

# CHAPITRE 1

## IDM ET L’APPROCHE MDA

### 1.1 Introduction

Au cours des dernières décennies, le développement de grands projets d'ingénierie, toujours plus complexes, a mis en évidence la nécessité de disposer d'outils, de méthodes et de processus permettant d'en assurer la maîtrise tout au long de leur cycle de vie.

L'ingénierie dirigée par les modèles (IDM), d'abord utilisée principalement dans le domaine des systèmes logiciel, a permis plusieurs améliorations significatives dans le processus de développement de systèmes complexes en se concentrant sur des préoccupations plus abstraites autour des modèles utilisés que sur la programmation classique (le code). Il s'agit donc d'une forme d'ingénierie générative dans laquelle tout ou partie d'une application est engendrée à partir de modèles. Un modèle est une abstraction, une simplification d'un système qui est suffisante, non seulement pour comprendre le système modélisé, mais également pour garantir son bon fonctionnement.

Dans la suite de ce chapitre, nous présentons les concepts essentiels ainsi que la terminologie que nous utiliserons tout au long de ce manuscrit. Nous abordons les principes clés de l'ingénierie IDM et les différentes variantes d'ingénierie centrées sur les modèles. Nous verrons que la maîtrise de la sûreté de fonctionnement joue une part prépondérante dans la conception et le développement des systèmes complexes.

### 1.2 Principes généraux de l’IDM

Suite à l'approche objet des années 80 et de son principe du « tout est objet », l'ingénierie du logiciel s'oriente aujourd'hui vers l'ingénierie dirigée par les modèles (IDM) et le principe du « tout est modèle ». Cette nouvelle approche peut être considérée à la fois en continuité et en rupture avec les précédents travaux. Tout d'abord en continuité car c'est la technologie objet qui a déclenché l'évolution vers les modèles. En effet, une fois acquise la conception des systèmes informatiques sous la forme d'objets communicant entre eux, il s'est

posé la question de les classer en fonction de leurs différentes origines (objets métiers, techniques, etc.).

L'IDM vise donc, de manière plus radicale que pouvaient l'être les approches des patterns et des aspects, à fournir un grand nombre de modèles pour exprimer séparément chacune des préoccupations des utilisateurs, des concepteurs, des architectes, etc. C'est par ce principe de base fondamentalement différent que l'IDM peut être considérée en rupture par rapport aux travaux de l'approche objet.

Alors que l'approche objet est fondée sur deux relations essentielles, «InstanceDe »et « HériteDe», l'IDM est basée sur un autre jeu de concepts et de relations. Le concept central de l'IDM est la notion de modèle pour laquelle il n'existe pas à ce jour de définition universelle. Néanmoins, de nombreux travaux s'accordent à un relatif consensus d'une certaine compréhension.

L'intérêt pour l'IDM a été fortement amplifié à la fin du 20<sup>ième</sup> siècle, lorsque l'organisme de standardisation OMG (Object management Group) a rendu publique son initiative MDA (Model Driven Architecture), qui peut être vue comme une restriction de l'IDM à la gestion de l'aspect particulier de dépendance d'un logiciel à une plateforme d'exécution. [1]

### 1.3 OMG et MDA

L'**Object Management Group** est une association américaine, fondée en 1989 et compte aujourd'hui un consortium de plus de 1000 entreprises du monde entier. Parmi les réalisations de l'OMG sont le Common Object Request Broker Architecture (CORBA), Unified Modeling Language (UML), Meta Object Facility (MOF), IDL (Interface Definition Language), XML Metadata Interchange (XMI) et le méta-modèle d'entrepôt commun (MCG). Toutes ces normes contribuent à rendre l'idée d'un modèle de développement axé sur une réalité. Environ vers 2001 OMG adopté un nouveau cadre appelé le modèle Driven Architecture (MDA). Contrairement aux autres standards de l'OMG le MDA offre un moyen d'utiliser des modèles au lieu du code source traditionnelle. Il reste à voir si cette nouvelle façon de développement logiciel sera acceptée parmi les développeurs et les entreprises. [2]

## 1.4 L'Approche MDA

### 1.4.1 Concepts de base

Regardons de près les différents concepts préliminaires de MDA. Pour bien comprendre leurs significations, nous proposons quelques définitions :[3]

➤ **Système** : on parle toujours de MDA dans le cadre d'un système existant ou à réaliser.

Un système peut tout inclure : un programme, une certaine combinaison de différentes parties de différents systèmes, une fédération de systèmes autonomes, une fédération d'entreprises, .... Toutefois, dans les systèmes, on s'intéresse surtout aux logiciels.

➤ **Point de vue** : un point de vue sur un système est une vision totale sur ce système S'intéressant à un aspect particulier. Par exemple, nous pouvons nous intéresser aux Flux d'informations, et donc décrire uniquement le cheminement des données dans Tout le système. La norme RM-ODP définit cinq points de vue :

- **Point de vue Entreprise** : il représente le logique métier (le Pourquoi). Il exprime les objectifs, les droits et les obligations des entités qui composent une application.

- **Point de vue Information** : il représente le système d'information du système (le Quoi). Il exprime la sémantique de l'application à l'aide de structures de données, de fonctions les manipulant et de propriétés d'intégrité sur ces données.

- **Point de vue Traitement** : il représente la conception fonctionnelle de l'application (Le Comment). Il exprime une vision définissant des entités fonctionnelles, en précisant leurs interactions et en déterminant les propriétés des fonctions correspondantes.

- **Point de vue Ingénierie** : il représente les plates-formes support de l'application (Le Où). Il décrit les fonctions et services de base que doit fournir l'infrastructure d'exécution pour que l'on puisse construire les mécanismes assurant l'interopérabilité en environnement hétérogène.

- **Point de vue Technologie** : il représente les produits que va utiliser l'application (Le Avec Quoi). Il expose les contraintes technologiques imposées par les mécanismes d'ingénierie correspondant à une infrastructure matérielle ou logicielle spécifique.

➤ **Architecture** : l'architecture d'un système est la spécification des parties et des connecteurs du système, ainsi que les règles d'interaction entre ces parties, utilisant les connecteurs.

➤ **Modèle** : le modèle d'un système est la spécification formelle des fonctions, de la structure et/ou du comportement de ce système dans son environnement, dans un certain but. Un

modèle est souvent représenté par des schémas et du texte. Le texte peut être exprimé dans un langage de modélisation ou en langage naturel.

- **Application** : le terme application désigne une fonctionnalité qui est en cours de développement. Un système décrit une ou plusieurs application(s) supportée (s) par une ou plusieurs plates-formes.
- **Plate-forme** : une plate-forme est un ensemble de sous -systèmes et de technologies, qui fournissent aux applications supportées, un ensemble cohérent de fonctionnalités à travers des interfaces et des gabarits masquant leurs détails d'implémentation.

Les plates-formes peuvent être de type objet (supporte les objets avec interfaces, requête/réponse aux services), de type batch (supporte une séquence de programmes indépendants s'exécutant consécutivement) ou de type flux de données (supporte un flux continu de données entre les composants logiciels).

Les plates-formes peuvent supporter plusieurs technologies (tels que CORBA, Java 2 Entreprise Edition) et sont implémentées par plusieurs fournisseurs (tels que WebSphere, WebLogic)

### 1.4.2 Principe de MDA

En novembre 2000, l'OMG, consortium de plus 1000 entreprises, initie la démarche MDA. En Dans la littérature le terme « démarche », « norme » ou « approche » qualifie le MDA. Il est intéressant de noter que MDA n'a pas encore de genre.

Ce standard a pour but d'apporter une nouvelle vision unifiée de concevoir des applications en séparant le logique métier de l'entreprise, de toute plate-forme technique.

En effet, la logique métier est stable et subit peu de modifications au cours du temps, Contrairement à l'architecture technique. Il est donc évident de séparer les deux pour faire face à la complexité des systèmes d'information et aux coûts excessifs de migration technologique. Cette séparation autorise alors la capitalisation du savoir logiciel et du savoir-faire de l'entreprise. A ce niveau, l'approche objet et l'approche composant n'ont pas su tenir leurs promesses. Il devenait de plus en plus difficile de développer des logiciels basés sur ces technologies. Le recours à des procédés supplémentaires, comme le patron de conception ou la programmation orientée aspect, était alors nécessaire.

Le standard MDA doit aussi offrir la possibilité de stopper l'empilement des technologies qui nécessite de conserver des compétences particulières pour faire cohabiter des systèmes divers et

variés. Ceci est permis grâce au passage d'une approche interprétative à une approche transformationnelle. Dans l'approche interprétative, l'individu a un rôle actif dans la construction des systèmes informatiques alors que dans l'approche transformationnelle, il a un rôle simplifié et amoindri grâce à la construction automatisée. [4]

### 1.4.3 Architecture général de MDA

Sur la Figure 1.1, l'architecture du MDA se découpe en quatre couches. Dans la Première couche, se trouvent le standard UML (Unified Modeling Language), MOF (Meta-Object Facility) et CWM (Common Warehouse Metamodel), Dans la couche suivante, se trouve aussi un standard XMI (XML Metadata Interchange) qui permet le dialogue entre les middlewares (Java, CORBA, .NET et web services). La troisième couche contient les services qui permettent de gérer les évènements, la sécurité, les répertoires et les transactions. Enfin, la dernière couche propose des Framework adaptables à différents types d'applications à savoir Finance, Télécommunication, Transport, Espace, médecine, commerce électronique et Manufacture,...). [3]

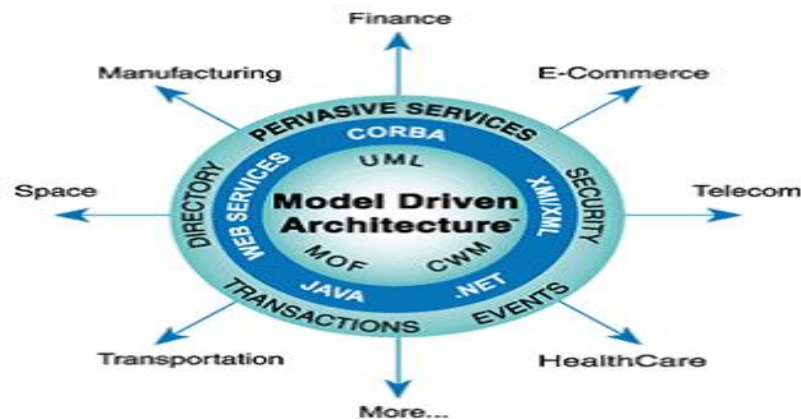


Figure 1.1 : architecture général de MDA

#### 1.4.4 Architecture MDA à quatre niveaux

L'OMG a défini une architecture à quatre niveaux d'abstraction, comme carte général pour l'intégration des méta-modèles, en se basant sur l'MOF comme le montre la figure 1.2. Dans cette architecture, les modèles de deux niveaux adjacents sont liés par une relation d'instanciation : [5]

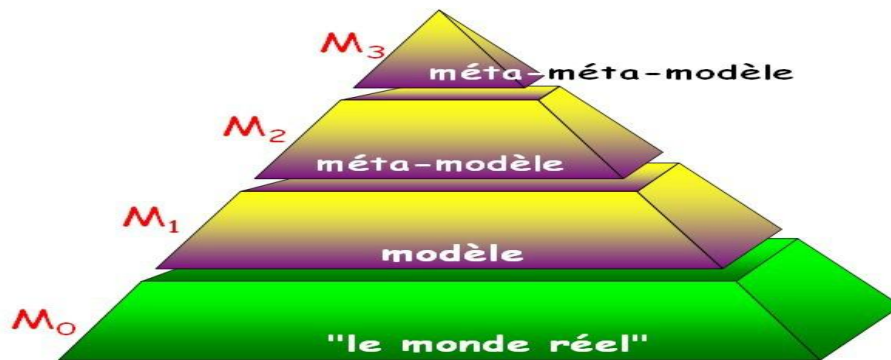


Figure 1.2 : Les quatre niveaux d'abstraction pour MDA

- **Le niveau M0** : Niveau des instances des modèles. Il définit des informations pour la modélisation des objets du monde réel.
- **Le niveau M1** : Ce niveau représente toutes les instances d'un méta-modèle. Les Modèles du niveau M1 doivent être exprimés dans un langage défini au niveau M2. UML est un exemple de modèles du niveau M1.
- **Le niveau M2** : Ce niveau représente toutes les instances d'un méta-méta-modèle. Il est composé de langages de spécifications de modèles d'information. Le méta-modèle UML qui est décrit dans le standard UML et qui définit la structure interne des modèles UML, appartient au niveau M2.
- **Le niveau M3** : Ce niveau définit un langage unique pour la spécification des Méta-modèles. Le MOF élément réflexif du niveau M3, définit la structure de tous les Méta-modèles du niveau M2.

### **1.4.5 Modèle, Méta-modèle et méta-méta-modèle dans MDA.**

#### **1.4.5.1 Modèle**

Ici, plusieurs définitions se côtoient. Nous retiendrons celle qui semble couvrir le mieux notre propos. [6]

Un modèle est une simplification d'un système, construit avec une intention de comprendre le système en adoptant un point de vue défini. Le modèle répondra à des questions à propos du système.

On précise cette définition:

- un modèle n'est qu'une représentation partielle (un aspect) du système (le réel),
- un système peut donc avoir plusieurs modèles. [7]

Généralement un modèle est une spécification ou une description de la fonction, de la structure et /ou du comportement d'un système et son environnement. Un modèle est souvent présenté par une combinaison de schémas et du texte, exprimée dans un langage de modélisation.

Dans l'architecture MDA, nous pouvons considérer que les modèles MDA sont des représentations, à différents niveaux d'abstraction, de l'information nécessaire à la production et à l'évolution d'applications informatiques. Les modèles réalisés dans MDA concourent donc tous plus ou moins selon leur niveau d'abstraction à la production ou à l'évolution d'applications informatiques. Comme tout modèle, les modèles MDA doivent être fortement connectés avec la réalité, en l'occurrence avec les applications informatiques. [8]

#### **1.4.5.2 Méta-modèle**

Un méta-modèle définit la structure que doit avoir tout modèle conforme à ce méta-modèle. Autrement dit, tout modèle doit respecter la structure définie par son méta-modèle. Par exemple, le méta-modèle UML définit que les modèles UML contiennent des packages, leurs packages des classes, leurs classes des attributs et des opérations.

Les méta-modèles fournissent la définition des entités d'un modèle, ainsi que les propriétés de leurs connexions et de leurs règles de cohérence. MOF les représente sous forme de diagrammes de classes.

Les diagrammes de classes permettent de représenter les notions d'un domaine et leurs propriétés, que ces notions ou entités soient organisées ou non sous forme d'objets. Cette

utilisation des diagrammes de classes a le double avantage de permettre de définir très précisément les méta-modèles et de les rendre eux aussi pérennes et productifs.

Un méta-modèle est donc une sorte de diagramme de classes qui définit la structure d'un ensemble de modèles. Les méta-modèles permettent de pérenniser la structure des modèles car ils sont eux-mêmes représentés sous forme de modèles (diagrammes de classes). Ils permettent d'associer des traitements aux modèles grâce notamment aux méta-opérations des méta-classes. Les liens entre méta-modèles permettent enfin de bien séparer les aspects métier des aspects techniques et donc de prendre en compte les plates-formes d'exécution. [8]

#### 1.4.5.3 Méta-méta-modèle

Un méta-modèle est un modèle, il est donc conforme à son propre méta-modèle. Par exemple, dans le cas d'XML, c'est une DTD XML qui joue le rôle de méta-méta-modèle pour les DTD.

**Remarque :** Pour limiter le nombre de niveaux d'abstraction, et éviter de définir un méta-méta-méta modèle, les méta-méta-modèles sont généralement conçus pour être *auto-descriptifs*, c'est-à-dire capables de se décrire eux-mêmes. [6]

#### 1.4.6 Modèles de MDA

##### 1.4.6.1 Modèle d'exigence CIM (*Computation Independent Model*)

Le CIM est l'abréviation anglaise de Computation Independent Model.

Les exigences du Système sont modélisées dans ce modèle qui décrit la situation dans laquelle le système sera utilisé.

Un tel modèle est parfois appelé Business Model ou Domain Model (c'est-à-dire un modèle de l'entreprise). Il ne montre pas les détails de la structure du système. Typiquement ce modèle est indépendant de l'implémentation du système. Le CIM correspond à la modélisation de l'entreprise sans parler encore de système informatique.

Il montre le système au sein de l'environnement dans lequel il opérera. Il aide ainsi pour représenter ce que le système devra exactement faire. Il se révèle utile, non seulement comme une aide pour comprendre le problème, mais aussi comme une source du vocabulaire partagé avec



d'autres modèles. Dans la spécification MDA, les exigences du CIM doivent pouvoir être suivies à la construction des autres modèles (PIM et PSM) qui les implémentent et vice versa.

Le CIM est constitué de deux modèles UML, des points de vue de l'entreprise et des Informations. Le CIM joue un rôle important pour combler le fossé entre les experts, du métier et des exigences, et les experts de la conception des artefacts, qui ensemble satisfont les exigences métier. [3]

#### ***1.4.6.2 PIM (Platform Independent Model)***

Le terme PIM est l'acronyme de l'anglicisme Platform Independent Model soit le modèle Indépendant de la plate-forme. Cela signifie que ce type de modèle n'a pas de dépendance Avec la plate-forme technique. Il décrit le système mais ne montre pas les détails de son utilisation sur la plate-forme. Le PIM représente ainsi le logique métier, spécifique au système, mais indépendant de la technique et de la technologie.

Le PIM correspond à la modélisation du système de manière indépendante à la plate-forme.

Ce modèle est concrètement représenté par un diagramme de classes en UML. Le PIM donne une sémantique aux classes à l'aide de stéréotypes selon le méta-modèle MDA et les profils UML. [3]

#### ***1.4.6.3 PSM (Platform Specific Model)***

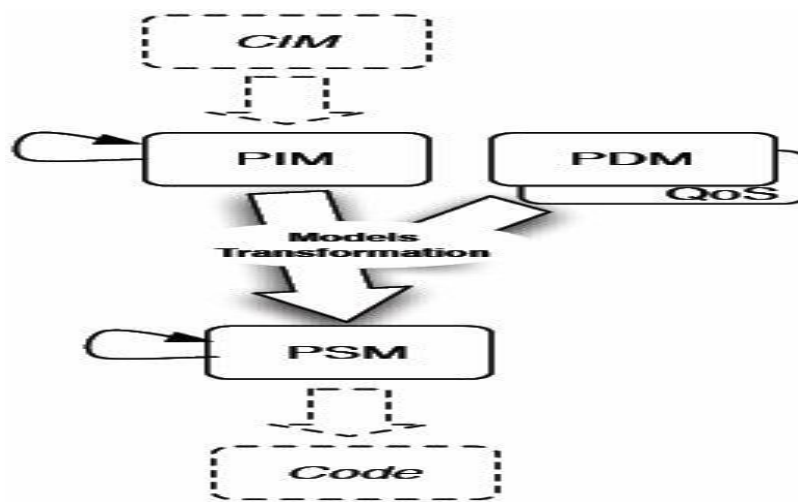
Le PSM, pour Platform Specific Model, est, quant à lui, un modèle dépendant de la Plate-forme technique. Ce type de modèle sert essentiellement de base à la génération de Code exécutable.

Ce modèle, produit à partir d'une transformation du PIM, est un modèle du même système que le PIM mais spécifié par rapport à la plate-forme. Il décrit aussi comment ce système utilisera la plate-forme choisie.

Un PSM peut fournir plus ou moins de détails selon son but. Un PSM peut être une implémentation, s'il fournit toutes les informations utiles pour construire le système. D'autre part, un PSM peut ressembler à un PIM pour définir un niveau d'abstraction intermédiaire entre le PIM et le PSM d'implémentation. [3]

#### 1.4.6.4 PDM (*Platform Description Model*)

Ce modèle est désigné par l'acronyme PDM pour Platform Description Model. Il correspond à un modèle de transformation du PIM vers un PSM d'implémentation. L'architecte doit choisir une ou plusieurs plates-formes pour l'implémentation du système avec les qualités architecturales désirées. Ce modèle propre à la plate-forme est utile pour la transformation du PIM en PSM. La démarche MDA est ainsi basée sur le détail des modèles dépendant de la plate-forme. Il représente les particularités de chaque plate-forme. Il devrait être fourni par le créateur de la plate-forme. [3]



**Figure 1.3 : Principe du Processus MDA**

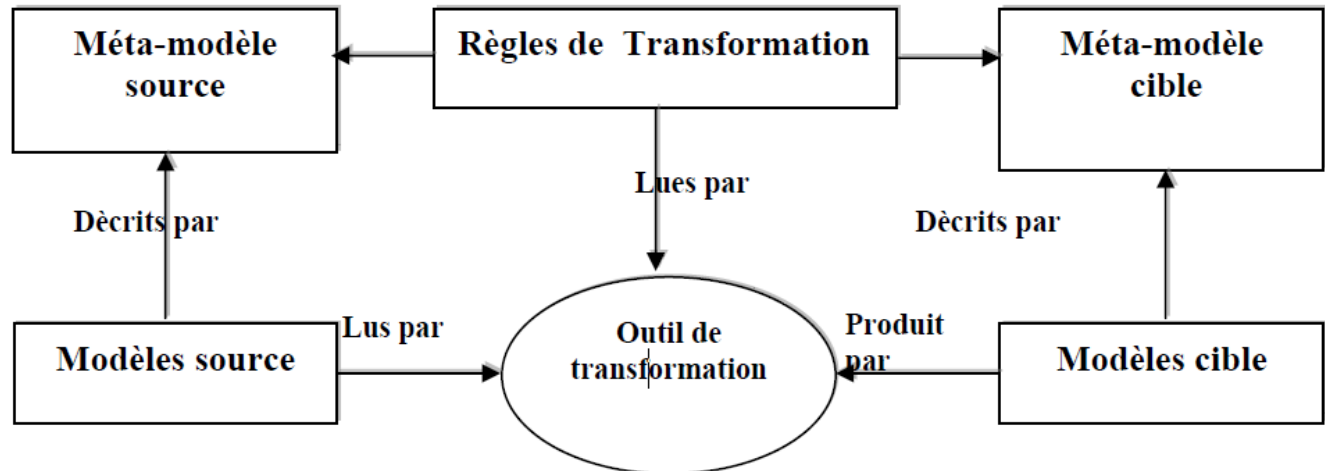
## 1.5 Transformation des modèles du MDA

### 1.5.1 Définition d'une transformation de modèle

La transformation : est une génération automatique d'un modèle cible (Target model) à partir d'un modèle source (source Model).

D'après la définition de la transformation, La transformation est un ensemble de règles de transformation qui décrivent ensemble comment un modèle dans un langage source est transformé en modèle dans un langage cible.

-Une règle de transformation est une description de comment un concept ou plus dans un langage source peut être transformé en un concept ou plus d'un langage cible.



**Figure 1.4 : Processus de transformation de modèles**

### 1.5.2 Principe de transformation des modèles

D'un point de vue utilisateur, une transformation prend un ou plusieurs modèles en entrée et génère un ou plusieurs modèles en sortie. Une transformation est le fruit de la combinaison de deux composantes : une définition de transformation et un outil de transformation.

Les transformations sont des processus permettant d'une part les manipulations de modèles, et d'autre part le passage d'un niveau de modèle à un autre.

Pour réaliser des transformations de modèles, les modèles doivent être exprimés dans un langage de modélisation étant lui-même défini à l'aide d'un méta-modèle. En partant des méta-modèles sources et cibles de la transformation, on distingue deux types de transformations : endogènes et exogènes.

Une transformation est dite endogène si les modèles utilisés sont issus du même méta-modèle. Mais lorsque les modèles sources et cibles sont de différents méta-modèles, la transformation est dite exogène ou translation. On peut subdiviser les deux catégories de transformations comme suit : [9]

✓ ***Transformations endogènes***

- Optimisation : améliorer les performances tout en maintenant la sémantique,
- Restructuration (*refactoring*) : transformation de la structure pour améliorer certains aspects de la qualité du logiciel,
- Simplification : transformation afin de réduire la complexité syntaxique.

✓ ***Transformations exogènes***

- Synthèse : transformation d’un certain niveau d’abstraction vers un niveau d’abstraction moins élevé.
- Rétro ingénierie: inverse de la synthèse.
- Migration : transformation d’un programme écrit dans un langage vers un autre langage du même niveau d’abstraction.

Un autre facteur important à prendre en considération dans les transformations concerne le niveau d’abstraction. On distingue les transformations horizontales et les transformations verticales.

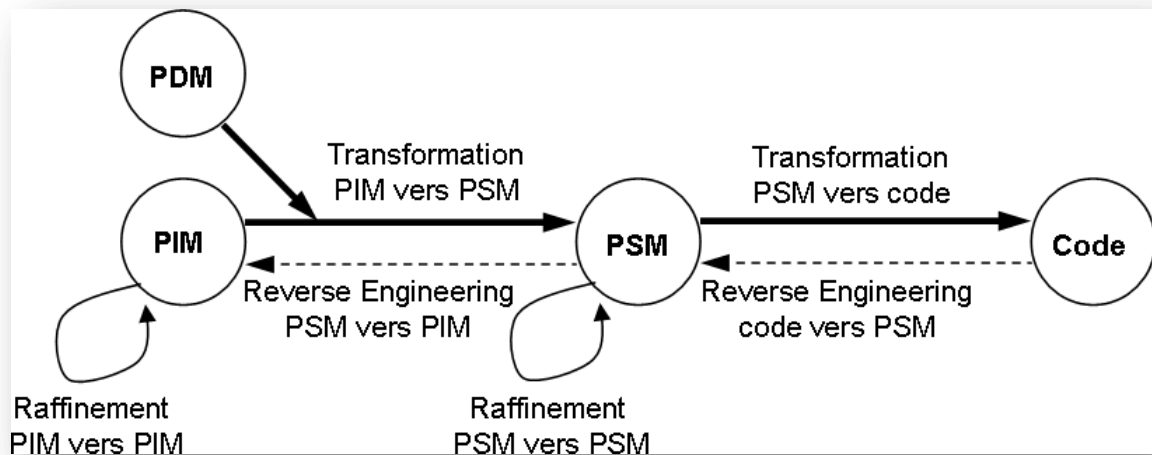
Une transformation horizontale est une transformation où les modèles sources et cibles sont du même niveau d’abstraction. À l’opposé, dans une transformation verticale, les modèles impliqués sont de différents niveaux d’abstraction.

Pour comprendre comment se passe une transformation, il est essentiel de définir différents modèles du MDA, qui sert tout d’abord à modéliser l’application puis par transformation successives vont permettre de générer du code. On s’intéresse ici qu’aux deux modèles suivant :

- Le PIM (*Platform Independent Model*) : Indépendant de toute plateforme, il ne connaît aucune des informations sur les technologies qui vont permettre le déploiement de l’application. On utilise en générale un diagramme UML pour le représenter.
- Le PSM (*Platform Specific Model*) : dépendant de la plateforme technique, il est essentiellement utilisé comme base pour la génération de code exécutable vers une plateforme.

Selon Quatre types de transformations sont définis entre ces deux modèles au sein des spécifications MDA.

- ***PIM vers PIM (raffinement)*** : Cette transformation constitue un enrichissement et une spécialisation du modèle, en y apportant des informations indépendantes des spécificités d'une technologie. La description d'un modèle de répartition, de persistance des données ou de composants peut être vue comme un raffinement.
- ***PIM vers PSM (projection)*** : C'est la traduction du modèle générique, vers une plateforme d'exécution.
- ***PSM vers PSM (réalisation)*** : C'est la mise en œuvre concrète d'un modèle générique sur une plate-forme d'exécution. Elle consiste en l'ensemble des phases qui mènent à un logiciel exécutable, telles que la génération de code source, la compilation, le déploiement, l'instanciation et l'initialisation des composants logiciels.
- ***PSM vers PIM (reverse-engineering)*** : Cette transformation permet l'élaboration du modèle générique à partir de l'implémentation existante d'un logiciel. En théorie, cette transformation est censée fournir un modèle générique décrivant l'application à partir d'une base de code accessible. En pratique, il est très complexe d'automatiser entièrement ce processus pour tout modèle PSM.
  
- ***Génération de code*** : Dans la pratique, certains travaux font la distinction entre les PSM Exécutable (ou code) et les PSM non exécutables, mais la génération de code n'est pas toujours considérée comme une transformation de modèles. Quand même qu'il est possible de passer d'un PSM non exécutable à du code et inversement. Il est important de souligner que le passage du code au PSM est une opération de rétro-ingénierie qui est assez complexe à réaliser. Si le code n'a pas été conçu dans la démarche du MDA, il faut faire appel aux techniques traditionnelles de rétro-ingénierie pour effectuer de telles opérations. [10]



**Figure 1.5 : les transformations des modèles dans l'approche MDA**

## 1.6 Les outils MDA

Les outils MDA peuvent être classifiés en plusieurs catégories, comme le montre Le Tableau : [3]

Catégorie	Description	Code utilisé
Editeur de Définition de Transformation	Sert à construire et modifier des définitions de transformation	EDT
Editeur de Modèle	Sert à construire et modifier des modèles	EM
Outil de Transformation	Outil réalisant des transformations MDA	OT
Générateur de Fichier de Code	Outil pouvant lire un modèle en entrée et reproduit un fichier de code (texte)	GFC
Repository de Définition de Transformation	Stockage des définitions de transformations	RDT

Valideur de Modèle	Outil pour vérifier la validité des modèles.	VM
Repository de Modèle	Base de données des modèles	RM
Parseur de Fichier de Code	Permet d'extraire les informations d'un fichier texte pour les stocker dans le repository de modèle	PFC
Editeur de Code (IDE)	Outil pour Editer, compiler, debugger	EC

**Tableau 1.1 : Classification des outils MDA**

### 1.7 Les langages de transformation MDA

Le Tableau présente quelques langages de transformation MDA. [3]

Langages	Description
UMLX	[UMLX]
ATL	ATLAS Transformation Language
YATL	Yet Another Transformation Language
BOTL	The Bidirectional Object Oriented Transformation
MOLA	MOLA(MOdel Transformation Language) Offers basic elements of a new graphical transformation language. [MOLA]
GreAT	Graph Rewriting and Transformation

**Tableau 1.2 : quelque langage de transformation MDA**

## 1.8 Le future de MDA

Nous pensons que le futur de MDA va dépendre de son utilisation (pour construire et maintenir des systèmes informatiques) et des résultats obtenus. Cependant, il faut savoir que MDA n’est plus simplement une théorie, mais une réalité présentant déjà des résultats positifs. D’autres travaux ont déjà démontré la viabilité de l’application de MDA, par exemple l’application de MDA aux services, aux systèmes embarqués, à CORBA, aux applications militaires, et aux systèmes d’information.

Plusieurs entreprises proclament avoir implémenté une approche MDA. Cependant, l’approche MDA n’est pas encore complètement expérimentée et plusieurs problèmes ne sont pas encore résolus. Par exemple, le langage de transformation autour de MDA n’est pas normalisé, mais les propositions soumises à l’OMG devraient offrir un avenir sûr pour MDA. De plus, méthodologies autour de MDA sont aussi nécessaires. [3]

## 1.9 Conclusion

Dans ce chapitre nous avons défini le contexte dans lequel se place ce mémoire et détaillé les principaux concepts du domaine de l’ingénierie des modèles. Nous avons mis en évidence le Rôle central des modèles et de l’outillage sous-jacent dans le processus de conception des systèmes.

Dans le cadre de l’approche IDM, nous avons présenté différentes techniques de traitement des modèles parmi lesquelles nous avons détaillé la méta-modélisation ainsi que les techniques de transformation de modèles. Ces techniques sont aujourd’hui des pivots majeurs pour l’innovation dans les domaines de recherche liés à la modélisation des systèmes complexes.