Types de modèles dans MDA

1. **Introduction** L'**Architecture Dirigée par les Modèles (MDA)** est une approche promue par l'OMG (Object Management Group) dont le but principal est de séparer la **spécification fonctionnelle** d'un système de son **implémentation technique**. Cela permet d'augmenter la productivité, la portabilité, l'interopérabilité, et la maintenance des applications face à l'obsolescence technologique.

Le cœur de la MDA repose sur la manipulation de modèles à différents niveaux d'abstraction et les **transformations** qui permettent de passer d'un niveau à l'autre, jusqu'à l'obtention du code.

2. Modèle CIM (Computation Independent Model)

2.1. Définition et Rôle

Le **CIM (Computation Independent Model)**, ou **Modèle Indépendant de Calcul**, est le niveau d'abstraction le plus élevé. Il est le point de départ du processus MDA.

- **Indépendance Totale :** Il est **totalement indépendant de toute solution informatique** (pas de classes, d'interfaces, de bases de données, ni de services).
- **Objectif :** Décrire le **contexte, les exigences et les processus métier** (le "quoi"). Il se concentre sur les aspects du domaine et l'environnement du système.
- **Public Cible :** Experts métier, analystes fonctionnels, et parties prenantes (utilisateurs finaux, management).

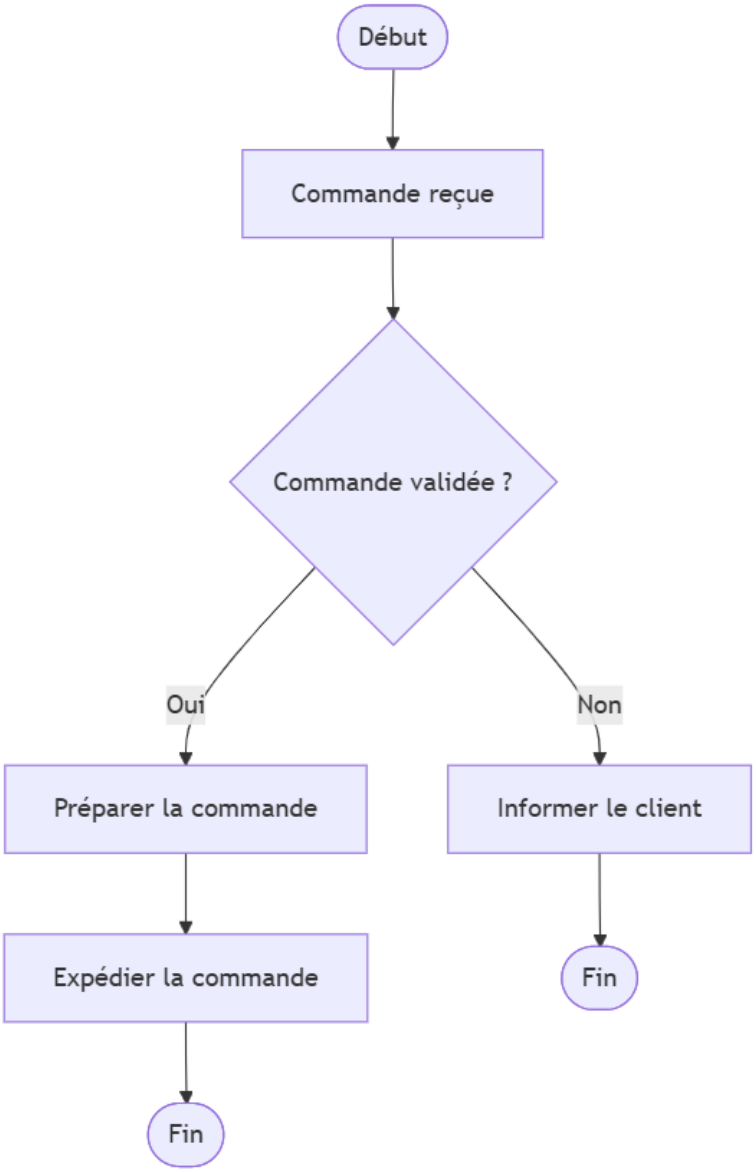
2.2. Exemples de Notations et Concepts

Le CIM utilise souvent des notations compréhensibles par les non-informaticiens :

Concept CIM	Notation Typique
Processus Métier	BPMN (Business Process Model and Notation)
Exigences	Langage naturel, histoires utilisateur
Acteurs / Rôles	Décrits par des cas d'utilisation UML à un niveau très orienté métier

2.3. Le **BPMN (Business Process Model and Notation)** est un standard de modélisation graphique normalisé par l'OMG (Object Management Group). Son objectif principal est de fournir une notation standardisée, compréhensible par tous les acteurs d'une organisation, des analystes métier aux développeurs techniques, en passant par les managers et les parties prenantes. Le BPMN permet de représenter de manière claire et précise les processus métier d'une entreprise sous forme de diagrammes, en utilisant un ensemble riche d'éléments visuels (objets de flux, objets de connexion, couloirs, artefacts). Ces éléments permettent de décrire les activités, les événements déclencheurs, les décisions, les acteurs impliqués et les flux de séquence qui constituent un processus.

Exemple Imaginons que nous suivons une commande passée sur un site internet. Voici ce qui se passe :



1. Le Déclenchement du Processus

- **Symbole :** ([Début]) (Cercle vert)
- **Explication :** C'est **l'événement de début**. Il représente le déclencheur qui lance tout le processus. Dans notre exemple, cela se produit dès qu'une **nouvelle commande arrive** dans le système. C'est le point de départ.

2. La Première Action

- **Symbole :** [Commande reçue] (Rectangle)
- **Explication :** C'est une **tâche**. Une tâche est une action unique, une étape de travail concrète qui doit être accomplie. Ici, la première action est d'enregistrer et de prendre en compte la commande reçue.

3. Le Point de Décision

- **Symbole :** {Commande validée ?} (Losange)
- **Explication :** C'est une **passerelle exclusive** , Son rôle est de poser une question et de diriger le processus vers un chemin différent en fonction de la réponse. C'est un embranchement.
  - La question est : "La commande est-elle validée ?"
  - **Il n'y a qu'un seul choix possible à la fois** : soit "Oui", soit "Non".

4. Les Deux Chemins Possibles

- **Chemin "Oui" (Commande valide) :**
  - [Préparer la commande] : Nouvelle **tâche**. L'équipe du entrepôt emballe les articles.
  - [Expédier la commande] : Dernière **tâche** de ce chemin. Le colis est remis au livreur.
  - ([Fin]) : **Événement de fin** (cercle rouge). Il marque l'achèvement **réussi** du processus pour ce chemin. Le client va recevoir sa commande.
- **Chemin "Non" (Commande invalide) :**
  - [Informer le client] : **Tâche**. Le service client envoie un email pour expliquer le problème (ex: paiement refusé, article indisponible).
  - ([Fin]) : **Événement de fin**. Il marque ici la fin du processus pour ce chemin, mais sur une **situation anormale**. La commande n'a pas pu être traitée.

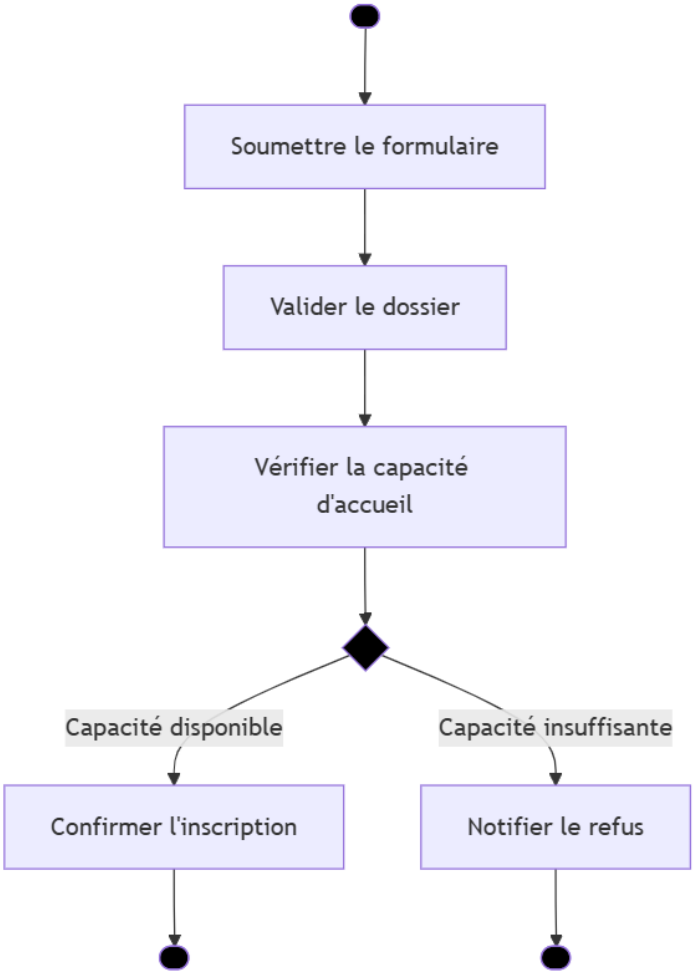
2.4. **Pourquoi cette modélisation est-elle utile ?**

- **Clarté :** Tout le monde (métier, technique, nouveaux employés) comprend le processus de la même manière.

- **Efficacité :** On identifie immédiatement le point de décision critique ("Commande validée ?") qui influence toute la suite.
- **Amélioration :** En le visualisant, on peut se demander : "Comment réduire le nombre de commandes qui prennent le chemin 'Non' ?".

2.5. Exemple Concret (Système d'Inscription Universitaire) :

Un diagramme BPMN montre le flux de travail pour l'inscription : "L'étudiant soumet le formulaire", "Le Secrétariat valide le dossier", "Le système vérifie la capacité d'accueil". Il n'y a aucune mention de bases de données, de serveurs ou de langage de programmation.



Flux du processus :

1. Le processus commence par la soumission du formulaire
2. Le secretariat valide le dossier
3. Le système vérifie la capacité d'accueil
4. **Décision** : Si capacité suffisante → confirmation, sinon → notification de refus
5. Le processus se termine dans les deux cas

2.6. Interet des artfac dans le diagramme BPMN

Pourquoi ces artefacts sont-ils utiles ?

Sans artefacts :

- Le diagramme montre seulement **CE QUI** se passe
- On voit les étapes et les décisions

Avec artefacts :

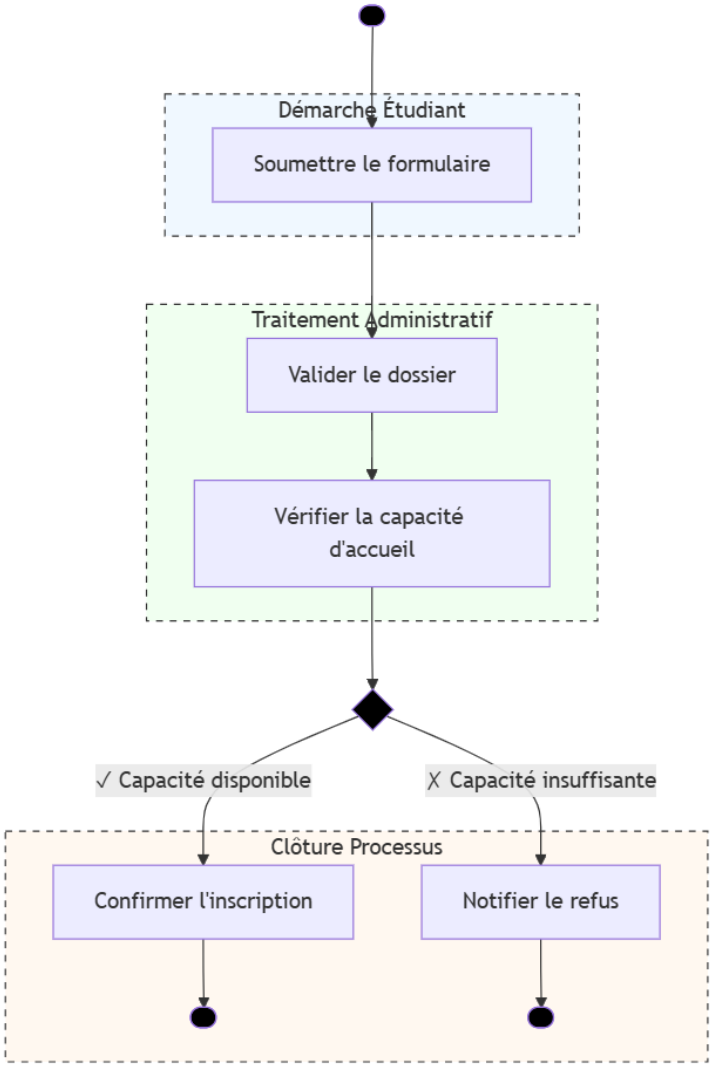
- Le diagramme explique aussi **POURQUOI** et **COMMENT**
- **Le Groupe** met en évidence une partie importante du processus
- **Les Annotations** documentent les règles métier directement sur le diagramme
- **Résultat:** Le processus est beaucoup plus compréhensible pour tous les acteurs

Bénéfices pour l'entreprise

1. **Documentation intégrée:** Les règles métier sont directement attachées aux éléments du processus
2. **Meilleure communication:** Les contraintes et règles sont explicites
3. **Formation accélérée:** Les nouveaux employés comprennent mieux la logique métier
4. **Audit facilité:** Les décisions et leurs justifications sont documentées

Ces artefacts transforment un simple enchaînement d'activités en une véritable cartographie intelligente du processus métier !

Solution de l’exemple de la scolarité avec artfac



Symbole : (rectangle avec bordure en pointillés et coins arrondis)

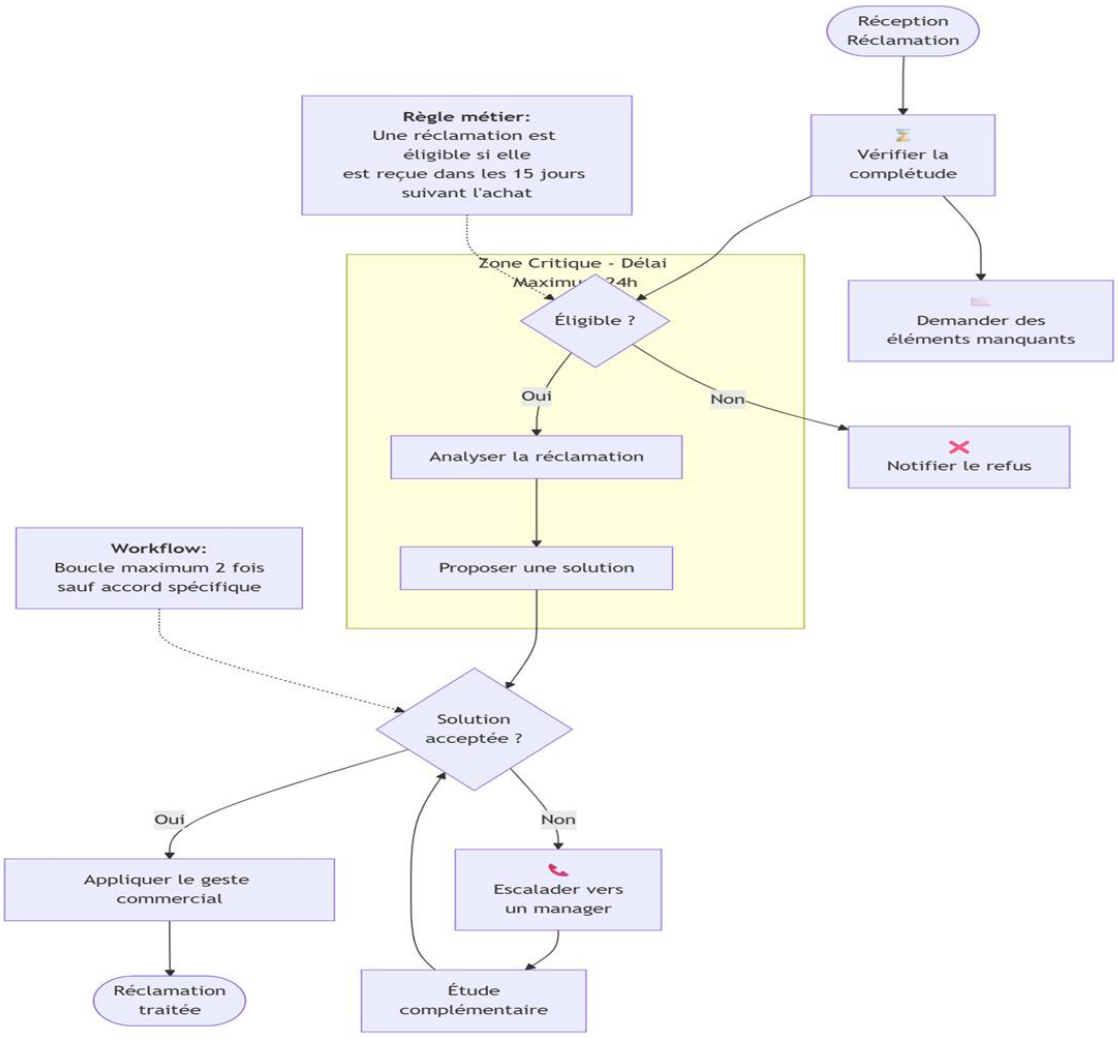
Rôle : Regrouper visuellement des activités liées sans affecter le flux.

2.7. Les principaux Artefacts en BPMN

3. **Groupe (Group)** : Regroupement visuel (bordure pointillée)
4. **Objet de Données (Data Object)** : Représente les documents/informations

5. **Annotation (Text Annotation)** : Commentaires explicatifs

Exemple avec artfac



Explications des Artefacts Utilisés

1. Groupe (Group)

- **Représenté par:** Le rectangle en pointillé autour des étapes "Éligible ?", "Analyser" et "Proposer une solution"
- **Utilité:** Permet de regrouper visuellement des activités qui font partie d'une même logique métier
- **Dans notre exemple:**
  - Nom: **"Zone Critique - Délai Maximum 24h"**
  - Il indique que ces trois étapes doivent être traitées rapidement
  - Met en évidence une contrainte temporelle importante

2. Annotations (Text Annotations)

- **Représenté par:** Les notes avec le symbole « » et reliées par des pointillés
- **Utilité:** Ajouter des informations explicatives, des règles métier ou des contraintes

Annotation 1: Règle métier

"**Règle métier:**
Une réclamation est éligible si elle est reçue dans les 15 jours suivant l'achat"

- **Reliée à:** La passerelle "Éligible ?"
- **Explication:** Donne le critère concret utilisé pour prendre la décision

Annotation 2: Workflow

"**Workflow:**
Boucle maximum 2 fois sauf accord spécifique"

- **Reliée à:** La passerelle "Solution acceptée ?"
- **Explication:** Précise une règle de gestion concernant la boucle de négociation

3. Modèle PIM (Platform Independent Model)

Définition et Rôle

Le **PIM (Platform Independent Model)**, ou **Modèle Indépendant de Plateforme**, est le modèle central de la MDA.

- **Indépendance de Plateforme :** Il est **indépendant de toute technologie d'implémentation spécifique** (Java EE, .NET, Spring, CORBA, etc.), mais utilise des **concepts de conception logicielle**.

- **Objectif :** Décrire la **structure et le comportement du système** (le "comment conceptuel") en utilisant des concepts orientés objet ou composant. Il sert de **machine conceptuelle** abstraite.
- **Public Cible :** Architectes logiciels et concepteurs.

Exemples de Notations et Concepts

Le PIM est principalement modélisé avec le langage **UML (Unified Modeling Language)**, mais sans les détails spécifiques à une plateforme.

Concept PIM	**Notation Typique**
Structure	**Diagrammes de Classes** (Classes, Attributs, Opérations, Associations)
Comportement	**Diagrammes de Séquence, d'État-Transition, d'Activité**
Architecture	**Diagrammes de Composants/Paquetages**

Exemple Concret (Système d'Inscription) :

- **Classe** Etudiant (avec attributs nom, matricule).
- **Interface** ServiceInscription (avec l'opération validerDossier(d: Dossier)).
- **Association** entre Etudiant et Cours (indiquant les cours auxquels l'étudiant est inscrit).
- **Absence de détail technique :** Aucune mention de JPA, de Hibernate, ni de protocole REST.

4. Modèle PSM (Platform Specific Model)

Définition et Rôle

Le **PSM (Platform Specific Model)**, ou **Modèle Spécifique de Plateforme**, est le niveau le plus proche de l'implémentation.

- **Spécificité de Plateforme :** Il est **lié à une technologie d'implémentation spécifique** (une seule plateforme), intégrant ses contraintes et ses mécanismes.
- **Objectif :** Spécifier les détails techniques nécessaires à la génération de code exécutable.
- **Public Cible :** Développeurs et architectes techniques.

Exemples de Notations et Concepts

Le PSM est souvent basé sur le **PIM**, enrichi par des **Profils UML** (ou d'autres mécanismes d'extension) qui introduisent les concepts de la plateforme cible.

Concept PSM	Exemple de Technologie (Java EE)
Entité persistante	**Stéréotype <<EJB Entity>>** ou **<<JPA Entity>>**
Service distribué	**Stéréotype <<Stateless Session Bean>>** ou **<<RESTController>>**
Mécanisme transactionnel	**Annotation @Transactional**

Exemple Concret (Système d'Inscription - PSM Java EE) :

- **Classe** Etudiant stéréotypée <<JPA Entity>> (générant l'annotation @Entity).
- **Classe** ServiceInscriptionImpl stéréotypée <<Stateless Session Bean>> (générant l'annotation @Stateless).
- L'opération validerDossier() est annotée avec @TransactionAttribute(REQUIRED).

5. Transformations entre Niveaux

Les transformations sont le **mécanisme clé** de la MDA, permettant de dériver un modèle cible à partir d'un modèle source. Elles sont généralement définies par des **règles** et exécutées par des outils.

Transformation Verticale (CIM PIM et PIM PSM)

Une transformation est dite **verticale** lorsqu'elle fait passer le modèle d'un niveau d'abstraction à un autre.

CIM PIM (Passage du Métier à la Conception Abstraite)

- **Rôle :** Traduire les exigences métier en une structure logicielle abstraite.
- **Règles Typiques :**
  - **Entité Métier Classe PIM** (UML).
  - **Processus Métier Opération PIM** sur une Classe de Service/Composant.
  - **Règle Métier Contrainte OCL** (Object Constraint Language) sur une Classe/Opération PIM.

PIM PSM (Passage de la Conception à l'Implémentation Spécifique)

- **Rôle :** Enrichir le modèle PIM avec les détails techniques de la plateforme choisie.
- **Règles Typiques :**
  - **Classe PIM** (destinée à être persistante) **Classe PSM** stéréotypée <<Entity>> (JPA).
  - **Classe PIM** (destinée à gérer la logique) **Classe PSM** stéréotypée <<SessionEJB>> (Java EE) ou <<RESTController>> (Spring/Express).
  - **Association PIM Association PSM** enrichie d'annotations de mapping de base de données (@OneToMany, etc.).

6. Langages de Transformation

L'OMG a standardisé le langage **QVT (Query/View/Transformation)**. D'autres langages, comme **ATL (ATLAS Transformation Language)**, sont couramment utilisés pour implémenter ces règles de transformation.

- **Exemple de Transformation Détaillée : CIM → PIM → PSM**

Nous allons suivre l'entité clé **"Client"** à travers les trois niveaux, en ciblant une plateforme **Microservices RESTful (Spring Boot)**.

Étape 1 : CIM (Focus sur le Rôle et les Données Métier)

- **Concept CIM : Abonné** (ou **Client**).
- **Exigence Métier :** Le système doit conserver les informations personnelles de l'Abonné (Nom, Adresse, Coordonnées) pour la facturation et le suivi.
- **Modélisation :** Décrite dans le glossaire métier et les diagrammes de flux d'information.

Étape 2 : Transformation CIM PIM

- **Règle de Transformation :** *Chaque Entité Métier primaire et persistante se transforme en une Classe PIM.*
- **Résultat PIM :** Création de la **Classe UML Client**.
    - **Attributs :** nom: String, adressePostale: String, email: String.
    - **Opération :** modifierAdresse(nouvelleAdresse: String) (Logique de gestion de l'état).

Étape 3 : Transformation PIM PSM (Cible : Spring Boot / REST / JPA)

Le PIM est enrichi pour le rendre spécifique à la plateforme Spring Boot (Spring Boot est une **plateforme Java** moderne et très populaire qui facilite le **développement rapide d'applications web et microservices**. Elle repose sur le framework **Spring**, mais simplifie énormément sa configuration et son utilisation).

- **Règle 1 (Persistance) :** *Toute Classe PIM destinée à être persistée devient une Entité JPA dans le PSM.*
    - **Résultat PSM :** La **Classe Client** est stéréotypée <<JPA Entity>> et on ajoute l'attribut id: Long stéréotypé <<Id>>.
- **Règle 2 (Exposition de service) :** *Le besoin d'accéder aux clients se traduit par une classe PSM de type contrôleur REST.*
    - **Résultat PSM :** Création d'une **Classe ClientController** stéréotypée **<<REST Controller>>** qui expose des opérations (GET, POST, etc.) pour manipuler les entités Client.
        - Opération getClient(id: Long) stéréotypée <<GET Mapping>> (avec l'URL /api/clients/{id}).

Étape 4 : PSM Code Source

Le PSM guide la génération du code final (Java dans cet exemple).

Élément PSM	Code Java Généré (PSM → Code)
Classe Client $\ll JPA Entity \gg$	java @Entity public class Client { @Id @GeneratedValue private Long id; private String nom; // ... }
Classe ClientController $\ll REST Controller \gg$	java @RestController @RequestMapping("/api/clients") public class ClientController { @GetMapping("/{id}") public Client getClient(@PathVariable Long id) { // ... } }

Conclusion : Le concept de **Client** (CIM) a été traduit en une **Classe** (PIM), puis spécifié par des **annotations techniques** (PSM) pour finalement produire du **code Java exécutable** (Code). Si l'on décidait de migrer vers **C#/.NET**, seules les étapes 3 et 4 (PSM et génération de code) changeraient, le PIM restant stable.

6. Conclusion

L'approche MDA structure le développement autour de trois modèles essentiels :

- **CIM** : Modèle métier pur, sans aspects techniques
- **PIM** : Conception logique indépendante de toute technologie
- **PSM** : Implémentation spécifique à une plateforme technique

Les **transformations successives CIM → PIM → PSM** garantissent une traduction fidèle des besoins métier vers la solution technique, en maintenant la cohérence et en réduisant les erreurs.
Cette méthodologie permet de créer des systèmes **alignés avec le métier**, **évolutifs** et **adaptables** aux changements technologiques, tout en accélérant le développement grâce à une approche systématique et traçable.