et les chercheurs en génie logiciel cherchent à enrichir les métamodèles qui sont utilisés dans la conception d'applications ou à en définir de nouveaux, à faciliter la création de nouveaux espaces de modélisation plus adaptés aux besoins des utilisateurs et à automatiser les différentes étapes de modélisation nécessaires à l'élaboration d'un produit fini. Ainsi, pour obtenir un produit logiciel 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 engendrée à partir de modèles. Dans cette nouvelle perspective, les modèles occupent une place de premier plan parmi les artéfacts de développement des systèmes, mais 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 puissants et 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 leurs points de variations sémantiques et donc aussi de décrire de manière précise les langages utilisés pour les représenter. On parle alors de métamodélisation.

Le travail présenté dans cet article est le fruit d'une étude bibliographique sur l'IDM, menée dans le cadre d'une thèse en cours portant sur le développement logiciel dirigé par les modèles, appelé encore *développement IDM*. Dans cette présentation, pour des raisons de place, nous avons choisi de mettre l'accent sur le

thème central du développement IDM qu'est la transformation de modèles et de ne pas aborder certains aspects néanmoins importants tels que la validation et le test (Mottu et al. 2008), la vérification ou la traçabilité (Amar et al, 2008). De nombreux travaux portent sur ces autres aspects de l'IDM, parmi lesquels on peut citer certains travaux de notre équipe IRIT-MACAO (Amar et al, 2008), (Anwar et al, 2007), (Combemale, 2008), (Nassar, 2005), (Ober et al, 2008), (Tran et al. 2007b).

Cet article est organisé comme suit. Dans la section 2, nous présentons d'abord les travaux ayant contribué à la genèse de l'IDM. Dans la section 3, nous décrivons des standards de l'OMG qui sont des métamodèles de référence pour l'IDM. Dans la section 4, la plus importante en taille, nous présentons la problématique et les objectifs de l'IDM, les transformations de modèles avec leur panorama d'outils et de langages dédiés, et les outils de métamodélisation. Nous concluons par la section 5 qui décrit aussi les axes de recherche sur lesquels nous travaillons actuellement.

2. Genèse de l'IDM

Au sens large du terme, l'Ingénierie Dirigée par les Modèles (IDM), utilisée encore sous le terme anglais de MDE (Model Driven Engineering), est la discipline qui place les modèles au centre des processus d'ingénierie logicielle. 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.

Pour mieux comprendre la philosophie et l'apport de l'IDM, nous présentons dans cette section les principales approches de développement logiciel basées sur les modèles qui ont fortement contribué à l'émergence de l'IDM : le MIC (Model-Integrated Computing) (MIC-website) (Sztipanovits et al, 1995), Model Driven Architecture (MDA) (OMG-MDA, 2001) et les usines logicielles de Microsoft (Software factories) (Greenfield et al, 2003) (Muller, 2006).

2.1. Model-Integrated Computing (MIC)

Les travaux autour du Model-Integrated Computing (MIC) (MIC-website) (Sztipanovits et al, 1995), ont proposé, 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 de DSMLs (Domain-Specific Modeling Languages), des outils de métaprogrammation, des techniques de vérification et de transformations de modèles.

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

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

Le niveau (MIPS) est constitué de langages de modélisation spécifiques à un domaine, et de chaînes d'outils pour la construction et l'analyse de modèles, la synthèse d'applications.

Le niveau *Application* représente les applications logicielles adaptables. Dans ce niveau, les programmes exécutables sont spécifiés en terme de composition de plates-formes (CORBA, etc.).

2.2. Model Driven Architecture (MDA)

Le MDA (Model-Driven Architecture) (OMG-MDA, 2001) est une initiative de l'OMG rendue publique en 2000. C'est une proposition à la fois d'une architecture et d'une démarche de développement. L'idée de base du MDA est la séparation des spécifications fonctionnelles d'un système des détails de son implémentation sur une plate-forme donnée. Pour cela, le MDA classe les modèles en modèles indépendants des plates-formes appelés PIM (Platform-Independent Models) et en modèles spécifiques appelés PSM (Platform-Specific Models). L'approche MDA permet de déployer un même modèle de type PIM sur plusieurs plates-formes (modèles PSM) grâce à des projections standardisées. Elle permet aux applications d'interopérer en reliant leurs modèles et favorise l'adaptabilité aux évolutions des plates-formes et des techniques. La mise en œuvre du MDA est entièrement basée sur les modèles et leurs transformations.

L'initiative MDA a donné lieu à une standardisation des approches pour la modélisation sous la forme d'une structure en 4 niveaux de modélisation (appelée communément *Pile de modélisation*) présentée dans la section 3 ci-après. 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 en Anglais). La figure 1 donne une vue générale d'un processus

MDA en faisant apparaître les différents niveaux d'abstraction associés aux modèles, depuis les modèles de besoins jusqu'au code qui s'exécute.

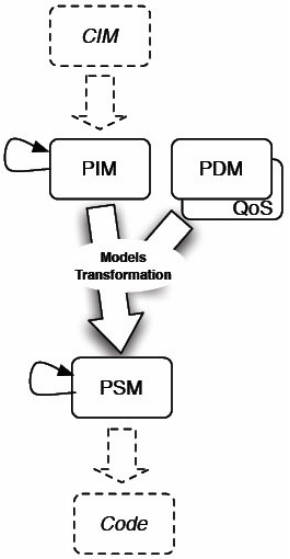

Figure 1. ***Principe du Processus MDA***

Aujourd'hui plusieurs outils respectent cette approche MDA. Parmi les plus récents nous pouvons citer :

- AndroMDA (AndroMDA, 2008) qui est une plate-forme de génération de code extensible qui transforme des modèles UML en composants qui peuvent êtres déployés sur une plate-forme donnée (JEE, Spring, .NET, etc.)

- OAW (OAW, 2008), l'acronyme de OpenArchitectureWare, qui est une plate-forme modulaire de génération développée en Java. Il supporte les modèles EMF, UML2, XML ou des modèles exprimés en JavaBeans, etc.

2.3. Les usines logicielles (Software Factories)

Les usines logicielles (Software Factories) (Greenfield et al, 2003) 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 logiciel et une chaîne de montage que l'on peut caractériser par les quatre points suivants, en prenant par exemple l'industrie automobile :

- Une chaîne de montage ne fabrique généralement qu'un seul type de voitures avec différentes combinaisons d'options.

- Les ouvriers sont relativement 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 dans une chaîne de montage sont très spécialisés et fortement automatisés. Ils 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 fabriquées sur place. Une chaîne de montage automobile ne fait généralement qu'un assemblage de pièces normalisées ou préfabriquées ailleurs.

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 point 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. L'environnement de développement Visual Studio .Net 2005 (Microsoft, 2005) a été conçu autour de ces idées et propose un environnement générique extensible pouvant initialement être configuré pour un ensemble de domaines d'applications prédéfinis.

2.4. Synthèse

Les approches à base de modèles mentionnées ci-dessus ont permis l'émergence de l'IDM dans la mesure en mettant l'accent un certain nombre d'idées clef. En précurseur, 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 *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). Le MDA a mis en évidence l'intérêt de séparer les spécifications fonctionnelles d'une application de son implémentation sur une plate-forme donnée. Suite à l'initiative de l'OMG, il sert de cadre normatif pour l'IDM.

3. L'IDM et les standards de l'OMG

L'OMG est un consortium regroupant des industriels et des chercheurs dont l'objectif principal est d'établir des standards pour résoudre, entre autres, les problèmes d'interopérabilité entre les systèmes d'information. Ces standards sont

centrés sur la notion de métamodèles qui décrivent la structure des modèles et sur celle de méta-métamodèles. Nous donnons les définitions de modèles, métamodèles et méta-métamodèles dans les paragraphes suivants. Ces trois notions sont schématisées par la figure 2 ci-après.

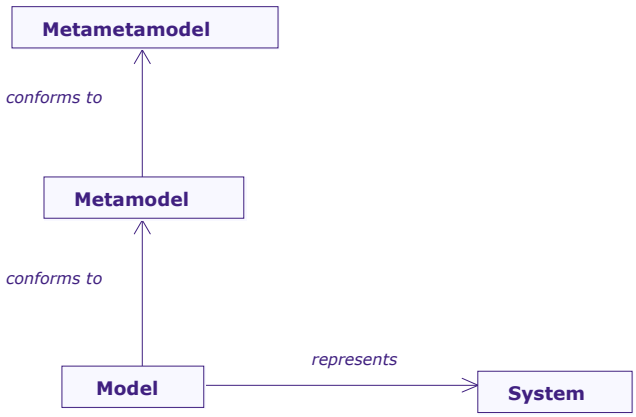

Figure 2. *Notions de base en ingénierie des modèles*

Un modèle peut être défini comme une abstraction d'un système réel qu'il représente. Le mot abstraction fait référence ici à un ensemble restreint d'informations traduisant une partie de la réalité du système, celle qui sera nécessaire à l'exploitation du modèle.

Un métamodèle est un modèle de modèles. C'est aussi un langage utilisé pour décrire des modèles, dans la mesure où il englobe un ensemble de concepts nécessaires à la description d'une famille de modèles donnée. A titre d'exemple, UML est un métamodèle qui offre des concepts permettant de décrire les différents modèles (Diagramme de classe, Diagramme de cas d'utilisation, …) d'un système.

Un méta-métamodèle quant à lui, permet de décrire un modèle de métamodèles ; autrement dit c'est un langage de description de métamodèles (par exemple le méta-métamodèle MOF (OMG-MOF, 2006).

Après avoir défini les notions de modèle, métamodèle et méta-métamodèle, nous présentons dans cette section certains standards de l'OMG relatifs à l'IDM : MOF (*Meta-Object Facility*), le standard pour établir de nouveaux langages de modélisation ; QVT (*Query, Views, Transformation*), le standard pour modéliser les transformations de modèles ; UML (*Unified Modeling Language*), le standard pour la modélisation orientée-objet ; XMI (*XML Metadata Interchange*) qui permet de

représenter les modèles sous forme de documents XML pour des besoins d'interopérabilité ; CWM (*Common Warehouse Metamodel*), le standard pour les techniques liées au entrepôts de données, et enfin DI (*Diagram Interchange*) qui permet la représentation au format XML des parties graphiques d'un modèle. Tous ces métamodèles ont comme objectif commun la pérennité des modèles dans un contexte de développement IDM.

3.1. MOF

Le MOF (OMG-MOF 2.0, 2006) se situe au sommet dans l'architecture à quatre niveaux de l'OMG (voir figure 3 ci-dessous). C'est un méta-formalisme, donc un formalisme, pour établir des langages de modélisation permettant eux-mêmes d'exprimer des modèles. Dans sa version 2.0, le méta-métamodèle MOF est constitué de deux parties : EMOF (Essential MOF), pour l'élaboration des métamodèles sans association, et CMOF (Complete MOF) pour l'élaboration des métamodèles avec associations. Il faut souligner que MOF s'auto-décrit pour pouvoir limiter l'architecture de l'OMG à quatre niveaux

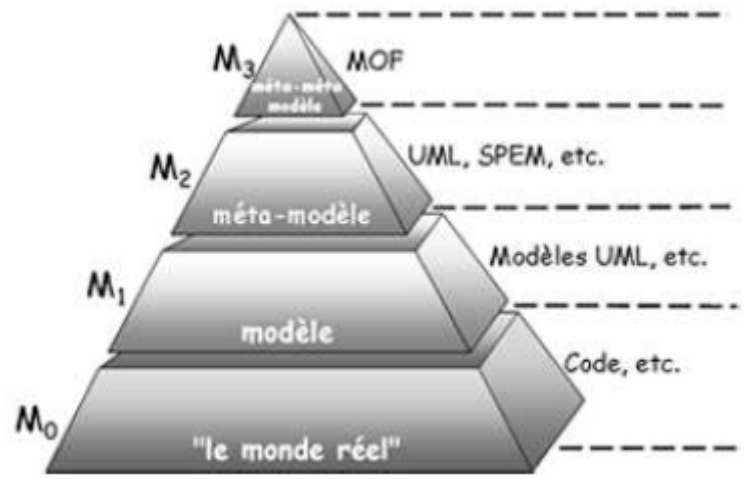

Figure 3. *Pyramide de modélisation de l'OMG (Bézivin, 2003)*

3.2. QVT

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. Par exemple, la transformation d'un

diagramme de classes UML en un schéma de base de données relationnelle est élaborée sous la forme d'un modèle de transformation conforme au métamodèle QVT. Nous verrons en détail cette notion de transformation de modèles dans la section 4.2.

L'appel à proposition de l'OMG d'avril 2002 pour la norme QVT (OMG-QVT, 2002) visait à atteindre les objectifs suivants :

– normaliser l'expression des correspondances (transformations) entre langages définis avec MOF ;

– exprimer des requêtes (Query) pour filtrer et sélectionner des éléments d'un modèle (y compris 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 métamodéle 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) présente un caractère hybride, en proposant trois langages de transformation (Figure 4). 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. *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 composante impérative de QVT est supportée par *Operational Mappings*. Ce langage 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. 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).

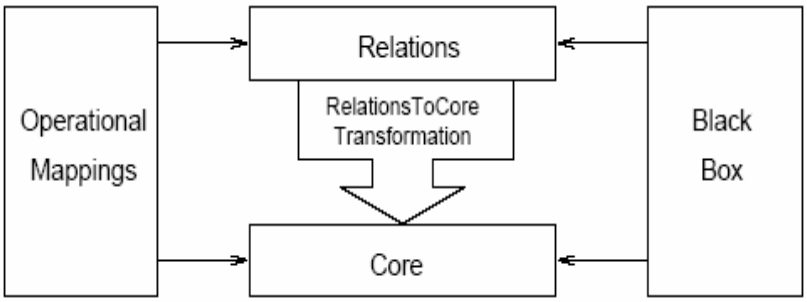

Figure 4. *Architecture du standard QVT (OMG-QVT 2.0, 2008)*

3.3. 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, le métamodèle UML est composé de deux paquetages : UML Superstructure (OMG-UML, 2008a) qui devient le standard en matière de modélisation orientée objet, et UML Infrastructure (OMG-UML, 2008b) qui décrit le noyau d'UML commun au MOF.

UML 2.0 Infrastructure (OMG-UML, 2005b) a été conçu de façon modulaire pour pouvoir être réutilisé par MOF 2.0 (OMG-MOF 2.0, 2006) et UML 2.0 Superstructure (OMG-UML, 2005a). 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 2.0 Superstructure.

UML 2.0 Infrastructure est constitué de plusieurs 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.) qui sont utilisés lors de la définition de la syntaxe abstraite des métamodèles, et enfin *Core ::Profiles*, permettant l'élaboration de profils UML.

3.4. 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 5 ci-dessous).

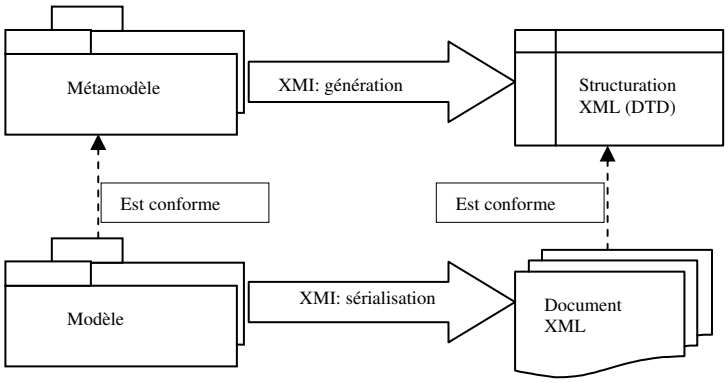

Figure 5. *XMI et la structuration de balises XML (BLANC, 2005)*

3.5. CWM

CWM (*Common Warehouse Metamodel*) (OMG-CMW, 2003) est le standard de l'OMG pour les techniques liées aux entrepôts de données. Il couvre le cycle de vie complet de modélisation, de construction et de gestion des entrepôts de données. CWM définit un métamodèle qui représente les métadonnées aussi bien au niveau métier qu'au niveau technique. CWM définit actuellement les métamodèles des principaux types d'entrepôts de données (Relationnel, Objet, XML,....) et propose des règles de transformation entre ceux-ci. Les métamodèles de données permettent de modéliser des ressources comme les bases de données relationnelles et les bases de données orientées objets.

3.6. Diagram Interchange

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 5 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 de toutes les 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.

4. L'Ingénierie dirigée par les modèles

L'Ingénierie Dirigée par les Modèles (IDM) est une discipline qui a pour vocation l'automatisation et la sûreté du développement des systèmes logiciels complexes — notamment les systèmes embarqués — en fournissant des outils et des langages permettant la transformation de modèles d'un niveau d'abstraction à un autre ou d'un espace technologique à un autre.

Dans cette section, nous présentons la problématique et les objectifs de l'IDM, la notion de transformation de modèles avec les outils et langages dédiés, et enfin les outils de méta-modélisation permettant de concevoir des DSMLs (Domain Specific Modeling Languages).

4.1. Problématique et objectifs

L'IDM a pour principal objectif de relever un certain nombre de défis du génie logiciel (qualité, productivité, 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 pour l'étude des différentes problématiques d'un système permettant à différents acteurs de s'intéresser aux différents aspects du système (sécurité, performance, consommation...). Avec cette approche, l'IDM est en passe de lever le verrou consistant à ne pouvoir tester un système que tardivement dans le cycle de vie.

Une des raisons majeures de l'apparition des architectures dirigées par les modèles repose sur la volonté de décrire précisément les besoins des clients par des CIM (Computational Independent Models) et le savoir-faire ou la connaissance métier d'une organisation dans des modèles abstraits indépendants des plates-formes (PIM - Platform Independent Models). Ayant isolé le savoir-faire métier dans des PIM, on a besoin soit de transformer ces modèles en d'autres PIM (besoin d'interopérabilité), soit de produire ou de créer des modèles PSM (Platform Specific Models) ciblant une plate-forme d'exécution spécifique (pour améliorer la portabilité et augmenter la productivité).

4.2. Transformation de modèles

La définition la plus générale et qui fait l'unanimité au sein de la communauté IDM (Bézivin, 2004) consiste à dire qu'*une transformation de modèles est la génération d'un ou de plusieurs modèles cibles à partir d'un ou de plusieurs modèles sources*. Dans l'approche par modélisation, cette 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 entre les métamodèles source et cible qui décrivent la structure des modèles cible et source. 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. 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 toutes les instances du 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. La figure 6 illustre ces deux étapes d'une transformation de modèles

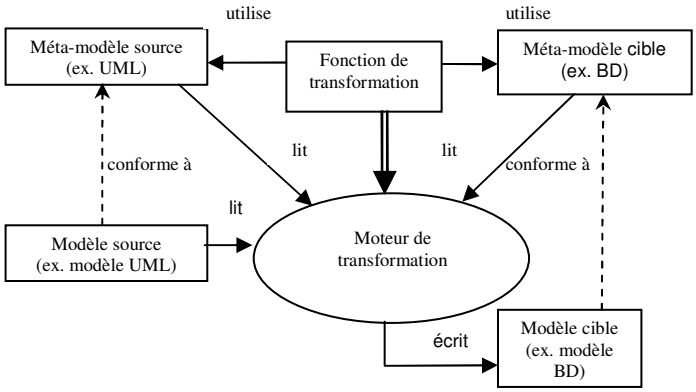

Figure 6. *Schéma de base d'une transformation de modèles (TOPCASED-WP5, 2008)*

4.2.1. Principales approches de transformation de modèles

Dans (Blanc, 2005) trois approches de transformations de modèles sont retenues: 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 informatique 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 des modèles cibles souhaités. Ces canevas sont des *modèles cibles paramétrés* ou des *modèles template*. 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 à travers son langage de transformation de modèles ATL que nous verrons dans la section 4.2.8.2.

4.2.2. Types de transformation

Une transformation de modèles met en correspondance des éléments des modèles cibles et sources. 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 situation est la transformation d'une classe UML munie de ses opérations et de ses attributs en une classe homonyme en Java ;

– 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 d'une telle transformation 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.

4.2.3. Axes de transformation

Selon la classification de (TOPCASED_WP5, 2008), la transformation de modèles peut se faire selon trois axes de transformation :

– 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 ou Model Refactoring) ;

– l'axe *métamodèle* permet de caractériser l'importance des métamodèles mis en jeu dans une transformation. Par analogie avec la 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. Exemple de transformation utilisant l'axe métamodéle: UML to SysML (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 transmis à la transformation (par exemple, le modèle source). Les informations transmises à la transformation sont appelées paramètres de la transformation. Nous parlerons de transformation *automatique* 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 qu'à l'exécution de la transformation.

4.2.4. Taxonomie des transformations

Partant de la nature des métamodèles source et cible, on distingue encore 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 issus du même métamodèle, et *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 des patrons sont des exemples de transformation endogène et horizontale ; tandis que la migration des 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-dessous résume les combinaisons possibles entre transformations de modèles.

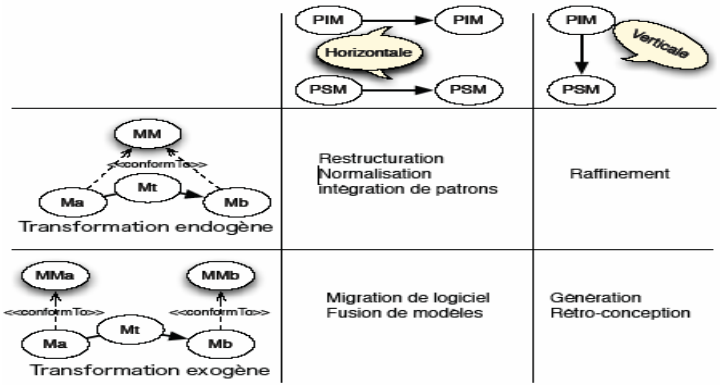

Figure 7. *Taxonomie des transformations de modèles (Combemale, 2007)*

4.2.5. Propriétés des transformations

Les principales propriétés qui caractérisent les transformations de modèles sont la réversibilité, la traçabilité, la réutilisabilité, l'ordonnancement et la modularité :

– r*éversibilité* : une transformation est dite réversible si elle peut se faire dans les deux sens. Exemple : Model to text et text to Model ;

– t*raçabilité* : 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 représentable par un modèle. Une instance de ce modèle est donc associée à chaque exécution d'une transformation tracée. La définition d'un

métamodèle de traces nous permet de structurer les traces qui seront générées par la plate-forme de traçabilité et ainsi de mieux les manipuler ;

– r*éutilisabilité* : la réutilisabilité permet de réutiliser des règles de transformation dans d'autres transformations de modèles. L'identification de patrons de transformation est un moyen pour mettre en œuvre cette réutilisabilité ;

– o*rdonnancement* : l'ordonnancement consiste à représenter les niveaux d'imbrication des règles de transformation. En effet, les règles de transformations peuvent déclencher d'autres règles ;

– m*odularité* : une transformation modulaire permet de 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.

4.2.6. 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 transformations de graphes 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. Comme souligné dans (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 de modèles complexes.

4.2.7. 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.

4.2.7.1. 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 ces modèles. 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 et de l'institutionnaliser 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 peut être ouvert depuis Eclipse ce qui permet de programmer en Java des transformations de modèles par le biais des classes d'implémentations associées à chaque élément d'UML. Cependant, 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 profil. MDA Modeler offre un support graphique de modélisation des profils, représentant les métaclasses référencées, ainsi que les stéréotypes utilisés et leurs propriétés et associations. La définition d'une transformation de modèles nécessite de référencer les métaclasses UML concernées par la transformation. Ces références contiendront le code de transformation qui sera en langage J s'il s'agit d'un composant MDA J ou en Java s'il s'agit d'un composant MDA Java. En résumé nous pouvons dire que MDA Modeler offre plusieurs mécanismes permettant de définir des opérations sur les modèles et ainsi de les rendre productifs. La figure 8 ci-dessous illustre les principes des outils MDA Modeler et UML Modeler.

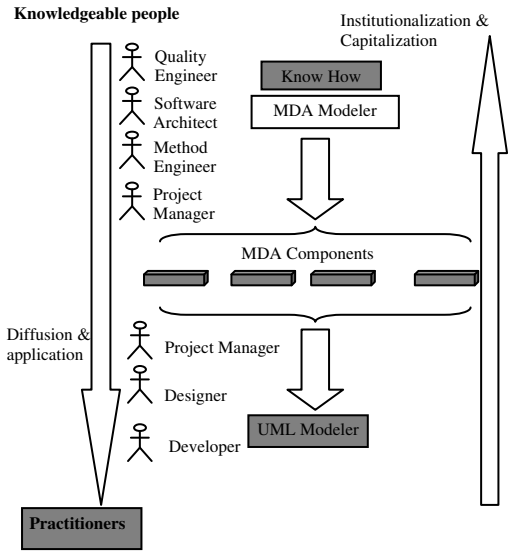

Figure 8. *Principes de MDA Modeler et UML Modeler (Softeam, 2008).*

4.2.7.2. 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 métamodéle. 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 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 la définition de la transformation par le biais des règles utilisant des classes d'implémentation, RSM pourra *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.

4.2.7.3. OptimalJ

Optimal J (Compuware-website, 2003) est le fruit d'un 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 transformations de modèles à modèles. L'outil propose l'ajout de patrons de transformations 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.

4.2.7.4. FUJABA

FUJABA (Burmester et al, 2004), 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 plusieurs activités de recherche notamment dans le domaine des applications distribuées, ses 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.

4.2.7.5. Synthèse

En résumé, nous pouvons dire que les outils intégrés aux AGL ont atteint un bon niveau de maturité (RSM et MDA Modeler) mais ils présentent certaines limitations notamment pour MDA Modeler qui ne peut faire que des transformations de modèles endogènes. RSM, même s'il permet de faire des transformations de modèles exogènes, est limité s'il s'agit de faire une transformation multiple de modèles (M vers N). 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.

4.2.8. 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 pour être plus ou moins intégrables dans les environnements de développement standard. Parmi ces outils nous pouvons citer Mia-Transformation de Mia-Software, et le plug-in ADT qui implémente le langage ATL 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.

4.2.8.1. Mia-Transformation

Mia-Transformation (MiaSoftware-website) est un outil qui exécute des transformations de modèles prenant en charge différents formats d'entrées et de sortie (XMI, API, etc.). Les transformations sont exprimées comme des règles d'inférence semi-déclaratives qui peuvent être enrichies en utilisant des scripts Java pour des services additionnels tels que la manipulation de chaînes.

4.2.8.2. ATL

ATL (ATLAS, 2005), acronyme de *ATLAS Transformation Language*, est un langage à vocation déclarative, mais en réalité hybride, qui 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.

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 de règles de transformation en ATL

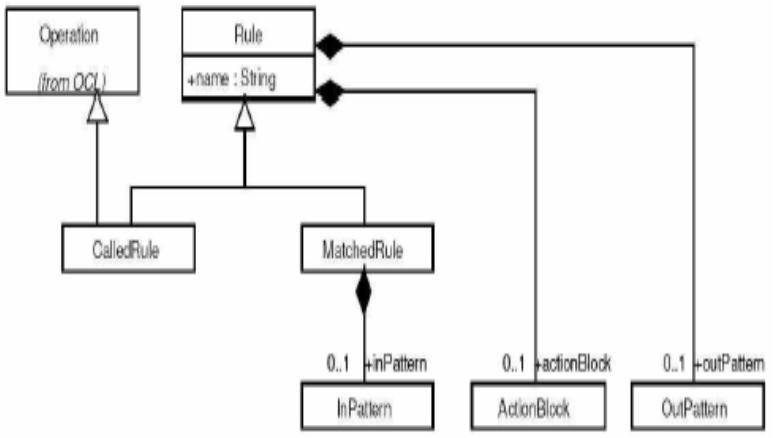

Figure 9. S*yntaxe abstraite d'une règle de transformation ATL (extrait du métamodèle ATL)*

ATL supporte deux modes d'exécution différents : le mode standard et le mode par raffinement. Dans le mode *standard*, les éléments sont crées 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.

4.3. 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 métaprogramme. Parmi ces outils, nous pouvons citer Kermeta (Triskell, 2005) de l'IRISA-INRIA Rennes, MetaEdit+ (Tolvanen et al, 2003) de MetaCase, XMF-Mosaic (Xactium-website) de la société Xactium, EMF/Ecore (Eclipse-EMF-website) de la fondation Eclipse, et l'outil TOPCASED (TOPCASED_WP5, 2008) (TOPCASED-website).

4.3.1. 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 metamodèlisation KERMETA (Triskell, 2005) (Jézéquel 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 2.0 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'action (Action Semantics, Xion).

Cette composition fait de KERMETA un véritable langage de métamodélisation exécutable. Comme UML 2.0 Infrastructure, KERMETA 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 des métamodèles de niveau M2.

4.3.2. MetaEdit +

MetaEdit+ (Tolvanen et al, 2003), (Muller, 2006) 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 transformations de modèles.

4.3.3. XMF-Mosaic

XMF-Mosaic (Xactium-website) est un outil destiné à développer des langages de modélisation et à déployer tous 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 domaines. 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. Ce langage peut être utilisé tout au long de la construction d'un projet, pour décrire les différents éléments développés comme pour les manipuler en mode console dans un programme. 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 dans le 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.

4.3.4. EMF/Ecore

EMF (Eclipse-EMF-website), qui signifie *Eclipse Modeling Framework*, est une plate-forme de modélisation et de génération de code qui facilite la construction d'outils et d'autres applications basées sur des modèles structurés. Il permet 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 pouvons citer :

– le métamodèle *Ecore* qui est un canevas de classes pour décrire les modèles *EMF* et manipuler les référentiels de modèles ;

– *EMF.Edit* qui 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.

4.3.5. TOPCASED

Le projet open source 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 métaprogrammation.

5. Conclusion

Nous avons présenté dans cet article un état de l'art sur le développement logiciel dirigé par les modèles. Comme nous l'avons dit dans l'introduction, nous avons plus particulièrement mis l'accent dans cette étude sur les concepts, langages et outils associés à la transformation de modèles – paradigme central de l'IDM.

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. Les transformations nécessitent des langages dédiés (ATL, QVT,…), des outils de métamodélisation flexibles (Kermeta, EMF/Ecore, TOPCASED,…) qui pourront assister les concepteurs de systèmes logiciels complexes.

Beaucoup de thèmes de recherche émergent au sein de la sphère IDM ; il devient donc essentiel de créer une synergie entre les différents groupes de recherche dans ce domaine. L'action transversale IDM (action-idm-website) a ainsi été lancée pour servir de cadre de réflexion et d'échange entre les communautés issues de domaines technologiques différents.

Travaux en cours et perspectives

Si des efforts importants ont été faits dans le domaine de l'IDM, cette discipline est néanmoins jeune et de nombreux axes de recherche restent à explorer. Nous sommes plus particulièrement impliqués dans les axes détaillés ci-dessous :

– *définition de DSL sous forme de profils UML* : pour supporter la modélisation par point de vue, nous avons défini un profil UML (Nassar, 2005) qui introduit la notion de classe multivue et propose une démarche de conception décentralisée passant par l'élaboration de modèles-vue. Nous travaillons actuellement sur la composition (fusion) de modèles-vues en considérant cette composition comme une

transformation exogène horizontale de UML vers VUML. Cette composition porte sur la fusion des diagrammes de classes (Anwar et al, 2007) et des machines multivue (Ober et al, 2008) ;

– *modélisation et mise en œuvre assistée du processus de développement IDM* : à ce jour les langages de description de processus – tels que SPEM (cf. travaux de (Bendraou, 2007) et (Combemale, 2008)) – ne prennent pas en compte spécifiquement les notions de transformation ; il est nécessaire de caractériser les transformations comme des activités particulières, dont les produits en entrée et en sortie doivent être des modèles conformes à un même métamodèle (transformation endogène) ou à plusieurs métamodèles (transformation exogène) ;

– *exécutabilité des modèles de processus* : l'intérêt de l'IDM est l'automatisation des transformations de modèles. Pour cela, il faut pouvoir rendre exécutables les modèles de processus qui représentent les transformations, et donc leur associer une sémantique opérationnelle. Les travaux décrits dans (Bendraou, 2007) et (Combemale, 2008) sont une première étape vers cet objectif ;

– *passage à l'échelle* : pour être vraiment utilisable dans l'industrie du logiciel, l'IDM doit pouvoir traiter des modèles de toute taille et de toute complexité, dans des environnements de développement par nature collaboratifs. Cette contrainte est loin d'être satisfaite à ce jour. C'est pour cette raison que des recherches s'engagent sur le passage à l'échelle de l'IDM, la notion de méga-modèles et de modèles infinis, la gestion collaborative de modèles, etc. Pour notre part, nous travaillons à la définition d'un métamodèle supportant la –description et la mise en œuvre de procédés IDM collaboratifs ;

– *traçabilité des processus de développement IDM* : si la traçabilité du processus de développement fait l'objet depuis longtemps de recherches au niveau des exigences, il est intéressant d'appliquer ces approches à la traçabilité des transformations, pour conserver notamment des traces d'exécution à des fins de documentation, ou pour rejouer des transformations Nous exploitons ces traces d'exécution pour rejouer une transformation afin de re-générer un modèle cible suite à l'évolution de propriétés associées à un modèle source (Amar et al, 2008) ;

– *réutilisabilité* : pour mettre en œuvre la réutilisation de transformations — considérées comme des modèles de procédés — l'un des moyens est de définir des patrons de procédé et d'être capable d'appliquer automatiquement ces patrons pour définir ou modifier une transformation. Nous avons élaboré un métamodèle supportant la représentation de patrons de procédés (Tran et al, 2007a) et travaillons actuellement à l'élaboration d'un environnement support ;

– *reverse-engineering, refactoring (re-usinage) sur les modèles* : l'une des caractéristiques importantes des transformations est leur potentielle réversibilité. Combinée à l'utilisation de patrons, cette réversibilité permet de faire du refactoring sur les modèles (Tran et al. 2007b), qu'ils soient des modèles de produit ou de procédé. Nous travaillons actuellement à la définition d'une base de patrons de procédés pour automatiser la mise en œuvre — sous forme de transformations — de ces opérateurs de refactoring.

6. Bibliographie

Action IDM, http://www.actionidm.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. *Revue RSTI série L'OBJET,* numéro spécial Ingénierie Dirigées par les Modèles. Vol 13, n°4/2007.

Amar B., Leblanc H., Dhaussy P., Coulette B. Modèle de traçabilité pour la mise en œuvre d'une technique de validation formelle. *Revue Génie Logiciel*, numéro spécial Journées Neptune, *NEPTUNE' 2008*, Paris 8-9 Avril 2008.

ATLAS, KM3: Kernel MetaMetaModel, Technical Report, LINA&INRIA, NANTES, aug. 2005.

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/, page 7-11 Juillet 2002,

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

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

Bézivin J. Sur les principes de base de l'ingénierie des modèles. *RTSI-L'Objet*, 10(4) : Page 145-157, 2004.

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

Breton E. Contribution à la représentation de processus par des techniques de métamodèlisation, Thèse de doctorat, Université de Nantes, Juin 2002.

Burmester S., Giese H., Niere J., Tichy M., Wadsack J., Wagner R., Wendehals L. and 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 (3), page 203-218, 2004.

Combemale B, Cregut X, Rougemaille S., Migeon F., Pantel M., Maurel C. Expériences pour décrire la sémantique en ingénierie des modèles. *IDM'06, Lille – Secondes Journées sur l'Ingénierie Dirigée par les Modèles*, Lille, 07 Juin 2006, Paris, Hermès, p. 18-19.

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

Combemale B. Approche de métamodèlisation 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/index.html?keywords=emf; http://www.eclipse.org/modeling/emf/

GME-website, GME: http://www.isis.vanderbilt.edu/projects/gme/

Greenfield J., Short K. Assembling Applications with Patterns, Models, Frameworks and Tools. *In OOPSLA'03,* Anaheim, California, USA, October 26-30 2003

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),* pp. 82-83, Anaheim, CA, USA, October 2003.

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. In *IDM'05, Paris Premières Journées sur l'Ingénierie Dirigée par les Modèles*, Posters & Démos, Paris, 30 Juin- 01 Juillet, 2005.

Jouault J., Kurtev I. On the Architectural Alignment of ATL and QVT, Rapport de recherche préliminaire, ATLAS Group, INRIA&LINA Université de Nantes, Novembre 2005

Meylan S, Tatibouet B., Etude et comparaison d'outils de transformation de modèles. Rapport de recherche, Janvier 2006, LIFC –Université de FRANCHE-COMTE, p.14-19

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, 2005

Mottu J.M., BAUDRY B., LE TRAON Y. Test de Transformation de Modèles : Expression d'Oracles. *IDM '08 : 4èmes Journées sur l'Ingénierie Dirigée par les Modèles*, 5-6 juin, Mulhouse, France, 2008

Muller Pierre Alain, De la modélisation objet des logiciels à la métamodèlisation 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.

Nassar M. Analyse/conception par objets et points de vue : le profil VUML. Thèse INPT, Toulouse, 28 septembre 2005.

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

Ober I., Coulette B., Lakhrissi Y. Behavioral modelling and composition of object slices using event observation. *In Proceedings of ACM/IEEE international conference MODELS'08,* Toulouse, October 1-3, 2008.

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

OMG, DI 2.0, http://www.omg.org/cgi-bin/doc?ptc/2003-09-01.pdf, 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, QVT 2.0 RFP, www.omg.org/docs/ad/02-04-10.pdf, Avril 2002

OMG, QVT Final Adopted Spec. , www.omg.org/docs/ptc/05-11-01.pdf, November 2005

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

OMG, UML 2.0 Superstructure, http://www.omg.org/cgi-bin/doc?formal/05-07-04.pdf, 2005a

OMG, UML 2.0 Infrastructure, http://www.omg.org/cgi-bin/doc?formal/05-07-05.pdf, 2005b

OMG, UML 2.2 Superstructure, http://www.omg.org/cgi-bin/doc?ptc/08-05-05, 2008a

OMG, UML 2.2 Infrastructure, http://www.omg.org/cgi-bin/doc?ptc/08-05-04, 2008b

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

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. *In Congrès INFORSID'07* (Perros-Guirec, France, May 2007). Pages 382-397. ,2007

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

Softeam, Garantir en permanence la cohérence Modèle code ; MDE vs RTE, White Paper 2000.

Sottet J., Calvary G., Favre J-M. Ingénierie de l'interaction homme-machine dirigée par les modèles. *IDM'05, Paris Premières Journées sur l'Ingénierie Dirigée par les Modèles*, Paris, 30 Juin- 01 Juillet, 2005.

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., et 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.

Tolvanen J.-P. , Rossi M. MetaEdit+: defining and using domain-specific modeling languages and code generators. *In Proceedings of the OOPSLA '03: 18th annual ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 92-93, Anaheim, California, October 26-30 2003.

TOPCASED-WP5, staff, Guide méthodologique pour les transformations de modèles. Rapport de recherche n° 8, Novembre 2008, IRIT/MACAO

TOPCASED-WP5, staff, 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 SEKE 2007*, July 9-11, Boston, 2007.

Tran H.N., Coulette B., Dong Thi B. T., “Modelling Process Patterns and their Application.” *ICSEA* 2007, Cap Estérel France, August 25-31, 2007. IEEE Computer Society Press 2007.

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

Vachet C., Laurillau Y., Carron B. Situation de mobilité et approche dirigée par les modèles. *UbiMob’06*, Paris, France, Septembre 2006

Vanwormhoudt G. Vérification de modèles avec EMF et OCL, Rapport de recherche, Ecole d’Ingénieurs TELECOM, Lille 1.

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

ANNEXE POUR LE SERVICE FABRICATION

A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. **ARTICLE POUR LA REVUE:**
   TSI — L'ingénierie dirigée par les modèles

2. **AUTEURS :**
   Samba Diaw — Rédouane Lbath — Bernard Coulette

3. **TITRE DE L'ARTICLE :**
   Etat de l'art sur le développement logiciel dirigé par les modèles

4. **TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :**
   Développement logiciel orienté modèles

5. **DATE DE CETTE VERSION :**
   12 janvier 2009

6. **COORDONNÉES DES AUTEURS :**

   – adresse postale :
   Laboratoire IRIT
   Université Toulouse 2- Le Mirail
   5, allées Antonio Machado 31058 Toulouse Cédex 9
   – téléphone : 05 61 50 38 96
   – télécopie : 05 61 50 41 73
   – E-mail: *{diaw, lbath, coulette}@univ-tlse2.fr*

7. **LOGICIEL UTILISE POUR LA PRÉPARATION DE CET ARTICLE :**
   Microsoft Office 2002 SP3

8. **FORMULAIRE DE COPYRIGHT A RETOURNER SIGNÉ PAR LES AUTEURS À TÉLÉCHARGER SUR :**
   http://www.revuesonline.com

LAVOISIER
SERVICE EDITORIAL – HERMES
14 rue de Provign
94236 Cachan cedex

Sylvie Viriot
Responsable Revues
Tel : 01-47-40-67-67
e-mail : revues@lavoisier.fr