Chapitre 3 Meta-Object Facility(MOF)

1. Introduction

Dans le domaine de l'ingénierie dirigée par les modèles , la nécessité de structurer, formaliser et standardiser les langages de modélisation est primordiale. C'est dans ce contexte qu'intervient le **Meta-Object Facility (MOF)**, un standard défini par l'**Object Management Group (OMG)**. Le MOF fournit une architecture de métamodélisation permettant de définir la syntaxe et la sémantique des langages de modélisation comme **UML** (Unified Modeling Language). Grâce à son architecture hiérarchique à quatre niveaux d'abstraction (M0 à M3), le MOF constitue la base conceptuelle sur laquelle reposent de nombreux standards de modélisation. Ce chapitre se propose d'explorer en détail les principes fondamentaux du MOF, son architecture, ses rôles dans la modélisation logicielle, ainsi que ses liens avec d'autres standards de l'OMG.

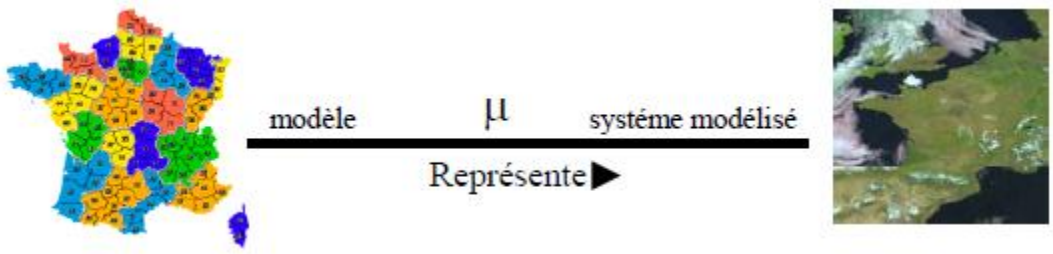
2. Définition et Relations Fondamentales du Modèle

Il n'existe pas de définition universelle du modèle, mais les différentes sources (UML, Seidwitz, Bézivin et Gérbé) convergent sur l'idée qu'un modèle est une **abstraction ou simplification d'un système** (appelé le Système Étudié) construite dans un **but précis** (par exemple, pouvoir répondre à des questions à la place du système réel).

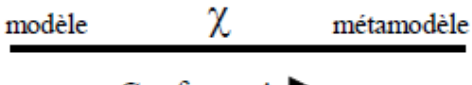
Deux relations complémentaires structurent cette notion :

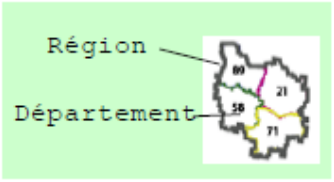
1. **RepresentationDe (μ)** : La relation unique qui lie le **modèle** au **système** qu'il modélise (ex: une carte géographique est un modèle de la région).

Par exemple une carte géographique peut jouer le rôle de *modèle*, alors que la région étudiée jouera celui de *système modélisé*.



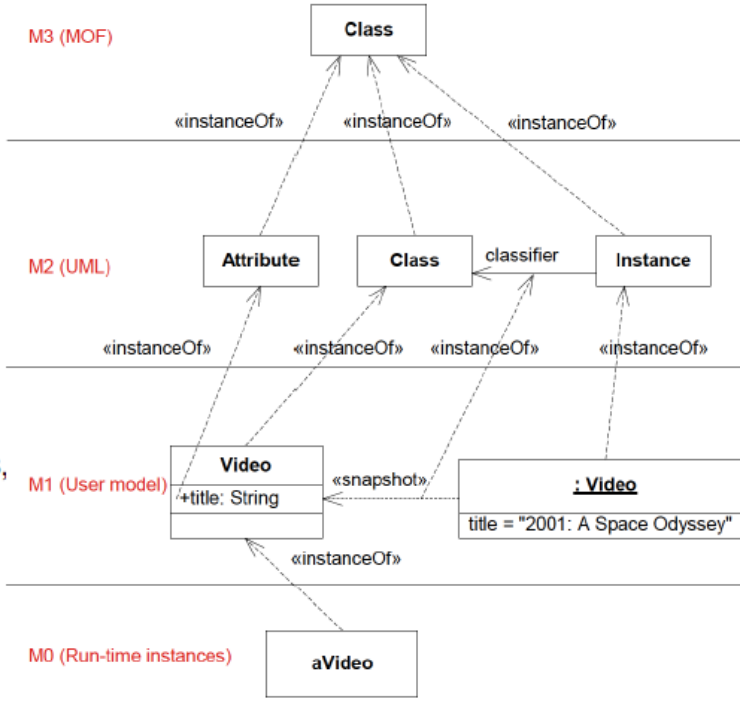
2. **ConformeA (Conformité)** : La relation qui garantit qu'un modèle respecte le langage dans lequel il est exprimé. Cette conformité se manifeste différemment selon les espaces technologiques (ex: un document XML conforme à un schéma, un objet conforme à sa classe).





3. **L'Architecture à 4 Niveaux**

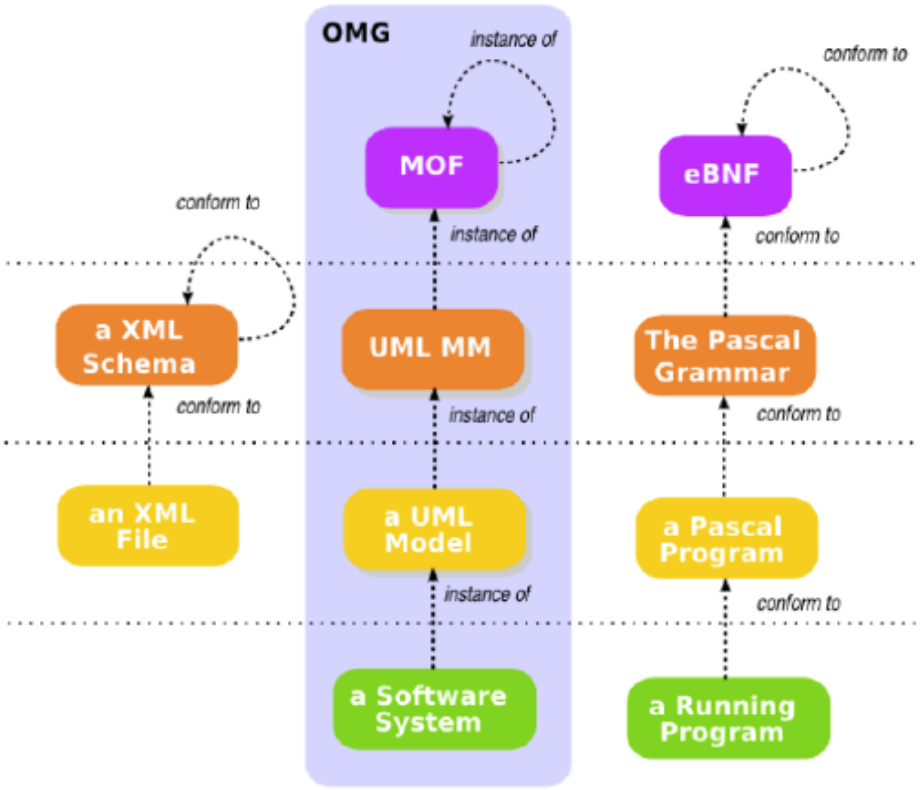
- M3 : Méta-méta-modèle des méta-modèles de M2... et de M3
- M2 : Méta-modèles des modèles de M1
- M1 : Modèles (Diagrammes de classes, de séquence, ...)
- M0 : Instances des modèles à l'exécution



3.1. Présentation des Niveaux

Chaque niveau **instancie** le niveau supérieur :

- Un modèle M1 est une instance d'un métamodèle M2
- Un métamodèle M2 est une instance de MOF M3
- MOF M3 est une instance de lui-même (réflexivité)



3.3. Principe Fondamental et Détail des 4 Niveaux

- **Niveau M0 : Les Instances (Données)**

Définition : Monde réel, données exécutables, objets avec valeurs concrètes.

Caractéristiques :

- Données opérationnelles
- États dynamiques du système
- Valeurs attribuées aux propriétés

Exemple concret - Système Bancaire :

java

```
// INSTANCES M0 - DONNÉES RÉELLES
Compte cpt_12345 = new Compte(
    numero = "FR7630001007941234567890185",
    solde = 1500.00,
    dateOuverture = "15/01/2024"
);
```

```
Client cli_789 = new Client(
   id = 789,
   nom = "aissani”,
   prenom = "mohamed",
   email = "mohaissani@email.com"
);

// LIEN entre instances
cpt_12345.setTitulaire(cli_789);
```

- **Niveau M1 : Les Modèles**

Définition : Description abstraite d'un système, conforme à un métamodèle.

Caractéristiques :

- Spécifique à un domaine métier
- Définit les classes, attributs, relations
- Independent de l'implémentation

Exemple concret - Modèle Bancaire :

```
class Client {
  - id : Integer
  - nom : String
  - prenom : String
  - email : String
}
class Compte {
  - numero : String
  - solde : Double
  - dateOuverture : Date
}
class Transaction {
  - id : Integer
  - montant : Double
  - date : Date
  - type : String
}
```

```
Client "1" -- "*" Compte : titulaire
Compte "1" -- "*" Transaction : historique
```

Représentation formelle :

```
// MODÈLE M1 - Spécification du système bancaire
model SystemeBancaire {
  class Client {
    attribute Integer id;
    attribute String nom;
    attribute String prenom;
    attribute String email;
  }

  class Compte {
    attribute String numero;
    attribute Double solde;
    attribute Date dateOuverture;
  }
  association Client_Compte {
    role titulaire : Client[1];
    role comptes : Compte[*];
  }
}
```

- **Niveau M2 : Les Métamodèles**

Définition : Langage de modélisation qui définit les concepts disponibles pour créer des modèles M1.

Exemples standards :

- UML (Unified Modeling Language)
- Ecore (Eclipse Modeling Framework)
- BPMN (Business Process Model and Notation)

Exemple concret - Métamodèle de Modélisation Objet :

```
// MÉTAMODÈLE M2 - Définition des concepts objet
metamodel UMLSimplifie {
```

```
// Concept de base
abstract class NamedElement {
  attribute String name;
}

// Définition d'une classe
class Class extends NamedElement {
  attribute Visibility visibility;
  attribute Boolean isAbstract;
  reference Property[*] ownedAttributes;
  reference Operation[*] ownedOperations;
}

// Définition d'une propriété
class Property extends NamedElement {
  attribute String type;
  attribute Multiplicity multiplicity;
  reference Class owningClass;
}

// Définition d'une opération
class Operation extends NamedElement {
  attribute String returnType;
  reference Parameter[*] parameters;
  reference Class owningClass;
}

// Types énumérés
enum Visibility { PUBLIC, PRIVATE, PROTECTED }
enum Multiplicity { ONE, MANY }
}
```

- **Niveau M3 : MOF (Meta-Object Facility)**

Définition : Méta-méta-modèle qui définit le langage pour spécifier les métamodèles.

Caractéristiques MOF :

- Standard OMG
- Réflexif (peut se décrire lui-même)
- Base pour de nombreux standards (UML, XMI, CWM)

Structure simplifiée de MOF :

```
// META-META-MODÈLE M3 - MOF CORE
meta-metamodel MOF {

  // Élément nommé de base
  abstract class NamedElement {
    attribute String name;
  }

  // Concept de Type
  abstract class Type extends NamedElement;

  // Concept de Class
  class Class extends Type {
    attribute Boolean isAbstract;
    reference Property[*] ownedAttributes;
    reference Operation[*] ownedOperations;
  }

  // Concept de Property
  class Property extends NamedElement {
    attribute Boolean isDerived;
    reference Type type;
    reference Class owningClass;
  }

  // Concept de DataType
  class DataType extends Type;

  // Concept de Package
  class Package extends NamedElement {
    reference Type[*] ownedTypes;
  }
}
```

Exemple 1

Exemples concrets pour chaque niveau

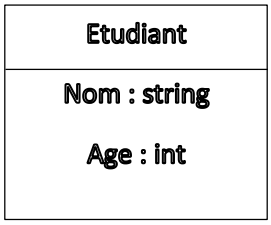
Pour bien comprendre l'architecture à 4 niveaux, voici un **exemple progressif d'un système de gestion d'étudiants** :

- **Niveau M0 – Objets concrets (exécution)**

Etudiant e = new Etudiant("nour", 22);
Ici, **e** est une instance réelle d'un objet (exécuté à l'exécution du programme).

- **Niveau M1 – Modèle UML**

Diagramme UML de classe :



Ce **modèle UML** décrit la structure du système. Il est utilisé pour raisonner sur le système avant son implémentation.

- **Niveau M2 – Métamodèle UML**

Définitions générales dans UML :

- Une **Classe** est un élément possédant :
  - un nom
  - des attributs (nom, type)
- Une **Association** relie deux classes
- Un **Attribut** a un nom et un type

Ce sont les **règles générales** que doivent respecter tous les modèles UML.

- **Niveau M3 – MOF**

Le **MOF** définit que :

- Une **Classe** est un **Classifier**.
- Un **Classifier** peut avoir des **StructuralFeatures**.
- Une **Property** est une **StructuralFeature**.

Ces concepts sont décrits dans le MOF pour pouvoir être utilisés dans des métamodèles comme UML.

Exemple 2 : Exemple de système réel à modéliser (niveau M0)

• Une pièce possède 4 murs, 2 fenêtres et une porte

• Un mur possède une porte ou une fenêtre mais pas les 2 à la fois

• Deux actions sont associées à une porte ou une fenêtre :

– ouvrir et fermer

• Si on ouvre une porte ou une fenêtre fermée, elle devient ouverte

• Si on ferme une porte ou une fenêtre ouverte, elle devient fermée

Pour modéliser ce système, il faut définir 2 diagrammes UML : niveau M1

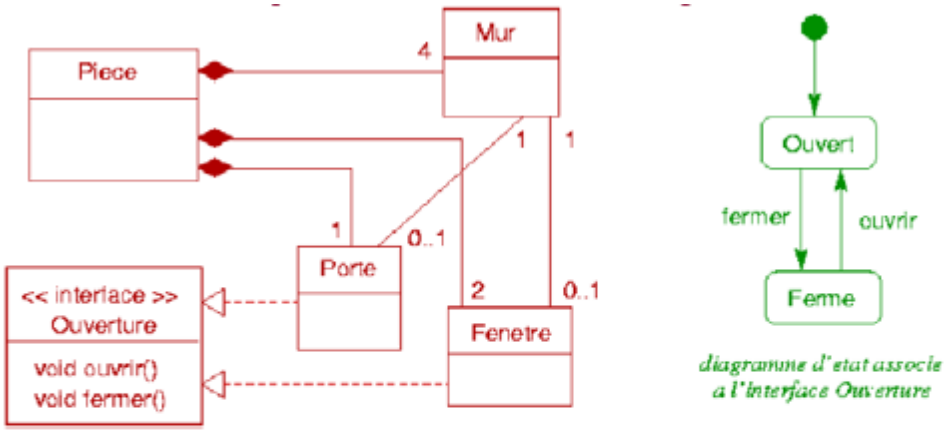
• Un diagramme de classe pour représenter les relations entre les éléments (portes, murs, pièce)

• Un diagramme d'état pour spécifier le comportement d'une porte ou d'une fenêtre (ouverte, fermée)

• On peut abstraire le comportement des portes et des fenêtres en spécifiant les opérations d'ouverture et fermeture dans une interface

– Le diagramme d'état est associé à cette interface

• Il faut également ajouter des contraintes OCL pour préciser les contraintes entre les éléments d'une pièce



context Mur **inv**: fenetre -> union(porte) -> size() <= 1
-- un mur a soit une fenêtre soit une porte (soit rien)
context Piece **inv**:
mur.fenetre -> size() = 2 -- 2 murs de la pièce ont une fenêtre
mur.porte -> size() = 1 -- 1 mur de la pièce a une porte

Les 2 diagrammes de ce modèle de niveau M1 sont des

diagrammes UML valides, parce qu'ils sont conformes au métamodèle UML.

• Les contraintes sur les éléments des diagrammes UML et leurs relations sont définies

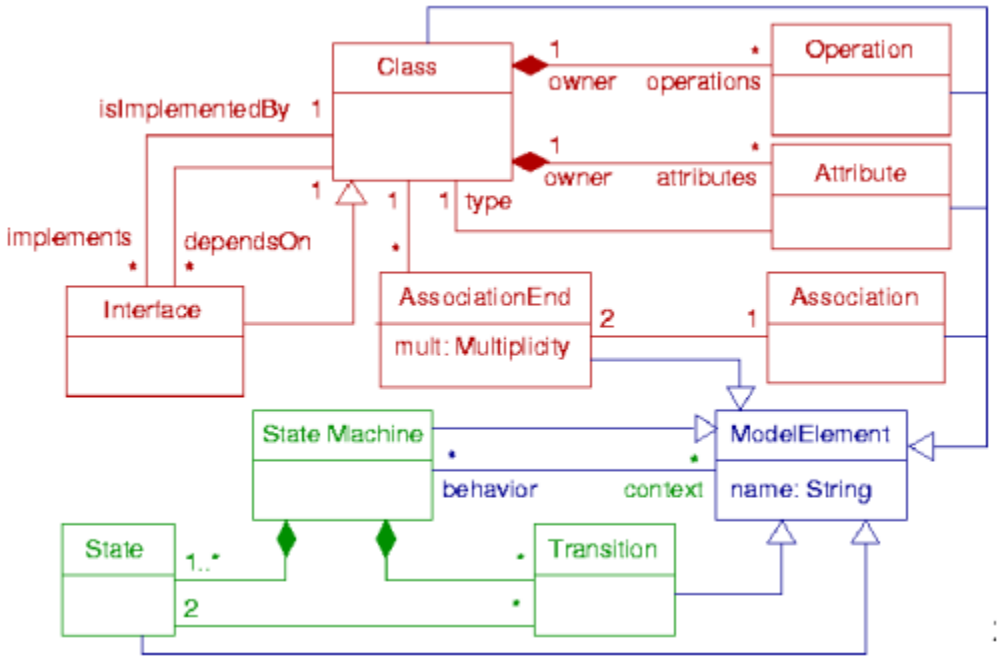
– Dans le métamodèle UML : niveau M2

– Un diagramme UML (de classes, d'états ...) doit être conforme au métamodèle UML

• Métamodèle UML

– Diagramme de classes spécifiant la structure de tous types de diagrammes UML

– Avec contraintes OCL pour spécification plus précise



Contraintes OCL, quelques exemples

– context Interface inv: attributes -> isEmpty()

Une interface est une classe sans attribut

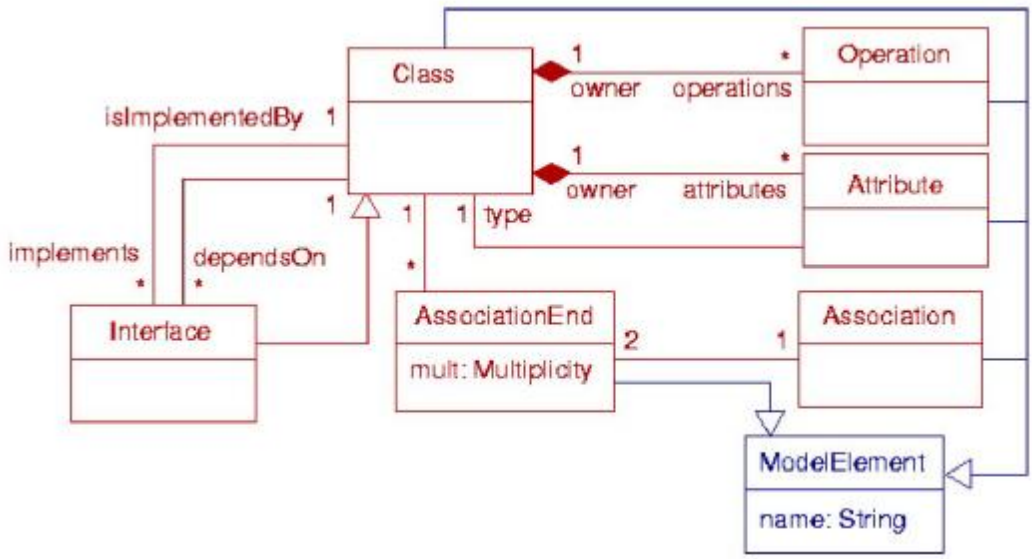
– context Class inv: attributes -> forAll (a1, a2 |a1 <> a2 implies a1.name <> a2.name)

2 attributs d'une même classe n'ont pas le même nom

– context StateMachine inv: transition -> forAll (t |self.state -> includesAll(t.state))

– Une transition d'un diagramme d'état ne connecte que des états de ce diagramme d'état

métamodèle M3



4. **Relations entre les Niveaux**

4.1 Relation d'InstanceOf

M0 "est une instance de" M1 "est une instance de" M2 "est une instance de" M3

Exemple complet :

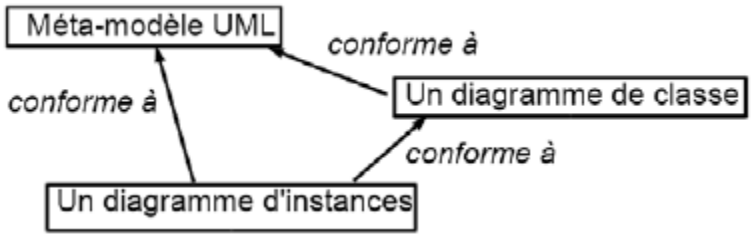
```
// Niveau M3 - MOF
Class { name, properties, operations }

// Niveau M2 - Métamodèle UML (instance de MOF)
instance Class of MOF!Class {
  name = "Class",
  properties = {name, attributes, methods},
  operations = {}
}

// Niveau M1 - Modèle Bancaire (instance de UML)
instance Client of UML!Class {
  name = "Client",
  attributes = {id, nom, prenom},
  methods = {}
}
```

```
// Niveau M0 - Données (instance de Modèle)
instance cli_001 of Client {
  id = 1,
  nom = "Aissani",
  prenom = "mohamed"
}
```

4.2 Relation de Conformité

Un modèle est **conforme** à son métamodèle s'il respecte :

- Toutes les contraintes syntaxiques
- Toutes les règles sémantiques
- Toutes les contraintes OCL définies



5. Rôle du Métamodèle et Méta-méta-modèle

5.1 Rôle du Métamodèle (M2)

- **Définir** le vocabulaire de modélisation
- **Spécifier** les règles de construction
- **Assurer** la cohérence des modèles
- **Permettre** l'interopérabilité entre outils

5.2 Rôle du Méta-méta-modèle (M3)

- **Fournir** un langage standard pour définir des métamodèles
- **Assurer** la réflexivité (MOF peut se décrire lui-même)
- **Permettre** la sérialisation standard (XMI)
- **Garantir** l'interopérabilité au niveau métamodèle

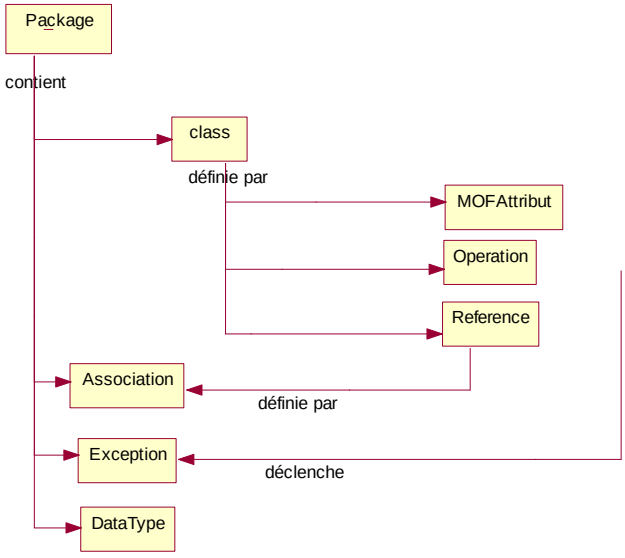
6. Le Meta-Object Facility (MOF)

6.1. Caractéristiques Clés du MOF

Le MOF se situe au niveau M3 et est une spécification de l'OMG fortement inspirée de l'approche orientée objet et d'UML.

1. **Standardisation et Interopérabilité** : Il permet de relier les différents modèles standards de l'OMG (UML, CWM) et assure l'interopérabilité des métamodèles.
2. **Base pour l'échange XMI** : Le MOF est le mécanisme par lequel les modèles sont sérialisés au format **XMI** (XML Metadata Interchange), facilitant l'échange de métadonnées.
3. **Langage Abstractionnel** : Le MOF fournit un modèle abstrait d'objets. Il n'a pas de syntaxe concrète définie, mais propose **UML** (graphique) et **MODL** (textuel) comme notations pour exprimer les méta-modèles.
4. **Réflexivité** : Les applications MOF peuvent manipuler des modèles à l'aide d'opérations génériques sans connaître le domaine spécifique.

6.2. Les Méta-Entités Principales du MOF Le MOF définit l'ensemble des concepts de base permettant de construire un métamodèle. Les méta-entités principales sont **Package, Class, MOFAttribut, Reference et Operation**.



4. Relations Structurelles du MOF

Le MOF définit un ensemble de relations (associations) qui permettent de structurer les méta-entités.

4.1 La Relation Generalizes (Héritage)

Cette relation est applicable aux classes et aux paquetages.

- Elle lie un **sous-type** (subtype) à son ou ses **super-types** (supertype).
- Elle est navigable du sous-type vers le super-type, permettant l'accès direct aux super-types via une référence supertypes.

4.2 La Relation Contains (Contenance)

C'est la relation de composition structurelle au sein d'un métamodèle.

- Elle permet de rattacher :
  - Classes, Associations et autres entités à un **Paquetage**.
  - Attributs et Opérations à leur **Classe**.
  - Paramètres à leur **Opération**.
- Elle est navigable dans les deux sens : un élément a une référence Container et un conteneur a une référence contents.

4.3 La Relation IsTypeOf (Typage)

Cette relation lie un élément de modélisation typé à son type.

- Elle lie le **typedElement** (ex: un Attribut, un Paramètre, une Extrémité d'Association) au **type** (Classe ou DataType).
- Elle est navigable du typedElement vers son type via une référence type.

4.4 La Relation DependsOn (Dépendance)

Elle représente les dépendances sémantiques entre les éléments de modélisation.

- Elle lie l'élément **dépendant** à tous les éléments dont il dépend (**provider**).
- Elle est navigable du dépendant vers le provider via une référence requiredElement.

4.5 La Relation Constraints

Cette relation permet de lier un élément de modélisation (le constrainedElement) à un ensemble de **contraintes** (comme les contraintes OCL) qui le référencent. Elle est navigable dans les deux sens.

7. Réflexivité de MOF

L'auto-définition du MOF signifie que le langage utilisé pour définir les règles des métamodèles est **suffisamment expressif** pour se définir lui-même. C'est l'ultime niveau d'abstraction de l'architecture de modélisation.

Pour le comprendre, il faut se référer à la hiérarchie classique de modélisation à **quatre couches** (M0 à M3) utilisée en IDM :

Pourquoi l'auto-définition est-elle nécessaire ?

La capacité du MOF à s'auto-définir (c'est-à-dire que la couche **M3** est l'instance de la couche **M3** elle-même) assure une **cohérence** et une **stabilité** fondamentales :

1. **Uniformité** : Elle garantit que les mêmes concepts et règles de modélisation (classes, propriétés, héritage, etc.) sont utilisés pour définir tous les langages de modélisation (M2).
2. **Fondement Solide** : Elle ancre l'ensemble de la pyramide de l'IDM. Si la couche la plus haute (le langage de définition des langages) est capable de se décrire, alors toute la structure en dessous est basée sur un socle unique et non-ambigu.

En termes simples : le MOF est le **langage des langages de modélisation**. Le fait qu'il s'auto-décrive signifie que vous pouvez utiliser le MOF pour décrire ce qu'est le MOF lui-même.

MOF peut se décrire lui-même :

```
// MOF se modélise lui-même
instance MOFClass of MOF!Class {
  name = "Class",
  isAbstract = false,
  ownedAttribute = {nameAttr, isAbstractAttr}
}

instance nameAttr of MOF!Property {
  name = "name",
  type = StringType,
  owningClass = MOFClass
}
```

8. Exemples par Domaine

8.1. Domaine : E-Commerce

NIVEAU M3 (MOF) :
  Class, Property, Association, DataType

NIVEAU M2 (Métamodèle E-Commerce) :
Produit, Commande, Client, Paiement, Livraison

NIVEAU M1 (Modèle de boutique) :
Livre, Vêtement, CommandeEnLigne, ClientPremium

NIVEAU M0 (Données) :
Produit#123: "Livre IDN", prix=29.90€
Client#456: "aissani mohamed", aissani@email.com

7.2 Domaine : Santé

NIVEAU M3 (MOF) :
Class, Property, Enumeration, Constraint

NIVEAU M2 (Métamodèle Santé) :
Patient, Médecin, RendezVous, Diagnostic, Traitement

NIVEAU M1 (Modèle hôpital) :
PatientAdulte, PatientEnfant, Consultation, Urgence

NIVEAU M0 (Données) :
Patient#789: "mounir", groupeSanguin="A+"
RendezVous#321: "2024-03-15 10:30", statut="confirmé"

8. Avantages de l'Architecture à 4 Niveaux

8.1 Avantages Techniques

- **Interopérabilité** entre outils via XMI
- **Réutilisabilité** des métamodèles
- **Extensibilité** des langages de modélisation
- **Validation** automatique des modèles

8.2 Avantages Méthodologiques

- **Séparation des préoccupations**
- **Abstraction progressive**
- **Maintenabilité** des modèles
- **Évolutivité** du patrimoine de modélisation

8.3 Avantages pour l'Ingénierie Dirigée par les Modèles

- **Génération** de code cohérente
- **Transformation** de modèles standardisée
- **Vérification** automatique de conformité
- **Gouvernance** des modèles

10. Conclusion

L'architecture à 4 niveaux de MOF constitue le **fondement théorique** de l'ingénierie dirigée par les modèles. Elle permet :

- Une **définition rigoureuse** des langages de modélisation
- Une **séparation claire** des préoccupations
- Une **interopérabilité** entre outils
- Une **évolution contrôlée** des standards