Chapitre 1 Initiative MDA

1. Introduction

En novembre 2000, l'Object Management Group (OMG) a proposé une approche nommée Model Driven Architecture (MDA) pour le développement et la maintenance des systèmes à prépondérance logicielle.

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 modèles. Le but est de favoriser l'élaboration de modèles de plus haut niveau.

Il s'agit de mettre à profit les compétences et résultats obtenus par exemple dans les domaines de la compilation des langages, des méthodes formelles, de la modélisation par objets, de la programmation par composants distribuée, des technologies du web, etc. Pour cette raison, nous ne parlerons plus de MDA mais d'*Ingénierie Dirigée par les Modèles* (IDM, ou MDE pour "Model Driven Engineering" dans la langue de Shakespeare).

Pour faire face à la complexité et à l'évolution croissante des applications, l'IDM ouvre de nouvelles voies d'investigation. En autorisant une appréhension des applications selon différents points de vues tout en intégrant comme fondamental la composition et mise en cohérence de ces perspectives, elle ne peut s'inscrire dans la pérennité que si elle prend ces racines dans des bases bien fondées établies par la théorie.

2. Evolution récente en ingénierie dirigée par les modèles

2.1. Pourquoi l'IDM ? Pourquoi faire ?

• Évolution permanente des technologies logicielles.

• Faire communiquer et interagir des éléments distants.

• Normaliser un standard qui sera utilisé par tous.

• Systèmes distribués : Middleware (Intergiciel).

• Les algorithmes de distribution et de calcul sont indépendants de la technologie et de la mise en oeuvre.

• Nécessité de découpler la logique métier et la mise en oeuvre technologique.
• Séparation des préoccupations.

• Besoin de modéliser et spécifier.

- A un niveau abstrait la partie métier.
- La plateforme de mise en œuvre.
- Projeter ce niveau abstrait à la plateforme.

2.2. Le MDA et l'IDM

L'approche MDA devient une variante particulière de l'Ingénierie Dirigée par les Modèles. L'IDM peut être vue comme une famille d'approches qui se développent à la fois dans les laboratoires de recherche et chez les industriels impliqués dans les grands projets de développement logiciels. Deux de ces industriels (IBM et Microsoft) ont récemment défini leur stratégie MDE et le moins que l'on puisse dire c'est qu'elles semblent converger.

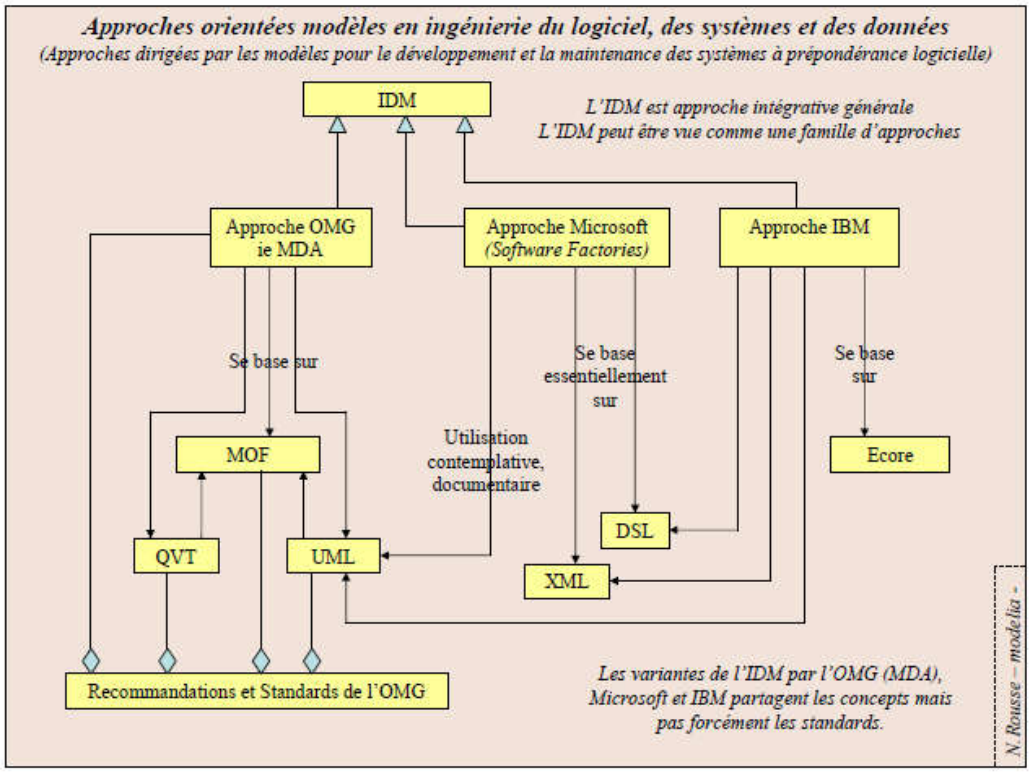

Figure 1 : IDM vue comme une famille d'approches.

L'idée initiale de l'OMG consistait à s'appuyer sur le standard UML pour décrire séparément les parties des systèmes indépendantes des plates-formes spécifiques (PIM ou Platform Independant Models) et les parties liées aux plates-formes (PSM ou Platform Specific Models).

L'OMG met surtout en avant actuellement une architecture dont le socle est le MOF (Meta-Object Facility) sur lequel s'appuie d'une part une collection de méta modèles dont UML n'est qu'un élément parmi d'autres et d'autre part une technologie émergente de transformation de modèles nommée MOF/QVT .

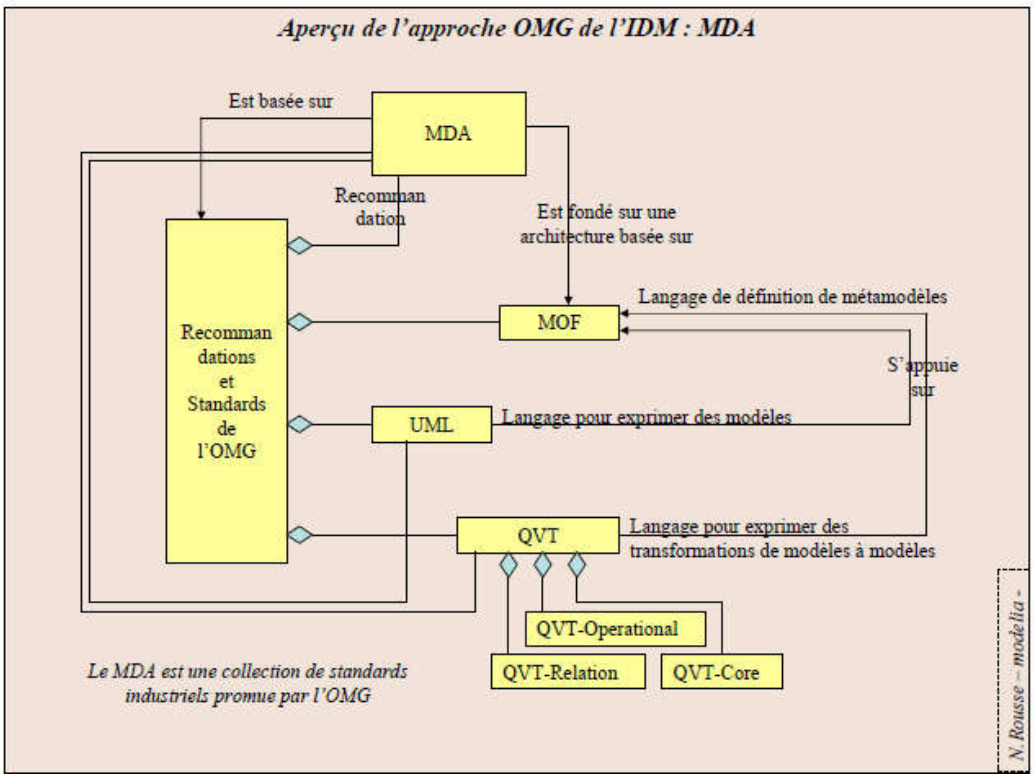

Figure 2 : Aperçus de l'approche OMG de l'IDM : MDA.

2.3. UML et l'IDM

Il faut donc séparer clairement les approches IDM du formalisme UML. Non seulement la portée de l'IDM est beaucoup plus large que celle d'UML, mais la vision IDM est aussi très différente de celle d'UML et parfois même en contradiction. UML est, dans ses versions 1.5 et 2.0, un standard assez monolithique obtenu par consensus à maxima, dont on doit réduire la portée à l'aide de mécanismes comme les profils.

Dans certains outils UML de première génération, le support des profils UML a été présenté comme une fonctionnalité importante. De plus certains de ces outils ont proposé des langages de décoration propriétaires. Le résultat de ces choix est bien souvent de créer des modèles qui ont parfois coûté cher à produire et qui vont être difficiles à réintégrer dans des approches IDM. Ces modèles restent liés à l'outil qui les a produits, ce qui est en contradiction avec les objectifs de l'approche de l'OMG prônant l'indépendance de la plate-forme et donc à fortiori des outils de développement.

2.4. Les objets et l'IDM

Il faut clairement séparer l'approche orienté objet de l'approche orientée modèle et de l'IDM. S'il est vrai que le standard MDA de l'OMG est directement fondé sur une technologie orientée objet, ce n'est là qu'un choix technologique. Dans bien des cas cependant la confusion est faite entre les concepts que l'on trouve dans l'IDM et leur

incarnation dans le monde objet. Par exemple la vision selon laquelle un modèle serait une "instance d'un" métamodèle a été largement relayée par l'OMG. La relation *InstanceDe* est trop souvent utilisée de manière informelle. Par exemple l'espace technologique des grammaires, des documents structurés XML, ou des bases de données relationnelles, ne sont pas fondés sur le concept d'objet.

2.5. Les méthodes de modélisation et l'IDM

La terminologie « Ingénierie Dirigée par les Modèles » laisse supposer que la nouveauté de cette approche réside dans l'utilisation systématique de modèles. Il n'en est pourtant rien et cette caractéristique est loin d'être originale. Il existe depuis longtemps en informatique des méthodes de modélisation, la plus connue en France étant Merise. Dans ces méthodes, les modèles sont vus comme étant la base de toute activité humaine d'ingénierie. Ces méthodes ont largement contribué à la diffusion du concept de modèle en informatique, mais leur impact sur la production effective du logiciel reste néanmoins mitigé. Plus de 50 ans après les débuts de l'informatique, les pratiques industrielles restent centrées sur le code, considéré comme étant le seul représentant fiable du logiciel.

Pour qu'un modèle soit "**productif**" il doit pouvoir être interprétable et manipulable par une machine. Il est considéré comme essentiel dans le cadre de l'IDM de pouvoir exprimer formellement le contenu d'un modèle et "quoi faire" avec ce modèle et ceci quel que soit le niveau d'abstraction. Ceci se traduit par le besoin d'exprimer formellement les transformations entre modèles, les rendant ainsi productifs. La caractéristique originale de l'Ingénierie Dirigée par les Modèles est plus dans l'utilisation systématique de méta-modèles, que dans l'utilisation systématique de modèles. L'IDM se distingue des méthodes de modélisations traditionnelles comme Merise par la préoccupation constante de rendre les (meta) modèles productifs plutôt que contemplatifs.

2.6. La programmation par aspects et L'IDM

Dans des approches comme la programmation par aspects on essaye de réaliser la synthèse des préoccupations logicielles sur la base du code, par exemple avec des extensions du langage Java comme AspectJ ou HyperJ.

Dans l'approche MDA on essaye de réaliser la séparation des aspects liés ou indépendants de la plate-forme par des modèles différents (PIM et PSM).

La séparation et le tissage des aspects sont donc au centre des approches IDM. Pour l'OMG, ce qui est prioritaire dans le MDA, c'est la séparation des aspects liés ou indépendants de la plate-forme mais pour l'IDM, de façon plus générale, les aspects à prendre en compte couvrent toutes les préoccupations qui peuvent apparaître lors du développement, de la maintenance et de l'évolution des systèmes à prépondérance logicielle (aspects fonctionnels mais aussi aspects non-fonctionnels comme la qualité de service, la sécurité, etc.)

3. Architecture du MDA

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).. 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 évènements, 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 lectronique, manufacture,…)

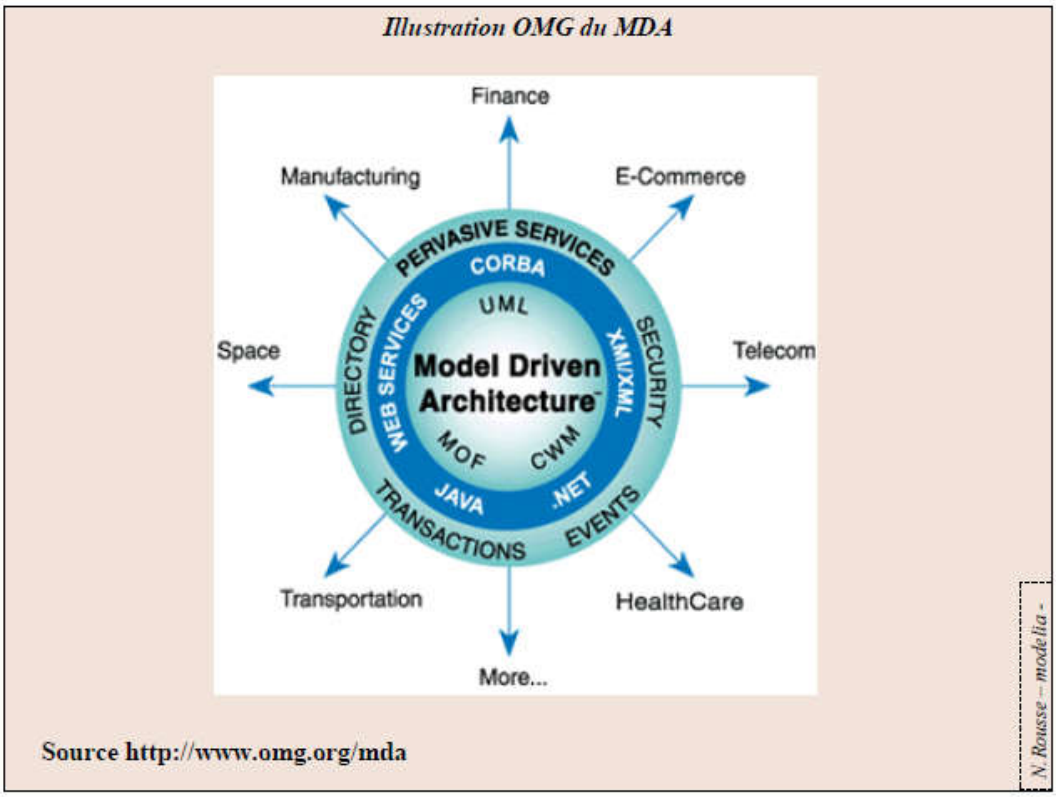

Figure 3 architecture MDA.

Pour créer une application, un architecte fait un schéma en UML par exemple, mais d'autres langages peuvent aussi être utilisés. L'architecte dirigera son application en évoluant de couche en couche pour aller vers le domaine d'application qui 'intéresse.

3.1. 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.

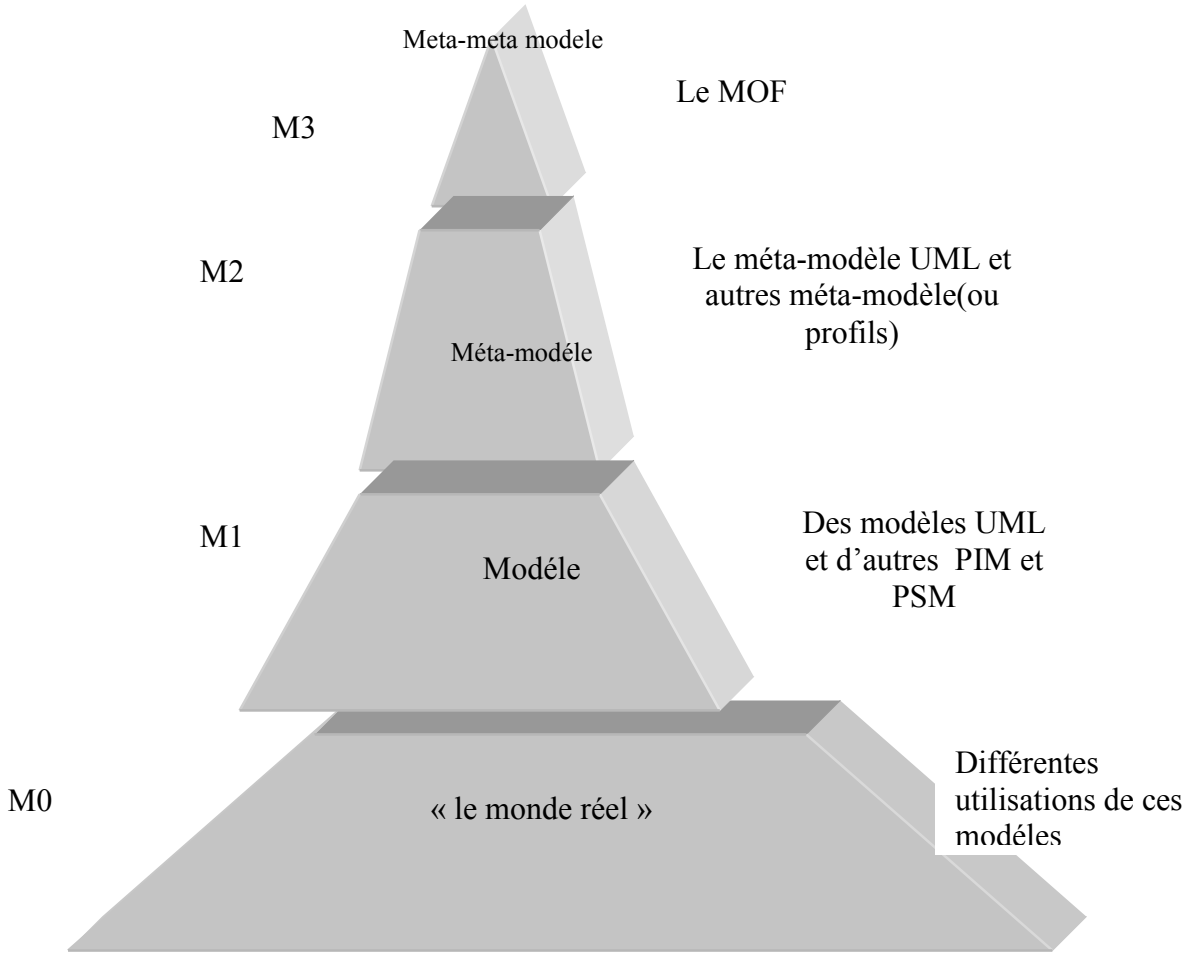

Figure 4 : architecture à quatre niveaux.

3.2. 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.

L'approche MDA distingue deux aspects principaux dans le processus de développement d'une application, l'aspect métier qui représente les fonctions de l'application, et l'aspect technique qui représente la technologie de mise en œuvre de l'application. Chaque aspect est exprimé par un ensemble de modèles, qui véhiculent l'information nécessaire à la génération

du code source de l'application. On passe d'une vue contemplative des modèles à une vue productive. MDA définit trois niveaux de modèles représentant les niveaux d'abstraction de l'application, le CIM, le PIM et le PSM :

Modèle CIM- Computational Independant Model

Les modèles d'exigence CIM décrivent les besoins fonctionnels de l'application, aussi bien les services qu'elle offre que les entités avec lesquelles elle interagit. Leur rôle est de décrire l'application indépendamment des détails liés à son implémentation. Les CIM peuvent servir de référence pour s'assurer que l'application finie correspond aux demandes des clients.

Modèle PIM- Platform Independant Model

Les modèles PIM sont les modèles d'analyse et de conception de l'application. La phase de conception à cette étape du processus suppose l'application de Design pattern, le découpage de l'application en modules et sous-modules, etc. Le rôle des PIM est de donner une vision structurelle et dynamique de l'application, toujours indépendamment de la conception technique de l'application.

Modèle PM- Platform Model

Rarement utilisé, un PM décrit la structure, et les fonctions techniques relatives à une plateforme d'exécution (systèmes de fichiers, de mémoire, de BDD…) et précise comment les utiliser. Le PM est associé au PIM pour obtenir le PSM.

Modèle PSM- Platform Specific Model

Le PSM est le modèle qui se rapproche le plus du code final de l'application. Un PSM est un modèle de code qui décrit l'implémentation d'une application sur une plateforme particulière, il est donc lié à une plateforme d'exécution.

Code source

Représente le résultat final du processus MDA, le code source est obtenu par génération automatique (partielle ou totale) du code de l'application à partir du PSM. Le code source obtenu peut toujours être enrichi ou modifié manuellement.

3.2.4 Le PDM (Plateform 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 Le passage entre ces différents modèles va se faire par une suite de transformations.

L'OMG veut proposer à terme l'automatisation de ces transformations par des outils logiciel. 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.

4. Les standards de l'OMG

L'OMG a défini des 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. 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 coeur 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 métamodèles 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.

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 termes de classes, d'associations, de paquetages, de types de données. Cela lui permet aussi de pouvoir définir d'autres méta-modèles.

4.2 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 la 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.

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.

4.3 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. 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.

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 les développements 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 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.

4.4 . XMI 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 métamodèles.

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 métadonnées 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.

4.5. HUTN (Human-Usable Textual Notation)

HUTN est un langage automatisable qui se veut plus simple que XMI. 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.

4.6. 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 .

4.7 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.