

Introduction à MDA : Principe

par [Pierre Parrend](#)

Date de publication : 04/12/2006

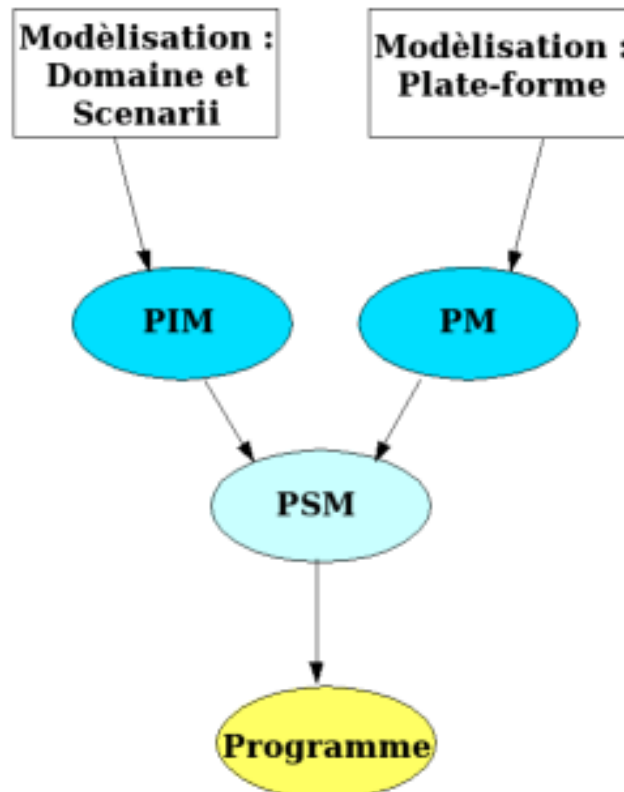
Dernière mise à jour :

Introduction générale au développement orienté modèle (MDA - Model Driven Architecture).

- I - Ce qu'est MDA : Le Principe
- II - Les Outils de MDA
- III - MDA dans la Pratique
 - III-A - La modélisation
 - III-B - L'intégration
 - III-C - La transformation
 - III-D - La génération de code
 - III-E - Le stockage et l'accès
- IV - Les Transformations
 - IV-A - Les Méta-modèles
 - IV-A-1 - Les niveaux de méta-modèles
 - IV-B - Les Mappings
 - IV-B-1 - Les mappings verticaux
 - IV-B-2 - Les mappings horizontaux
- V - Une Méthode
 - V-A - Approche linéaire
 - V-B - Approche incrémentale

I - Ce qu'est MDA : Le Principe

Il s'agit de modéliser l'application que l'on veut créer de manière indépendante de l'implémentation cible (niveau matériel ou logiciel). Ceci permet une grande réutilisation des modèles.



Les modèles ainsi créés (PIM - Platform Independent Model) sont associés à des modèles de plate-forme (PM - Platform Model), et transformés, pour obtenir un modèle d'application spécifique à la plate-forme (PSM - Platform Specific Model).

Des outils de génération automatique de code permettent ensuite de créer le programme directement à partir des modèles.

Cette approche permet de plus de faire évoluer facilement, à partir des modèles, les applications : le développement d'un nouveau module, quand tous les modèles nécessaires sont disponibles, peut ne pas prendre plus de quelques minutes.

II - Les Outils de MDA

Pour obtenir une telle efficacité, plusieurs outils conceptuels sont mis à disposition. La technologie MDA (Model Driven Architecture) est supportée par l'OMG (Object Management Group), qui propose également UML (Unified Modeling Language) et Corba (Object Request Broker).

Ces outils sont :

- **UML**, largement utilisé par ailleurs, qui permet une mise en oeuvre aisée de MDA en offrant un support connu,
- **XMI**, XML Metadata Interchange, qui propose un formalisme de structuration des documents XML de telle sorte qu'ils permettent de représenter des méta-données d'application de manière compatible,
- **MOF**, Meta Object Facility, spécification qui permet le stockage, l'accès, la manipulation, la modification, de méta-données,
- **CWM**, base de données pour méta-données.

L'OMG n'a pas jugé utile de standardiser un processus associé à ces outils. Leur rôle est de répondre aux besoins des utilisateurs de manière générique, et non de proposer de solutions définitives pour certains types d'applications précises.

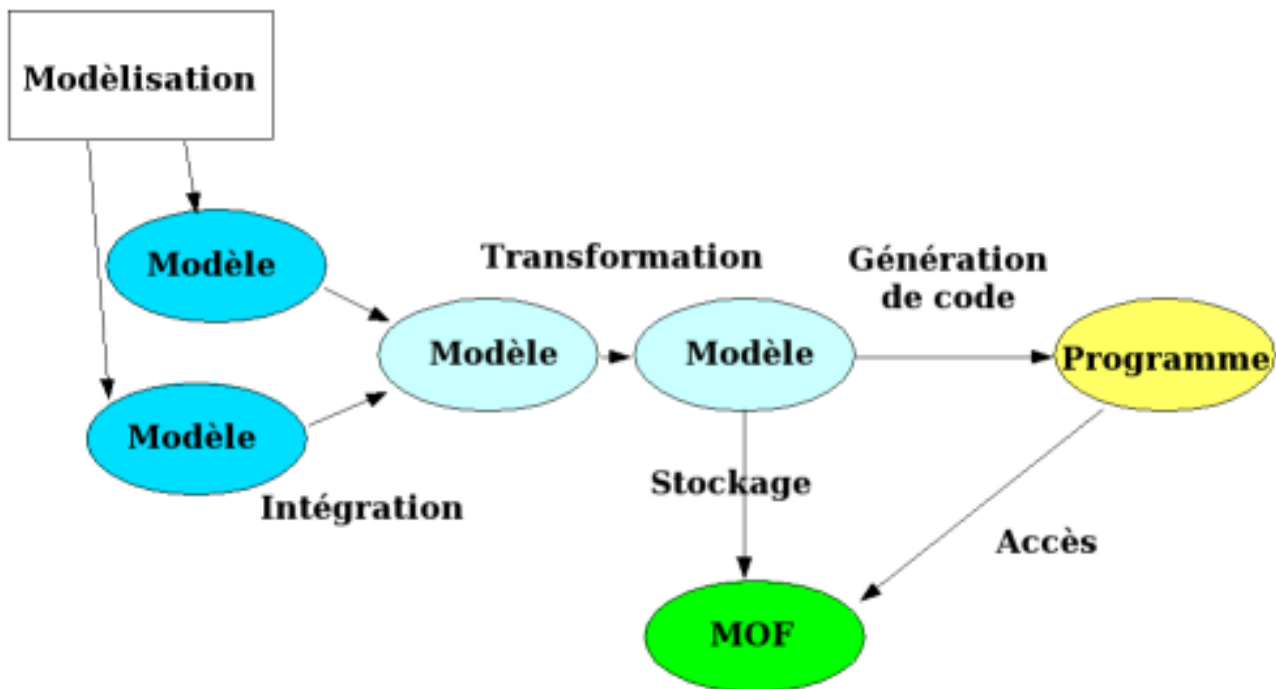
Un processus de génie logiciel exploitant les possibilités de MDA a cependant été proposé : le 'Model-Driven Software Development' (<http://www.mdsd.info/>).

III - MDA dans la Pratique

Un certain nombre d'étapes et de transformation sont identifiables dans un processus MDA :

- modélisation UML,
- transformation intermédiaires,
- génération de code.

Cette suite de transformations est illustrée par l'image suivante :



III-A - La modélisation

C'est une étape préalable. Elle se fait en utilisant les outils conceptuels d'UML (Unified Modeling Language). Elle aboutit à un ensemble de modèles de la future application, représentés par des diagrammes de classes. Chaque domaine fonctionnel de l'application est représenté indépendamment des autres.

A partir de ces diagrammes de classes UML, des modèles manipulables sont générés (au format XMI).

III-B - L'intégration

Plusieurs modèles sont utilisés pour la création d'une application. Typiquement, il s'agit de modèles représentant des fonctionnalités différentes. Ils sont assemblés en un seul modèle, qui représente l'application, et sont à partir de là manipulables comme un unique modèle.

III-C - La transformation

A partir de modèles génériques, il s'agit de préciser ce que sera l'application : format de données, réalisation des fonctionnalités. Le(s) modèle(s) de l'application sont donc complétés, affinés.

Typiquement, il s'agit à ce niveau là de transformer un méta-modèle (modèle de modèle, qui indique les contraintes que doit respecter l'application), en modèle fonctionnel, dotés de services particuliers.

Il peut également s'agir de transformer un modèle fonctionnel indépendant de l'application (PIM - Platform Independent Model) en modèle prenant en compte les contraintes de déploiement (PSM - Platform Specific Model).

III-D - La génération de code

Lorsque le modèle est complet, le code est généré. L'application peut alors être déployée.

Dans la pratique, le code généré doit être complété : l'implémentation des différentes méthodes, par exemple, n'est pas explicitée dans le modèle.

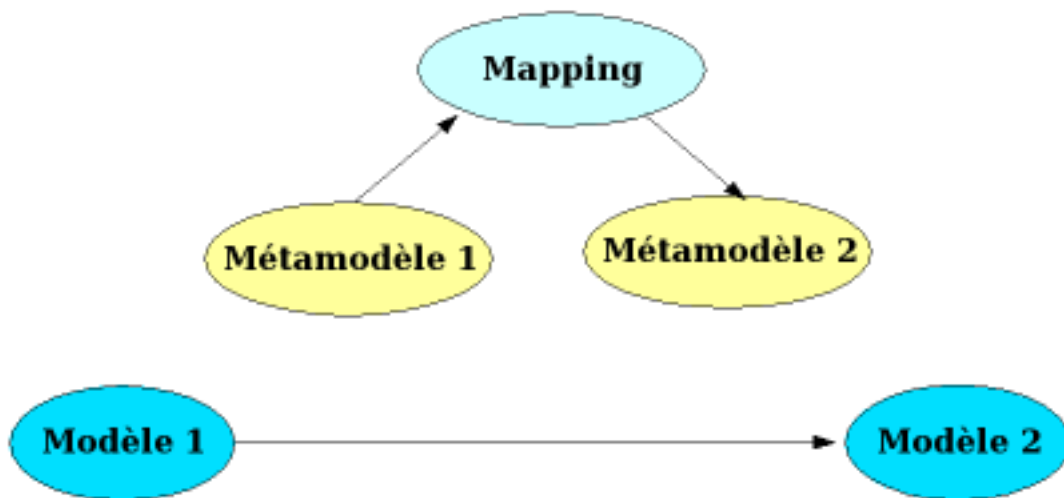
En cas d'extension ultérieure du modèle, il est indispensable de disposer d'un environnement de développement qui conserve le code ajouté. Si ce n'est pas le cas, le développement incrémental est rendu beaucoup plus laborieux, et la création de grandes applications est compromise.

III-E - Le stockage et l'accès

Les modèles peuvent être réutilisés pour le développement d'autres applications, mais ils sont également accessibles par les applications elles-mêmes. La spécification MOF (Metadata Object Facility) permet de définir des bases de données de modèles accessibles de manière transparente par les applications.

IV - Les Transformations

La transformation de modèle MDA se fait par mapping entre un modèle initial et un modèle cible. Chaque modèle doit être décrit par un méta-modèle, qui recense les caractéristiques de ce modèle. Le mapping est alors défini comme une traduction entre le méta-modèle initial et le méta-modèle cible. La figure suivante présente le principe de la transformation.



Pour effectuer la transformation, un moteur de transformation est indispensable.

IV-A - Les Méta-modèles

Les méta-modèles sont exprimés couramment sous format XML. Ils peuvent représenter un langage connu, par exemple UML ou Java. Ils peuvent également être conçus spécifiquement pour un domaine d'application donné.

Deux approches existent pour créer un langage de méta-modèle :

- **MOF** : On crée à partir de rien un langage nouveau, en respectant les spécifications du MOF (Meta Object Facility),
- **UML** : On crée une extension d'UML, à l'aide de stéréotypes, de contraintes (langage OCL).

IV-A-1 - Les niveaux de méta-modèles

On définit quatre niveaux de méta-modèles :

- le niveau zéro est l'implémentation du programme,
- le niveau 1 est le modèle du programme (classes)
- le niveau 2 est le méta-modèle du programme, c'est à dire le langage
- le niveau 3 est le méta-modèle des méta-modèles, ou méta-méta-modèle, c'est à dire la définition d'un méta-modèle. Il s'agit donc de la spécification MOF.

Il faut noter que les niveaux ne sont pas absolu : en cas de nécessité, on peut avoir plusieurs modèles intermédiaires.

M3	MOF – Meta modèle de définition de langage
M2	Modèle de de langage
M1	Classes d'un programme
M0	Instanciation

IV-B - Les Mappings

On distingue deux types de mappings : les mapping verticaux, qui changent le niveau d'abstraction, et les mapping horizontaux, qui le conservent.

IV-B-1 - Les mappings verticaux

On en distingue deux sortes :

- Mappings de modèle d'analyse vers le modèle d'implémentation (ou mappings de raffinement). Ils permettent de préciser le modèle de l'application, en fonction de l'implémentation souhaitée. Souvent, un seul modèle initial suffit. Parfois, des contraintes sont nécessaires (sinon le modèle cible est incomplet), ou bien plusieurs modèles sont utilisés comme source.
- Mappings d'abstraction. C'est la transformation inverse. Elle permet une meilleure compréhension du code (reverse engineering), ou la migration d'une plate-forme vers une autre.

IV-B-2 - Les mappings horizontaux

- Mappings de représentation. On l'utilise pour passer d'un format à un autre, le second disposant de plus d'outils de manipulation et de représentation. Peu utile s'ils ne sont pas réversibles.
- Mappings d'optimisation, pour améliorer la performance.
- Mappings de reconstruction, pour améliorer la maintenabilité et la lisibilité du code.

Les mappings sont réalisés soit par des règles générales, soit par corrélation, c'est à dire par appariement de propriétés du modèle source avec des propriétés du modèles cible.

V - Une Méthode

Maintenant que les outils et les principes de MDA ont été présentés, voici une méthode de développement d'application correspondante. Elle a pour vocation d'améliorer la productivité, c'est à dire le temps de création de la première version de l'application, de même que l'évolutivité, c'est à dire la possibilité d'améliorer et d'étendre cette application.

On distinguera une approche linéaire, exploitable pour des petites applications ou des sous-systèmes, et une approche incrémentale, qui doit permettre de mettre en oeuvre des applications de plus grandes dimensions.

V-A - Approche linéaire

Phase de spécifications

- choix des principes de l'application (et donc des modèles)
- adaptation de ces modèles
- validation de l'architecture au niveau fonctionnel

Phase de design

- raffinement automatique (intégration de modèles existants)
- raffinement manuel

Phase d'implémentation

- génération de code pour la plate-forme-cible
- complétion du code généré
- intégration d'outils pré-existants

Phase de validation

- bon fonctionnement de l'application
- validation des spécifications

V-B - Approche incrémentale

Cette approche met en évidence la puissance de MDA. L'évolution permanente du logiciel est intégrée dans la méthodologie de développement, ce qui a deux avantages principaux :

- lors de la première réalisation d'une application, des versions fonctionnelles intermédiaires sont disponibles, ce qui permet d'accélérer la mise en production - éventuellement d'une version incomplète, préférable à un retard pur et simple,
- lors de l'évolution d'une application, la méthodologie est conservée, et la validation est donc facilitée.

Prototype

Réalisation d'une architecture comportant les fonctionnalités minimales (IHM et fonctionnalités clés) selon la méthode linéaire.

Architecture complète

Intégration des différents principes de fonctionnement : type de client (lourd/léger), support de mobilité, etc., selon la méthode linéaire.

On obtient un squelette d'application, comportant tous ses éléments, mais sans implémentation.

Mise en place des fonctionnalités

Compléter chaque fonctionnalité l'une après l'autre, selon la méthode linéaire.

Récurtivité

Les étapes 2, 3 peuvent être réalisés par suite d'affinements successifs. Chaque affinement doit permettre de nouveaux types d'usage (extension des fonctionnalités), les version intermédiaires doivent être exploitables.

Validation de la version finale

Evolutions

Elles peuvent se faire selon le même principe que les affinements succesifs ayant conduit à la version complète du produit.