Généralités sur l'approche MDA

Article rédigé par Lamia Gaouar

Présentation

Les technologies sont en constante évolution. Afin de bénéficier de avancées technologiques, il est nécessaire d'adapter les applications à ces technologies, or cette opération coûte cher aux entreprises car il est souvent nécessaire de réécrire le code entièrement. Lorsqu'il n'existe pas de capitalisation des fonctions de l'application et que le développement repose généralement sur le code source. La séparation des préoccupations est apparue comme la solution nécessaire au problème. Ainsi, spécifications fonctionnelles et spécifications techniques sont prises en compte séparément par l'approche MDA.

L'approche MDA (Model Driven Architecture) est un processus de l'ingénierie dirigée par les modèles. Après la technologie procédurale, la technologie objet et la technologie des composants, l'approche MDA est à l'origine de l'ingénierie des modèles. Elle a été proposée par l'OMG en 2000. L'approche MDA est basée sur la séparation des préoccupations. Elle permet prendre en compte, séparément, aspect métier et aspect technique d'une application, grâce à la modélisation. Le code source de l'application est obtenu par génération automatique à partir des modèles de l'application. Les modèles ne sont plus seulement un élément visuel ou de communication mais sont, dans l'approche MDA, un élément productif et le pivot du processus MDA.

Les modèles

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 oeuvre 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. Du plus abstrait, le CIM au plus concret le PIM :

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 dérit 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 finale 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.

Les transformations de modèles

Les modèles CIM, PIM et PSM constituent les étapes principales de l'approche MDA. Chacun de ces modèles contient des informations nécessaires à la génération du code source de l'application. Le code est obtenu par génération automatique à partir du PSM, le PSM est obtenu par transformation successive des modèles CIM vers PIM et des modèles PIM vers PSM. La transformation de modèles est une étape importante du processus MDA, c'est grâce aux transformations que les modèles deviennent des éléments productifs de MDA. L'exécution des transformations permet d'assurer un lien de traçabilité entre les différents modèles du processus MDA. Ces liens sont un gage de qualité du processus de développement logiciel dans MDA.

Transformations CIM vers PIM

Les modèles CIM expriment les besoins des utilisateurs. Cette étape consiste à construire, partiellement, des modèles PIM à partir des CIM. Le but est de retranscrire les informations contenues dans les CIM vers les modèles PIM. C'est ce qui va permettre de

s'assurer que les besoins de l'utilisateur sont véhiculés et respectés tout au long du processus MDA.

Transformations PIM vers PIM

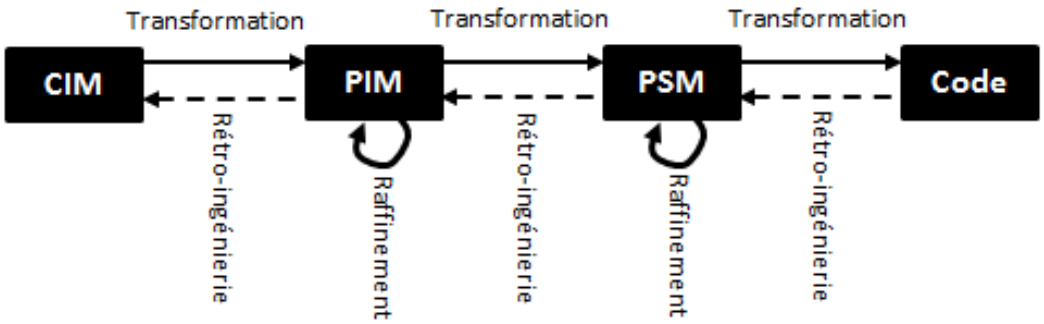
Les modèles PIM modélisent l'aspect structurel et dynamique d'une application. Cette étape s'exprime par l'enrichissement des modèles PIM. Enrichir des PIM consiste à leur rajouter de l'information utile et à spécifier leur contenu. Les PIM sont ainsi raffinés et les informations qu'ils contiennent, précisées.

Transformations PIM vers PSM

Cette étape consiste à créer des modèles PSM à partir des informations fournies par les modèles PIM, en y ajoutant des informations techniques relatives à la plateforme d'exécution cible. C'est ainsi que le lien avec la plateforme d'exécution se forme. Un PSM fournit les informations utiles à la génération du code de l'application et sont dépendants de la plateforme d'exécution. Il est possible de créer autant de PSM qu'il y a de plateformes cibles.

Transformations PSM vers code

La transformation des modèles PSM en code source consiste à générer le code source de l'application, de façon totale ou partielle, à partir des modèles PSM de l'application. Cette étape n'est pas à proprement dit considérée comme une transformation par MDA. En effet, une transformation selon MDA est définie par la transformation d'un modèle vers un autre modèle, chacun d'eux étant structurés par leur métamodèle. Or le code source n'a pas de métamodèle, la transformation des PSM vers le code est plutôt considérée comme une retranscription textuelle du modèle PSM.



- Les transformations de modèles dans MDA -

Transformations inverses

Transformations inverses ou rétro-ingénierie, signifie la construction de modèles à partir d'applications existantes. Dans ce cadre, MDA identifie aussi les transformations inverses : Code vers PSM, PSM vers PIM et PIM vers CIM.

Les étapes du développement MDA

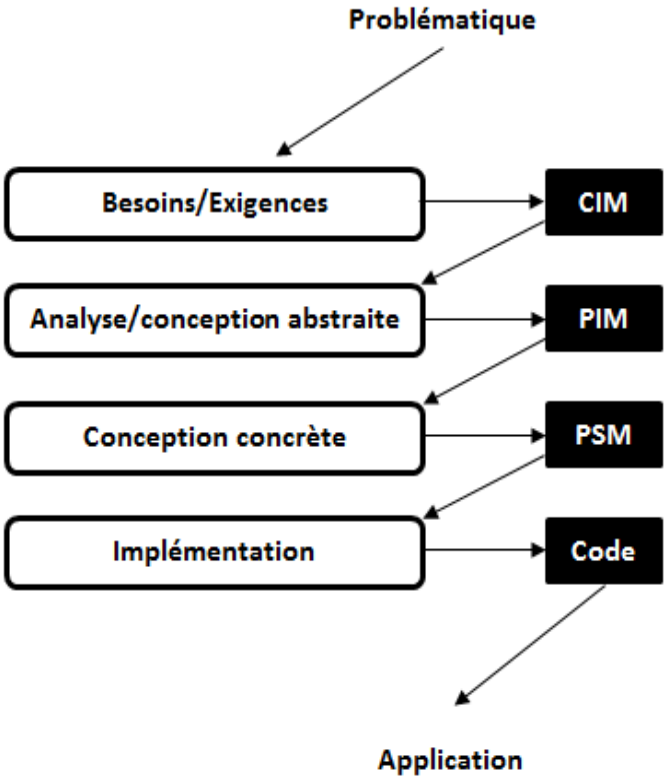
Le développemnt d'une application débute toujours par le ressenssement des besoins et exigences de l'utilisateur. Comme dans l'approche MDA, l'étude d'un problème débute par la spécification des besoins métiers qui sont ensuite modélisés par un ou plusieurs modèles CIM.

Une fois les fonctions de l'application définies, vient ensuite l'étape d'analyse et de conception abstraite de l'application. Les PIM sont générés à partir des modèles CIM (transformation CIM vers PIM). Les CIM et les PIM sont des modèles pérennes puisqu'ils sont indépendants des plateformes d'exécution. la définition des modèles CIM et PIM constitue l'analyse fonctionnelle de l'application.

L'étape suivante est l'étude technique du développement de l'application. L'application est décrite dans son environnement d'exécution, le PSM est obtenu à partir des modèles PIM (transformation PIM vers PSM) associé aux spécifications techniques de la plateforme cible. Un PSM est spécifique à une plateforme particulière et n'est donc pas pérenne.

Le code source de l'application est obtenu par génération automatique à partir du PSM, c'est le rôle du générateur de code. La génération du code peut être partielle ou totale, cela dépend du niveau de détail de l'information véhiculée par le PSM.

Les besoins de l'utilisateur sont initialement modélisés dans les CIM. Tout au long du processus MDA ces besoins sont retranscrits vers les modèles PIM et PSM, jusqu'au code à travers les divers étapes de transformations de modèles et de génération de code. Ce lien de traçabilité qui existe, du modèle CIM jusqu'au code, est assuré grâce aux transformations de modèle, et permet d'assurer que l'application à l'arrivé respecte bien les besoins de l'utilisateur.



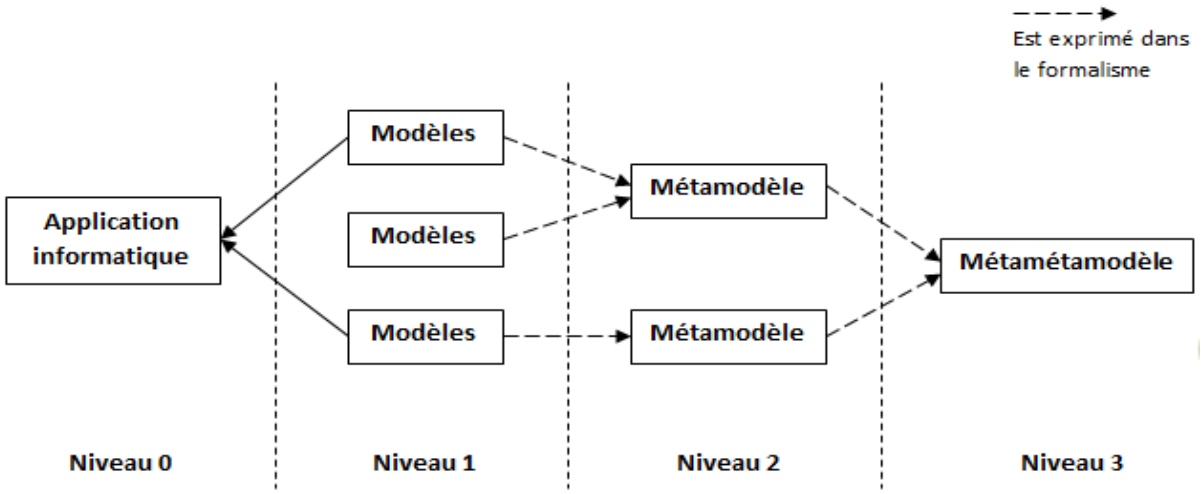
- Architecture MDA –

L'architecture à 04 niveaux de MDA

Lorsqu'on rédige un programme en Java (ou dans tout autre langage), celui-ci respecte la grammaire du langage Java, c'est le formalisme de modélisation. Ce formalisme est lui-même définit par une grammaire BNF (Bachus Naur Form) qui se définie elle-même. La grammaire BNF est appelée dans ce cas un métaformalisme. Il en va de même pour l'élaboration des modèles dans l'approche MDA.

Les niveaux méta

Dans MDA, formalisme de modélisation et métaformalisme sont appelé respectivement métamodèle et métamétamodèle. Ainsi chaque modèle est exprimé dans le formalisme de son métamodèle, qui lui-même est définit par son métamodèle, autrement dit le métamétamodèle. Le métamétamodèle se définit lui-même. Les relations entre les notions de modèle, métamodèle et métamétamodèle forment l'architecture à 04 niveaux de MDA :



- Les niveaux méta de MDA -

Dans l'approche MDA, les applications informatiques sont les entités à modéliser. Les modèles représentent l'information, partielle ou totale, nécessaire à la construction de ces applications. Dans MDA, **une application informatique est doncreprésentée par un ou plusieurs modèles**.

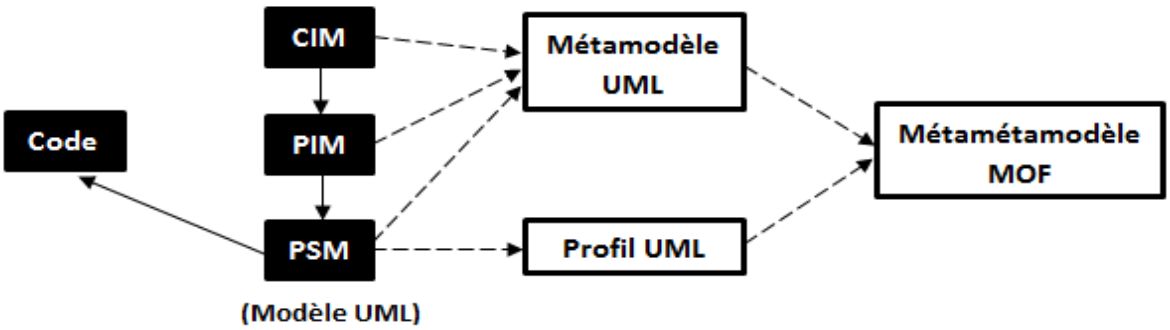
Un métamodèle définit la structure (et non la sémantique) des modèles conforment à ce métamodèle. Un métamodèle est un diagramme de classes qui définit les entités du modèle, ainsi que les propriétés de leurs connexions et de leurs règles de cohérence. Pour être valide, **chaque modèle doit être conforme à son métamodèle.** Cette relation de conformité est primoridale dans l'approche MDA, il est possible ainsi de construire des outils capables de manipuler les modèles.

Le métamétamodèle est pour ainsi dire le métamodèle des métamodèles. Et entretien donc avec eux la même relation qu'a un métamodèle avec ses modèles. **Les métamodèles sont des modèles conforment à leur métamétamodèle.**

Dans MDA, quel que soit le niveau méta, tous les éléments sont considérés comme des modèles. Par conséquent, les métamodèles et les métamétamodèles sont aussi des modèles.

On aurait pu continuer à monter dans les niveaux méta mais le fait est que le métamétamodèle, utilisé par MDA, se définit lui-même. Autrement dit, il est son propre métamodèle ce qui amène à 04 niveaux de hiérarchie. Il n'existe en effet qu'un seul métamétamodèle utilisé par MDA et qui est le standard MOF (Meta Object Facility) qui a pour caractéristique de se définir lui-même.

L'architecture de MOF1.4



- L'architecture de MOF1.4 -

Le métamétamodèle MOF1.4

L'OMG a définit le standard MOF (Meta Object Facility). La version 1.4 de MOF est celle utilisée par les modèles publics de l'OMG et donc, elle est largement répandue. Le rôle du métamétamodèle MOF1.4 est de structurer les métamodèles de l'architecture. Il représente les métamodèles sous forme de diagrammes de classes. Le diagramme de classes qui modélise le métamodèle est composé de méta-classes, les méta-classes de méta-attributs, …etc. Cette nomination sert à différencier avec les diagrammes de classes des modèles. Nous avons dit que le métamétamodèle MOF1.4 est le métamodèle des métamodèles, par conséquent, il est lui-même représenté sous forme de diagramme de classes.

Le métamodèle UML et les profils UML

MDA recommande l'utilisation du métamodèle UML pour l'élaboration des modèles CIM et PIM, et l'utilisation des profils UML pour l'élaboration des modèles PSM. Un métamodèle UML est un diagramme de classes conforme au standard MOF1.4. Les modèles CIM, PIM et PSM sont des modèles UML conforment au métamodèle UML.

En plus d'être un standard de modélisation très largement répandu, UML définit un ensemble de diagrammes pour la modélisation des différents aspects (fonctionnels, structurels, dynamiques, ...) des applications orientées objets, indépendamment des

plateformes d'exécution ce qui le rend idéal pour l'élaboration des modèles CIM et PIM.

UML définit aussi la notion de profils UML. Un profil UML est une extension – ou adaptation – d'UML à un domaine pariculier. Dans le context MDA, les profils UML peremttent de décrire la strcture d'une plateforme d'exécution particulière grâce à des diagrammes UML stéréotypés. Un profil UML correspond à une plateforme unique, il est dépendant des plateformes d'exécution et donc idéal pour la modélisation des PSM. Une autre approche possible pour l'élaboration des PSM, se sont les métamodèles de plateformes. Un PM (platforem Model) est un métamodèle conforme au standard MOF1.4 et décrit une plateforme particulière. Profil UML ou métamodèle de plateformes, les deux approches sont recommandées par MDA.

L'architecture de MOF2.0

Comme mentionné plus haut, le métamodèle MOF ressemble fortement aux diagrammes de classes UML. L'objectif de la version 2.0 de MOF, est d'un coté de rassembler les points communs entre UML et MOF et de l'autre d'en expliciter les différences. MOF2.0 est beaucoup plus complexe que sa version précédente, mais ne modifie en rien l'architecture à 04 niveaux de MDA. MOF2.0 et UML2.0 sont toujours respectivement, le métamétamodèle et le métamodèle de MDA.

UML2.0 Infrastructure

L'OMG à définit le métamodèle UML2.0 Infrastructure. La définition d'un nouveau métamodèle ne change pas l'ordre de hiérarchie de l'architecture à 04 niveaux de MDA. UML2.0 Infrastructure à pour unique but de rassembler les méta-classes communes (celles concernant les diagrammes de classes comme *package*, *class*, …) entre UML et MOF dans un seul métamodèle. UML2.0 Infrastructure est le métamodèle commun entre UML2.0 Superstructure et MOF2.0, et a pour seul rôle d'être réutilisé par eux.

Le métamodèle UML2.0 Infrastructure est découpé en une trentaine de packages pour en faciliter la réutilisation. L'intégration des packages d'UML2.0 Infrastructure par UML2.0 Superstructure et MOF2.0 se fait par un le bais d'un mécanisme complexe, le merge. Pour simplifier, le merge est une sorte de copier-coller des éléments du package mergé vers le package mergeur. UML2.0 Infrastructure utilise lui-même le merge entre certains de ces packages.

UML2.0 Superstructure

Dans sa version 2.0, le standard UML s'appelle UML2.0 Superstructure. Ce nouveau nom permet de le distinguer du métamodèle UML2.0 Infrastructure. UML2.0 Superstructure intègre certaines parties du métamodèle UML2.0 Infrastructure par le méchanisme du merge, mais intègre aussi tous les concepts nécessaire à l'élaboration des diagrammes UML (diagramme cas d'utilisation, de classes, de séquences, …). UML2.0 Superstructure est plus complexe qu'UML2.0 Infrastructure étant donné son rôle dans l'architecture à 04

niveaux de MDA.

MOF2.0

Théoriquement MOF2.0 est toujours considéré comme l'unique métamétamodèle de MDA. Techniquement, MOF2.0 est composé de deux parties : EMOF pour Essential MOF et CMOF pour Complete MOF. EMOF permet l'élaboration de métamodèles sans méta-associations (référence à la propriété d'*association* d'UML dans les métamodèles), il intègre le package Basic de UML2.0 Infrastructure. Et CMOF permet l'élaboration de métamodèles avec associations, il intègre le package Construct de UML2.0 Infrastructure. Les relations d'intégration entre MOF2.0 et UML2.0 Infrastructure se font aussi par le machanisme du merge. Donc avec MOF2.0 vous avez la possibilité de créer des métamodèles avec ou sans méta-association simplement en vous basant sur EMOF ou CMOF.

Le standard MOF est à la base de l'architecture à 04 niveaux de MDA. Tous les métamodèles basés sur MOF ont une structure commune, cette conformité assure une cohérence entre les modèles élaborés durant le processus. Cette cohérence est un gage de la qualité MDA. Un point qu'il faut préciser, ce n'est pas l'architecture à 04 niveaux qui structure l'approche MDA, mais c'est l'ensemble des métamodèles qui la compose. Ainsi, on peut très bien développer ses propres métamodèles pour adapter MDA au contexte de son application/projet.

Les technologies de mise en oeuvre

L'approche MDA et préconise l'utilisation de standards de l'OMG, parmi eux UML, MOF ET XMI, mais n'exclut en rien l'utilisation d'autres outils ou langages.

Les modèles

Pour l'élaboration des modèles, MDA préconise l'utilisation du standard UML. UML définit un ensemble de diagrammes permettant de modéliser tous les aspects d'une application orientée objets (aspect statique, dynamique, fonctionnel, …). Pour chaque modèle MDA correspond un diagramme UML approprié :

CIM

Les modèles UML utilisés pour les modèles d'exigence sont : le diagramme de cas d'utilisations et le diagramme de séquences. Ces diagrammes permettent d'exprimer les besoins de l'utilisateur et sont indépendants des plateformes d'exécution.

PIM

Les modèles UML utilisés pour la phase analyse et conception abstraite sont : le diagramme de classes, le diagramme d'activités, le diagramme d'états et le diagramme de séquences. Ces diagrammes permettent de modéliser la structure et la dynamique de

l'application indépendamment de la plateforme d'exécution.

PSM

Les modèles UML utilisés pour la conception concrète sont : le diagramme de classes et le diagramme de composants. MDA conseille l'utilisation des profils UML pour l'élaboration des PSM. Un PSM basé sur un profil UML est un modèle UML mais dit d'application, il est spécifique à la plateforme décrite par le profil UML.

L'autre approche possible et aussi recommandée par MDA est l'utilisation des modèles de plateforme. Le PM (Platform Model) d'une plateforme particulière associé au PIM de l'application permet d'obtenir le PSM de la plateforme décrite par le PM.

Les profils UML présentent l'avantage de faciliter la transformation PIM vers PSM par rapport aux PM, mais ces derniers permettent une plus grande expressivité pour les plateformes.

UML modélise une application orientée objets sous forme d'un ensemble d'objets liés par des relations, mais il ne permet pas de définir le comportement de ces objets. C'est pour corriger cette lacune que l'OMG a défini les standards OCL (Object Constraint Language) et AS (Action Semantics). UML décrit une opération d'un objet que par son nom, ses paramètres et les exceptions qu'elle lève, OCL et AS se greffent à UML pour ajouter plus de précision à cette description en offrant les outils pour permettre la définition du corps de ces opérations. Les standards OCL et AS sont indépendants des plateformes d'exécution, ils sont principalement utilisés pour l'élaboration des PIM :

OCL

OCL est un langage textuel de contraintes, il permet de formuler des contraintes sur n'importe quel élément d'un modèle UML. Il est particulièrement utile pour exprimer les pré- et post- conditions d'une opération d'un objet. Une contrainte OCL est une expression qui exprime une condition. Elle ne permet cependant pas de créer, modifier ou supprimer des éléments du modèle UML. Elle ne génère donc aucun effet de bord. Le résultat de l'évaluation d'une expression OCL est de type booléen (vrai ou faux), il permet de savoir si la contrainte est oui ou non respectée. Avant sa version 2.0, OCL utilise le langage naturel (l'anglais), aucune règle n'est posée quant à l'expression de ces conditions. Avec l'arrivée d'OCL2.0, les contraintes OCL sont désormais définies par un métamodèle. Ce qui permet aux modèles de contraintes OCL d'être pérennes, productifs et intégrés à MDA. OCL complète UML dans l'expression de ses modèles. Malgré qu'il permette d'exprimer des contraintes sur des éléments UML, OCL n'est pas un langage de programmation.

AS

L'OMG à définit aussi le langage AS. AS permet de spécifier des actions de type création,

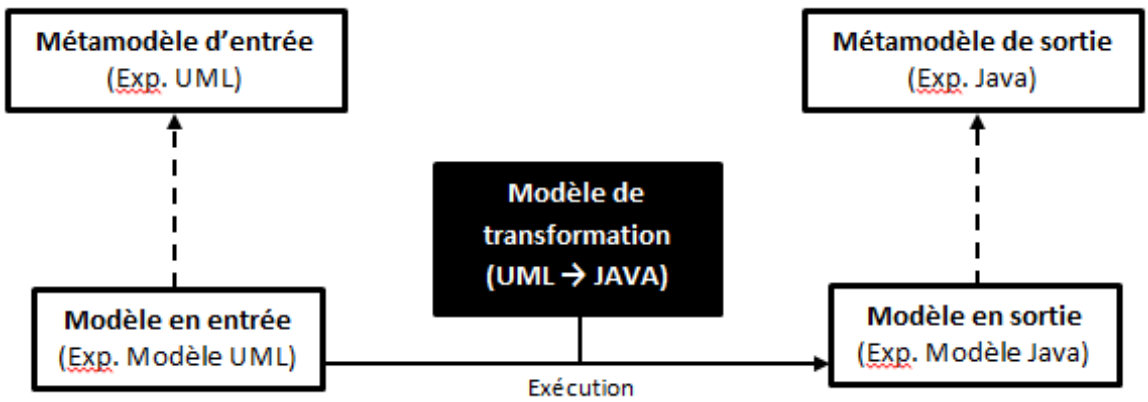
suppression ou encore modification sur n'importe quel élément d'un modèle UML. Une action AS, modifie l'état du modèle concerné après son exécution. Avec AS, les modèles UML gagnent en expressivité.

Comme pour OCL, le standard AS s'appuie sur un métamodèle, mais aucune syntaxe précise n'est définie pour la formulation des actions AS. Chacun est donc libre de choisir la syntaxe qu'il souhaite.

AS a été proposé pour enrichir le corps des opérations UML, d'actions, qui ne pouvaient être formulées par UML ou par OCL, puis être traduites vers un langage de programmation. Grâce à la nature des actions que propose AS (création modification, suppression), il est alors même possible d'utiliser AS pour la transformation de modèles.

Les transformations

L'approche MDA préconise de modéliser les transformations elles-mêmes. MDA considère une transformation comme une application. Un modèle de transformation est une fonction qui prend un ensemble de modèles en entrée et rend un ensemble de modèles en sortie. Les modèles, en entrée et en sortie, sont structurés par leur métamodèle.



Il existe trois façons différentes de modéliser une fonction de transformation. Le principe reste le même, la différence réside dans la façon de formuler les règles de transformations :

Approche par programmation

Dans cette approche, l'idée est de programmer une transformation de modèle de la même façon que n'importe quel application informatique, en utilisant les langages orientés objets. Et où les données à manipuler représentent les modèles impliqués dans la transformation.

Approche par template

L'approche par template consiste à définir des modèles templates et à remplacer leurs

paramètres par les valeurs des modèles sources. Les modèles templates, ou modèles cibles paramétrés, sont des canvas des modèles cibles, ils contiennent des paramètres qui seront par la suite, lors de la transformation, substitué par les valeurs des modèles sources. Cette approche utilise un langage particulier pour la définition des modèles templates.

Approche par modélisation

Par analogie à l'approche MDA, l'approche par modélisation modélise aussi les règles de transformations en se basant sur l'ingénierie dirigée par les modèles. Dans cette approche, un modèle de transformations est structuré par son métamodèle, et les modèles sont indépendants des plateformes d'exécution. L'objectif de cette approche est de pérenniser les modèles de transformations et de les rendre productifs. L'OMG a défini le standard MOF2.0 QVT (Query/View/Transformation) comme métamodèle pour structurer les modèles de transformations de modèles. Les modèles de transformations, conformes au métamodèle MOF2.0 QVT, expriment les règles de correspondance entre le métamodèle source et le métamodèle cible nécessaires à la transformation.

Informatisation des modèles

Les modèles sont une entité abstraite et n'ont pas d'existence physique à proprement dit. Pour pouvoir les manipuler, il est nécessaire de pouvoir les informatisés. La représentation informatique des modèles dans l'approche MDA se fait soit de façon textuelle, plus adaptée aux stockages des modèles et aux échanges entre applications. Soit sous forme d'objets de programmation, plus adaptée aux manipulations informatiques des modèles.

La représentation textuelle des modèles se fait par le standards XMI (XML Metadata Interchange). Les modèles sont représentés sous forme de document XML ce qui favorise leur pérennité. La représentation des modèles sous forme d'objets Java est réalisée avec le framework EMF (Eclipse Modeling Framework). EMF offre la possibilité d'effectuer des opérations sur les modèles (transformation, validation, …).

Par analogie à la relation "modèle - métamodèle", XMI et EMF génèrent automatiquement, à partir du métamodèle du modèle à représenter, le formalisme de représentation des modèles. C'est à dire, pour représenter le modèle sous forme de document XML, XMI génère sa DTD à partir du métamodèle du modèle. Le document XML qui définit le modèle est donc structuré par la DTD générée à partir du métamodèle. Pour représenter le modèle sous forme d'objets Java, EMF génère un ensemble de classes Java à partir du métamodèle du modèle. Les objets Java, qui expriment le modèle, sont des instances des classes générées à partir du métamodèle.

Objectifs de l'approche MDA

L'OMG a définit l'approche MDA pour répondre aux problèmes liés à l'évolution continues des

technologies. MDA est à l'origine de l'ingénierie dirigée par les modèle. Tout ce qui constitue l'approche MDA, la modélisation, les transformations, les technologies de mises en oeuvres, … ont étaient pensés afin de permettre à MDA de remplir les objectifs suivants :

La pérennité des savoir-faire

La technologie évolue plus vite que les métiers de l'entreprise. Il semble alors plus intéressant de capitaliser les savoir-faire de l'entreprise indépendamment des préoccupations techniques. L'approche MDA vise principalement à rendre les spécifications métiers des entreprises pérennes, indépendamment des technologies de mise en oeuvre. Cette pérennité est possible grâce aux modèles, qui sont par nature des entités pérennes, et à l'utilisation d'UML qui est un standard stable et largement répandu.

Les gains de productivité

Les modèles sont au cœur du processus MDA, ils facilitent la communication entre experts du domaine et développeurs, mais pas seulement. Dans MDA les modèles deviennent un outil de production grâce à l'automatisation des transformations de modèles. Le modèle devient un élément qui véhicule une information utile, précise et nécessaire, permettant l'obtention du code source de l'application par génération automatique du code. L'automatisation des transformations que permettent les modèles dans le processus de développement MDA, entraîne un gain de productivité.

La prise en compte des plateformes d'exécution

A partir du PIM, MDA permet d'intégrer les spécifications techniques d'une plateforme d'exécution dans les transformations PIM vers PSM, et d'obtenir ainsi un PSM spécifique à la plateforme d'exécution, à partir duquel le code source de l'application est généré. Grâce à ce procédé, une application tire pleinement partie des fonctionnalités de la plateforme d'exécution cible. Le développement d'applications multiplateforme, ou encore la migration logicielle, sont facilitées puisqu'il suffit à partir du PIM, d'effectuer les transformations PIM vers PSM en y intégrant à chaque fois, le modèle de la plateforme concernée. Obtenant ainsi pour chaque plateforme, son PSM à partir duquel le code source sera généré.

Quelques outils

Voici quelques uns des outils disponibles sur le marché pour le développement d'applications par l'approche MDA.

- Rational Rose est un modeleur qui permet la création et l'édition de diagrammes UML. Il peremt également de générer le code source Java ou C++ de ces modèles.

- ArgoUML est un logiciel de création de diagrammes UML programmé en Java. Il est multiplateformes. Il permet aussi la génération de code vers Java, C++, PHP, C# et SQL.

- BOUML est un modeleurs de diagrammes UML. Il est multipalteformes et permet la génération de code source vers Java, C++, Python, MySQL. Il supporte aussi la rétro-ingénierie.

- MagicDraw est un modeleur commercial. C'est un outil graphique de modélisation UML et dispose de nombreuses fonctionnalités.

- Acceleo est un générateur de code permettant de traduire les modèles en code source. Il est aussi compatible avec de nombreux modeleurs.

MDA en entreprise

Sur le site de l'OMG, est disponible à cette adresse un ensemble de *Success Stories* relatant de l'expérience de diverses organisations dans leur applications de l'approche MDA. En voici quelques unes :

Swisslog Software AG

Swisslog est un fournisseur de solutions intégrées pour l'automatisation logistique. Elle propose des services dans le cadre de la planification et la mise en œuvre de solutions logistiques intégrées en milieu hospitalier, et propose des services complets pour l'automatisation des centres de distribution et des entrepôts.

Swisslog fait le choix de l'approche MDA pour apporter flexibilité, adaptabilité et un champ d'application plus vaste au développement d'une nouvelle génération de systèmes de gestion. Dans ce projet, MDA à permit de formuler des modèles d'application indépendant des technologies d'implémentation, s'adaptant plus facilement aux nouvelles technologies et plateformes. Avec MDA, Swisslog se voit tirer pleinement partie de l'expertise du domaine, de l'architecture et des fonctionnalités du support technique utilisé pour l'implémentation de ses logiciels.

Lockheed Martin

Lockheed Martin est la première entreprise américaine et mondiale de défense et de sécurité. Ses activités englobent la création et la fabrication de véhicules, plus particulièrement des avions et des voitures militaires.

Lockheed Martin fait appel à l'approche MDA pour le développement du programme informatique F-16 Modular Mission Computer. Leur but est d'assurer un support multiplateforme au système informatique F-16 MMC, d'augmenter la productivité et la qualité. Dans un contexte aussi exigent que le développement de logiciels avioniques, l'approche MDA a su démontrer son efficacité. Le temps de développement s'est vu réduit de 20% sur le programme F-16 MMC, qui a pu atteindre une compatibilité multi-plateformes complète. Cette efficacité est due entre autre à la génération automatique de code qui évite les bugs habituellement introduits dans la phase de

codage manuel. Ou encore aux modèles d'application exprimés d'une manière totalement indépendante des plateformes d’exécution, les supports matériel et logiciel peuvent être mis à niveau sans impact sur les modèles d'application.