

MÉMOIRE D8
présentation de MDA
Model Driven Architecture

JULIOT Etienne

version 1.0
10/06/2002

Rédacteur : Etienne Juliot - juliot@laposte.net
Destinataire : Christian Atiobgé
Relecteur : Jean Bézivin

Mots clés : modélisation, middleware, OMG, UML, méta-modèle, architecture, composant

Table des matières

1	Introduction	3
2	État de l'art	3
2.1	Évolution des langages	3
2.2	Évolution des middlewares	4
2.3	Évolution des méthodes	5
3	Utilité	6
4	Définition	6
5	Architecture de MDA	7
5.1	Le PIM - Platform Independant Model	8
5.2	PSM - Platform Specific Model	9
5.3	Services Facilities	10
5.4	UML Profils	12
6	Mise au point	13
6.1	Abstraction	13
6.2	Automatisation	13
7	Futur	14
8	Conclusion	14

Table des figures

1	Modèle UML de l'architecture de MDA	6
2	Architecture global de MDA	7
3	Dynamique générale de la mise au point d'une architecture MDA	8
4	Exemple de modélisation UML d'un objet PIM	9
5	Exemple de modèle PSM Corba	9
6	Méta-modélisation des EJBs dans MDA	10
7	Architecture des services vis à vis du reste de MDA	11
8	Spécialisation de la génération des services	11
9	Identification du rôle de l'expert du domaine et des designers	12
10	Représentation d'un modèle selon différents niveaux d'abstraction	13
11	Liaison entre un modèle PIM abstrait et un PSM Corba	13

1 Introduction

Depuis maintenant quelques années, on voit émerger régulièrement de nouvelles architectures logicielles à base de composants. Ce qui à l'origine devait permettre une réutilisabilité extrêmement importante se voit aujourd'hui soumis aux modes et à des évolutions très importantes. Les entreprises ne veulent plus investir des sommes colossales pour suivre cette incessante remise en cause de l'existant. Elles veulent pérenniser leurs architectures.

L'OMG a ainsi mis à contribution son expérience dans les middlewares (Corba) et dans la modélisation (UML) pour mettre au point une norme de méta-modélisation des architectures à base de composants : le Model Driven Architecture.

Ce méta-modèle définit une approche orientée modèle dans laquelle les composants révèlent une sémantique indépendante de toute plateforme, permettant ainsi de mieux se concentrer sur la logique métier. MDA définit également des abstractions de services et de comportements communs aux middlewares. Sachant qu'en plus, MDA se base sur des travaux déjà très largement adoptés comme références (XMI, UML, UML Profils, ...), la mise au point d'un méta-modèle spécifié suivant cette norme permettra d'automatiser au maximum la génération du modèle spécifique à une plateforme particulière¹.

2 État de l'art

Pour comprendre les raisons qui ont poussé à la création de MDA, il est nécessaire de faire un retour en arrière afin de suivre l'évolution des langages de programmation, des architectures middlewares et des méthodes de génie logiciel.

2.1 Évolution des langages

Au fur et à mesure de l'évolution des langages et des méthodes de programmation, plusieurs grandes générations se sont succédées. La première prend ses origines au début de l'histoire de l'informatique, à l'époque des langages procéduraux et fonctionnels. On peut discerner deux langages majeurs qui ont marqué cette période : le Cobol et le C (encore très largement utilisé).

L'évolution des langages a ensuite été guidée par un souci constant d'améliorer la productivité, la réutilisabilité et la facilité de maintenance et d'évolution des logiciels. A la suite de la constatation d'une stabilité plus forte en se concentrant sur les entités du système plutôt que sur ses fonctionnalités, la programmation orientée objets s'est vu privilégiée. Le premier de ces langages fut le smalltalk, suivi du bien connu C++. Aujourd'hui, le Java est de très loin le langage le plus prometteur dans ce milieu (le C# n'étant que dérivé de Java).

Suite à l'évolution des réseaux (dont Internet) et à l'augmentation de la complexité des logiciels, des architectures à bases de composants ont vu le jour². Ces middlewares se caractérisent par une vision plus macroscopique que l'approche objet. Un composant est un agrégat d'objets formant un ensemble cohérent et autonome. Chaque composant peut être distribué sur un réseau, ce qui leur permet de s'appeler entre eux sans se rendre compte qu'ils puissent être physiquement situés à l'autre bout de la planète.

¹Corba, EJB, .NET, .GNU, WebServices,...

²cf. le chapitre suivant pour leur historique

Cette distribution géographique a de nombreux avantages parmi lesquels on trouve la communication inter-entreprises, la répartition de charges, les architectures n-tiers, la réplication d'EIS, ...

En parallèle du développement des architectures à base de composants, un langage de description de données connaît également un engouement sans réserve : le XML (eXtended Markup Language)[11]. Celui-ci se voit ainsi utilisé dans tous les domaines et pour des utilisations assez variées. Porté par IBM, Microsoft et W3C, des protocoles utilisant XML ont été mis au point pour décrire et permettre la communication entre composants logiciels distribués sur Internet. Ces WebServices se servent ainsi de trois protocoles majeurs : UDDI (annuaire), WSDL (description d'interfaces) et SOAP (communication). Grâce à l'utilisation de protocoles ouverts et standardisés, on retrouve ici aussi l'indépendance vis à vis des langages, plateformes et fournisseurs d'applicatifs.

En résumé, on peut donc identifier les différentes évolutions suivantes ³ :

famille	Langages
procédural	Fortran, Cobol, C
objet	Smalltalk, C++, Java
composant	CCM, DCOM, EJB
webservice	UDDI, WSDL, SOAP

Ce tableau qui représente l'évolution des langages ne signifie pas qu'il vaut mieux utiliser les WebServices à la place du C. Tout dépend du contexte de l'application et des méthodes de mise au point. Chaque génération est plus ou moins dépendante de la précédente, et on peut facilement parler de couches d'abstraction.

2.2 Évolution des middlewares

Depuis maintenant quelques années, le concept de programmation orientée objets (POO) a évolué pour regrouper plusieurs objets de même catégorie ensemble. Ceci permet une forte séparation des modules d'un programme et une grande réutilisabilité. De plus, ces groupes d'objets peuvent être facilement déployés sur des machines distantes, et communiquer ensemble de manière transparente. Cette approche est nommée : la conception orientée composants. Des frameworks de composants ont été mis en place pour simplifier et uniformiser la conception de composants. Sur plateformes Windows, Microsoft a ainsi créé les composants COM, puis DCOM, leur équivalent en environnement distribué.

De nombreuses entreprises se sont regroupées au sein d'un organisme international nommé l'OMG (Object Management Group) afin de standardiser les composants. Leur résultat est une norme nommée Corba, indépendante de toute plateforme, de tout langage et de toute entreprise. C'est le standard le plus ouvert et certainement le plus puissant à l'heure actuelle. Il a par contre l'inconvénient d'être compliqué à mettre en place. Avec la dernière version de la spécification Corba 3, l'OMG a défini précisément la notion de composants autonomes sous le nom de CCM (Corba Component Model). Ce modèle semble très puissant mais il arrive un peu tard. En effet, l'OMG lui-même considère que Corba est une norme qui est en perte de vitesse⁴.

Avec l'explosion de l'utilisation de Java du côté des serveurs, Sun a lui aussi standardisé au sein de sa distribution J2EE (Java Enterprise Edition) les composants Java. Les EJBs (Enterprise Java Bean) ont ainsi connu un engouement extraordinaire. Ils ont l'avantage d'arriver au bon moment et de répondre simplement à la majorité des exigences des développeurs.

³Ce tableau est indicatif, il ne peut résumer à lui seul l'évolution complète des langages

⁴“Corba was a powerful first step, but we have more steps to take” Fred Waskiewicz, directeur des standards OMG

Pour répondre à cet engouement pour Java, Microsoft a mis en place une nouvelle stratégie entièrement tournée vers les WebServices : .Net . Cette architecture multi-langages propose de tirer partie de composants locaux, de services métiers accessibles sur un réseau et des services gérés par Microsoft pour l'authentification et le paiement sécurisé. Pour l'instant, cette architecture est mono-plateforme (Windows bien sûr) et ne marche que sur le serveur d'application de Microsoft, mais un projet libre du nom de Mono⁵ est en train de le porter sur d'autres plateformes, dont Linux.

Depuis peu, d'autres architectures à bases de composants sont apparues. On peut par exemple citer :

- .GNU qui visent à utiliser les WebServices comme brique de base des composants, mais sans tout centraliser dans une compagnie ou un autre tiers
- Bonobo qui représente une sur-couche logicielle à Corba dans le cadre de l'utilisation de Gnome (un des desktops de GNU/Linux)

On peut aussi énumérer les principales architectures middlewares actuelles, passées et à venir :

Middleware	Mainteneur	Langages
EJB	Sun et JCP (BEA, Oracle, IBM, Borland, ...)	Java
.Net	Microsoft	C#, VB.NET, ASP.NET, ...
WebServices	WS-Consortium (IBM, Microsoft, ...)	SOAP, UDDI, WSDL + XML, HTTP
CCM	OMG	Corba
.GNU	GNU	Java, C#

2.3 Évolution des méthodes

En plus de l'évolution des langages et des composants, les méthodes de conception se sont raffinées au fur et à mesure du temps.

Pour les langages procéduraux, la méthode à la mode était SADT. Cette méthode générait beaucoup de documents et partait du principe de boîtes noires et de raffinements successifs.

Du côté des bases de données, la domination de Merise a été longtemps incontestée (même encore aujourd'hui)⁶.

Avec l'avènement de la programmation orientée objets, plusieurs méthodes sont rentrées en compétition. Plutôt que de disperser les concepteurs dans des méthodes concurrentes avec pourtant les mêmes objectifs, les créateurs des méthodes OMT, Booch et OOSE se sont regroupés pour mettre en place une spécification commune.

C'est ainsi qu'est né UML (Unified Model Language)⁷, standardisé peu de temps après par l'OMG.

UML n'est pas une méthode, c'est plutôt une syntaxe. La norme décrit comment représenter des cas d'utilisation, des diagrammes de classes, des scénarios, ... mais elle n'impose pas de processus de mise au point. On peut dire qu'aujourd'hui, UML est de très loin la norme de représentation objet la plus utilisée, la plus efficace et la plus universelle.

UML évolue régulièrement. Un langage complémentaire à UML (nommé OCL et inclus dans la norme dans sa version 1.3) a été créé pour le formaliser un peu plus. Le méta-modèle d'UML a lui-aussi été standardisé dans une spécification nommée MOF. Le MOF peut se décrire en XML suivant la spécification XMI. On peut ainsi générer ou interpréter de l'UML indépendamment de

⁵<http://www.mono.org>

⁶Cette constatation est valable surtout en France

⁷<http://www.omg.org/uml>

L'objectif de MDA est de créer une représentation UML de la logique métier et de lui associer des caractéristiques MDA. Ensuite, il suffit de demander une génération automatique des composants en fonction de l'architecture à composants choisie. Du travail complémentaire est encore bien sûr nécessaire afin de raffiner le modèle obtenu en fonction du contexte choisi (mais le travail est largement allégé).

Ainsi, le passage d'une architecture .NET à une architecture J2EE, par exemple, ne consiste qu'à reprendre la logique métier spécifiée en MDA et à redemander une génération, mais cette fois, avec une cible différente.

5 Architecture de MDA

Pour bien comprendre comment fonctionne MDA, il peut être intéressant de simuler chronologiquement la conception d'une application suivant le modèle MDA (cette chronologie n'est pas strictement dans cet ordre car plusieurs étapes se font en parallèles).

MDA se découpe en 4 principales couches (ou étapes) qui réfèrent à chaque fois à des standards déjà adaptés par l'industrie ou en cours de normalisation.

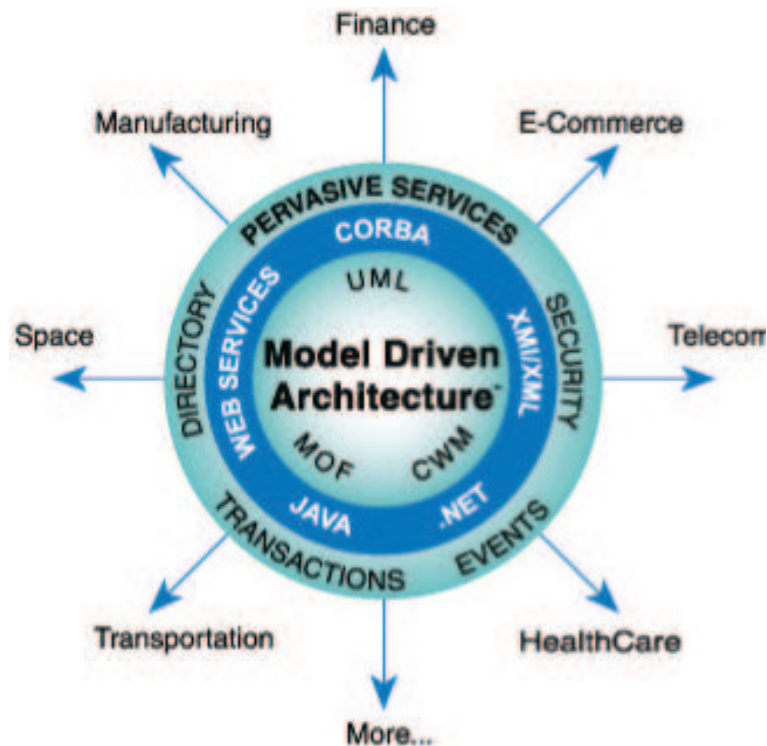


FIG. 2 – Architecture global de MDA

Au coeur de MDA, on retrouve les technologies UML, MOF et CWM spécifiées par l'OMG pour modéliser la logique métier de l'application.

Ce modèle métier est spécialisé ensuite dans une technologie middleware. On retrouve les standards actuels tels que les EJBs, Corba, .NET et les WebServices. L'anneau extérieur du cercle représente les services. Ceux-ci permettent par exemple de gérer la transaction, la persistance, les évènements.

Enfin, la couche spécifique au domaine prend place à la périphérie du noyau. Elle se base sur les profils UML et permet de proposer des frameworks spécifiques au domaine d'application de l'application (Espace, Télécommunication, Mécanique, etc.).

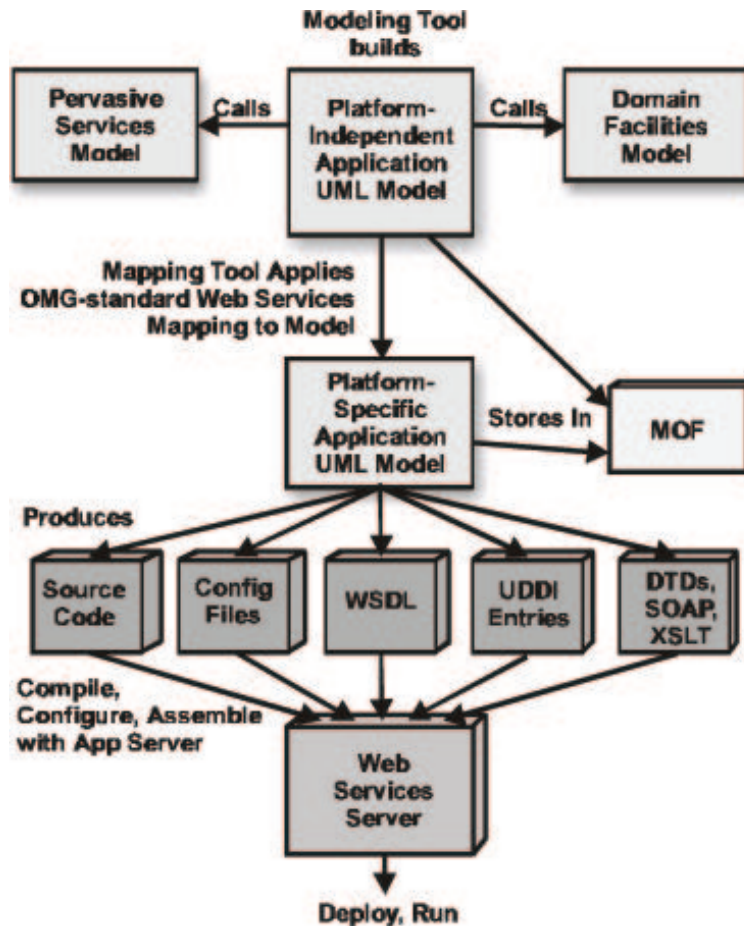


FIG. 3 – Dynamique générale de la mise au point d'une architecture MDA

Chacune de ces couches va être approfondie dans les chapitres suivants.

5.1 Le PIM - Plateforme Independant Model

La première étape lors de la conception d'une application est de se concentrer sur l'intérêt de l'application : la logique métier. Cette étape permet la mise au point de l'intelligence du modèle, totalement spécifique à l'application, mais indépendante de la technique.

Dans un cycle de développement "en Y", le PIM est considéré comme l'architecture métier. Elle est mise au point par un architecte spécialisé dans le domaine de l'application.

Pour mettre au point cette partie, trois standards sont utilisés :

- UML : pour modéliser
- MOF : pour méta-modéliser (par l'intermédiaire de XMI)
- CWM : pour modéliser les flux entre datawarehouses et le traitement en temps réel d'informations tout en contrôlant les ressources

Concrètement, l'architecte met au point ces diagrammes de classes, et il les stéréotype en suivant le méta-modèle MDA, en vue de donner une sémantique aux classes.

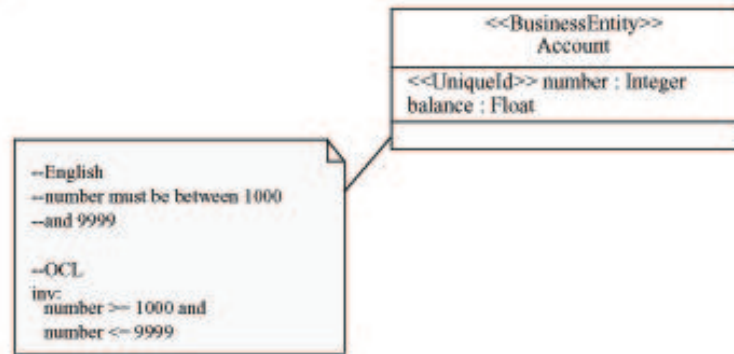


FIG. 4 – Exemple de modélisation UML d’un objet PIM

De plus, l’utilisation d’UML comme moyen pour donner une sémantique MDA à des objets est judicieuse car cette méthode de représentation est très complète et permet de pousser loin la modélisation. On peut par exemple utiliser les pré/post condition par l’intermédiaire de OCL.

Par comparaison, le format IDL promu par l’OMG pour Corba et pour représenter l’interface de classes indépendamment du langage, est moins concis et moins formel (pas d’invariant, pas de contraintes sur la valeur “null”, ...).

Une fois le PIM mis au point, il devient ainsi bien plus facile de procéder à des contrôles de cohérence sur le modèle que si on travaillait directement sur un modèle spécifique à une plateforme.

5.2 PSM - Platform Specific Model

Une fois le PIM mis en place, l’architecte technique prend la main sur le projet. Il s’agit cette fois de générer le modèle décrit dans des termes métiers vers une plateforme spécifique de middleware. Au fur et à mesure de l’évolution des outils de génération MDA, cette phase sera de plus en plus automatisée. Elle aura à terme la même puissance qu’ont aujourd’hui les outils de génération de code à partir de modèles UML.

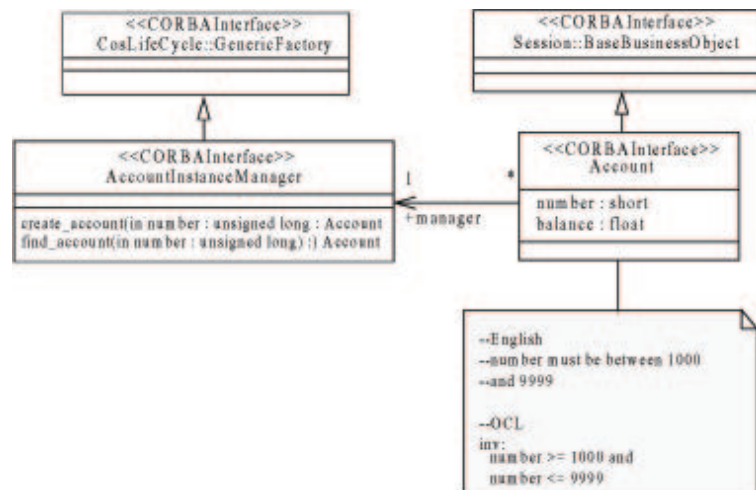


FIG. 5 – Exemple de modèle PSM Corba

La génération consiste à créer à la fois des éléments spécifiques à la plateforme cible tels que

les fichiers IDL en Corba, les descripteurs de déploiement XML en Java ou les fichiers WSDL en .NET) mais également des modèles UML représentant les classes du modèle métier dans le contexte d'un middleware.

Cette phase peut être considérée comme un raffinement du modèle décrit dans la PIM.

Bien entendu, l'OMG standardise cette phase de génération pour obtenir au final des architectures inter-opérables et pour garder une cohérence entre les PSM. Actuellement, seule la transformation vers les composants CCM de Corba 3.0 est standardisée, mais celle vers les EJBs ne saurait tarder.

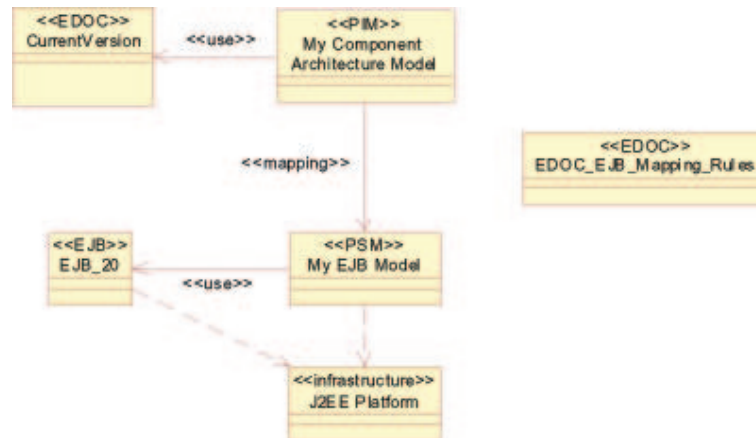


FIG. 6 – Méta-modélisation des EJBs dans MDA

A court terme, une quantité certaine d'efforts sera nécessaire pour mapper complètement le PIM vers un PSM, mais on peut espérer que l'intégralité de cette phase sera automatisée d'ici quelques années (à condition pour l'architecte de bien spécifier la sémantique des composants dans le PIM).

En utilisant le standard MDA, il devient aussi plus aisé de rendre l'interopérabilité entre plateformes plus immédiate et la création de ponts entre technologies plus efficace. L'OMG va même jusqu'à comparer MDA à un "ORB Internet" intégrant toutes les plateformes middleware passées, présentes et futures.

Si une nouvelle plateforme middleware apparaît au cours de l'élaboration d'un logiciel, il suffit de développer un mappeur entre le PIM et ce nouveau PSM pour transformer l'application vers ce middleware. Pour les entreprises, le gain est très important car il permet de suivre les technologies sans avoir à tout remodeler. On peut analoguer cette évolution des méthodes de travail avec celle qui a eu lieu lors du passage des méthodes d'"extrem programming" (réfléchir en codant) vers les méthodes de modélisation, qui ont permis de faciliter le changement de langage d'implémentation et de se concentrer sur des problèmes de plus haut niveau.

5.3 Services Facilities

Lorsqu'on regarde chacune des technologies middleware une à une, on constate qu'à chaque fois, des services annexes aux technologies ont été développés pour simplifier le travail des développeurs. On peut remarquer qu'à chaque fois, les mêmes services reviennent et qu'ils offrent presque tout le temps les mêmes fonctionnalités.

On retrouve sous différentes formes les services suivants⁸ :

⁸Cette liste n'est pas exhaustive

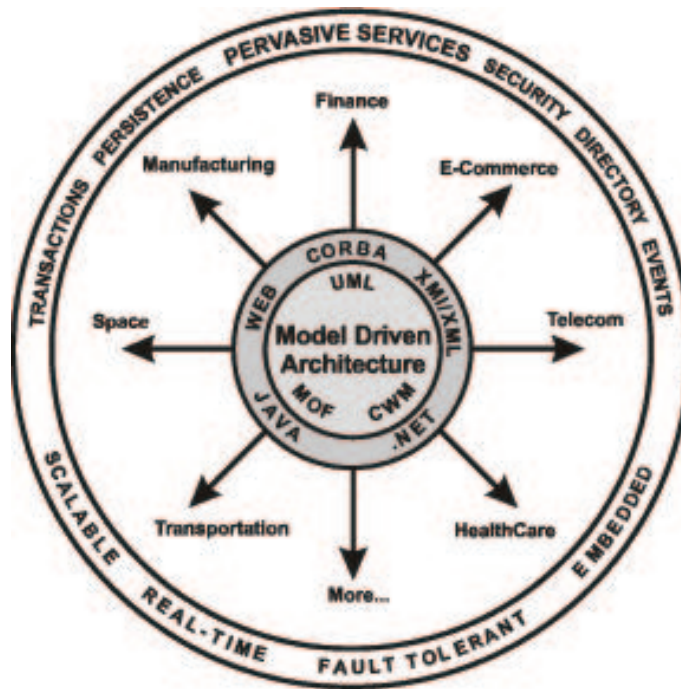


FIG. 7 – Architecture des services vis à vis du reste de MDA

- Service de nommage (ex : UDDI, CosNaming, JNDI, LDAP), pour retrouver la référence à un composant distant
- Service de transaction (ex : JTA), pour permettre de gérer la consistance des actions
- Service de sécurité (ex : WS-Security, SSL), pour crypter les communications et permettre une authentification
- Service d'évènements (ex : CosNotify, JMS), pour mettre au point une architecture événementielle
- Service de persistance (ex : EJB Entity, CosPersistent), permet de gérer la persistance transparente (dans une base de donnée par exemple)

Ainsi, l'OMG a défini un méta-modèle par dessus ces services pour identifier leurs parties communes. Ce méta-modèle est bien entendu spécifié comme un PIM, avec des procédures de mapping standardisées vers des plateformes. De cette manière, on peut dire qu'une partie de MDA est spécifiée grâce à MDA.

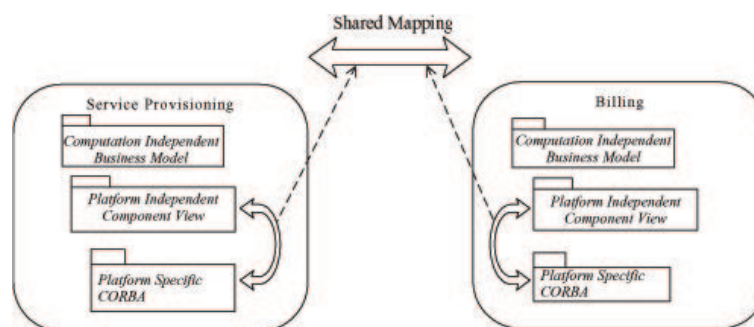


FIG. 8 – Spécialisation de la génération des services

Dans MDA, l'application est évidente : il s'agit de mettre en place ces méta-services au moment de la mise au point du PIM. On obtient un niveau d'abstraction supplémentaire vis à vis du middleware. En effet, savoir comment est gérée concrètement le mécanisme de transaction n'intéresse pas le modelleur. Il a juste besoin de savoir si tel ou tel objet fait partie d'une transaction ou non. Seul le spécialiste de la plateforme s'y intéressera précisément (et encore, si un besoin pointu se fait sentir).

5.4 UML Profils

Depuis 1996, l'OMG a mis une grande partie de ses forces dans une communauté nommée *Domain Task Forces*. Celle-ci a pour objectif de spécifier des architectures communes à des catégories particulières d'applications. En effet, les applications d'un même domaine d'activité (par exemple : la comptabilité, le commerce électronique, les banques, etc.) ont des caractéristiques communes. La factorisation de celles-ci aboutirait à des architectures pré-cablées. On évite ainsi de travailler "from scratch" (autrement appelé syndrome de la page blanche) et de réinventer à chaque fois la roue.

Pour chaque domaine d'activité, les principaux acteurs à la fois techniques, applicatifs, utilisateurs se sont mis autour d'une table pour se mettre d'accord sur les objectifs à atteindre et sur les frameworks y répondant au mieux.

Une partie de ces travaux ont déjà aboutis en prenant le nom d'*UML Profils*. Cet ajout à la norme UML sera une de ses plus grosse évolution.

Cet engouement de l'industrie vis à vis de ces travaux est dû à une volonté de standardiser et de normaliser l'élaboration des processus métier (des initiatives similaires existent d'ailleurs dans le monde du XML avec BTML, ebXML, ...). Outre l'interopérabilité plus aisée entre les applications tirant parties de ces frameworks, les profils UML permettent de gagner du temps et de minimiser les risques lors de l'élaboration d'un logiciel.

Les profils UML peuvent être mise au point dans un cadre dépassant l'application visée. Ces profils peuvent donc être réutilisables et ils permettent d'exposer une frontière plus nette entre l'expertise d'un domaine d'activité et son implémentation informatique. L'expert du domaine obtient ainsi une place bien identifié dans le processus d'élaboration du modèle et bien discernée de celle du designer.

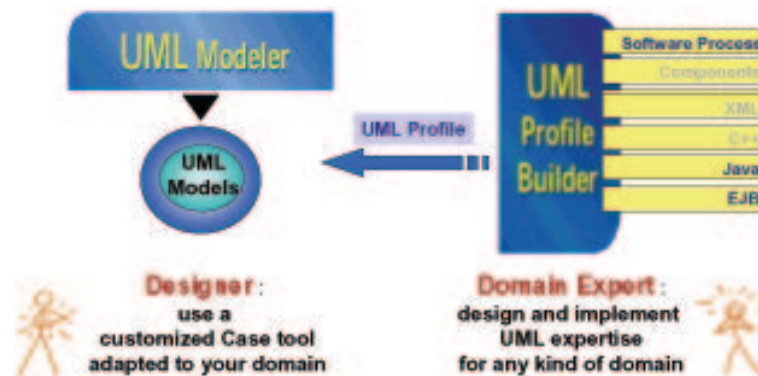


FIG. 9 – Identification du rôle de l'expert du domaine et des designers

Dans le cadre de MDA, les profils UML s'intègrent parfaitement car ils tirent partis des informations de sémantiques portées par le modèle PIM pour automatiquement générer les classes et le framework associés au domaine cible.

6 Mise au point

6.1 Abstraction

Pour une même application, il est très souvent utile de représenter sa logique métier à plusieurs niveaux de détail. UML propose des mécanismes très simples d'abstraction, comme par exemple les packages ou l'utilisation de pattern façade. MDA apporte un complément non négligeable en permettant de représenter un même modèle selon son PIM (où on ne se concentre que sur la logique métier) et son équivalent PSM (la même chose, mais avec la technique en plus).

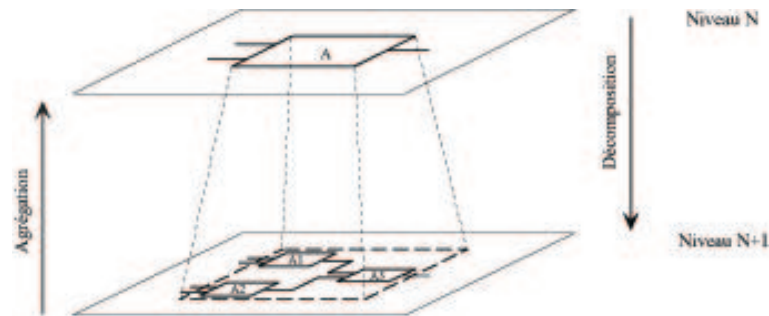


Figure 1. Représentation d'un modèle selon différents niveaux d'abstraction

FIG. 10 – Représentation d'un modèle selon différents niveaux d'abstraction

Les modèles ont les mêmes points d'entrée et de sortie mais leur mécanisme interne est plus ou moins détailler. Il permet de travailler à des niveaux micro ou macroscopique, en déléguant le travail à des spécialistes différents.

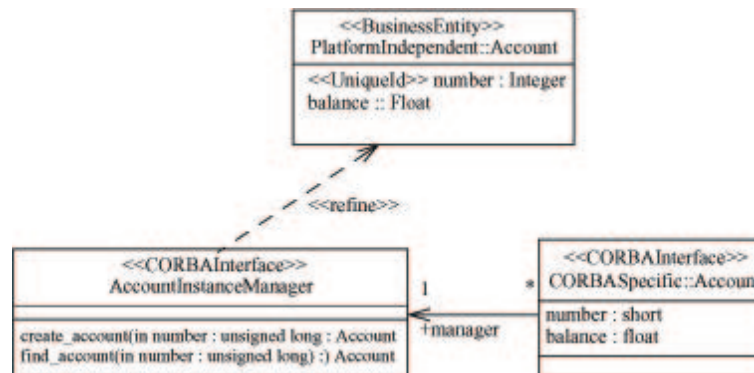


FIG. 11 – Liaison entre un modèle PIM abstrait et un PSM Corba

L'abstraction de PIM à PSM n'est pas la seule. Il est également possible de raffiner un PIM vers un autre PIM, et un PSM vers un PSM (pour avoir différents niveaux de granularité d'un même modèle). Entre chaque modèle, il faut absolument garantir la cohérence. MDA propose ainsi par des mécanismes de synchronisation inspirés du reverse engine entre modèles.

6.2 Automatisation

En combinant la génération du modèle spécifique vers le modèle technique (PIM vers PSM) et l'utilisation de domaines métiers standardisés (UML Profils), on obtient une automatisation très

importante. On peut estimer qu'avec des outils MDA complets, le temps de développement d'une application peut quasiment être réduit de moitié, tout en évitant bons nombres de bugs liés à la technique. De plus, le portage d'une architecture vers une autre devient bien plus rapide et bien plus automatisé.

Les techniques de travail risquent fortement d'être changées par l'adoption de MDA car de nouveaux métiers pourraient apparaître. La génération successive de différents types de modèles peut ainsi requérir à terme des spécialistes formés à une certaine étape du processus de développement MDA.

7 Futur

Sur le papier, MDA possède tous les atouts pour s'imposer. Mais mieux vaut se garder de faire des prédictions hâtives.

MDA n'en est pour l'instant qu'à ses débuts, et l'OMG a seulement proposé de grandes idées pour son fonctionnement. Tout le méta-modèle MDA n'est pas spécifié et quelques zones d'ombres existent encore.

Un des moteurs majeurs du succès de MDA est la mise au point d'outils l'exploitant. Pour l'instant, seules quelques AGL (dont Objectteering de Softeam, qui est très actif dans ce domaine) s'y intéressent mais on ne peut espérer d'outils performant avant 2004. Par contre, ces outils seront certainement basés sur XMI, ce qui les rendra indépendant de tout atelier UML (comme c'est trop rarement le cas actuellement, en bonne partie à cause du quasi-monopole de Rational Rose).

Malgré tous ses avantages, est-ce que les entreprises investiront dans MDA ? Actuellement, quelques grands comptes (dont IBM) ont dors et déjà annoncé leur intérêt et la prochaine adaptation de leur méthodes de travail. Mais pour s'imposer, MDA requiert une adhésion massive. Or, celle-ci peut être soumise à la mode (comme on le voit pour les services Web et XML) ou à son ergonomie d'utilisation (Corba a par exemple échoué en bonne partie à cause de sa difficulté d'utilisation).

Une autre cause d'un échec possible de MDA serait la rationalisation des middlewares. En effet, si les différents rôles entre middlewares sont identifiés convenablement et qu'une stabilisation de ceux-ci apparaît, l'intérêt d'un méta-modèle pourrait être remis en cause⁹.

De plus, des outils de génération de modèles, généralement basés sur XML et générant du XMI, existent déjà. Ils ont l'avantage d'être efficaces, faciles à utiliser et d'être déjà au point. Mais ils ne possèdent ni la standardisation de MDA, ni sa richesse et généralement pas non plus son indépendance vis à vis du middleware.

8 Conclusion

En conclusion, MDA arrivera au moment même où les entreprises n'ont plus envies de dépenser de l'argent pour suivre la mode des middlewares. L'OMG a mis au point une spécification robuste et parfaitement adaptée à l'évolution des technologies et des méthodes de travail.

MDA aide les architectes à pérenniser leurs architectures, aide les développeurs à générer automatiquement des éléments techniques récurrents et fastidieux, aide les managers à identifier les tâches de chaque expert dans le processus de mise au point, et aide les entreprises à gagner

⁹Par exemple, une architecture typique est l'utilisation d'EJBs en interne et les Web Services en externe (inter-entreprises).

de l'argent en gagnant du temps et en minimisant les risques d'engagement dans une mauvaise technologie.

Références

- [1] MDA
<http://www.omg.org/mda>. OMG.
- [2] Model Driven Architecture
Document number ormsc/2001-07-01 *Architecture Board ORMSC*. OMG, 9 juillet 2001.
- [3] Developing in OMG's Model-Driven Architecture
Jon Siegel and the OMG Staff Strategy Group. Revision 2.6 OMG, Novembre 2001.
- [4] Model Driven Architecture - White Paper
Richard Soley and the OMG Staff Strategy Group. Draft 3.2 OMG, 27 novembre 2000.
- [5] MDA - When a major software industry trend meets our toolset, implemented since 1994
Philippe Desfray. <http://www.softeam.fr>. Softeam, 2001.
- [6] Architecture orientée objet
Modélisation multivue et simulation à événements discrets
Antoine Aïllor, JF Santucci. Université de Corse.
- [7] Modélisation objet avec UML
Pierre-Alain Muller. Eyrolles, 1999.
- [8] Modélisation objet avec UML
Pierre-Alain Muller. Eyrolles, 1999.
- [9] UML 1.4
<http://www.irisa.fr/manifestations/seminaires-2000/10nov00/Bezivin/index.htm>.
Jean Bezivin D'après textes de l'OMG, 7 novembre 2001.
- [10] UML 2.0
<http://cgi.omg.org/cgi-bin/doc?ad/01-02-39.pdf>. OMG.
- [11] XML
<http://www.w3c.org/>. W3C.
- [12] Corba
<http://www.omg.org/corba>.
OMG.
- [13] Entreprise JavaBean
<http://java.sun.com/j2ee>.
Sun.
- [14] .Net
<http://www.microsoft.com/net>.
Microsoft.
- [15] .GNU
<http://www.dotgnu.org>. GNU.