

**CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS
CENTRE D'ENSEIGNEMENT DE LYON**



**Examen probatoire du diplôme d'ingénieur C.N.A.M.
en INFORMATIQUE**
option ingénierie et intégration informatique : système de conduite

présenté par

Sylvain ANDRE

MDA (Model Driven Architecture) principes et états de l'art.

Soutenu le 05 novembre 2004

JURY

M. Alain CABANES
M. Bertrand DAVID
M. Claude GENIER
M. Jacques KOULOUMDJIAN

Remerciements

Je tiens à remercier toutes celles et tous ceux qui ont contribué à ce mémoire en m'accordant un peu de leur temps.

J'ai apprécié leur aide et leurs remarques qui m'ont été précieuses.

Merci à Xavier Blanc (LIP6), Bruno Traverson (EDF), Gregory de Fombelle (Thales), Philippe Desfray (Softeam), Xavier Bonifay (IFCS), Nicolas Farcet (Thales), Madame Perrot (Softeam), Bertrand David (Ecole Centrale de Lyon) et Claude Lacroix (CNAM Lyon).

TABLE DES MATIÈRES

<u>INTRODUCTION</u>	1
<u>1. VERS UNE COMPLEXITÉ GRANDISSANTE</u>	2
<u>2. PRINCIPES DU MDA</u>	2
<u>2.1 Architecture du MDA</u>	3
<u>2.2 Les différents modèles du MDA</u>	4
<u>2.2.1 LE CIM (COMPUTATION INDEPENDENT MODEL)</u>	4
<u>2.2.2 LE PIM (PLATFORM INDEPENDENT MODEL)</u>	4
<u>2.2.3 LE PSM (PLATFORM SPECIFIQUE MODEL)</u>	4
<u>2.2.4 LE PDM (PLATEFORM DESCRIPTION MODEL)</u>	4
<u>2.3 La transformation des modèles du MDA</u>	5
<u>2.3.1 DE PIM VERS PIM</u>	5
<u>2.3.2 DE PIM VERS PSM</u>	5
<u>2.3.3 DE PSM VERS PSM</u>	6
<u>2.3.4 DE PSM VERS PIM</u>	6
<u>2.4 Les standards de l'OMG</u>	6
<u>2.4.1 L'ARCHITECTURE À QUATRE NIVEAUX</u>	7
<u>2.4.2 LE MOF</u>	7
<u>2.4.3 L'UML ET L'OCL</u>	8
<u>2.4.4 LES PROFILS UML</u>	9
<u>2.4.5 XMI 2.0 (XML METADATA INTERCHANGE)</u>	9
<u>2.4.6 HUTN (HUMAN-USABLE TEXTUAL NOTATION)</u>	10
<u>2.4.7 CWM (COMMON WAREHOUSE METAMODEL)</u>	10
<u>2.4.8 PATRON DE CONCEPTION (DESIGN PATTERN)</u>	10
<u>3. ETAT DE L'ART</u>	10
<u>3.1 Retour sur les standards</u>	10
<u>3.1.1 UML 2.0</u>	10
<u>3.1.2 MOF 2.0</u>	11
<u>3.2 L'utilisation des standards pour la transformation</u>	12
<u>3.2.1 LA TRANSFORMATION PAR ANNOTATION OU MARQUAGE</u>	12
<u>3.2.2 LA TRANSFORMATION PAR MÉTA-MODÈLE</u>	12
<u>3.2.3 L'APPROCHE EN DOUBLE Y</u>	13
<u>3.3 Les projets de recherche sur le MDA</u>	14
<u>3.3.1 LE PROJET ACCORD</u>	15
<u>3.3.2 LE PROJET MODATHÈQUE</u>	15
<u>3.3.3 MODFACT</u>	15
<u>3.3.4 LE PROJET MODELWARE</u>	16
<u>3.4 Bilan</u>	16
<u>3.4.1 LES AVANTAGES DU MDA</u>	16
<u>3.4.2 LES INCONVÉNIENTS DU MDA</u>	17
<u>3.4.3 RETOUR SUR EXPÉRIENCE</u>	17
<u>CONCLUSION</u>	18
<u>BIBLIOGRAPHIE</u>	19
<u>GLOSSAIRE</u>	24

Introduction

L'OMG* (Object Management Group) a lancé un gigantesque chantier dans le monde du génie logiciel. L'OMG propose de revoir notre façon de penser une application. Cette révolution s'inscrit dans la durée et s'appuie sur des standards éprouvés, regroupés au sein du MDA* (Model Driven Architecture). Cette démarche apporte un changement important dans la conception des applications. Elle introduit une séparation nette dans la logique métier de l'entreprise et la logique d'implémentation.

Cette révolution est d'autant plus importante qu'elle change non seulement notre façon de voir mais aussi notre façon de faire. En effet, elle propose de mettre à disposition des développeurs des outils d'automatisation de génération de code à partir de la logique métier. Le MDA est en cours de développement et n'en est qu'à ses débuts. La phase d'utilisation à grande échelle est prévue dans les dix à quinze prochaines années.

Le MDA est actuellement en phase de conception et de nombreuses voix sont explorées. Certaines commencent à être bien connues tandis que d'autres sortent tout juste de l'ombre. Nous verrons d'abord les motivations qui ont poussé l'OMG à proposer la démarche MDA. Puis, nous aborderons les principes du MDA en présentant les modèles et les standards, piliers de cette démarche. Enfin, nous étudierons le rôle des standards dans la transformation des modèles, nous détaillerons les projets de recherche qui visent à l'obtention, à la concrétisation et à la divulgation de nouveaux outils. Et pour finir, nous ferons le point sur les différentes utilisations du MDA aux travers d'exemples vécus.

1. Vers une complexité grandissante

Pour rester compétitives, les sociétés de l'industrie de développement du logiciel doivent réduire leurs coûts de développement ainsi que leurs coûts de maintenance. Tâches compliquées, car les systèmes* deviennent de plus en plus complexes. Par exemple, chez Thales, la taille des systèmes est multipliée par 5 ou 10 tous les 5 ans. Quels que soient les systèmes développés, la taille du code varie de 100 000 à 1 000 000 de lignes. Il devient alors difficile de maintenir correctement ces logiciels à moindre coût et ceci dans des délais raisonnables. D'autant plus que la complexité augmente avec la taille du code, engendrant des bogues et des retards dans la livraison finale. Il n'est pas rare alors que « 23% des développements applicatifs soient des échecs!» que « 84% des projets informatiques critiques n'atteignent pas leurs objectifs!» (The Standish Group International Inc.) et que « le coût moyen d'une mise à jour technologique soit de 18 % du coût original du projet ! » (CIO Magazine) [CON04].

Dans ce contexte, les chefs d'entreprise hésitent à faire des mises à jour sans la garantie d'un retour sur investissement. S'ils franchissent le pas, ils sont contraints alors de financer à nouveau au prix fort l'évolution de leurs logiciels fraîchement acquis et développés à grands frais. Cette évolution consiste à « redéveloppe » et porter ce logiciel vers une nouvelle plate-forme*, alors que globalement la logique métier de l'entreprise n'a pas changé. En effet, au cours de ces dernières années, du fait de l'évolution incessante et inévitable, les technologies se sont succédées. Par exemple, CORBA* (Common Object Request Broker Architecture) qui devait permettre une communication entre applications différentes ne s'est pas imposé en tant que standard du Middleware*. Cela s'explique par sa complexité et l'apparition d'architectures* concurrentes plus simples comme EJB* (Entreprise Java Bean) et .Net. Ainsi, les architectures se suivent et ne se ressemblent pas.

Pour permettre l'interopérabilité entre les différents systèmes, réduire les coûts de développement et augmenter l'évolutivité, l'OMG* (Object Management Group) propose la démarche MDA* (Model Driven Architecture), où la notion de modèle devient essentielle et centrale.

2. Principes du MDA

En novembre 2000, l'OMG, consortium de plus 1000 entreprises, initie la démarche MDA. En fait, au sens strict du mot, c'est un « cadre standard qui regroupe des standards » selon la définition de Philippe Desfray. Mais dans la littérature le terme « démarche », « norme » ou « approche » qualifie le MDA. Il est intéressant de noter que MDA n'a pas encore de genre. Certains disent le MDA, d'autre la MDA, enfin certains ne mentionnent pas l'article et disent MDA. Dans la suite du rapport, nous emploierons le masculin pour le désigner et nous le qualifierons de démarche ou d'approche.

Ce standard a pour but d'apporter une nouvelle façon de concevoir des applications en séparant la logique métier de l'entreprise, de toute plate-forme technique. En effet, la logique métier est stable et subit peu de modifications au cours du temps, contrairement à l'architecture technique. Il est donc évident de séparer les deux pour faire face à la complexification des systèmes d'information et des forts coûts de migration technologique. Cette séparation autorise alors la capitalisation du savoir logiciel et du savoir-faire de l'entreprise. A ce niveau, l'approche objet et l'approche composant n'ont pas su tenir leurs promesses. Il devenait de plus en plus difficile de développer des logiciels basés sur ces technologies. Le recours à des procédés supplémentaires, comme le patron de conception ou la programmation orientée aspect, était alors nécessaire. Le standard MDA doit aussi offrir la possibilité de stopper l'empilement des technologies qui nécessite de conserver des compétences particulières pour faire cohabiter des systèmes divers et variés. Ceci est permis grâce au passage d'une approche

interprétative à une approche transformationnelle. Dans l'approche interprétative, l'individu a un rôle actif dans la construction des systèmes informatiques alors que dans l'approche transformationnelle, il a un rôle simplifié et amoindri grâce à la construction automatisée.

La démarche MDA propose à terme de définir un modèle métier indépendant de toute plate-forme technique et de générer automatiquement du code vers la plate-forme choisie. Pour cela, l'accent est mis non plus sur les approches objet mais sur les approches modèle. Le but est de favoriser l'élaboration de modèles de plus haut niveau.

2.1 Architecture du MDA

Sur la figure 1 ci-dessous, l'architecture du MDA se découpe en quatre couches. L'OMG s'est basé sur plusieurs standards. Au centre, se trouve le standard UML* (Unified Modeling Language), MOF* (Meta-Object Facility) et CWM* (Common Warehouse Metamodel). Ils seront développés au chapitre 2.4 à la page 6. Dans la couche suivante, se trouve aussi un standard XMI* (XML* Metadata Interchange) qui permet le dialogue entre les middlewares (Java, CORBA, .NET et web services). La troisième couche contient les services qui permettent de gérer les événements, la sécurité, les répertoires et les transactions. Enfin, la dernière couche propose des frameworks spécifiques au domaine d'application (Finance, Télécommunication, Transport, Espace, médecine, commerce électronique, manufacture,...) [CAR03, JUL03].

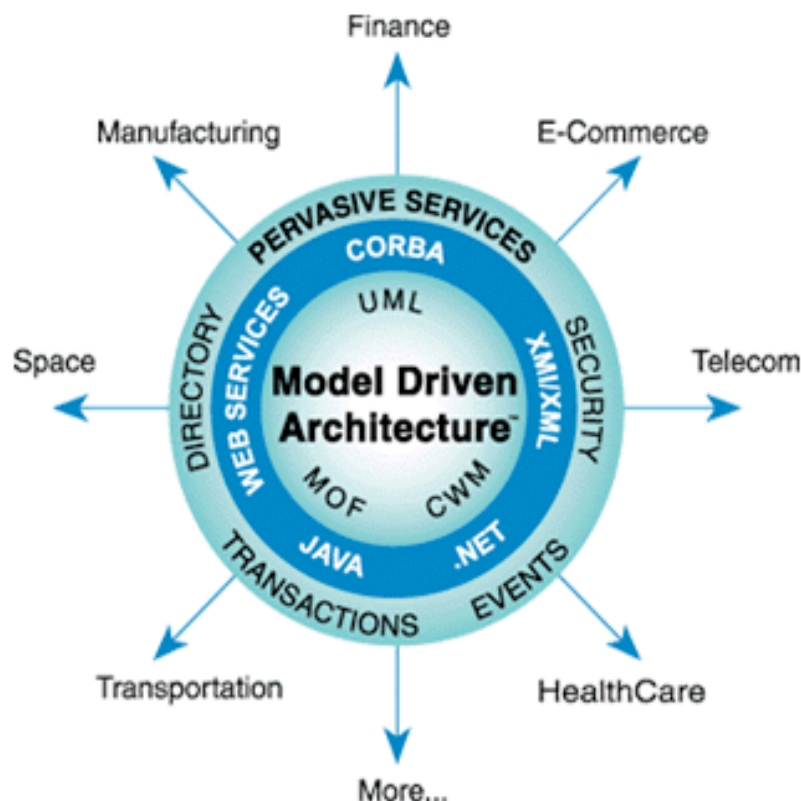


Figure 1 : architecture du MDA.

Pour créer une application, un architecte fait un schéma en UML par exemple, mais d'autres langages peuvent aussi être utilisés. En partant du centre de la figure 1 ci-dessus, l'architecte dirigera son application en évoluant de couche en couche pour aller vers le domaine d'application qui l'intéresse.

2.2 Les différents modèles du MDA

Le MDA est composé de plusieurs modèles, « descriptions abstraites d'une entité du monde réel utilisant un formalisme donné » qui vont servir dans un premier temps à modéliser l'application, puis par transformations successives à générer du code. Aujourd'hui, la frontière entre les différents modèles n'est pas encore bien explicitée, ni formalisée. Néanmoins, il est possible de donner une description de chacun d'eux.

2.2.1 Le CIM (Computation Independent Model)

Il est indépendant de tout système informatique. C'est le modèle métier ou le modèle du domaine d'application. Le CIM permet la vision du système dans l'environnement où il opérera, mais sans rentrer dans le détail de la structure du système, ni de son implémentation. Il aide à représenter ce que le système devra exactement faire. Il est utile, non seulement comme aide pour comprendre un problème, mais également comme source de vocabulaire partagé avec d'autres modèles. L'indépendance technique de ce modèle lui permet de garder tout son intérêt au cours du temps et il est modifié uniquement si les connaissances ou les besoins métier changent. Le savoir faire est recentré sur la spécification CIM au lieu de la technologie d'implémentation. Dans les constructions des PIM* (Platform Independent Model) et des PSM* (Platform Specific Model), il est possible de suivre les exigences modélisées du CIM qui décrivent la situation dans lequel le système est utilisé, et réciproquement [OMG03].

2.2.2 Le PIM (Platform Independent Model)

Il est indépendant de toute plate-forme technique (EJB, CORBA, .NET,...) et ne contient pas d'informations sur les technologies qui seront utilisées pour déployer l'application. C'est un modèle informatique qui représente une vue partielle d'un CIM. Le PIM représente la logique métier spécifique au système ou le modèle de conception. Il représente le fonctionnement des entités et des services. Il doit être pérenne et durer au cours du temps. Il décrit le système, mais ne montre pas les détails de son utilisation sur la plate-forme. A ce niveau, le formalisme utilisé pour exprimer un PIM est un diagramme de classes en UML qui peut-être couplé avec un langage de contrainte comme OCL* (Object Constraint Language) (voir le chapitre 2.4.3 à la page 8). Il existe plusieurs niveaux de PIM. Le PIM peut contenir des informations sur la persistance, les transactions, la sécurité,... Ces concepts permettent de transformer plus précisément le modèle PIM vers le modèle PSM [BEZ02a, JUL03, OMG03].

2.2.3 Le PSM (Platform Specific Model)

Il est dépendant de la plate-forme technique spécifiée par l'architecte. Le PSM sert essentiellement de base à la génération de code exécutable vers la ou les plates-formes techniques. Le PSM décrit comment le système utilisera cette ou ces plates-formes. Il existe plusieurs niveaux de PSM. Le premier, issu de la transformation d'un PIM, se représente par un schéma UML spécifique à une plate-forme. Les autres PSM sont obtenus par transformations successives jusqu'à l'obtention du code dans un langage spécifique (Java, C++, C#, etc.) Un PSM d'implémentation contiendra par exemple des informations comme le code du programme, les types pour l'implémentation, les programmes liés, les descripteurs de déploiement [BEZ02a, JUL03, OMG03].

2.2.4 Le PDM (Platform Description Model)

Cette notion n'est pas encore bien définie par l'OMG, pour l'instant il s'agit plus d'une piste de recherche. Un PDM contient des informations pour la transformation de modèles vers une plate-forme en particulier et il est spécifique de celle-ci. C'est un modèle de transformation qui va permettre le passage du PIM vers le PSM. Normalement, chaque fournisseur de plate-forme devrait le proposer [BEZ03a, MOG03].

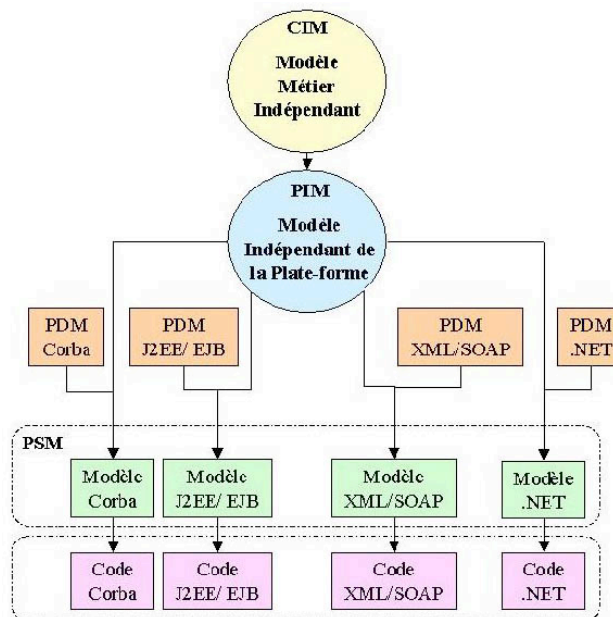


Figure 2 : exemple d'utilisation des modèles pour réaliser une application.

Le passage entre ces différents modèles va se faire par une suite de transformations (voir figure 2). L'OMG veut proposer à terme l'automatisation de ces transformations par des outils logiciels. Il sera alors possible de passer du modèle d'analyse au déploiement de la solution en générant automatiquement du code. Mais dans un premier temps, ce passage de modèle à modèle va s'effectuer, de façon semi-automatique ou assisté, puis progressivement de manière partielle pour arriver enfin à une génération totale du code. Ces transformations seront de plus en plus automatisées au fur et à mesure de l'évolution des outils.

2.3 La transformation des modèles du MDA

Le MDA identifie plusieurs transformations pendant le cycle de développement. Il est possible de faire quatre types de transformations différentes.

2.3.1 De PIM vers PIM

Ces transformations sont utilisées pour enrichir, filtrer ou spécialiser les informations des modèles sans rajouter aucune information liée à la plate-forme. Un exemple de transformation PIM vers PIM est de masquer des éléments afin de s'abstraire des détails fonctionnels. Un autre exemple est le passage du modèle d'analyse à celui de conception. Cependant, ces transformations ne sont pas toujours automatisables. Le fait de passer d'un PIM à un autre PIM est appelé raffinement. Ce processus consiste à introduire des détails supplémentaires dans le modèle. Il est aussi utilisé pour le passage de PSM à PSM [BEZ02a].

2.3.2 De PIM vers PSM

Un fois le PIM suffisamment raffiné pour pouvoir être spécialisé vers une plate-forme donnée, il peut alors être transformé en PSM. Cette opération consiste à ajouter au PIM des informations propres à une plate-forme technique. Les principales plates-formes visées sont J2EE, .NET ou CORBA, ...C'est le PDM, un profil (les profils sont abordés à la page 9 au chapitre 2.4.4) ou le MOF (Meta-Object Facility) (voir page 7 au chapitre 2.4.2) qui contient les caractéristiques de transformation. Il est alors possible de passer d'un modèle indépendant à un modèle dépendant. Les règles de transformation devront être généralisées et capitalisées pour obtenir dans le futur une automatisation importante [BEZ02a].

2.3.3 De PSM vers PSM

Une transformation PIM vers PSM n'est pas toujours suffisante pour permettre la génération de code d'où la nécessité de passer de PSM à PSM en utilisant des formalismes intermédiaires. Par exemple, pour générer un code C++, à partir d'un formalisme en UML, un passage d'UML vers SDL* (Specification and Description Language) puis de SDL vers C++ pourrait être utilisé. La transformation PSM à PSM (raffinement) s'effectue lors de phases de déploiement, d'optimisation ou de reconfiguration [BEZ02a].

2.3.4 De PSM vers PIM

Cette transformation est utilisée pour revenir à un modèle indépendant de plate-forme (PIM) à partir d'un modèle spécifique de plate-forme (PSM) ou éventuellement du code. C'est une opération de rétro-ingénierie (reverse engineering) qui est assez complexe à réaliser et difficilement automatisable. Ces transformations sont néanmoins nécessaires pour permettre l'intégration d'applications existantes dans le processus MDA [BEZ02a].

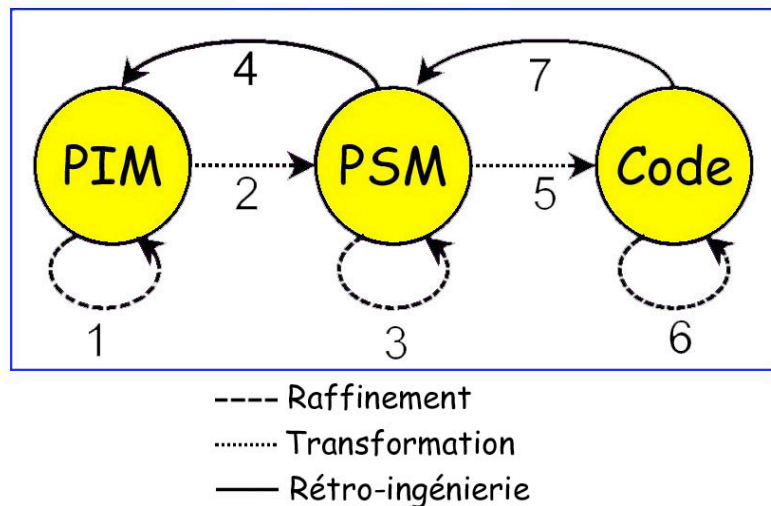


Figure 3 : transformations des modèles.

La figure 3 ci-dessus synthétise les différentes transformations possibles des modèles. Le PIM représente le niveau de modèle le plus abstrait. Par passages successifs, les modèles deviennent de plus en plus concrets jusqu'à l'obtention du code. Le code peut-être assimilé par certains à un PSM exécutable, de même que la génération de code n'est pas toujours considérée comme une transformation de modèle. La figure montre aussi qu'il est possible à partir du code de recréer un PSM puis un PIM. Toutefois, le passage du code au PSM ne peut s'effectuer que si la démarche MDA a été respectée, sinon les techniques traditionnelles de rétro-ingénierie doivent être appliquées [BEZ02d].

2.4 Les standards de l'OMG

Nous venons de détailler les différents modèles du MDA ainsi que le sens des transformations, mais nous n'avons pas décrit comment passer d'un modèle à un autre. Nous expliquerons ce point au chapitre 3.2 à la page 12 après avoir examiné l'architecture en couche de modèle puis les différents standards qui permettent ces transformations.

2.4.1 L'architecture à quatre niveaux

Cette architecture est hiérarchisée en quatre niveaux. En partant du bas :

- Le niveau M0 (ou instance) correspond au monde réel. Ce sont les informations réelles de l'utilisateur, instance du modèle de M1.
- Le niveau M1 (ou modèle) est composé de modèles d'information. Il décrit les informations de M0. Les modèles UML, les PIM et les PSM appartiennent à ce niveau. Les modèles M1 sont des instances de méta-modèle de M2.
- Le niveau M2 (ou méta-modèle), il définit le langage de modélisation et la grammaire de représentation des modèles M1. Le méta-modèle UML qui est décrit dans le standard UML, et qui définit la structure interne des modèles UML, fait partie de ce niveau. Les profils UML, qui étendent le méta-modèle UML, appartiennent aussi à ce niveau. Les méta-modèles sont des instances du MOF.
- Le niveau M3 (ou méta-méta-modèle) est composé d'une unique entité qui s'appelle le MOF. Le MOF permet de décrire la structure des méta-modèles, d'étendre ou de modifier les méta-modèles existants. Le MOF est réflexif, il se décrit lui-même.

La figure 4 ci-dessous représente les quatre niveaux. Les niveaux M1, M2 et M3 appartiennent au monde de la modélisation [BEZ02a, BEZ02d].

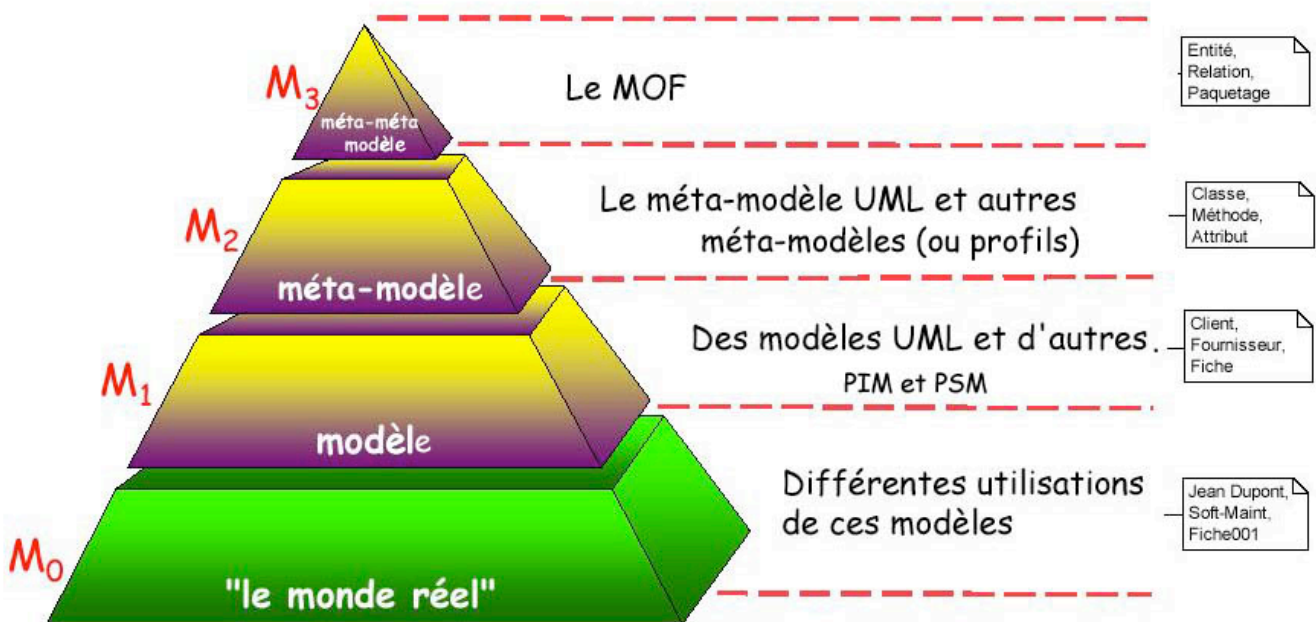


Figure 4 : architecture à quatre niveaux.

2.4.2 Le MOF

Il fait partie des standards définis par l'OMG et il peut être vu comme un sous-ensemble d'UML. Le MOF et l'UML constituent le cœur de la technologie MDA car ces deux standards permettent de créer des modèles technologiquement neutres. Le MOF spécifie la structure et la syntaxe de tous les méta-modèles comme UML, CWM et SPEM ; ils ont le même formalisme. Il spécifie aussi des mécanismes d'interopérabilité entre ces méta-modèles. Il est donc possible de les comparer et de les relier. Grâce à ces mécanismes d'échanges, le MOF peut faire cohabiter plusieurs méta-modèles différents. Il définit pour chaque méta-modèle :

- Un modèle abstrait d'objets MOF génériques et leurs associations.
- Un ensemble de règles pour exprimer un méta-modèle MOF à l'aide d'interface IDL* (Interface Definition Language).
- Un ensemble de règles sur le cycle de vie et la composition des éléments d'un méta-modèle MOF.
- Une hiérarchie d'interfaces réflexives permettant de découvrir et de manipuler des modèles basés sur des méta-modèles MOF dont l'interface n'est pas connue.

Ainsi, une application MOF peut manipuler un modèle à l'aide d'opérations génériques sans connaissance particulière de la structure du modèle. Le MOF est un méta-méta-modèle ouvert et il doit pouvoir supporter un grand nombre d'utilisations, de même qu'il doit pouvoir aussi être supporté par un grand nombre d'applications.

Le MOF propose un framework de modélisation objet qui pourra supporter tout type de méta-modèles et ceux-ci seront enrichis avec de nouveaux concepts. Les principaux concepts sont les suivants :

- Des classes qui permettent de définir les types des méta-objets du MOF.
- Des associations qui permettent de modéliser des relations entre deux méta-objets.
- Des types de données qui permettent de modéliser les autres données (types primitifs, etc...).
- Des paquetages qui rendent ces modèles modulaires, en regroupant des classes et des associations. Un méta-modèle est toujours défini par un paquetage.

Le MOF s'auto-définit en terme de classes, d'associations, de paquetages, de types de données. Cela lui permet aussi de pouvoir définir d'autres méta-modèles [BLA04, CAR03, LEM00, MAR03b, OMG03, OUM03, SRI03].

2.4.3 L'UML et l'OCL

Le langage UML (Unified Modeling Language) a été adopté par l'OMG comme standard de modélisation de système informatique en novembre 1997. Les concepts définis par l'UML sont très proches de ceux définis par le MOF. Ainsi, le MOF utilise les représentations graphiques de l'UML. L'UML a apporté au domaine de la méta-modélisation sa notation graphique et ses concepts objet. Le langage UML permet la modélisation de systèmes indépendamment de toute démarche ou de plateforme. C'est pourquoi, dans le cadre du MDA, UML peut-être utilisé pour décrire des plates-formes (PDM), des organisations ou des situations (PIM) ou la plupart des systèmes logiciels (PSM). L'OMG veut intégrer l'UML au sein même des applications de développement afin d'éviter que le passage au code marque la fin de l'utilisation des modèles UML. Ce passage au code exécutable devra se faire de façon automatique, solutionnant ainsi bon nombre de problèmes de maintenabilité et d'évolutivité des logiciels. Aujourd'hui, la version officielle d'UML est la 1.5 [BEZ02a, CAR03]. La version 2.0 est abordée au chapitre 3.1.1 page 10.

Depuis sa version 1.1, UML intègre le langage OCL (Object Constraint Language) qui est un langage d'expression permettant de décrire des contraintes sur des modèles objet. Une contrainte est une restriction sur une ou plusieurs valeurs d'un modèle non représentable en UML. OCL est un langage sans effet de bord, il est incapable de modifier quoi que ce soit au sein des objets ou même des variables. Il donne des descriptions précises et non ambiguës du comportement du logiciel en complétant les diagrammes. Il définit des pré-conditions, des post-conditions ou des invariants pour une opération. Il permet aussi la définition des expressions de navigation ou des expressions booléennes. OCL est un langage de haut niveau d'abstraction parfaitement intégré à UML qui permet de trouver des erreurs beaucoup plus tôt dans le cycle de vie de l'application. Cette vérification d'erreurs est rendue possible grâce à la simulation du comportement du modèle. OCL est interprété par des outils et par conséquent le passage vers différentes plates-formes technologiques est réalisable de façon semi-automatique ou automatique [BEZ03a, FAV04, SUNb, WAR].

2.4.4 Les profils UML

Un profil UML permet d'adapter le langage UML à un domaine qu'il ne pouvait couvrir correctement. Les profils sont utilisés pour la génération de PIM ou de PSM mais aussi pour passer du PIM au PSM. Les spécificités de chaque plate-forme peuvent être modélisées grâce aux mécanismes d'extension d'UML définis par les profils UML. Par exemple, les stéréotypes permettent l'ajout de nouveaux éléments au méta-modèle, les valeurs marquées (tagged values) permettent l'ajout de propriétés à une méta-classe et les contraintes permettent l'ajout ou la modification de règles [CHE03b]. Il est possible d'associer les stéréotypes, les valeurs marquées et les contraintes à tout concept UML (classe, attribut, association, cas d'utilisation). Ces éléments permettent d'établir une correspondance entre les concepts UML et les concepts du domaine représentés par le profil. Le profil est aussi composé d'un ensemble de règles de présentation, de conception ou de transformation. Ce sont ces règles qui rendent le profil opérationnel [BEL04, BEZ02d, BLA04, GER03, JUL03, MUL04, OMG03, OUM03, PEL02].

L'OMG a défini plusieurs profils ou chacun d'eux a un rôle particulier dans les transformations.

Par exemple :

- Le profil EDOC (Enterprise Distributed Object Computing, version 1) vise à faciliter le développement de modèles d'entreprises, de systèmes ou d'organisations.
- Le profil EAI (Enterprise Application Integration, version 1) simplifie l'intégration d'applications en normalisant les échanges et la traduction des méta-données.
- Le profil CORBA, version 1 pour exprimer la sémantique de CORBA IDL en utilisant une notation UML et pour supporter CORBA IDL dans des outils UML.
- Le profil modélisation temps réel (Schedulability Performance and Time, version 1) pour modéliser des applications en temps réels.
- Le profil Test (version 1 adoptée) permet la spécification de tests pour les aspects structurels (statiques) ainsi que pour les aspects comportementaux (dynamiques) des modèles UML.
- Le profil QoS (Quality of Service, version 1 en cours de finalisation) représente la qualité de service et de tolérance aux erreurs.
- Le profil CORBA Component Model (CCM, version 1 en cours de finalisation) étend le profil CORBA et permet les transformations de PIM à PSM ou de PSM à PSM.
- Le profil SPEM (Software Process Engineering Metamodel, version 1) est défini à la fois comme un profil UML et comme un méta-modèle MOF. SPEM définit les façons d'utiliser UML dans les projets logiciels et permet la création de modèles de processus (pour les PIM exclusivement).

Il existe de nombreux autres profils, publics ou propriétaires.

2.4.5 XMI 2.0 (XML Metadata Interchange)

XMI est le langage d'échange entre le monde des modèles et le monde XML (eXtensible Markup Language). C'est le format d'échange standard entre les outils compatibles MDA. XMI décrit comment utiliser les balises XML pour représenter un modèle UML en XML. Cette représentation facilite les échanges de données (ou méta-données) entre les différents outils ou plates-formes de modélisation. En effet, XMI définit des règles permettant de construire des DTD* (Document Type Definition) et des schémas XML à partir de méta-modèle, et inversement. Ainsi, il est possible d'encoder un méta-modèle dans un document XML, mais aussi, à partir de document XML il est possible de reconstruire des méta-modèles. Les méta-modèles MOF et UML sont décrits par des DTD et les modèles traduits dans des documents XML conformes à leur DTD correspondante. XMI a l'avantage de regrouper les méta-données et les instances dans un même document ce qui permet à une application de comprendre les instances grâce à leurs méta-données. Ceci facilite les échanges entre applications et certains outils pourront automatiser ces échanges en utilisant un moteur de transformation du type XSLT [BLA04, BEZ02a, BEZ02c, LEM00, OUM03, SRI03].

2.4.6 HUTN (Human-Usable Textual Notation)

HUTN est un langage automatisable qui se veut plus simple que XML. Il est utilisé pour générer un langage textuel facilement compréhensible qui sera différent pour chaque modèle. HUTN a pour but d'être un langage générique : chaque langage généré sera conforme à une grammaire commune [BEZ03a].

2.4.7 CWM (Common Warehouse Metamodel)

Enfin, le dernier standard de l'OMG évoqué ici est le CWM qui représente une démarche d'échange des méta-données entre différents entrepôts de données (Relationnel, Objet, XML,...). CWM définit un méta-modèle qui décrit les méta-données métiers mais aussi les méta-données techniques. Il définit les étapes de la modélisation, de la construction et de la transformation des entrepôts de données. Ainsi, CWM apporte l'interopérabilité entre différents logiciels. Ceux-ci peuvent alors échanger leurs méta-données grâce au format CWM. Le premier logiciel traduit ses méta-données vers le format CWM, et le second du format CWM vers ses méta-données [BEZ02a].

2.4.8 Patron de conception (Design pattern)

Bien que n'étant pas un standard MDA, les patrons de conception sont importants car ils vont à terme permettre la génération automatique de code. Un patron de conception est une solution éprouvée d'un problème récurrent de conception dans un contexte défini. C'est un méta-modèle de conception représentant un savoir-faire particulier. Ces techniques permettent d'augmenter la productivité en adoptant certaines structures établies et réutilisables. Elles pourront faciliter la tâche de l'architecte en lui apportant des solutions à des situations similaires déjà vécues [SUNa, CHE03a].

3. Etat de l'art

MDA est en cours d'élaboration. C'est une démarche qui s'inscrit dans la durée et qui ne sera réellement aboutie que dans dix à quinze ans, peut-être plus. Toutefois, il est d'ores et déjà possible de faire un point sur son état d'avancement, sur ses orientations futures ainsi que sur les différents projets lancés pour promouvoir cette démarche. L'OMG, de nombreuses sociétés et partenaires travaillent à sa construction et les retombées sont déjà significatives.

3.1 Retour sur les standards

L'OMG a défini de nouvelles versions pour ces standards. Il les a fait évoluer pour les intégrer dans la démarche MDA. Ces standards convergent tous dans la même direction pour pouvoir être homogènes et s'enrichir mutuellement, chacun apportant des éléments à l'autre.

3.1.1 UML 2.0

UML 2.0 est considérablement influencé par la démarche MDA. Cette nouvelle version est un outil de dialogue et d'échange entre les humains mais aussi, et c'est la nouveauté, entre les humains et les machines. La version 2.0 d'UML est en phase de test et devrait être opérationnelle fin 2004 ou début 2005. Cette nouvelle version, dotée d'une infrastructure plus solide, facilite l'application du MDA grâce notamment à un rapprochement avec le MOF. Le langage d'UML 2.0 est enrichi et devient modulaire. Il est composé de différents sous-langages. Cette segmentation facilite son apprentissage puisque ne seront appris que les sous-langages nécessaires pour le développement du projet.

UML 2.0 améliorera, dans un premier temps, l'approche de la modélisation par composants* qui était une faiblesse d'UML 1.x. Il sera désormais possible de concevoir un composant sans connaître a priori son environnement, puis de le connecter.

Ensuite, la nouvelle version apportera la modélisation d'applications contrairement à UML 1.x qui était basé sur la modélisation visuelle. UML 2.0 offrira la possibilité de transformer des modèles en code de façon automatique et permettra le déploiement et la configuration complète d'application. Ceci se fera bien sûr indépendamment du langage et de la plate-forme cible puisque UML 2.0 séparera le contenu logique de la présentation physique. De plus, UML 2.0 permettra de simuler le modèle au niveau conception, rendant la future application beaucoup plus sûre puisque des tests de débogage auront été faits avant la compilation du code. Les comportements d'un programme seront donc étudiés à l'avance grâce aux diagrammes d'activité ou de séquence, mais aussi grâce à une évolution du langage d'expression de contrainte OCL 2.0. UML 2.0 intègre aussi de nouveaux diagrammes comme le diagramme d'architecture (structural diagram) pour améliorer les spécifications architecturales.

La version 2.0 d'UML standardise et autorise la définition de nouveaux profils de modèles afin d'étendre l'usage d'UML à de nouveaux domaines d'application. Les profils peuvent être appliqués à UML mais aussi à n'importe quel méta-modèle du MOF. Dans la version 2.0 d'UML, il n'y a plus de valeurs marquées. Les stéréotypes prennent plus d'importance et sont utilisés à la place des valeurs marquées. De même, la représentation dynamique des systèmes est améliorée grâce à une meilleure adaptation des graphes d'activité aux machines à états.

Sur le marché se profilent déjà les premiers outils UML 2.0. Cependant, alors que l'objectif initial était de faciliter l'interopérabilité, des profils propriétaires apparaissent. Ces profils ne sont pas publics et les outils proposent rarement un éditeur de profils. Il risque d'y avoir autant de versions d'UML 2.0 que d'éditeurs. L'utilisateur se verra contraint, lors d'un portage d'un modèle d'un outil vers un autre, de réécrire ces modèles ce qui serait un comble pour ce standard [BOR04, REM04].

3.1.2 MOF 2.0

MOF 2.0 a été développé en prenant en compte la nouvelle infrastructure d'UML 2.0, permettant ainsi à UML 2.0 d'être utilisé afin de représenter des méta-modèles de MOF 2.0. MOF 2.0 introduit une importante nouveauté, c'est la norme de QVT* (Query View Transformation) qui est un langage de requêtes et de transformation de modèles. Le QVT est un effort de normalisation des mécanismes, des techniques et des notations nécessaires à la transformation de modèles source en modèles cible. L'objectif du QVT est de définir et d'établir une manière standard pour interroger les modèles du MOF, pour créer des vues et pour définir les transformations de modèles [RAP04]. Pour cela, trois éléments fondamentaux du MOF 2.0 sont utilisés.

- Une question (query) est une expression bien formée dans un langage d'interrogation défini comme OCL 2.0 par exemple. Un modèle est évalué par une question qui renvoie les objets, instances de ce modèle satisfaisant à la condition formulée dans la question.
- Une vue (view) est un modèle qui est dérivé d'un modèle différent sur des aspects spécifiques. Elle dépend d'un modèle source. Elle peut être obtenue en appliquant des transformations à ce modèle. Si un changement est effectué sur un modèle source alors la vue change elle aussi.
- Une transformation (transformation) produit un modèle cible en un modèle source. La définition d'une transformation spécifique décrit le rapport entre le modèle source et le modèle cible au niveau du méta-modèle. Ainsi, la transformation définie est applicable à tous les modèles, instances du méta-modèle source.

Ces deux standards sont importants dans la démarche MDA. Ils ont été spécialement repensés pour améliorer la transformation des modèles et la génération de code.

3.2 L'utilisation des standards pour la transformation

L'essentiel de la démarche MDA réside dans la capacité de passer d'un modèle à un autre. Les techniques de transformation de modèles sont donc au cœur du MDA. Ces transformations peuvent être manuelles, assistées par des outils ou entièrement automatiques. Les transformations étudiées dans ce paragraphe seront essentiellement des transformations de PIM vers PSM car ce sont ces transformations qui seront appelées à être utilisées couramment. D'ailleurs les professionnels du développement ne s'y sont pas trompés et ils proposent déjà des outils pour effectuer ces tâches.

Aujourd'hui plusieurs axes de recherche pour la transformation des modèles sont explorés.

3.2.1 La transformation par annotation ou marquage

Cette technique de transformation expérimentale commence tout juste à être mise en pratique. Elle consiste à annoter (tagger) un PIM à l'aide d'annotations (tag) puis, à convertir ce PIM en PSM à l'aide des règles de transformation issues des profils UML ou à l'aide des patrons de conception (voir figure 5 ci-dessous). Les annotations sont faites manuellement par l'utilisateur. Des traces de cette transformation sont conservées. En les utilisant, il est possible de revenir du PSM vers le PIM car une correspondance entre les éléments source et cible est maintenue. Le terme annotation peut-être remplacé par celui de marquage. Depuis la version 2 du MOF, il est aussi possible d'utiliser les méta-modèles pour annoter les PIM mais cela reste encore au stade de la recherche [BEL03a, BEL04, BEZ03a, JOA03a, JOA03b, JOA03c, MUL04, OMG03].

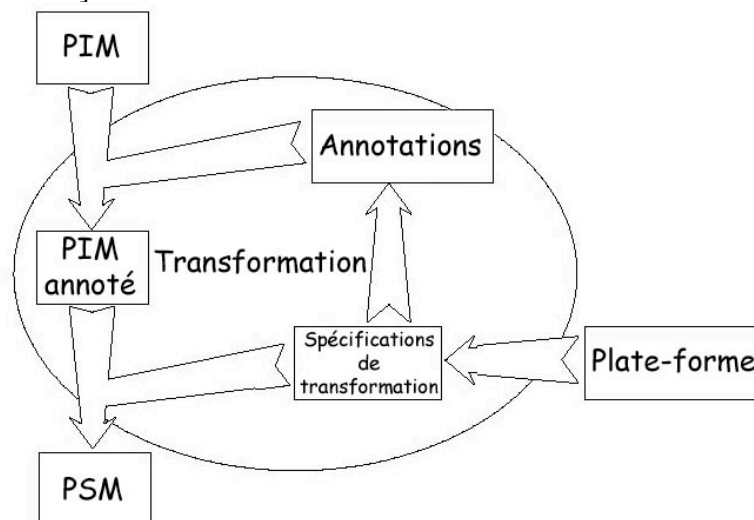


Figure 5 : transformation par annotation ou marquage

3.2.2 La transformation par méta-modèle

Cette technique est actuellement étudiée. Elle est en cours de prototypage et n'est donc pas encore implémentée dans des outils. La transformation s'effectue grâce aux méta-modèles. Les profils UML sont exclus de cette transformation car les méta-modèles peuvent être exprimés dans d'autres langages que l'UML. Elle comporte deux étapes représentée sur la figure 6 page 13. La première consiste à spécifier les règles de transformation décrivant la correspondance entre le langage source et le langage cible. La deuxième consiste à appliquer ces règles au PIM pour produire le PSM.

Le méta-modèle permet de décrire le PIM. Le PIM utilise le même langage que celui de son méta-modèle, donc tout modèle a un méta-modèle. Par exemple, le PIM peut être décrit en UML avec un méta-modèle UML et le PSM peut être spécifique d'une plate-forme particulière comme EJB. La transformation se base sur un langage source (ici UML) et sur un langage cible (ici Java). Dans notre exemple, toute classe en UML est convertie en classe Java grâce à la transformation [BEL03a, BEZ03a, JOA03a, JOA03b, JOA03c, MUL04, OMG03]

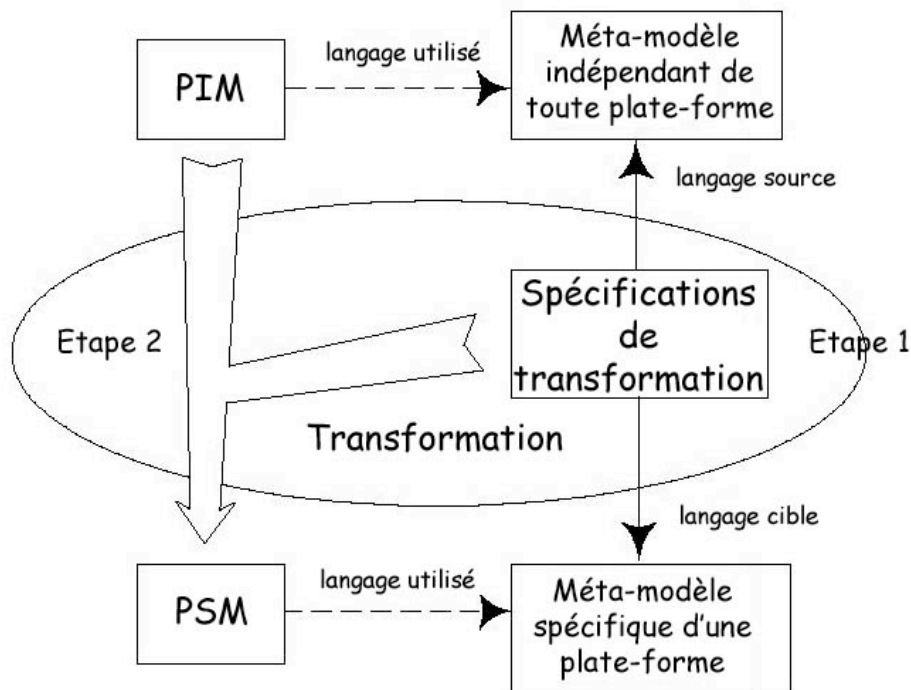


Figure 6 : transformation par méta-modèle

Des approches hybrides entre ces transformations sont pratiquées. Elles permettent encore plus de souplesse et de précision dans la création des modèles PSM. Dans ces paragraphes ont été présentées les techniques les plus utilisées. Il existe d'autres types de transformations comme la fusion de modèle ou la transformation par addition d'informations mais elles restent au stade théorique.

Il est important de remarquer que la transformation par les profils UML et celle par les méta-modèles, bien que similaire dans l'objectif, ont des différences notables. En effet, les méta-modèles ont un langage de plus haut niveau par rapport à celui des profils UML. La conséquence s'en ressent lors d'échanges de fichier XML. Un fichier issu du méta-modèle sera plus clair et concis que celui issu d'un profil UML. Cependant, certains reprochent aux méta-modèles d'être plus complexes à mettre en œuvre et regrettent le manque d'outils facilitant et masquant cette tâche. Ils préfèrent les profils UML qui sont le gage d'une certaine assurance de la qualité structurelle.

3.2.3 L'approche en double Y

Cette approche est actuellement étudiée chez Thales. Elle est intéressante car elle synthétise les recherches qui se font actuellement sur la transformation des méta-modèles. L'approche double Y découle de l'approche simple Y qui est devenue insuffisante car elle ne sépare pas les exigences non fonctionnelles des exigences fonctionnelles. Le cycle simple Y a seulement deux branches, l'une contient les spécifications fonctionnelles, l'autre les informations spécifiques à une plate-forme [BEL04, BEZ03a, BEZ02c, BEZ02d]. L'approche double Y permet d'avoir une vue globale de toutes les étapes de la démarche MDA. Elle permet de conserver une indépendance vis à vis des plates-formes durant la majorité de la phase de développement. Sur la Figure 7 page 14 en partant d'en haut à gauche, nous avons le CIM. Toutefois, dans la pratique peu de personnes utilisent l'appellation « CIM ». Ce tout premier modèle est alors appelé PIM fonctionnel. Il peut être raffiné en de nouveaux PIM si nécessaire. Sur la branche en face de celle du CIM, nous avons les exigences non fonctionnelles et la qualité de service (performance, sécurité, persistance...), qui sont indépendantes d'une plate-forme et/ou d'une architecture particulière. Ces deux branches se rejoignent et par une transformation elles donnent un nouveau modèle toujours indépendant d'une plate-forme mais avec en plus des exigences non

fonctionnelles et/ou des qualités de service. La transformation est effectuée de différentes façons. Par exemple, elle peut-être soit manuelle, soit automatique, ou intermédiaire ; et utiliser une transformation par annotation ou une transformation par méta-modèle. Le PIM issu de la transformation peut être raffiné si besoin. La branche en bas à droite contient des informations spécifiques à une plate-forme. Grâce à ces informations, le dernier PIM non fonctionnel va être transformé en PSM. Ce PSM est donc un modèle spécifique à une plate-forme avec des exigences fonctionnelles et non fonctionnelles. Il va par raffinements successifs donner du code [BEZ02d, FAR03a, FAR03b].

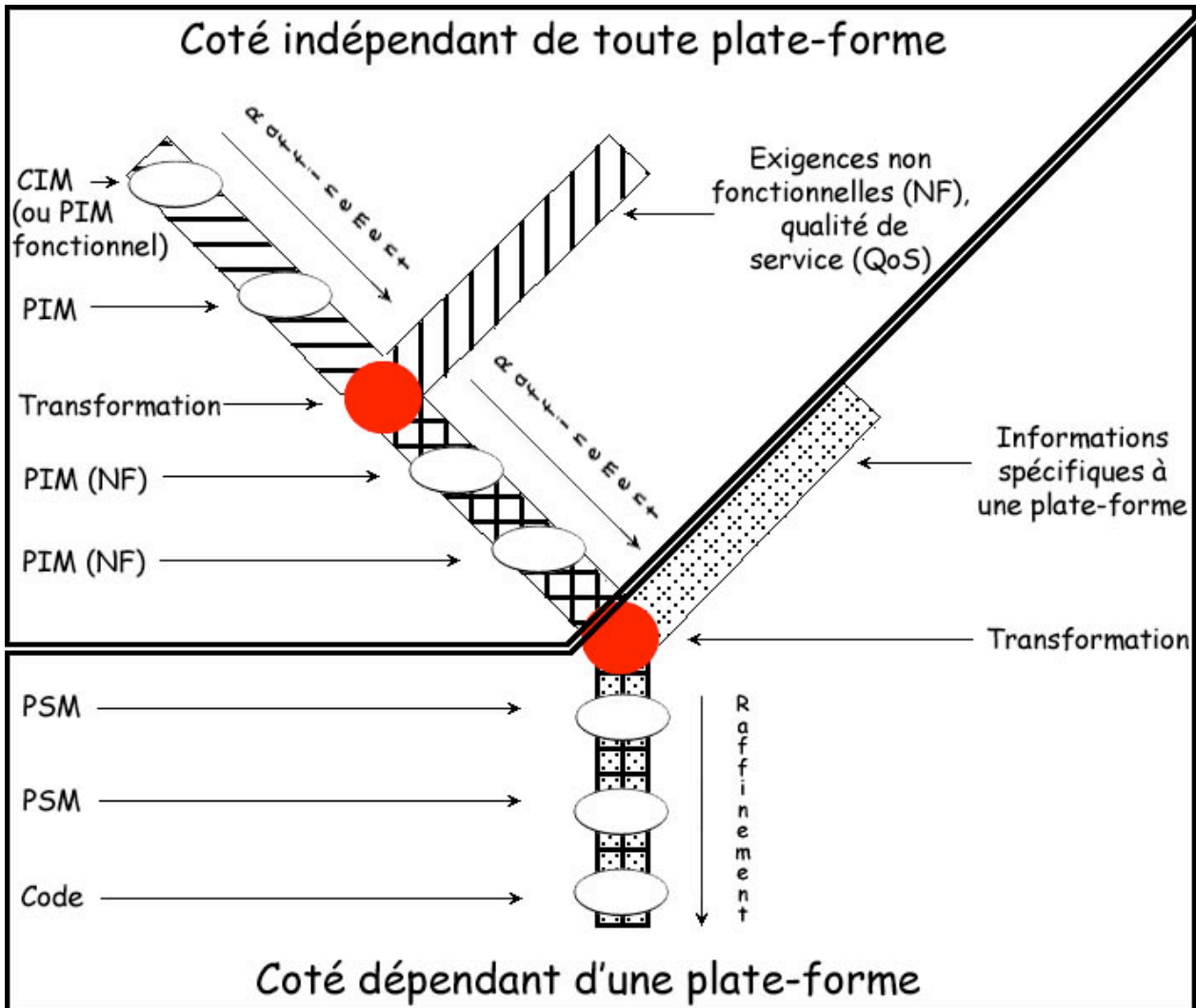


Figure 7 : cycle de transformation en double Y.

3.3 Les projets de recherche sur le MDA

Citons quatre projets dont trois français qui participent à l'élaboration des standards de l'OMG. Le premier a pris fin l'année dernière, les deux suivants sont en cours et le dernier vient tout juste de commencer. Cela montre la volonté de promouvoir le MDA mais aussi la dynamique de développement du MDA.

3.3.1 Le projet ACCORD

Le projet ACCORD (Assemblage de Composants par Contrats en environnement Ouvert et Réparti) regroupe des industriels (France Télécom, EDF), un éditeur d'outils (Softteam) et des organismes de recherche (INRIA, LIFL, CNAM, ENST, ENST-Bretagne). Initié en 2001, c'est un projet français créé dans le cadre du réseau RNTL "Réseau National des Techniques Logicielles" et co-financé par le ministère de l'industrie. Ce projet a pris fin en 2003. Il avait pour objectif de proposer aux architectes et intégrateurs de systèmes un cadre d'analyse et de conception fondé sur l'explicitation de la sémantique de contrats* pour décrire et assembler des composants métier. Le projet ACCORD a fait avancer la compréhension des techniques de développement par composant, ainsi que leurs utilisations, notamment au niveau international. Pour cela, il a été décomposé en trois parties. La première consistait à définir un modèle abstrait de composant et d'assemblage à l'aide du langage UML et de profils génériques. La deuxième spécifiait les règles de projection du modèle ACCORD vers les modèles CCM* et EJB*. Et la troisième, visait à démontrer que la mise en œuvre des deux premières parties était possible grâce à deux cas d'étude réalisés chez France Télécom et EDF. Ces trois parties ont été accomplies avec succès et elles ont débouché sur la création d'un outil basé sur l'atelier objecteering de la société Softteam. Il permet la transformation de modèle abstrait vers un modèle CCM ou EJB. Le projet a aussi abouti à la création d'un langage de modélisation adapté à l'approche composant et compatible avec UML 2.0 (extension d'UML avec les concepts additionnels de connecteurs complexes de contrats). Ce projet a ainsi permis aux différents partenaires de participer à la standardisation de spécifications, comme le profil CCM, proposée par l'OMG [CAR, TRA03].

3.3.2 Le projet MoDAthèque

Ce projet, lui aussi français, est porté par des industriels (Thales, France Télécom, EDF), des éditeurs d'outils (Softteam, Sodifrance, Omondo, W4) et des organismes de recherche (INRIA, CEA-LIST, LIP6, CNAM, IMAG). Proposé en janvier 2003, il est en cours et durera 24 mois. Son financement de 8,5 millions d'euros est assuré pour moitié par des industriels et des éditeurs, et pour l'autre moitié par le Ministère de la Recherche. Ce projet a pour but de bâtir une plate-forme libre qui apportera aux industriels des solutions pour la mise en œuvre de chaînes méthodologiques MDA, ainsi qu'une interopérabilité des outils de manipulation de modèles. Dans un premier temps, ces solutions qui se veulent flexibles et ouvertes, offriront la capitalisation des savoir-faire assurant ainsi des gains de productivité et une qualité de développement. Ensuite, l'architecture ouverte de la plate-forme MoDAthèque permet à une grande diversité d'outils d'entrer dans la compétition, fournissant aux utilisateurs un choix adapté à leurs besoins. De plus, ce projet donne la possibilité à des sociétés et laboratoires français de démontrer leur savoir-faire et leur avance dans un domaine où il existe une forte compétitivité internationale, notamment celle des compagnies américaines. Enfin, le projet MoDAthèque devrait renforcer la coopération entre les différents partenaires du projet. Ils pourront bénéficier de la diffusion de leurs résultats sous forme de standard ou prendre en compte les tendances des futurs standards. La réussite de cette plate-forme pourra se mesurer au travers de son ouverture et de sa capacité à fédérer les utilisateurs et les contributeurs du MDA [PAR03].

3.3.3 ModFact

C'est un projet ObjectWeb Open Source sous licence GPL* (General Public License). Il a débuté en septembre 2002 et a pour objectif de construire une plate-forme d'expérimentation pour l'ingénierie des modèles. Le LIP6* (Laboratoire d'Informatique de Paris 6) a déjà développé plusieurs outils pour la manipulation de modèles. En partenariat avec des industriels et à travers des projets de recherche, le LIP6 a fourni un framework pour la construction d'applications réparties selon une approche orientée modèle. ModFact inclut plusieurs projets. Le projet "Repository" offre le service de génération de référentiels MOF. Il autorise le stockage de modèles sous forme d'objets Java et la manipulation de ces

modèles se fait via les API* (Application Program Interface) proposées par les référentiels. Le projet "QVT" offre le service de transformations de modèles. Il permet la définition de règles de transformation de modèles. Enfin, le projet "Model bus", en cours d'élaboration, offrira le service d'interopérabilité entre outils MDA. Le Model Bus permettra la construction de réseaux d'outils MDA où les outils sont considérés comme étant des fournisseurs de services (transformation de modèles, stockage de modèles, etc...). Le Model Bus servira à connecter les outils sur le bus. Les utilisateurs utiliseront ces services à l'instar d'ORB, si ce n'est que le Modèle Bus n'est pas fondé sur un bus d'objets mais sur un bus de modèles [BOU03].

3.3.4 Le projet modelware

Le projet modelware vient tout juste de commencer (septembre 2004). C'est un projet européen doté de 20 millions d'euros pour une durée de 2 ans. Il a trois objectifs principaux. Le premier est de développer une solution pour permettre une augmentation de 50 % de la productivité de développement des logiciels. La solution proposée est une plate-forme de développement pour l'environnement MDA. Le deuxième objectif est de mener à bien son industrialisation. Et enfin le troisième, est de s'assurer de son adoption par des industriels. De nombreux partenaires participent à ce projet, notamment les industriels Thales, France Telecom, IBM,..., les éditeurs d'outils Softeam, IBM,... et des organismes de recherche INRIA, LIP6,... [POT04].

Ici ne sont cités que quelques projets. Il en existe une vingtaine d'autres à travers le monde, notamment dans la recherche de nouveaux langages de transformation de modèles.

3.4 Bilan

L'approche MDA paraît très séduisante de premier abord. Toutefois, il ne faut pas s'y méprendre, si elle a ses fervents adeptes, elle a aussi ses détracteurs qui avancent prudemment dans sa direction. Nous verrons dans un premier temps les avantages de cette approche, puis les inconvénients qui peuvent en découler, pour enfin mettre en évidence quelques retours sur expérience.

3.4.1 Les avantages du MDA

Ils sont nombreux et attendus ! Le principal avantage prôné par l'OMG est l'indépendance de la logique métier vis à vis de la plate-forme technologique pour éviter la valse des technologies. De cet avantage découlent de nombreux autres. En effet, le fait de séparer la logique métier du code, permet de déceler des possibilités d'erreur avant qu'elles ne soient commises. Les ingénieurs passent d'une démarche correctrice à une démarche préventive, voire prédictive. Cela engendre, des gains de temps pendant le développement de l'application mais aussi des gains de qualité du code. Ainsi, grâce à la transformation automatique du modèle vers le code, les programmes sont développés plus rapidement. Le gain est de 34% selon une étude de middleware company mais cela peut augmenter avec l'expérience, et le code est de bonne qualité car exempt de bogues. De plus, l'application ainsi créée est plus performante et présente une meilleure maintenabilité. Les gains de productivité n'en sont qu'accrus, de même que le retour sur investissement. Le passage par les modèles oblige les architectes à favoriser l'analyse et la modélisation de la logique métier. Les avantages architecturaux n'en sont que meilleurs, même si les spécifications sont plus longues qu'en utilisant d'autres méthodes. Les bénéfices seront tangibles lorsque l'application sera étendue. Il y a une réelle adéquation de l'application et des besoins de l'utilisateur. Il ne faut pas non plus occulter les améliorations de la portabilité entre middlewares, où des plates-formes hétérogènes ne seront plus un frein au développement informatique de l'entreprise. Les systèmes deviennent interopérables, les architectes allant même choisir la plate-forme la mieux adaptée sans tenir compte du contexte préexistant. En effet, les transformations offrent désormais la possibilité de développer plusieurs produits pour différentes plates-formes ou middlewares.

3.4.2 Les inconvénients du MDA

Bien que la démarche comporte des avantages indéniables, elle peut en rebuter certains. En effet, pour développer avec la démarche MDA, cela nécessite des architectes expérimentés qui connaissent et maîtrisent le développement orienté objet ainsi que les patrons de conception. Et c'est une denrée rare au vu de ce sondage où seulement 7 % des sondés comprennent les enjeux de l'approche MDA contre 59 % qui n'ont jamais entendu parlé de MDA. De même pour la nouvelle version d'UML, seulement 6% comprennent les enjeux de la version 2 d'UML contre 46 % qui ne connaissent pas UML 2.0 (SDTime 2002). D'autant plus qu'il n'est pas établi, qu'UML soit un langage efficace pour la programmation. Le MDA reste compliqué à mettre en œuvre. Pour faciliter son utilisation, il est nécessaire d'avoir des outils d'automatisation. Or aujourd'hui, ces outils ne communiquent pas tous entre eux et certains sont spécifiques à une plate-forme en particulier. Il faut noter aussi que les plates-formes pourraient évoluer plus rapidement que les outils de transformation entraînant ainsi un manque d'interopérabilité. De plus, le passage automatisé de PIM vers PSM est loin d'être achevé, d'autant que la distinction entre les deux n'est pas franche. Même s'il existe des outils qui permettent de faire ce passage automatiquement, le passage du PIM vers le code n'est pas complet, avec seulement 80 % de code généré à l'heure actuelle. L'automatisation complète, pour certains projets, devrait voir le jour vers 2008. Il y a donc une carence d'outils dans certains domaines. Enfin, le manque de compatibilité de XMI ainsi que des difficultés à le programmer, ne facilite pas la tâche aux utilisateurs du MDA. Les premières versions n'étaient pas compatibles entre elles et il était par conséquent difficile d'échanger des modèles. C'est notamment ce que reproche Microsoft au MDA, allant même jusqu'à dire qu'XMI n'était pas une bonne approche. Il reproche aussi la complexité du MOF et les limites d'UML qui ne sont pas assez claires. De fait, Microsoft souhaite implémenter à sa façon sa plate-forme de développement de modèles [FRA04].

3.4.3 Retour sur expérience

Des sociétés ont déjà franchi le pas et elles ont mis en œuvre la démarche MDA. Par exemple, la banque Wells Fargo, quatrième banque des Etats-Unis d'Amérique, a pu mettre en pratique cette démarche. Ses dirigeants avaient une stratégie très agressive en rachetant des banques concurrentes. Ces banques avaient un système informatique complètement différents de celui de Wells Fargo. Pour les intégrer au sein de leur système informatique, les ingénieurs ont utilisé l'approche MDA. Ils ont modélisé des modèles métier (PIM) qu'ils ont maintenu indépendant de toutes plates-formes. Les ingénieurs peuvent intégrer ou changer de technologies sans que le modèle métier ne soit affecté [MOG03].

Une autre société Magnet Communications qui travaille elle aussi dans le milieu bancaire pour des institutions financières, souhaitait développer ses applications sous J2EE. Elle avait une équipe de développeurs compétente mais ils ne proposaient pas un code de bonne qualité. En modélisant les applications en UML et en créant des PIM, 90 % du code a été généré automatiquement. L'automatisation a eu l'avantage de délivrer un code exempt d'erreur[OMGa].

Enfin un dernier exemple avec Swisslog qui est un fournisseur global de solutions intégrées de chaînes d'approvisionnement. En utilisant la démarche MDA pour développer leur logiciel de logistique, les ingénieurs ont permis une utilisation efficace de leurs ressources, grâce à la séparation de la logique métier de la plate-forme technique. Lors de la conception de l'application, leur productivité et leurs retours sur investissements ont été accrus grâce à l'automatisation de la production de code. Cette automatisation a aussi accru la qualité de leur application tout en améliorant la mise en place des normes de la société[OMGb].

Ces retours sur expérience sont plutôt convaincants. Ils ont pu être réalisés grâce à des outils MDA conçus par des sociétés de software. Il existe aujourd'hui une cinquantaine d'outils sur le marché. Cependant, ils n'implémentent pas la philosophie MDA de bout en bout. Certains ne proposent la transformation que vers une seule plate-forme, d'autre sont loin de tout automatiser. Il faut donc bien choisir son outil en fonction de la plate-forme de destination et du travail à réaliser.

Conclusion

La démarche MDA, bien qu'assez récente, suscite un réel intérêt chez bon nombre d'industriels et de développeurs. En effet, cette démarche est prometteuse et répond à des attentes légitimes non comblées par les technologies objet ou composant. Elle autorise la séparation de la logique métier de l'entreprise de son implémentation physique. Cette nouvelle approche bouleverse complètement notre façon de concevoir une application et ouvre de nouveaux horizons pour le génie logiciel qui ne sera plus dépendant des évolutions technologiques.

Toutefois, même si la démarche MDA est soutenue par l'OMG, ce qui est un gage de sécurité et de pérennité, il se peut que cette approche soit un jour concurrencée par des projets de même envergure, mais plus simples d'utilisation. Il suffit de regarder l'exemple de CORBA qui du fait de sa complexité a permis l'émergence de middleware comme .NET ou EJB au maniement plus facile. La démarche MDA est encore trop compliquée et réservée à de grandes entreprises ou des laboratoires de recherche. D'ailleurs, Microsoft a des vues divergentes face à cette démarche. Cependant, l'OMG est prêt à faire une passerelle de MDA vers la solution Microsoft comme il l'a fait pour CORBA et DCOM* (Distributed Component Object Model).

Pour que le MDA se diffuse auprès des développeurs, le savoir doit être essaimé et de nouveaux outils doivent être développés. Pour produire ces outils qui font encore aujourd'hui défaut, plusieurs projets ont été lancés pour que l'approche MDA tienne ses promesses. Ces outils permettront l'automatisation des transformations ainsi que la génération automatique de code à partir de modèles, permettant aux architectes de se consacrer pleinement aux tâches de modélisation métier. La démarche MDA deviendra une architecture viable, dès lors que ces transformations seront concrétisées et intégrées dans les outils.

A l'heure de la mondialisation, cette approche est importante car l'Europe pourra rivaliser avec d'autres pays comme l'Inde, ou des pays de l'Est comme la Roumanie où le coût de développement est moindre. L'Europe pourra rester compétitive en conservant son savoir-faire et la maîtrise de la production des logiciels. Les entreprises qui ne voudraient pas adhérer à cette démarche en développant leurs logiciels clients, s'élimineront d'elles-mêmes par la voie de la sélection sur la rentabilité.

Bibliographie

Tous les liens Internet ont été vérifiés le 20 octobre 2004.

- [ARB04] ARBELET L., *L'adhésion au standard MDA reste timorée*, 01 Informatique, [en ligne], 11 juin 2004, Disponible sur : <http://www.01net.com/article/245351.html>
- [BEL03a] BELAUNDE M., *L'approche MDA et les expérimentations à France Télécom*, [en ligne], 26 juin 2003, Disponible sur : <http://aristote1.aristote.asso.fr/Presentations/CEA-EDF-2003/Conferences/MarianoBelaunde/ExperimentationsMDA-EcoleDEte-26Juin2003.ppt>
- [BEL03b] BELAUNDE M., *L'approche MDA et les expérimentations à France Télécom*, 16 - 27 Juin 2003, Ecole d'Eté d'Informatique CEA - EDF - INRIA 2003 [en ligne], Disponible sur : http://aristote1.aristote.asso.fr/Presentations/CEA-EDF-2003/Conferences/MarianoBelaunde/Video/Conference_5_FranceTelecom.rm
- [BEL04] BELAUNDE M., *Le MDA* [en ligne], 9 janvier 2004, Disponible sur : http://www.bretagne.ens-cachan.fr/DIT/People/Claude.Jard/sem_20_04_2004_FTRD_trans9.pdf
- [BEZ02a] BEZIVIN J. et BLANC X., *MDA vers un nouveau paradigme(1)*, Développeur Référence, [en ligne], 15 juillet 2002, v2.16, pp. 7-11, Disponible sur : <http://www.weblmi.com/devreference>
- [BEZ02b] BEZIVIN J. et BLANC X., *Promesses et interogations (2)*, Développeur Référence, [en ligne], 5 septembre 2002, v2.17, pp. 8-12, Disponible sur : <http://www.weblmi.com/devreference>
- [BEZ02c] BEZIVIN J. et BLANC X., *Standards et travaux (3)*, Développeur Référence [en ligne], 2 octobre 2002, v2.18, pp. 8-10, Disponible sur : <http://www.weblmi.com/devreference>
- [BEZ02d] BEZIVIN J., *Les nouveaux défis des systèmes complexes et la réponse MDA de l'OMG*, [en ligne], 2002, Disponible sur : <http://www.lifl.fr/jfiadsma2002/talks/jfiadsma2002-Bezivin.pdf>
- [BEZ03a] BEZIVIN J., *Ecole d'Eté d'Informatique CEA EDF INRIA 2003*, Ingénierie des modèles logiciels [en ligne], 2003, Disponible sur : <http://aristote1.aristote.asso.fr/Presentations/CEA-EDF-2003/Cours/JeanBezivin/IndexJeanBezivin.html>
- [BEZ03b] BEZIVIN J., *Object to Model Paradigm Change with the OMG/MDA Initiative*, [en ligne], 2003, Disponible sur : <http://rangiroa.essi.fr/cours/mda/03-bezivin.ppt>
- [BLA02] BLANC X et DESFRAY P., *Du profil UML au composant MDA (4)*, Développeur Référence [en ligne], 2 décembre 2002, v2.22, pp. 2-5, Disponible sur : <http://www.weblmi.com/devreference>

- [BLA04] BLANC X., *Ingénierie des modèles*, [en ligne], juillet 2004, Disponible sur : <http://www-src.lip6.fr/homepages/Xavier.Blanc/courses/XB-Model-Engineering.ppt>
- [BLA] BLANC X., *Modèles et XML*, [en ligne], Disponible sur : http://www.adae.gouv.fr/upload/documents/blanc_xml_journeadae.pdf
- [BON04] BONDE L., *Notes de lecture*, MDA Guide, [en ligne], Disponible sur : http://www.lifl.fr/~bonde/Notes/NotesDeLecture_1_2004.pdf
- [BOR04] BORDERIE X., *Passer à UML 2*, JDN Développeurs, [en ligne], 15 septembre 2004, Disponible sur : <http://developpeur.journaldunet.com/tutoriel/cpt/040915-passer-a-uml2.shtml>
- [BOU03] BOUZITOUNA S., *ModFact un support pour le MDA*, Laboratoire d'informatique de Paris 6, [en ligne], Disponible sur : <http://meta.lip6.fr/file/presentation/ModFact.zip>
- [BRE01] BRETON E. et BEZIVIN J., *Un méta-modèle de gestion par les activités*, [en ligne], 20 p., Disponible sur : <http://www.sciences.univ-nantes.fr/lina/atl/publications/cite.pdf>
- [CARI03] CARIOU E., *Contribution à un Processus de Réification d'Abstractions de Communication*, Thèse, Université de Rennes I, [en ligne], 17 juin 2003, 150 p., Disponible sur : <http://www.irisa.fr/triskell/publis/2003/Cariou03a.pdf>
- [CLA04] CLAPAUD A., *L'approche MDA remet en selle les ateliers de génie logiciel*, 01 Informatique, [en ligne], 01 mai 2004, Disponible sur : <http://www.01net.com/article/245015.html>
- [CAR] CARREZ C., *Projet RNTL ACCORD, Assemblage de Composants par Contrats en environnement Ouvert et Réparti*, [en ligne], Disponible sur : <http://www.infres.enst.fr/projets/accord/index.html>
- [CAR03] CARON P-A., *Spécialisation d'un environnement de conception de systèmes flexibles aux Environnements Informatiques pour l'Apprentissage Humain*, Mémoire de DEA, Université des Sciences et Technologies de LILLE, [en ligne], 2 juillet 2003, 53 p., Disponible sur : http://noce.univ-lille1.fr/cms/uploaddocs/Rapport_de_Stage_DEA_Informatique_ver_8.pdf
- [CHE03a] CHEVAILLIER P., *Techniques avancées de programmation objet*, [en ligne], novembre 2003, ;Disponible sur : <http://www.enib.fr/info/gl2/cours/pattern.pdf>
- [CHE03b] CHEVAILLIER P., *Méta-programmation*, [en ligne], décembre 2003, Disponible sur : <http://www.enib.fr/info/gl2/cours/metaprogram.pdf>
- [CON04] CONSYST SQL., *MDA/MDD, pas simplement un nouvel acronyme!*, [en ligne], 2004, Disponible sur : http://www.consyst-sql.com/images_f/WWW/WP_MDA_short_2135_1_1.pdf
- [FRA04] FRANKEL D., *Domain-Specific Modeling and Model Driven Architecture*, MDA journal, [en ligne], janvier 2004, 10 p .;Disponible sur : http://www.bptrends.com/publicationfiles/01-04_COL_Dom_Spec_Modeling_Frankel-Cook.pdf

- [FAR03a] FARCET N., *MDA assessment and deployment within Thales*, [en ligne], 2003, Disponible sur : <http://aristote1.aristote.asso.fr/Presentations/CEA-EDF-2003/Conferences/NicolasFarcet/MIRROR-EDF-CEA-summer-school-2003-publication.ppt>
- [FAR03b] FARCET N., *MDA assessment and deployment within Thales*, 16 - 27 Juin 2003, Ecole d'Eté d'Informatique CEA - EDF - INRIA 2003 [en ligne], Disponible sur : http://www.aristote.asso.fr/Presentations/CEA-EDF-2003/Conferences/NicolasFarcet/Video/Conference_5_Thales.rm
- [FAV04] FAVRE J-M., *OCL : Object Constraint Language Introduction*, [en ligne], mars 2004, Disponible sur : <http://www-adele.imag.fr/~jmfavre/ENSEIGNEMENT/GI-03-04/IDM/OCL-Introduction-11-6pp.pdf>
- [FER04] FERRET r. et DEGOS J., RRD contre OptimalJ, *Le Monde Informatique*, 21 mai 2004, No 1027.
- [GER03] GERARD S., *profil UML pour les systèmes embarqués de l'automobile*, [en ligne], 03 février 2000;Disponible sur : http://www.supelec-rennes.fr/sic/JOURNEES/00_02_03/gerard.ppt
- [JOA03a] JOAQUIN M., *MDA Model Driven Architecture*, [en ligne], 2003;Disponible sur : http://www.joaquin.net/MDA/Model_Driven_Architecture_Tutorial.pdf
- [JOA03b] JOAQUIN M., *The several styles of Model Driven Architecture*, [en ligne], 2003, Disponible sur : http://www.joaquin.net/MDA/The_Several_Styles_of_MDA.pdf
- [JOA03c] JOAQUIN M., *What's a Platform -- An Overview of Model Driven Architecture*, [en ligne], 2003, Disponible sur : http://www.joaquin.net/MDA/What%27s_a_Platform--An_Overview_of_Model_Driven_Architecture.pdf
- [JUL03] JULIOT E., *MDA Model Driven Architecture*, Mémoire de DESS Génie Informatique, Université de Nantes, [en ligne], 10 juin 2002, 16 p., Disponible sur : <http://docs.happycoders.org/orgadoc/dev/mda/toutmda.pdf>
- [LAI04] LAI M., 2004, *Pensez objet avec UML et JAVA*, Dunod, 3ème édition, Paris, 219 p.
- [LEM00] LEMESLE R., *Techniques de modélisation et de Méta-Modélisation*, Thèse, Université de Nantes, [en ligne], 26 octobre 2000, 276 p., Disponible sur : <http://www.sciences.univ-nantes.fr/info/lrsg/Recherche/mda/richard.pdf>
- [MAR03a] MARVIE R., *model driven engineering*, [en ligne], 2003, Disponible sur : http://www.lifl.fr/~marvie/cours/marvie_mde.pdf
- [MAR03b] MARVIE R., *Vers des patrons de méta-modélisation*, [en ligne], 10 janvier 2003, Disponible sur : <http://www-valoria.univ-ubs.fr/Jacques.Malenfant/ALP.OCM/Journee2003/Marvie.pdf>
- [MOG03] MOGHRABI X., *L'approche Model-Driven Architecture, crédible pour développer un progiciel de gestion intégré*, Mémoire de DEA, Ecole des mines d'Albi, [en ligne], 2003, 63 p., Disponible sur : http://noce.univ-lille1.fr/cms/uploaddocs/Rapport_de_Stage_DEA_Informatique_ver_8.pdf

- [MUL02] MULLER A., *Assemblage par vues de composants logiciels*, Mémoire de DEA, Laboratoire d'Informatique fondamentale de Lille, [en ligne], juin 2002, 61 p., Disponible sur : http://www.lifl.fr/~mullera/publi/memoire_muller.pdf
- [MUL04] MULLER A., *Transformation de modèles : d'un modèle abstrait aux modèles EJB et CCM*, [en ligne], 2004, Disponible sur : http://www.lifl.fr/lmo2004/slides_lmo2004/muller.pdf
- [OMG03] OMG, *MDA Guide Versions 1.0.1*, [en ligne], juin 2003, Disponible sur : <http://www.omg.org/docs/omg/03-06-01.pdf>
- [OMGa] OMG, *Magnet Communications, Inc.*, MDA Success Story [en ligne], Disponible sur : http://www.omg.org/mda/mda_files/MagnetCommunications.htm
- [OMGb] OMG, *Swisslog Software AG*, MDA Success Story [en ligne], Disponible sur : http://www.omg.org/mda/mda_files/Aonix_Swisslog_SuccessStory.pdf
- [OUM03] OUM OUM SACK P-M., *Interopérabilité des Outils d'assistance à l'évolution des systèmes logiciels*, Mémoire de DEA, Laboratoire d'Informatique du Littoral, [en ligne], 26 juin 2003, 77 p., Disponible sur : <http://lil.univ-littoral.fr/~oumoumsack/publi/MemDocFinal.pdf>
- [PAR03] PARIGOT D., *MoDAthèque Plate-forme MDA*, [en ligne], janvier 2003, Disponible sur : <http://www-sop.inria.fr/oasis/personnel/Didier.Parigot/MDA/resumer.html>
- [PARI03] PARISOT T., *La programmation de mieux en mieux encadrée*, Le Monde Informatique, 19 décembre 2003, No 1007.
- [PEL02] PELTIER M., *Transformation entre un profil UML et un méta-modèle MOF : application du langage MTRANS*, [en ligne], Janvier 2002, Disponible sur : <http://www.lirmm.fr/LMO2002/pres/Lmo2002Peltier.pdf>
- [POT04] POTIER D., *Modelware*, [en ligne], juin 2004, Disponible sur : <http://www.artist-embedded.org/PastEvents/Rome04/Tuesday/ETP-Rome04-Potier-MODELWARE.pdf>
- [REM02] REMY C., *MDA et UML s'imposent*, Le Monde Informatique, 13 décembre 2002, No 963.
- [REM04] REMY C., *UML 2 modélise les applications*, Le Monde Informatique, 07 novembre 2003, No 1001.
- [RAP04] RAPELA D., *MODA-TEL (Deliverable 3.3), MDA modelling and application principles* [en ligne], mai 2004, 75 p., Disponible sur : <http://www.modatel.org/~Modatel/pub/deliverables/D3.3-final.pdf>
- [SRI03] SRIPLAKICH P., *Techniques des transformations de modèles basées sur la méta-modélisation*, Mémoire de DEA Systèmes Informatiques Répartis, Université Pierre et Marie Curie, [en ligne], septembre 2003, 64 p., Disponible sur : <http://modfact.lip6.fr/ModFactWeb/qvt/SimpleTRL.pdf>
- [SUNa] SUNYE G., *Introduction aux patrons de conceptions*, [en ligne], Disponible sur : <http://www.unantes.univ-nantes.fr/modules/d3/>

- [SUNb] SUNYE G., *Conception par contrats avec UML OCL – Object Constraint Language*, [en ligne], Disponible sur : <http://www.unantes.univ-nantes.fr/modules/d1/ocl>
- [TRA03] TRAVERSON B., projets RNTL ACCORD Bilans et perspectives, EDF R&D, 13 novembre 2003, [Par mail], Non Disponible en ligne.
- [VAR04] VARANDAT M., *Avec ou sans UML, la technologie progresse*, 01 Informatique, [en ligne], 28 mai 2004, Disponible sur : <http://www.01net.com/article/243233.html>
- [WAR] WARMER et KLEPPE., *OCL, un survol*, [en ligne], Disponible sur : https://cours.ele.etsmtl.ca/academique/mgl/mgl806/Cours/ThemeOCL_01.pdf

Glossaire

API :	Application Program Interface	16
Architecture :	L'architecture d'un système est la spécification de ses différentes parties avec leurs interconnexions qui le constituent.	3,13,15,18
CCM :	CORBA Component Model	9,15
CIM :	Computation Independent Model	4,13
Composant :	Module logiciel autonome qui assure un service. Ce module peut être installé sur différentes plates-formes. En vue de son utilisation ou de sa réutilisation, il exporte différents attributs ou méthodes sous forme d'interfaces. De plus, il possède des points d'entrée permettant sa configuration et est capable de s'auto décrire. L'objectif affiché de la notion de composant est la réutilisation de modules logiciels, que ce soit lors de la construction de nouvelles applications par assemblage de composants existants, lors de l'intégration de nouveaux composants dans des logiciels existants.	2,11,15,18
Contrat :	Il spécifie les propriétés et les contraintes liées aux différents éléments du modèle d'assemblage. C'est une mise en relation entre plusieurs spécifications. Cela permet une vérification de compatibilité.	15
CORBA :	Common Object Request Broker Architecture	2-5,18
CWM :	Common Warehouse Metamodel	3,7,10
DCOM :	Distributed Component Object Model	18
DTD :	Document Type Definition	9
EAI :	Enterprise Application Integration	9
EDOC :	Enterprise Distributed Object Computing	9
EJB :	Entreprise Java Bean	2,4,12,15,18
Framework :	Infrastructure logicielle qui facilite la conception des applications par l'utilisation de bibliothèques de classes ou de générateurs de programmes.	3,8,15
GPL :	General Public License	15
HUTN :	Human-Usable Textual Notation	10
IDL :	Interface Definition Language	8
LIP6 :	Laboratoire d'Informatique de Paris 6	15-16
MDA :	Model Driven Architecture (Architecture basée sur les modèles ou Architecture dirigée par les modèles)	1-18
Middleware :	Permet la communication entre des clients et des serveurs ayant des structures et une implémentation différentes. Il permet l'échange d'informations dans tous les cas et pour toutes les architectures. Enfin, le middleware doit fournir un moyen aux clients de trouver leurs serveurs, aux serveurs de trouver leurs clients et en général de trouver n'importe quel objet atteignable.	2,3,16,18
MOF :	Meta-Object Facility	3,5,7-11,12,15
OCL :	Object Constraint Language	4,8,11

OMG :	Object Management Group	1-10,14-16,18
Plate-forme :	Ensemble ou sous-ensemble de systèmes et de technologies qui fournissent un ensemble cohérent de fonctionnalités au travers d'interfaces.	2-6,8-9,11-17
PDM :	Platform Description Model	4-5,8
PIM :	Platform Independent Model	4-9,12-14,17
PSM :	Platform Specific Model	4-9,12-14,16
QoS :	Quality of Service	9
QVT :	Query View Transformation	11,16
SDL :	Specification and Description Language	6
SPEM :	Software Process Engineering Metamodel	7,9
Système :	Pris au sens large, il peut s'agir d'un programme, d'un ensemble d'applications logicielles,...	2-4,8-9,11,15-17
UML :	Unified Modeling Language	3-4,6-7,8,9,10-11,12,15,17
XMI :	XML Metadata Interchange	3,9,10,13
XML :	eXtensible Markup Language	3,9-10