1

L'architecture MDA

Ce chapitre présente de façon globale l'architecture MDA (Model Driven Architecture) telle que définie par l'OMG (Object Management Group). La suite de l'ouvrage revient en détail sur tous les points introduits ici.

Chacun s'accorde à penser que l'ingénierie logicielle guidée par les modèles est l'avenir des applications. De Bill Gates — *« Modéliser est le futur, et je pense que les sociétés qui travaillent dans ce domaine ont raison »* — à Richard Soley, le directeur de l'OMG, en passant bien entendu par les pionniers du monde UML, comme Graddy Booch, Jim Rumbaugh ou Ivar Jacobson et bien d'autres, tous affirment qu'il est temps d'élaborer les applications à partir de modèles et, surtout, de faire en sorte que ces modèles soient au centre du cycle de vie de ces applications, autrement dit qu'ils soient productifs.

Les modèles

Les modèles offrent de nombreux avantages. Ceux qui pratiquent UML ou tout autre langage de modélisation les connaissent bien. L'avantage le plus important qu'ils procurent est de spécifier *différents niveaux d'abstraction,* facilitant la gestion de la complexité inhérente aux applications.

Les modèles très abstraits sont utilisés pour présenter l'architecture générale d'une application ou sa place dans une organisation, tandis que les modèles très concrets permettent de spécifier précisément des protocoles de communication réseau ou des algorithmes de synchronisation. Même si les modèles se situent à des niveaux d'abstraction différents, il est possible d'exprimer des *relations de raffinement* entre eux. Véritables liens de traçabilité, ces relations sont garantes de la cohérence d'un ensemble de modèles représentant une même application.

La diversité des possibilités de modélisation ainsi que la possibilité d'exprimer des liens de traçabilité sont des atouts décisifs pour gérer la complexité.

Un autre avantage incontestable des modèles est qu'ils peuvent être présentés sous format graphique, facilitant d'autant la communication entre les acteurs des projets informatiques. Les modèles graphiques parmi les plus utilisés sont les modèles relationnels, qui permettent de spécifier la structure des bases de données. La représentation graphique de ces modèles offre un gain significatif de productivité.

Les mauvaises langues prétendent que modéliser est la meilleure façon de perdre du temps puisque, *in fine,* il faut de toute façon écrire du code. De même, au fameux dicton énonçant qu'un bon schéma vaut mieux qu'un long discours, on entend parfois répliquer qu'à un schéma peuvent correspondre plus de mille discours, selon la façon dont on l'interprète.

Ces critiques visent juste en cas d'absence de maîtrise des bonnes pratiques de modélisation, c'est-à-dire de l'ingénierie des modèles. C'est pourquoi il est essentiel d'acquérir de bonnes pratiques de modélisation afin de déterminer comment, quand, quoi et pourquoi modéliser et d'exploiter pleinement les avantages des modèles.

L'OMG a défini MDA (Model Driven Architecture) en 2000 dans cet objectif. L'approche MDA préconise l'utilisation massive des modèles et offre de premières réponses aux comment, quand, quoi et pourquoi modéliser. Sans prétendre être une Bible de la modélisation, répertoriant toutes les bonnes pratiques, elle vise à mettre en valeur les qualités intrinsèques des modèles, telles que pérennité, productivité et prise en compte des plates-formes d'exécution. MDA inclut la définition de plusieurs standards, notamment UML, MOF et XMI.

L'OMG

L'OMG (Object Management Group) est un consortium à but non lucratif d'industriels et de chercheurs, dont l'objectif est d'établir des standards permettant de résoudre les problèmes d'interopérabilité des systèmes d'information *(http://www.omg.org).*

Le principe clé de MDA consiste en l'utilisation de modèles aux différentes phases du cycle de développement d'une application. Plus précisément, MDA préconise l'élaboration de modèles d'exigences (CIM), d'analyse et de conception (PIM) et de code (PSM).

L'objectif majeur de MDA est l'élaboration de modèles pérennes, indépendants des détails techniques des plates-formes d'exécution (J2EE, .Net, PHP ou autres), afin de permettre la génération automatique de la totalité du code des applications et d'obtenir un gain significatif de productivité.

D'autres modèles, comme les modèles de supervision, de vérification ou d'organisation d'entreprise, ne sont pas encore intégrés dans l'approche MDA mais le seront rapidement sans difficulté, compte tenu de son ouverture.

Le modèle d'exigences CIM (Computation Independent Model)

La première chose à faire lors de la construction d'une nouvelle application est bien entendu de spécifier les exigences du client. Bien que très en amont, cette étape doit fortement bénéficier des modèles.

L'objectif est de créer un modèle d'exigences de la future application. Un tel modèle doit représenter l'application dans son environnement afin de définir quels sont les services offerts par l'application et quelles sont les autres entités avec lesquelles elle interagit.

La création d'un modèle d'exigences est d'une importance capitale. Cela permet d'exprimer clairement les liens de traçabilité avec les modèles qui seront construits dans les autres phases du cycle de développement de l'application, comme les modèles d'analyse et de conception. Un lien durable est ainsi créé avec les besoins du client de l'application.

Les modèles d'exigences peuvent même être considérés comme des éléments contractuels, destinés à servir de référence lorsqu'on voudra s'assurer qu'une application est conforme aux demandes du client.

Il est important de noter qu'un modèle d'exigences ne contient pas d'information sur la réalisation de l'application ni sur les traitements. C'est pourquoi, dans le vocabulaire MDA, les modèles d'exigences sont appelés des CIM (Computation Independent Model), littéralement « modèle indépendant de la programmation ».

Avec UML, un modèle d'exigences peut se résumer à un diagramme de cas d'utilisation. Ces derniers contiennent en effet les fonctionnalités fournies par l'application (cas d'utilisation) ainsi que les différentes entités qui interagissent avec elle (acteurs) sans apporter d'information sur le fonctionnement de l'application.

Dans une optique plus large, un modèle d'exigences est considéré comme une entité complexe, constituée entre autres d'un glossaire, de définitions des processus métier, des exigences et des cas d'utilisation ainsi que d'une vue systémique de l'application.

Si MDA n'émet aucune préconisation quant à l'élaboration des modèles d'exigences, des travaux sont en cours pour ajouter à UML les concepts nécessaires pour couvrir cette phase amont.

Le rôle des modèles d'exigences dans une approche MDA est d'être les premiers modèles pérennes. Une fois les exigences modélisées, elles sont censées fournir une base contractuelle variant peu, car validée par le client de l'application. Grâce aux liens de traçabilité avec les autres modèles, un lien peut être créé depuis les exigences vers le code de l'application.

Le modèle d'analyse et de conception abstraite PIM (Platform Independent Model)

Une fois le modèle d'exigences réalisé, le travail d'analyse et de conception peut commencer. Dans l'approche MDA, cette phase utilise elle aussi un modèle.

L'analyse et la conception sont depuis plus de trente ans les étapes où la modélisation est la plus présente, d'abord avec les méthodes Merise et Coad/Yourdon puis avec les méthodes objet Schlear et Mellor, OMT, OOSE et Booch. Ces méthodes proposent toutes leurs propres modèles. Aujourd'hui, le langage UML s'est imposé comme la référence pour réaliser tous les modèles d'analyse et de conception.

Par conception, il convient d'entendre l'étape qui consiste à structurer l'application en modules et sous-modules. L'application des patrons de conception, ou Design Patterns, du GoF (Gang of Four) fait partie de cette étape de conception. Par contre, l'application de patrons techniques, propres à certaines plates-formes, correspond à une autre étape. Nous ne considérons donc ici que la conception abstraite, c'est-à-dire celle qui est réalisable sans connaissance aucune des techniques d'implémentation.

MDA et UML

MDA considère que les modèles d'analyse et de conception doivent être indépendants de toute plate-forme d'implémentation, qu'elle soit J2EE, .Net, PHP, etc. En n'intégrant les détails d'implémentation que très tard dans le cycle de développement, il est possible de maximiser la séparation des préoccupations entre la logique de l'application et les techniques d'implémentation.

UML est préconisé par l'approche MDA comme étant le langage à utiliser pour réaliser des modèles d'analyse et de conception indépendants des plates-formes d'implémentation. C'est pourquoi dans le vocabulaire MDA ces modèles sont appelés des PIM (Platform Independent Model).

Précisons que MDA ne fait que préconiser l'utilisation d'UML et qu'il n'exclut pas que d'autres langages puissent être utilisés. De plus, MDA ne donne aucune indication quant au nombre de modèles à élaborer ni quant à la méthode à utiliser pour élaborer ces PIM.

Quels que soient le ou les langages utilisés, le rôle des modèles d'analyse et de conception est d'être pérennes et de faire le lien entre le modèle d'exigences et le code de l'application. Ces modèles doivent par ailleurs être productifs puisqu'ils constituent le socle de tout le processus de génération de code défini par MDA. La productivité des PIM signifie qu'ils doivent être suffisamment précis et contenir suffisamment d'information pour qu'une génération automatique de code soit envisageable.

Le modèle de code ou de conception concrète PSM (Platform Specific Model)

Une fois les modèles d'analyse et de conception réalisés, le travail de génération de code peut commencer. Cette phase, la plus délicate du MDA, doit elle aussi utiliser des modèles. Elle inclut l'application des patrons de conception techniques.

MDA considère que le code d'une application peut être facilement obtenu à partir de modèles de code. La différence principale entre un modèle de code et un modèle d'analyse ou de conception réside dans le fait que le modèle de code est lié à une plate-forme

d'exécution. Dans le vocabulaire MDA, ces modèles de code sont appelés des PSM (Platform Specific Model).

Les modèles de code servent essentiellement à faciliter la génération de code à partir d'un modèle d'analyse et de conception. Ils contiennent toutes les informations nécessaires à l'exploitation d'une plate-forme d'exécution, comme les informations permettant de manipuler les systèmes de fichiers ou les systèmes d'authentification.

Il est parfois difficile de différencier le code des applications des modèles de code. Pour MDA, le code d'une application se résume à une suite de lignes textuelles, comme un fichier Java, alors qu'un modèle de code est plutôt une représentation structurée incluant, par exemple, les concepts de boucle, condition, instruction, composant, événement, etc. L'écriture de code à partir d'un modèle de code est donc une opération assez triviale.

Pour élaborer des modèles de code, MDA propose, entre autres, l'utilisation de profils UML. Un profil UML est une adaptation du langage UML à un domaine particulier. Par exemple, le profil UML pour EJB est une adaptation d'UML au domaine des EJB. Grâce à ce profil, il est possible d'élaborer des modèles de code pour le développement d'EJB.

Le rôle des modèles de code est principalement de faciliter la génération de code. Ils sont donc essentiellement productifs mais ne sont pas forcément pérennes. L'autre caractéristique importante des modèles de code est qu'ils font le lien avec les plates-formes d'exécution. Cette notion de plate-forme d'exécution est très importante dans MDA car c'est elle qui délimite la fameuse séparation des préoccupations.

Transformation des modèles

Nous venons de passer en revue les trois types de modèles les plus importants pour MDA que sont les CIM, PIM et PSM. Nous avons aussi vu qu'il était important de bien établir les liens de traçabilité entre ces modèles. En fait, MDA établit ces liens automatiquement grâce à l'exécution de transformations des modèles.

Les transformations de modèles préconisées par MDA sont essentiellement les transformations CIM vers PIM et PIM vers PSM. La génération de code à partir des PSM n'est quant à elle pas considérée comme une transformation de modèle à part entière. MDA envisage aussi les transformations inverses : code vers PSM, PSM vers PIM et PIM vers CIM.

Nous ne saurions trop souligner l'importance des transformations de modèles. Ce sont elles qui portent l'intelligence du processus méthodologique de construction d'application. Elles sont stratégiques et font partie du savoir-faire de l'entreprise ou de l'organisation qui les exécute, car elles détiennent les règles de qualité de développement d'applications.

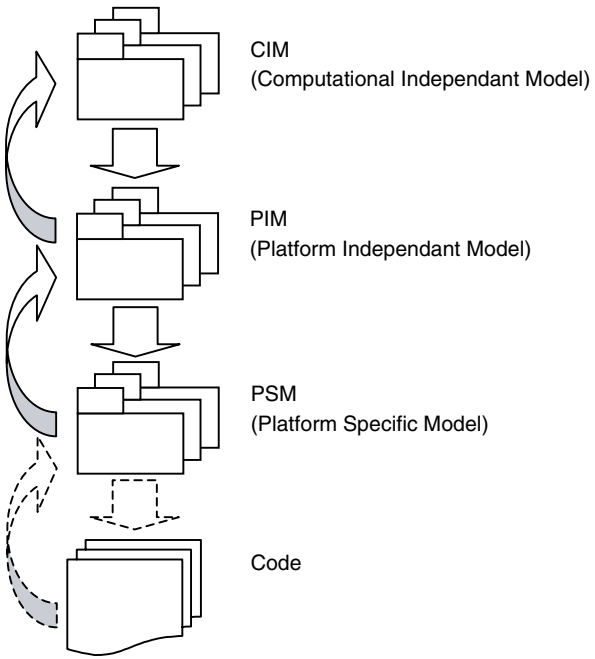
Conscient de cela, MDA préconise de modéliser les transformations de modèles elles-mêmes. Après tout, une transformation de modèles peut être considérée comme une application. Il est dès lors naturel de modéliser ses exigences, son analyse et sa conception et ses modèles de code afin de générer automatiquement le code de la transformation.

Architecture générale de l'approche MDA

La figure 1.1 donne une vue générale de l'approche MDA. Nous constatons que la construction d'une nouvelle application commence par l'élaboration d'un ou de plusieurs modèles d'exigences (CIM). Elle se poursuit par l'élaboration des modèles d'analyse et de conception abstraite de l'application (PIM). Ceux-ci doivent en théorie être partiellement générés à partir des CIM afin que des liens de traçabilité soient établis. Les modèles PIM sont des modèles pérennes, qui ne contiennent aucune information sur les plates-formes d'exécution.

Pour réaliser concrètement l'application, il faut ensuite construire des modèles spécifiques des plates-formes d'exécution. Ces modèles sont obtenus par une transformation des PIM en y ajoutant les informations techniques relatives aux plates-formes. Les PSM n'ont pas pour vocation d'être pérennes. Leur principale fonction est de faciliter la génération de code. La génération de code à partir des modèles PSM n'est d'ailleurs pas réellement considérée par MDA. Celle-ci s'apparente plutôt à une traduction des PSM dans un formalisme textuel.

Figure 1.1
Aperçu global de l'approche MDA



Si la vocation première de l'approche MDA est de faciliter la création de nouvelles applications, elle procure en outre de nombreux avantages pour la rétroconception d'applications existantes. C'est pourquoi les transformations inverses — PSM vers PIM et PIM vers

CIM — sont aussi identifiées. Ces transformations n'en sont toutefois encore qu'au stade de la recherche.

Technologies de modélisation

Nous venons de voir la primauté des modèles dans l'approche MDA. C'est cela qui la différencie principalement des approches classiques de génie logiciel telles que OMT (Object Management Technique), OOSE (Object Oriented Software Engineering) ou BCF (Business Component Factory), qui placent les objets ou les composants au premier plan.

Nous avons vu aussi que MDA préconisait l'élaboration de différents modèles, modèle d'exigences CIM, modèle d'analyse et de conception abstraite PIM et modèle de code et de conception concrète PSM. En réalité, MDA est beaucoup plus général et préconise de modéliser n'importe quelle information nécessaire au cycle de développement des applications. Nous pouvons donc trouver des modèles de test, de déploiement, de plate-forme, etc.

Afin de structurer cet ensemble de modèles, MDA définit la notion de formalisme de modélisation.

Le formalisme de modélisation MOF (Meta Object Facility)

Un formalisme de modélisation est un langage qui permet d'exprimer des modèles. Chaque modèle est donc exprimé dans un certain formalisme de modélisation. Les modèles d'exigences ont leur propre formalisme, qui est différent du formalisme permettant l'expression des modèles d'analyse et de conception abstraite. Rappelons d'ailleurs que le formalisme préconisé pour l'expression des modèles d'analyse et de conception est UML.

Un formalisme définit les concepts ainsi que les relations entre concepts nécessaires à l'expression de modèles. Nous reviendrons largement sur ce sujet dans la suite de l'ouvrage. Le formalisme UML d'expression des modèles d'analyse et de conception définit, entre autres, les concepts de classe et d'objet ainsi que la relation précisant qu'un objet est l'instance d'une classe.

Les notions de modèles et de formalisme de modélisation ne sont pas suffisantes pour mettre en œuvre MDA. Nous avons vu qu'il était aussi très important de pouvoir exprimer des liens de traçabilité ainsi que des transformations entre modèles. Pour pouvoir faire cela, il est indispensable de travailler non pas uniquement au niveau des modèles, mais aussi au niveau des formalismes de modélisation. Il faut exprimer des liens entre les concepts des différents formalismes. Par exemple, il faut pouvoir exprimer que le concept de classe UML doit être transformé dans le concept de classe Java.

Pour cela, MDA préconise de modéliser les formalismes de modélisation eux-mêmes. L'objectif est de disposer d'un formalisme permettant l'expression de modèles de formalismes de modélisation. Dans le jargon de MDA, un tel formalisme est appelé un

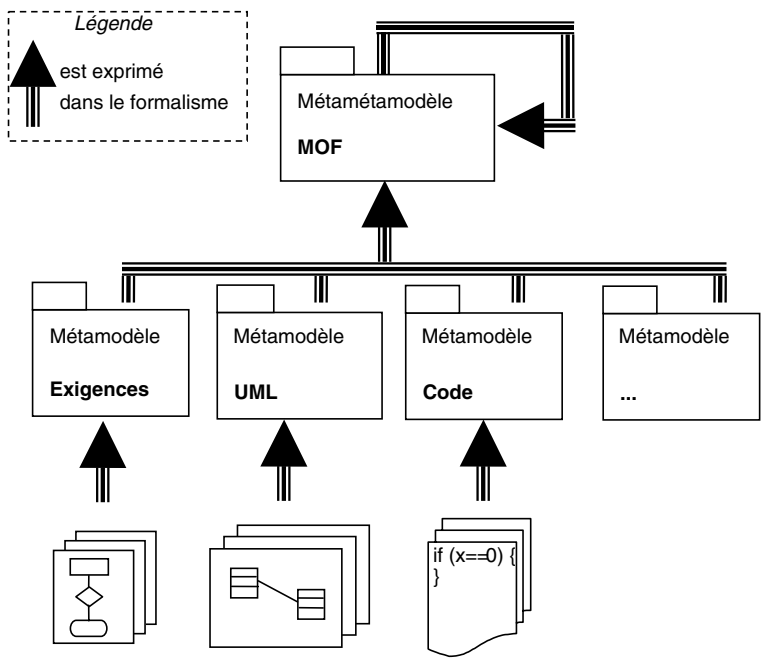
métaformalisme, et les modèles qu'il permet d'exprimer sont appelés des *métamodèles.* Nous pouvons donc faire une analogie entre métamodèles et formalismes de modélisation.

La question qui se pose alors est de savoir s'il est possible de construire un *métaméta-formalisme* permettant d'exprimer des métaformalismes, voire un *métamétaméta…-formalisme,* et ainsi de suite. MDA répond que seuls trois niveaux sont nécessaires : le modèle, le formalisme de modélisation, aussi appelé métamodèle, et le métaformalisme. Pour enrayer la montée dans les niveaux méta, MDA fait en sorte que le métaformalisme soit à lui-même son propre formalisme. Un métamétaformalisme n'est dès lors plus nécessaire.

Dans MDA, il n'existe qu'un seul métaformalisme, le MOF (Meta Object Facility). Aussi appelé *métamétamodèle,* le MOF permet d'exprimer des formalismes de modélisation, ou métamodèles, permettant eux-mêmes d'exprimer des modèles.

La figure 1.2 illustre ces concepts de modèle, de formalisme de modélisation (métamodèle) et de métaformalisme (métamétamodèle). Dans la suite de l'ouvrage, nous utilisons plutôt le vocabulaire MDA, c'est-à-dire modèle, métamodèle et métamétamodèle.

Figure 1.2
Modèle, métamodèle (formalisme de modélisation) et métamétamodèle (métaformalisme)



Le chapitre 2 développe en détail les caractéristiques du MOF. Il précise les relations entre modèles, métamodèles et métamétamodèle et explique comment élaborer un méta-modèle.

Métaformalisme

L'idée d'un métaformalisme n'est pas neuve en soi pour qui est habitué à manipuler des grammaires de langage. Dans le monde XML, un formalisme est représenté sous forme de schéma XML. Un schéma XML définit les concepts ainsi que les relations entre concepts permettant l'expression de documents XML, et les schémas XML sont eux-mêmes des documents XML. Cela n'est possible que parce qu'il existe un schéma des schémas XML, autrement dit un métaformalisme. Dans le monde XML, le schéma des schémas XML est aussi le dernier niveau car il est à lui-même son propre schéma.

La même analogie peut être faite entre les langages de programmation et la BNF (Bachus Naur Form), le langage utilisé pour exprimer les syntaxes des langages. Dans pratiquement tous les manuels de référence des langages (Java, Ada, C++, etc.), on trouve une définition BNF. Par exemple, un programme Java est exprimé dans le formalisme Java qui est défini par une grammaire en BNF, cette dernière se définissant elle-même.

Le métamodèle UML

La section précédente a introduit les notions de modèle, métamodèle et métamétamodèle. Nous avons vu que MDA préconisait l'utilisation de différents modèles et que chacun de ces modèles était conforme à un métamodèle, lui-même conforme au métamétamodèle MOF.

L'univers de MDA est donc partitionné par un ensemble de métamodèles. Chacun de ces métamodèles est dédié à une étape particulière de MDA (exigences, analyse et conception, code). D'un point de vue purement théorique, MDA n'impose aucune contrainte quant à l'utilisation de tel ou tel métamodèle pour chacune de ces étapes. Il n'en va pas de même pour la réalisation, puisque MDA préconise actuellement l'utilisation du métamodèle UML pour l'étape d'analyse et de conception abstraite et conseille de recourir aux profils UML pour élaborer des modèles de code et de conception concrète à partir de modèles UML.

UML pour les PIM

Le métamodèle UML est le métamodèle le plus connu de l'approche MDA. Le sujet de cet ouvrage n'étant pas UML, nous nous contenterons de préciser son rôle dans MDA.

Le métamodèle UML définit la structure que doit avoir tout modèle UML. Il précise, par exemple, qu'une classe UML peut avoir des attributs et des opérations. Le chapitre 3 donne une présentation plus détaillée du métamodèle UML.

D'un point de vue plus conceptuel, le métamodèle UML permet d'élaborer des modèles décrivant des applications objet. UML définit plusieurs diagrammes permettant de décrire les différentes parties d'une application objet. Par exemple, les diagrammes de classes permettent de décrire la partie statique des applications alors que les diagrammes de séquences ou d'activités permettent d'en définir la partie dynamique.

Les modèles UML sont indépendants des plates-formes d'exécution. Ils sont utilisés pour décrire aussi bien les applications Java que les applications C# ou PHP. Plusieurs outils du marché proposent des générateurs de code vers ces différents langages de programmation.

Pour toutes ces raisons, il est évident que le métamodèle UML constitue le métamodèle idéal pour l'élaboration des PIM (Platform Independent Model). Rappelons qu'un PIM est un modèle d'analyse et de conception d'une application et qu'il se doit d'être indépendant d'une plate-forme d'exécution.

UML pour les PSM

Le métamodèle UML définit la notion de *profil UML*. Un profil UML permet d'adapter UML à un domaine particulier. Par exemple, le profil UML pour EJB permet d'adapter UML au domaine des EJB. Les modèles UML réalisés selon ce profil ne sont plus vraiment des modèles UML mais plutôt des modèles d'application EJB.

Les profils UML ciblant des plates-formes d'exécution permettent, par définition, d'adapter UML à des plates-formes d'exécution. Le point intéressant à souligner dans un contexte MDA est que les modèles réalisés selon ces profils ne sont plus des modèles indépendants des plates-formes d'exécution mais, au contraire, des modèles dépendants de ces plates-formes. Ces modèles ne sont donc plus des PIM mais des PSM (Platform Specific Model).

Grâce aux profils ciblant des plates-formes d'exécution, il est possible d'utiliser UML pour élaborer des PSM. Ces PSM sont bien des modèles de code et ne peuvent donc être confondus avec du code. Par contre, étant donné qu'ils sont liés aux plates-formes d'exécution, ils facilitent grandement la dernière étape du MDA, qui est la génération de code.

MDA conseille l'utilisation de profils UML pour l'élaboration de PSM car cela a le mérite de faciliter les transformations PIM vers PSM puisque PIM et PSM sont tous deux des modèles UML.

L'autre approche possible, qui est aussi conseillée par MDA, est de définir les métamodèles propres aux plates-formes. Cette autre approche présente l'inconvénient de ne pas faciliter les transformations PIM vers PSM mais a l'avantage d'offrir une plus grande liberté dans l'expression des plates-formes.

Les profils UML pour EJB et UML pour CORBA ont été standardisés par l'OMG. D'autres profils, tels que UML pour Java ou UML pour C#, ne sont pas standardisés mais sont disponibles dans des produits tels que Rational Software Modeler ou Softeam MDA Modeler *(voir le chapitre 8)*.

UML et CIM

L'élaboration des CIM avec UML fait débat. Rappelons qu'un CIM est un modèle d'exigences d'une application. Ceux qui connaissent UML pourraient argumenter que les diagrammes de cas d'utilisation UML peuvent être considérés comme des CIM. Ces diagrammes permettent en effet de décrire les fonctionnalités qu'offre une application à son environnement. Il existe cependant d'autres approches d'expression des besoins, telles que celles supportées par DOORS ou Rational Requisite Pro. Voilà pourquoi MDA ne fait aucune préconisation quant à l'utilisation d'UML ou non pour exprimer les CIM.

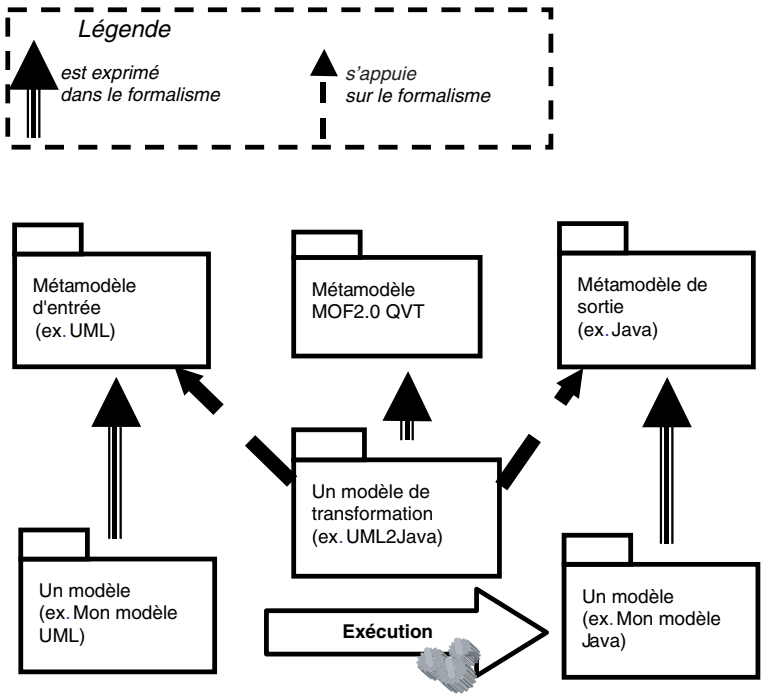
Modélisation de la transformation de modèles avec QVT

Nous avons déjà mentionné le fait que les transformations de modèles étaient au cœur de MDA et qu'il était important de les modéliser. Pour ce faire, l'OMG a élaboré le standard MOF2.0 QVT (Query, View, Transformation). Ce standard définit le métamodèle permettant l'élaboration de modèles de transformation. La transformation d'UML vers la plate-forme Java, par exemple, est élaborée sous la forme d'un modèle de transformation conforme au métamodèle MOF2.0 QVT.

Le standard MOF2.0 QVT cible les transformations modèle vers modèle. Pour simplifier, disons qu'un modèle de transformation est considéré comme une fonction prenant un modèle en entrée et rendant un modèle en sortie. Étant donné que les modèles d'entrée et de sortie disposent chacun de leur métamodèle, on parle aussi de métamodèle d'entrée et de métamodèle de sortie du modèle de transformation.

La figure 1.3 illustre les relations existantes entre le métamodèle MOF2.0 QVT, un modèle de transformation (UML2Java), son métamodèle d'entrée (UML) et de sortie (Java) et une exécution du modèle de transformation.

Figure 1.3
Transformation de modèles



Les métamodèles d'entrée et de sortie d'un modèle de transformation peuvent être identiques. Un modèle de transformation peut bien évidemment transformer des modèles UML en… modèles UML.

Les profils UML qui peuvent être utilisés pour élaborer les modèles PSM sont considérés comme des métamodèles à part entière. Par exemple, un modèle de transformation peut avoir le métamodèle UML comme métamodèle d'entrée et le profil UML pour EJB comme métamodèle de sortie. Un tel modèle de transformation peut spécifier une transformation UML vers EJB.

Il n'y a pas vraiment de consensus sur le statut de la génération de code. Celle-ci a beau être considérée comme une transformation modèle vers modèle par certains, cela ne fait pas l'unanimité. Il est souvent plus pratique de s'exprimer sous forme textuelle pour manipuler le code que sous forme de métamodèle. Un standard en cours de construction actuellement à l'OMG permettra d'ailleurs de définir un langage spécifiquement dédié à la génération de code à partir de modèles.

Le chapitre 7 détaille les transformations de modèles dans MDA et présente plus précisément les différentes approches possibles pour réaliser une transformation de modèle et les illustre à partir d'un même exemple.

Liens vers XML et Java avec XMI, JMI et EMF

Les modèles sont des entités abstraites qui n'ont pas besoin de représentation informatique pour exister. MDA utilisant les modèles à des fins de productivité, il est cependant nécessaire qu'ils disposent de représentations concrètes afin de pouvoir être manipulés informatiquement.

MDA définit deux façons différentes de représenter les modèles : soit sous forme de documents textuels, soit sous forme d'objets de programmation. La représentation textuelle est plus adaptée au stockage des modèles sur disque dur ou aux échanges de modèles entre applications tandis que la représentation objet est plus adaptée à la manipulation informatique (transformation, exécution, validation, etc.).

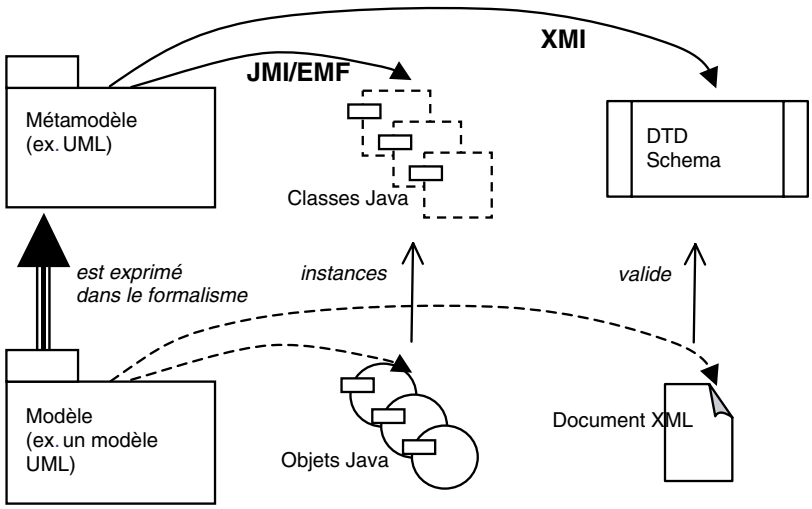
Les standards et frameworks MDA qui définissent la façon de représenter informatiquement les modèles sont XMI, JMI et EMF. Le standard XMI (XML Metadata Interchange) définit la façon de représenter les modèles sous forme de document XML. Le standard JMI (Java Metadata Interface) et le framework EMF (Eclipse Modeling Framework) définissent la façon de représenter les modèles sous forme d'objets Java. EMF, par exemple, permet la manipulation des modèles dans la plate-forme Eclipse.

XMI, JMI et EMF fonctionnent selon le même principe. Que ce soit en XML ou en Java, un format de représentation se définit par sa structure. Définir un format de représentation XML se fait en construisant une DTD ou un schéma XML. Définir un format de représentation objet se fait en construisant un ensemble de classes Java.

Le principe de fonctionnement de XMI, JMI et EMF est de générer automatiquement la structure des formats de représentation des modèles à partir de leur métamodèle. L'idée est de tirer parti de l'analogie qui existe entre la relation entre un modèle et son métamodèle et la relation qui existe entre un document XML et sa DTD ou entre des objets et leur classe.

La figure 1.4 illustre ce principe de fonctionnement. Nous constatons que XMI, JMI et EMF définissent des règles permettant de générer automatiquement les structures des formats de représentation de modèles à partir de leur métamodèle. Par exemple, XMI appliqué à UML permet la génération automatique d'une DTD permettant de représenter les modèles UML sous forme de documents XML. De la même manière, JMI et EMF permettent la génération automatique de classes Java permettant de représenter les modèles UML sous forme d'objets Java.

Figure 1.4
Principe de fonctionnement des standards XMI et JMI



Grâce aux standards XMI et JMI et au framework EMF, il est possible de représenter informatiquement tout modèle. XMI concrétise la pérennité des modèles en ce qu'il offre un format de représentation XML. JMI et EMF sont les socles opérationnels de MDA en ce qu'ils permettent la construction d'opérations de productivité sur les modèles.

Précisons qu'il existe des passerelles entre ces deux standards et qu'il est possible de passer d'une représentation à une autre sans difficulté.

Le chapitre 5 présente en détail les principes de fonctionnement du standard XMI illustrés par un exemple, tandis que le chapitre 6 se penche sur JMI et EMF et explique à partir d'un exemple comment manipuler un modèle à l'aide d'un programme Java.

L'étude de cas PetStore

Les sections précédentes ont présenté l'approche MDA dans sa globalité ainsi que les technologies utilisées. Afin d'illustrer concrètement MDA, nous l'avons appliqué à une étude de cas.

L'étude de cas retenue est la fameuse application Java PetStore des BluePrints de Sun. PetStore est une application relativement classique de commerce électronique. Un utilisateur peut naviguer sur un site Web pour voir quels sont les différents articles à la vente afin d'ajouter dans son panier ceux qu'il désire acheter. Quand il le décide, il peut passer commande et régler tous les articles qu'il a mis dans son panier. La figure 1.5 illustre la page d'accueil de cette application.

Nous avons choisi PetStore car c'est une application très documentée sur le Web. Même si l'objectif initial de cette application était de servir à tester les serveurs d'applications J2EE, PetStore est aujourd'hui utilisée pour des objectifs beaucoup plus étendus, tels que la comparaison .Net, Java et PHP. Grâce à cette documentation présente sur le Web, nous pouvons ne pas trop nous attarder sur la description de PetStore et nous concentrer sur notre objectif, qui est d'illustrer l'approche MDA.

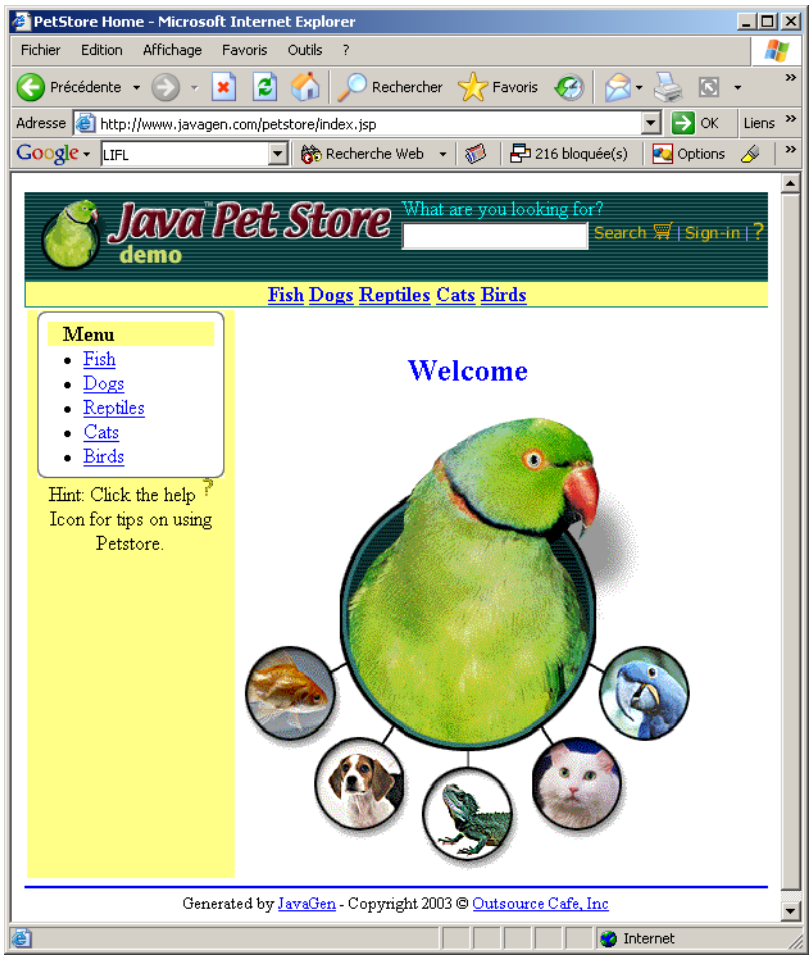


Figure 1.5
Page d'accueil du site Web de PetStore

Pour illustrer comment appliquer l'approche MDA et souligner les avantages concrets qu'elle apporte, nous allons détailler les différents modèles et transformations de modèles nécessaires à la construction de cette application selon les principes MDA.

Nous avons fait le choix de présenter les modèles CIM et PIM de cette application sous forme de modèles UML. Nous préférons utiliser l'approche qui consiste à considérer que les CIM peuvent être élaborés en utilisant UML car cela facilite la présentation de l'approche MDA sur une application de la taille de PetStore. Le seul modèle pérenne de PetStore est donc un modèle UML qui conjugue CIM et PIM.

Conformément à l'approche MDA, ce PIM de PetStore a dû être transformé vers différentes plates-formes d'exécution. Nous avons sélectionné les plates-formes J2EE/EJB et PHP. Le fait d'avoir choisi deux plates-formes nous permet de mettre en lumière les solutions qu'offre MDA pour résoudre la séparation des préoccupations entre le métier et la technique d'une application.

Le travail le plus important pour mettre en œuvre l'approche MDA réside dans l'élaboration du PIM et dans la définition des transformations PIM vers PSM. Pour PetStore, nous avons élaboré les transformations UML vers J2EE/EJB et UML vers PHP permettant de transformer le modèle PIM de PetStore vers les modèles PSM correspondant à ces plates-formes.

Concernant ces plates-formes, nous avons fait le choix d'utiliser les deux techniques possibles, à savoir l'utilisation de profils UML et de métamodèles MOF. Le choix d'un profil pour EJB et d'un métamodèle pour PHP permet de bien comprendre les différences existant entre ces deux approches préconisées par MDA pour représenter les plates-formes.

Après avoir transformé les PIM vers des PSM, nous effectuerons manuellement différentes opérations de raffinement sur les PSM afin qu'ils puissent servir à la génération de code. En effet, les transformations PIM vers PSM ne permettant pas d'obtenir un modèle directement exploitable, il est nécessaire de raffiner encore ce modèle afin d'effectuer la dernière opération, qui est la génération de code.

Pour finir, nous avons généré le code des deux PSM obtenus. Cette dernière étape nous permet d'illustrer les solutions qu'offre MDA pour s'abstraire des langages de programmation. Elle nous permet en outre de clarifier la portée de MDA en précisant ce qu'il est possible de faire aujourd'hui avec MDA et ce qu'il n'est pas possible de faire.

La figure 1.6 illustre les différentes étapes de MDA réalisées sur l'étude de cas PetStore, notamment les efforts d'élaboration d'un PIM, la transformation de ce PIM en plusieurs PSM (Java et PHP) et la génération de code à partir de ces PSM.

L'étude de cas PetStore nous permettra de mesurer la taille du travail à fournir pour passer vers une autre plate-forme, ici .Net.

Le chapitre 12 et dernier de l'ouvrage donne tous les détails de la réalisation de PetStore selon les principes de MDA.

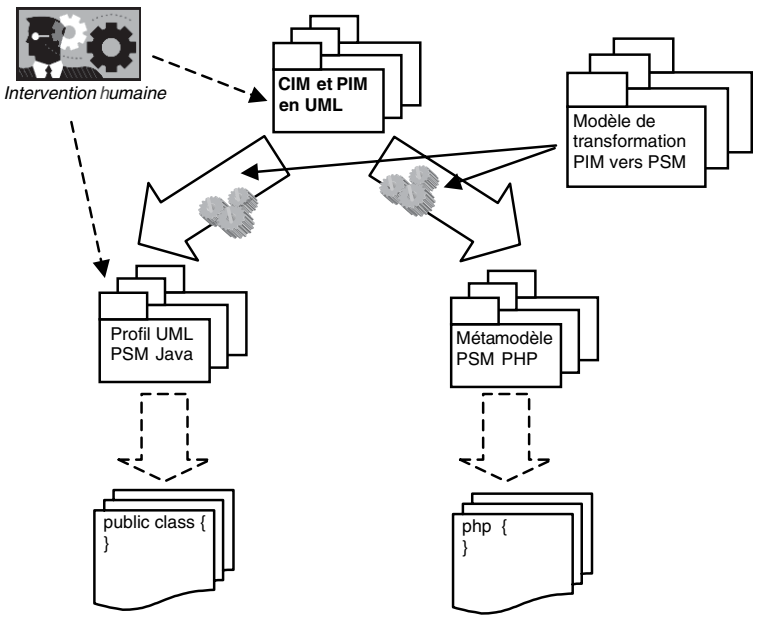


Figure 1.6
Application de MDA à l'étude de cas PetStore

Avantages attendus de MDA

Maintenant que nous avons introduit l'approche MDA et que nous avons brièvement décrit la façon dont nous l'illustrerons sur une étude de cas, prenons un peu de recul afin de mesurer pleinement les avantages attendus de cette approche.

Commençons par rappeler brièvement ce qui s'est passé ces dernières années dans le domaine de la construction des applications réparties.

Les premiers travaux effectués à l'OMG pour faciliter la construction et la maintenance des applications réparties ont porté sur l'élaboration de la spécification CORBA. L'objectif de CORBA était de fournir un environnement standard et ouvert permettant à tout type d'application d'interopérer. À l'époque (1990), il fallait résoudre tous les problèmes d'interopérabilité en standardisant les communications réparties et les interfaces des différentes entités composant l'application répartie.

Pour diverses raisons, CORBA n'a pas été un franc succès. D'autres intergiciels ont à leur tour tenté de résoudre le problème de l'interopérabilité en offrant toujours plus de services aux concepteurs d'applications (EJB, DCOM, Web Services).

De cette succession d'intergiciels est né le *paradoxe des intergiciels.* En effet, les intergiciels qui ont été initialement conçus pour faire face à la complexité des applications réparties ont finalement apporté beaucoup plus de complexité qu'ils n'en ont enlevée. Le problème n'est pas tant leur propre complexité, qui est inévitable, que le fait que les

applications qui les utilisent dépendent fortement des services offerts par ces intergiciels. De ce fait, changer d'intergiciel devient un cauchemar tant l'application est engluée dans l'intergiciel. Certains vont jusqu'à appeler *spaghettiware* les applications réparties qui utilisent les intergiciels. L'évolution d'une application a un coût prohibitif, car il faut inévitablement plonger en son sein pour couper tous les liens avec les intergiciels que l'on veut changer.

Pour résoudre ce problème, la solution envisagée a consisté à tout mettre en œuvre pour assurer de manière industrielle la fameuse séparation des préoccupations entre le métier des applications et la technique des intergiciels. De là est né MDA.

Les trois avantages recherchés étaient alors les suivants :

- La pérennité des savoir-faire, afin de permettre aux entreprises de capitaliser sur leur métier sans avoir à se soucier de la technique.
- Les gains de productivité, afin de permettre aux entreprises de réduire les coûts de mise en œuvre des applications informatiques nécessaires à leur métier.
- La prise en compte des plates-formes d'exécution, afin de permettre aux entreprises de bénéficier des avantages des plates-formes sans souffrir d'effets secondaires.

Pérennité des savoir-faire

Le métier des entreprises n'évolue que faiblement par rapport aux technologies qu'elles utilisent pour construire leurs applications informatiques. Fort de ce constat, il est évident que séparer les aspects métier des aspects techniques lors de la construction d'une application est une bonne pratique. Cela permet de rendre les spécifications métier pérennes, indépendamment des spécifications techniques.

L'avantage le plus important qu'offre MDA est précisément celui de la pérennité des spécifications métier. Son ambition est de faire en sorte que les PIM aient une durée de vie de plus de dix ans.

Nous comprenons mieux pourquoi MDA s'appuie si fortement sur la modélisation des spécifications métier. Par nature, les modèles sont des entités facilement pérennes, car ils permettent de représenter des informations à différents niveaux d'abstraction tout en masquant les détails inutiles. C'est donc la modélisation des spécifications métier qui permet de les rendre pérennes.

Le point le plus important pour rendre des modèles métier pérennes est le formalisme utilisé pour les élaborer. Ce formalisme doit être particulièrement stable et sa sémantique largement partagée par les personnes qui utiliseront les modèles.

Sur ce point, remarquons que MDA a pris ses précautions. Tout d'abord, MDA préconise l'utilisation du langage UML. Fruit de plusieurs années de travail collaboratif des différents acteurs du domaine, nous pouvons considérer qu'il est stable et que sa sémantique est largement partagée. De plus, l'OMG ne fait que préconiser l'utilisation d'UML. Il n'est donc pas lié définitivement avec ce langage. Si, demain, un nouveau langage apparaît

pour remplacer UML, il sera toujours possible de transformer les modèles UML vers ce nouveau langage. Les modèles PIM sont donc nécessairement pérennes.

Un autre critère important de pérennité des modèles est le rôle qu'ils jouent dans le cycle de vie de l'application. Rien ne sert de rendre un modèle pérenne si celui-ci n'est d'aucune utilité. N'est pérenne que l'information utile. Cela contredit l'opinion trop répandue selon laquelle seul le code est utile. Certes le code contient l'information nécessaire et suffisante pour faire en sorte que l'application tourne, mais est-il pour autant la seule source d'information utile ?

À cet égard, MDA propose une approche assez novatrice, consistant à faire remonter l'information utile dans les modèles. Le code n'est dès lors considéré que comme une représentation textuelle interprétable par les machines de l'information utile. Dans MDA, l'information utile est localisée pour l'essentiel dans les PIM et, dans une moindre mesure, dans les transformations PIM vers PSM. Insistons une nouvelle fois sur l'importance des liens de traçabilité entre modèles afin que l'approche soit réaliste.

Abstraction de l'information utile

L'approche qui vise à faire remonter en abstraction l'information utile a déjà été utilisée plusieurs fois dans l'histoire de l'informatique. Nous pouvons considérer que cette même approche a été utilisée pour faire remonter dans les langages de programmation tels que le C ou le Pascal l'information utile qui était contenue dans le code assembleur. Nous pourrions même considérer que cette approche a été utilisée pour faire remonter dans le code assembleur l'information qui était contenue dans les cartes perforées.

En ce qui concerne les technologies utilisées par MDA, la pérennité se retrouve quasiment dans tous les standards. Le standard qui porte en lui le plus de pérennité est vraisemblablement UML, qui permet l'expression des PIM. MOF, qui permet d'exprimer les métamodèles, est également gage de pérennité, car c'est grâce à lui que s'accomplit l'indépendance à l'égard des langages de modélisation.

MOF2.0 QVT est lui aussi central pour la pérennité puisque, sans transformation, il ne serait pas possible de faire remonter l'information utile au niveau des PIM. XMI complète ce socle pérenne en permettant de représenter les modèles sous forme de documents XML et en assurant ainsi un stockage « neutre » des modèles.

L'aboutissement de tous ces standards permettra, par exemple, aux développeurs de s'approprier les modèles UML sous une forme de stockage en un « XML neutre », indépendant des outils de modélisation.

Comme le principal apport de MDA est de pérenniser les modèles PIM sur des durées d'environ quinze ans, il lui est souvent reproché de ne concerner que les grosses applications. S'il est vrai que MDA est beaucoup plus profitable à ces dernières qu'aux petites et moyennes, ses principes et technologies n'en restent pas moins intrinsèquement indépendants de la taille des applications. Même des applications constituées de seulement une ou deux classes tirent un bénéfice de l'élaboration d'un modèle UML, lequel peut ensuite être transformé automatiquement vers un modèle de code lié à une plate-forme (CORBA ou J2EE/EJB, par exemple) puis *in fine* vers du code Java ou C#.

Plusieurs outils ciblent d'ailleurs en partie ce marché des petites applications, notamment Poseidon.

Gains de productivité

Il est toujours important pour une entreprise d'augmenter sa productivité afin de rester compétitive. Une façon bien connue d'augmenter la productivité est d'automatiser certaines étapes de production. C'est ce que fait MDA en automatisant les transformations de modèles.

Ce deuxième avantage apporté par MDA de gains substantiels de productivité grâce à l'automatisation des transformations de modèles est très novateur. Avant MDA, les modèles n'étaient que des dessins que l'on accrochait fièrement aux murs de son bureau. Ils permettaient au mieux de communiquer avec les différents membres de l'équipe. À peine imprimés, ces modèles étaient bien souvent en total décalage avec le code de l'application et devenaient inutilisables. Les qualités attendues des modèles relevaient essentiellement de la communication (clarté, lisibilité, simplicité, etc.).

MDA vise une forte intégration des modèles dans le processus de production de l'application puisque les modèles sont directement utilisés pour générer le code de l'application. Les qualités attendues des modèles sont donc tout autres. MDA a besoin de modèles précis, complets et bien définis pour pouvoir les transformer. Le statut des modèles change dès lors du tout au tout, et ils deviennent un élément de production des applications, au même titre que le code ou le binaire.

Pour effectuer ce changement de statut, MDA définit la nature du support informatique (stockage, API) des modèles. Nous avons vu que les standards JMI et le framework EMF permettaient de fournir des représentations concrètes pour les modèles. Ces standards, fondés sur le MOF, permettent d'automatiser les opérations effectuées sur les modèles. Le standard MOF2.0 QVT, par exemple, peut être vu comme un standard qui utilise JMI et EMF en proposant un langage de haut niveau pour exprimer les transformations de modèles.

Les principes à l'œuvre dans ces standards préexistaient à MDA, et certains outils tels que Objecteering ou Rational Rose proposent depuis le début des années 90 d'informatiser les modèles. Ces outils ont défini leur propre format de représentation concret et proposent aux utilisateurs des moyens pour programmer et donc automatiser les opérations sur les modèles.

Ces outils ont été des précurseurs à l'époque de leur conception. Les opérations qu'ils proposent sont principalement de la génération de code et de la génération de documentation sur les modèles UML. Aujourd'hui, de plus en plus d'outils proposent différentes opérations sur différents modèles, comme des opérations de génération et d'exécution de tests, de validation et de vérification, de simulation, d'exécution, de supervision, etc. Ces opérations complexes sont de plus en plus automatisées, engendrant des gains de productivité tout au long des étapes du cycle de vie des applications.

Ce dernier point soulève encore des questions pour savoir quel est le domaine d'application couvert par MDA. À voir se multiplier l'automatisation des opérations sur les modèles, nous pourrions en conclure que MDA est une sorte d'usine à gaz, dans laquelle entre un modèle pour y subir une multitude d'opérations afin qu'en sorte automatiquement une application neuve. En réalité, MDA ne fait aucune préconisation, autre que CIM, PIM et PSM, quant à l'ensemble des opérations à utiliser. Il est donc possible d'utiliser un nombre incalculable d'opérations sur les modèles et de clamer que nous suivons l'approche MDA, alors même que MDA ne préconise pas encore de méthode susceptible de définir clairement quelles sont les bonnes et les mauvaises pratiques.

MDA est une approche, et non une méthode, fondée sur l'utilisation de modèles pour assurer la séparation des préoccupations. Pour ce faire, MDA utilise les modèles comme des éléments productifs. Appliquer réellement MDA nécessite inévitablement de définir une méthode. Une telle méthode, propre à chaque entreprise, permet de bien appliquer MDA dans le contexte de l'entreprise. Aujourd'hui, plusieurs entreprises sont en train de définir leur propre méthode MDA. Toutes suivent l'approche MDA mais diffèrent dans leur degré d'automatisation et dans la diversité des opérations qu'elles proposent.

Dans notre étude de cas, nous utilisons une méthode relativement simpliste, qui consiste à définir en premier lieu les CIM et PIM de l'application sous forme de modèle UML puis à transformer automatiquement ces PIM en PSM, à appliquer manuellement quelques modifications sur ces PSM et à générer automatiquement le code. Cette méthode MDA n'est que partiellement automatisée et donc partiellement productive. Elle permet néanmoins d'obtenir des gains significatifs de productivité.

Prise en compte des plates-formes d'exécution

Les applications réparties d'aujourd'hui s'exécutent de plus en plus sur des plates-formes hétérogènes. Il n'est pas rare de voir une application s'exécuter en partie sur une plate-forme Java et en partie sur une plate-forme .Net, par exemple. Il est important pour les entreprises de dompter cette offre hétérogène afin de tirer parti du meilleur de chaque plate-forme.

MDA prend en compte les plates-formes en intégrant leur description aux modèles lors des transformations PIM vers PSM. Il est dès lors possible d'établir une stratégie propre à l'intégration de telle ou telle plate-forme.

Le troisième avantage qu'offre MDA est d'intégrer le support des plates-formes directement dans les modèles. Cet avantage est peut-être le plus déterminant. L'objectif premier de MDA est de permettre de migrer facilement d'une plate-forme à une autre. Grâce à ce support des plates-formes dans les modèles, nous pouvons penser, d'une part, que la migration de plate-forme sera beaucoup moins coûteuse et, d'autre part, qu'il sera possible de capitaliser le savoir-faire sur les plates-formes. Ces perspectives sont toutefois à conjuguer au futur car elles ne sont pas encore pleinement accomplies à l'heure actuelle.

La prise en compte des plates-formes par MDA se fait en deux endroits : dans les PSM et dans les transformations PIM vers PSM.

Concernant les PSM, souvenons-nous qu'un PSM est un modèle qui dépend fortement d'une plate-forme. Son métamodèle contient toute l'information relative à la plate-forme. Rappelons que ce métamodèle peut être un profil UML tel que ceux que nous avons déjà présentés. Pour l'instant, les métamodèles ou profils de plates-formes ne font qu'énumérer les concepts de la plate-forme nécessaires à connaître pour pouvoir l'utiliser. Par exemple, le métamodèle de la plate-forme J2EE contient les concepts de classe Java, de page JSP, de servlet, d'EJB, etc. Un modèle conforme à ce métamodèle représente une application J2EE.

Il n'existe pas encore de réel métamodèle de plate-forme. Un tel métamodèle permettrait d'élaborer des modèles représentant des plates-formes déployées et d'effectuer toutes sortes d'opérations sur ces modèles, telles que des tests de montée en charge ou de la supervision.

Les transformations PIM vers PSM contiennent toute l'information permettant à un PIM d'utiliser une plate-forme déterminée. Au début de MDA, l'OMG pensait qu'il appartenait aux constructeurs de plate-forme de fournir ces transformations. L'idée était qu'après la sortie d'une nouvelle plate-forme, les transformations permettant de l'exploiter seraient disponibles, engendrant un coût de migration quasi nul.

Peu après la sortie de MDA, de nombreux avis contraires se sont fait entendre, estimant que ce n'était pas aux constructeurs de plate-forme de fournir ces transformations mais plutôt aux utilisateurs, seuls à même de savoir comment exploiter une plate-forme dans leur contexte. Ce deuxième courant s'est avéré beaucoup plus intéressant et beaucoup plus réaliste, chacun capitalisant sa façon d'exploiter les plates-formes.

Ce courant inclut tous ceux qui réalisent les frameworks techniques d'entreprise. Ces frameworks, fondés sur les plates-formes existantes, permettent d'adapter les plates-formes aux besoins des entreprises. Il existe, par exemple, des frameworks techniques d'entreprise sur J2EE, qui empêchent l'utilisation de certaines fonctionnalités de la plate-forme, jugées peu viables pour le contexte de l'entreprise, comme les EJB Entity Container Managed Persistency dans certains frameworks. L'utilisation des frameworks par les applications se retrouve alors dans les transformations PIM vers PSM.

Ce caractère multiplate-forme est certainement l'avantage le plus apprécié par les utilisateurs de MDA. MDA est au demeurant la seule approche qui considère que les applications ne seront jamais plus déployées sur une unique plate-forme.

Synthèse

Ce chapitre a présenté MDA dans ses grandes lignes. Nous avons vu que MDA utilisait fortement les modèles et avons indiqué les différents types de modèles employés par MDA, à savoir CIM, PIM et PSM.

Nous avons présenté brièvement les différentes technologies qui permettent de mettre en œuvre MDA et avons introduit l'application de MDA à l'étude de cas Java PetStore de

Sun. Nous avons dégagé pour finir les principaux avantages de MDA, qui sont la pérennité, la productivité et la prise en compte des plates-formes.

Dans la suite de l'ouvrage, nous détaillons chacun des points soulevés dans ce chapitre. Afin d'organiser ces explications, nous avons choisi d'utiliser les trois avantages de MDA comme grille de lecture.