Chapitre 3 Meta-Object Facility(MOF)

1. Introduction

Au cours des deux dernières décennies l'approche orientée objet est peu à peu devenue l'approche de choix pour le développement de logiciels dans l'industrie. Or, l'objectif de l'IDM est de proposer une approche intégrant de manière homogène différents espaces technologiques dans laquelle l'approche orientée-objet et la technologie correspondante ne représentent qu'un cas particulier.

L'approche orientée objet est basée sur deux relations fondamentales: la relation *InstanceDe* qui permet d'introduire la notion de classe et la relation *HériteDe* qui permet d'introduire la notion de superclasse. La première relation, appelée *ReprésentationDe* est liée à la notion de *modèle*, alors que la relation *EstConformeA* permet de définir la notion de modèle par rapport à celle de *métamodèle*.

2. Modèles et *Représentation De modèles*

Il n'existe pas à ce jour de définition universelle de ce qu'est un modèle (tout comme il n'existe pas de définition universelle de ce qu'est un objet, un aspect, etc.)

On cite trois définitions tirés de la littérature.

La définition ci-dessous provient du standard UML

*"A **model** is an abstraction of a physical **system**, with a certain purpose."*

La mention système "physique" est clairement de trop, car cela signifierait qu'il n'est pas possible de définir des concepts abstraits comme la notion de cours, de session ou de transaction dans de tels modèles.

Seidwitz rapporte des travaux récents concernant l'étude des concepts fondamentaux de l'IDM. La définition suivante :

*"A **model** is a set of statements about some **system under study** (SUS)."*

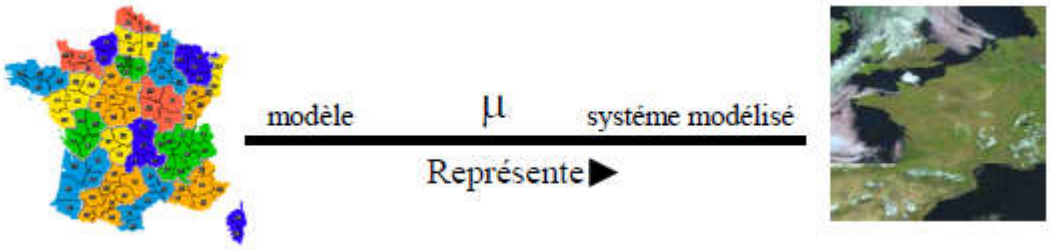
Finalement, une définition plus complète est proposée par Bézivin et Gérbé :

*"A **model** is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual **system**."*

Comme le suggèrent ces définitions provenant de sources différentes, il existe un consensus sur le fait que ***modèle*** et ***système étudié*** sont deux rôles complémentaires basés sur une unique relation liant un modèle au système qu'il modélise. Cette relation est appelée ***ReprésentationDe*** dans Notée µ pour éviter toute connotation et confusion.

Ces définitions ne sont donc pas restreintes aux modèles et systèmes informatiques.

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



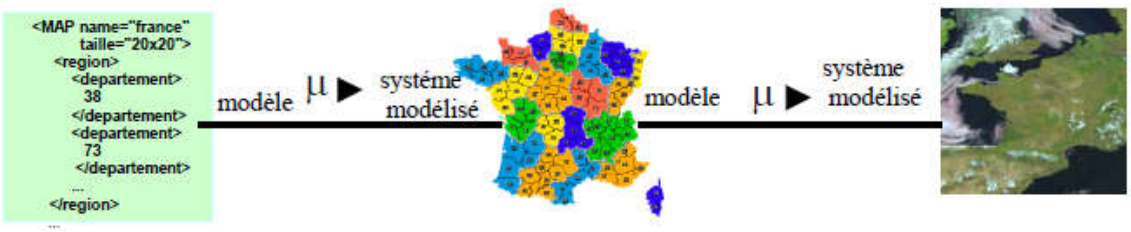
Par exemple la région considérée dans Le modèle que l'on fait d'un tel système peut lui être décrit dans un langage précis. C'est par exemple le cas d'une carte numérique dont on connaît parfaitement la structure.

Un modèle informel peut trouver son utilité dans le cadre d'activités humaines. Par exemple la cartographie a montré l'utilité de tels modèles aux cours des siècles. L'utilisation de modèles informels en informatique est aussi une pratique courante. Les limites des modèles contemplatifs sont néanmoins connues.

Dans le cadre de l'Ingénierie Dirigée par les Modèles on s'intéresse aux modèles formalisés pour les rendre productifs en automatisant leur traitement et leur exploitation informatique. Dans le contexte de l'IDM, Warmer et ses collègues donnent la définition suivante:

*"A **model** is a description of (part of) a **system** written in a **well-defined language**"*

Par exemple une description (formelle ou informelle) d'une carte géographique permet d'obtenir un modèle de cette carte (qui joue ainsi le rôle de système modélisé); la carte jouant à son tour le rôle de modèle par rapport à la région.



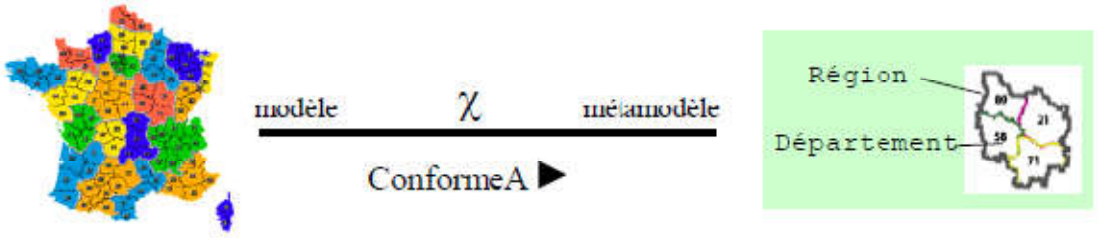
Pour qu'un modèle soit productif, et c'est là l'objectif premier de l'IDM, il faut qu'il puisse être manipulé par une machine et donc que le langage dans lequel ce modèle est exprimé soit clairement défini. La notion de métamodèle est au coeur même de l'IDM. Bien que certains aspects relatifs à cette notion restent sujets à controverses, on peut noter néanmoins une convergence dans les définitions que l'on trouve dans la littérature:

*"A **meta-model** is a model that defines the language for expressing a **model**".*

La notion de métamodèle mène à l'identification d'une seconde relation nommée ***ConformeA***.

La force de l'IDM est de reconnaître par ailleurs que la relation de conformité a plusieurs incarnations selon les Espaces Technologiques considérés. Par exemple dans l'espace technologique XML, on dira qu'un document XML est conforme à un schéma ou à une DTD; dans l'espace technologique des grammaires on dira qu'une phrase ou un programme est conforme à une grammaire donnée; dans l'espace technologique des bases de données on dira que le contenu d'une base de données est conforme au schéma de cette base de données, dans l'espace technologique orienté-objet on dira qu'un objet est conforme à sa classe. La relation *InstanceDe* correspond donc à l'incarnation de la notion de conformité dans le cadre orientée objet mais ce n'est là qu'un cas particulier.

La longue histoire de la cartographie, représentative s'il en est de l'évolution de la notion de modèle au cours des ages, montre qu'il est désormais jugé indispensable d'associer à chaque carte la description du "langage" utilisé pour réaliser cette carte. Ceci se fait notamment sous la forme d'une légende explicite. La carte doit être conforme à cette légende, sinon elle n'est pas utilisable. Bien évidemment plusieurs cartes peuvent être conformes à une même légende.



Désormais la cartographie n'est plus liée uniquement à la géographique. Il existe une foule de cartes spécialisées décrivant différents domaines: cartes géopolitiques, économiques, minières, archéologiques, vinicoles, etc. A chaque fois il s'agit de définir un langage spécialisé de domaine et de définir une légende appropriée. L'importance de définir et de développer ces formalismes est reconnu comme étant une activité à part entière, tout comme le développement et l'évolution de modèles représentant des régions particulières.

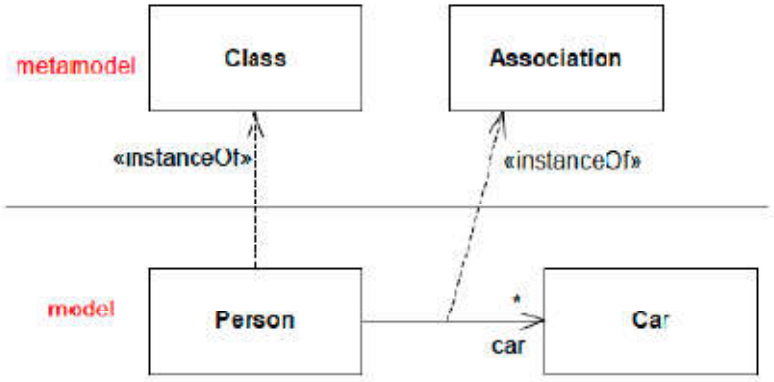
Il en est de même pour l'Ingénierie Dirigée par les Modèles. Les métamodèles deviennent des entités de première classe. Il doit entre autre être possible de définir des langages spécifiques de domaine (DSL) pour adresser les spécificités liées à tels secteurs d'activités, à tel métier, à tel point de vue, à tel niveau d'abstraction, ou à telles plates-formes technologiques.

3. Méta-modèles

Modèle d'un ensemble de modèles, Grammaire décrivant un langage de modélisation,
Définit la syntaxe abstraite des modèles

Pourquoi des méta-modèles ?
Pour éditer et valider les modèles Pour transformer des modèles ; Règles de transformation entre méta-modèles

Exemple d'une classe:

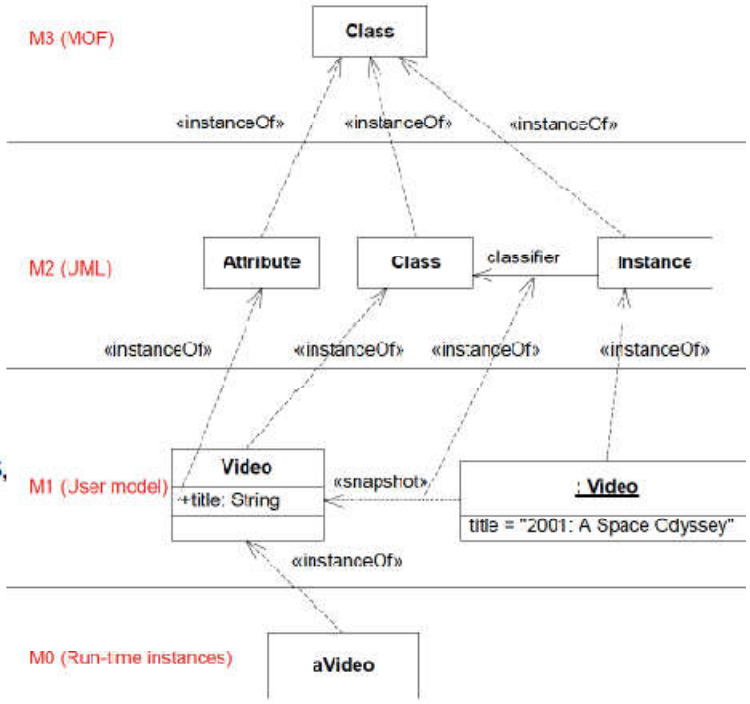


4. Méta-méta-modèles
Modèle d'un ensemble de méta-modèles, Grammaire décrivant les grammaires décrivant des langages de modélisation, Définit la syntaxe abstraite des méta-modèles

Remarque :
Comment faire pour ne pas entrer dans une boucle infinie ?
Définir le méta-méta-modèle à l'aide de lui-même Méta-circularité !

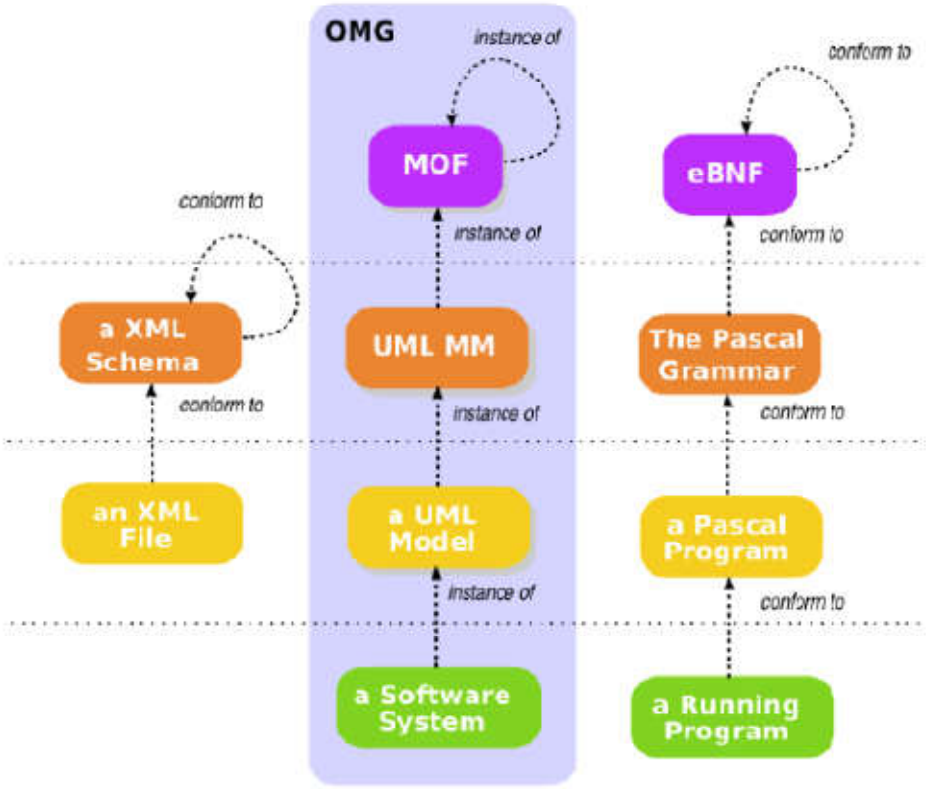
4 niveaux de (méta-) modélisation selon l'OMG

- 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



Exemple :

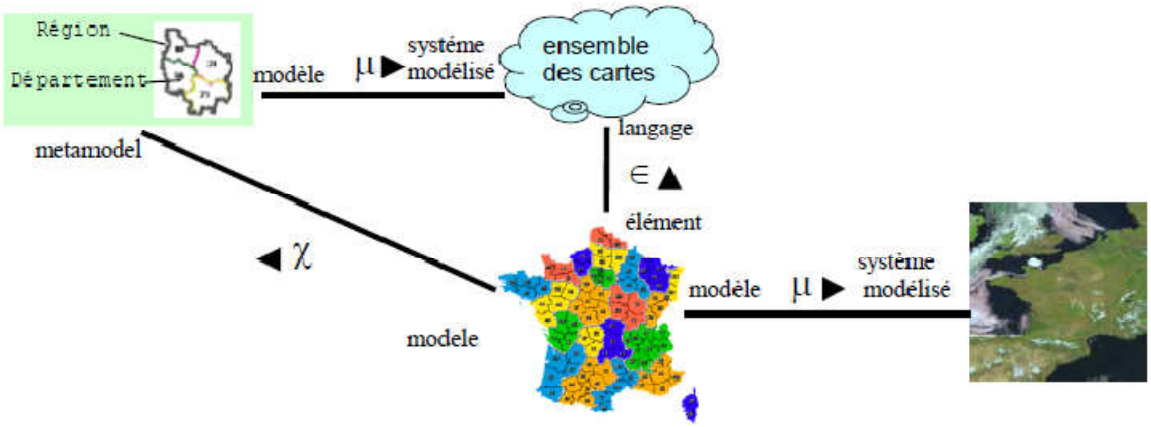
Méta-modélisation, XML Schemas, et Grammaires



5. Méta modèles vs. Langages

Bien souvent la notion de langage et de métamodèle sont confondus. Bien que ces deux concepts soit proches, ils sont néanmoins différents. Revenons sur la définition donnée auparavant: *"A **meta-model** is a model that defines the **language** for expressing a **model**"* . Un langage est un système abstrait alors qu'un métamodèle est une définition explicite de ce langage. Il faut donc distinguer le langage, qui joue le rôle de système, du (ou des) méta modèle(s) qui jouent le rôle de modèle(s) de ce langage.

Si l'on reprend l'exemple des cartes, on peut considérer par exemple le langage "des cartes IGN". Ce langage *est* l'ensemble de toutes les cartes décrites selon des conventions définies par l'Institut Géographique National. Comme le suggère la figure suivante, une légende *n'est pas* un langage (un ensemble) mais un outil concret pour définir et appréhender un langage de cartes. Une légende est une *ReprésentationDe* de ce langage. Elle joue le rôle de *modèle* par rapport au langage et de *métamodèle* par rapport à une carte particulière qui doit être *ConformeA* cette légende. La carte, quant à elle, est un élément du langage.



En effet un langage est un ensemble (au sens mathématique du terme) de phrases, alors qu'une grammaire est un modèle de ce langage (de cet ensemble). La notion de grammaire correspond, dans l'Espace Technologique Langage, à la notion de métamodèle en utilisant la terminologie unificatrice IDM. D'un point de vue pratique c'est la relation de conformité qui est utilisée et qui est implantée par exemple par un analyseur. Dans la langue parlée, remarquons qu'on assimile souvent langage et grammaire. Un métamodèle est un moyen concret de *définir* un langage, ce n'est pas un langage.

Exemple : 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

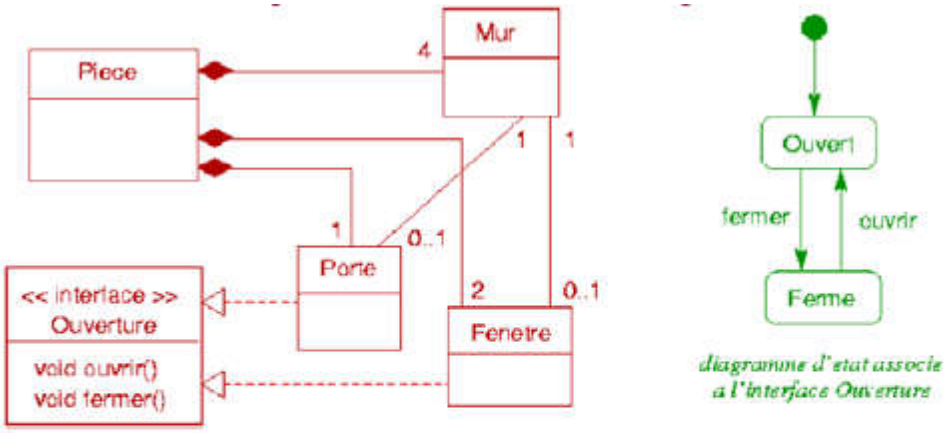
Pour modéliser ce système, il faut définir 2 diagrammes UML : niveau M1

• 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

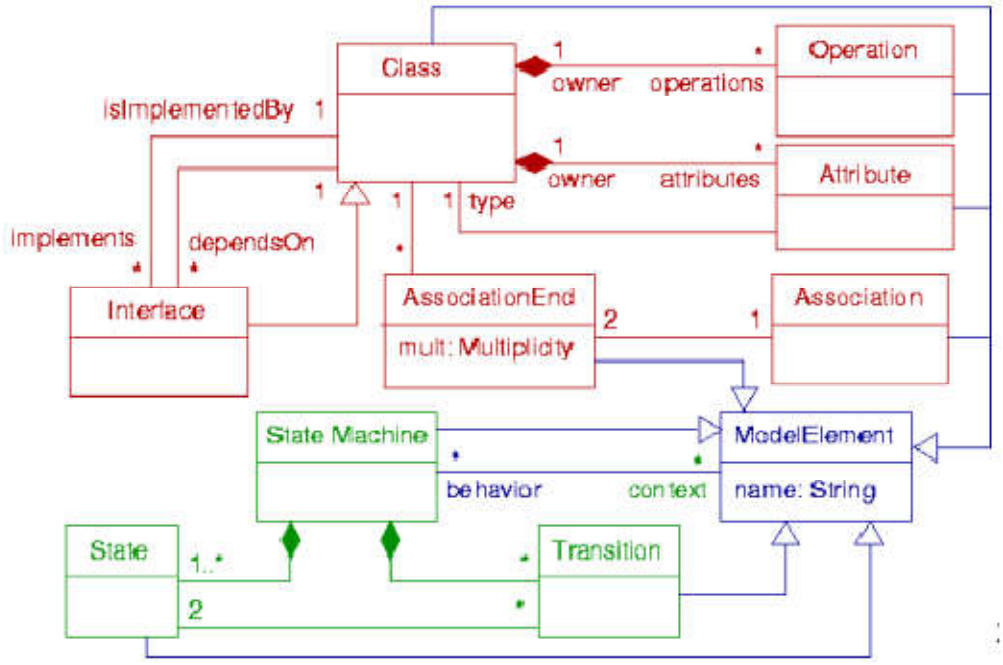
– Le diagramme d'état est associé à cette interface
• Il faut également ajouter des contraintes OCL pour préciser les contraintes entre les éléments d'une pièce



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.

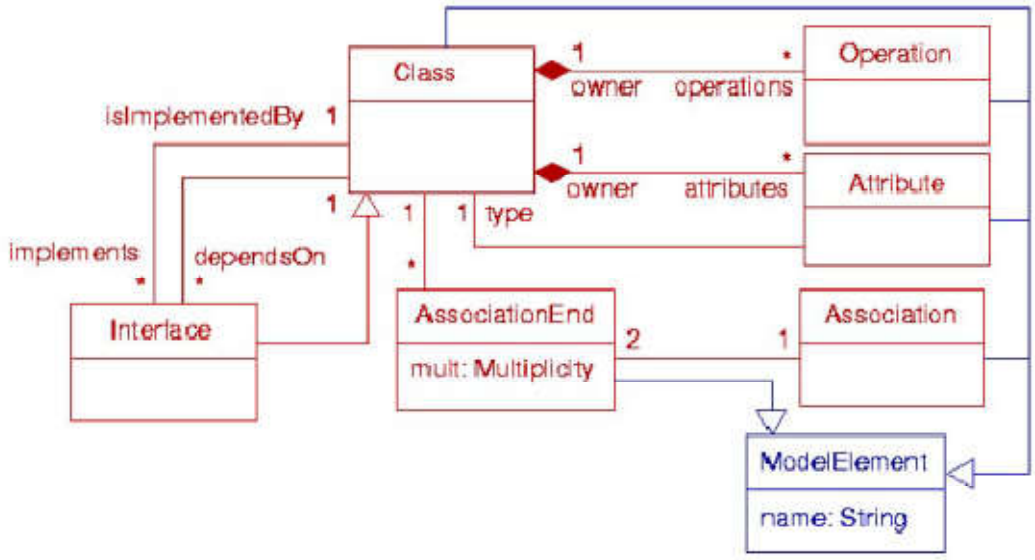
• Les contraintes sur les éléments des diagrammes UML et leurs relations sont définies
– Dans le métamodèle UML : niveau M2
– Un diagramme UML (de classes, d'états ...) doit être conforme au métamodèle UML
• Métamodèle UML
– Diagramme de classes spécifiant la structure de tous types de diagrammes UML
– Avec contraintes OCL pour spécification plus précise



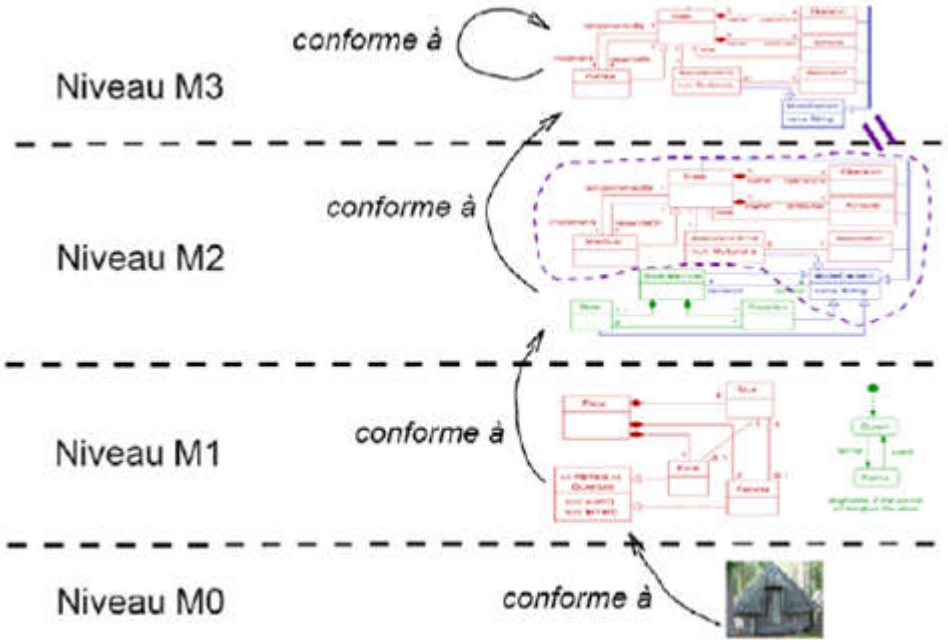
Contraintes OCL, quelques exemples
– context Interface inv: attributes -> isEmpty()
Une interface est une classe sans attribut
– context Class inv: attributes -> forAll (a1, a2 |a1 <> a2 implies a1.name <> a2.name)
2 attributs d'une même classe n'ont pas le même nom
– context StateMachine inv: transition -> forAll (t |self.state -> includesAll(t.state))

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

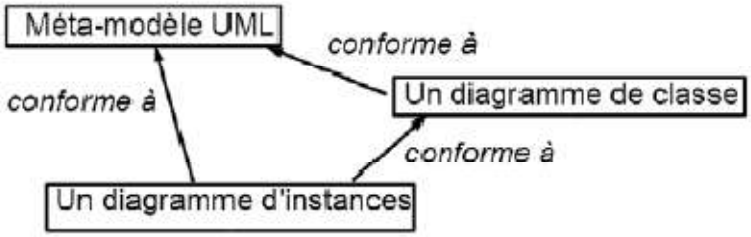
métamodèle M3



Hiérarchie de modélisation



Un diagramme d'instances est particulier car
– Doit être conforme au métamodèle UML
  • Qui définit la structure générale des diagrammes d'instances
– Doit aussi être conforme à un diagramme de classe
  • Diagramme de classe est un métamodèle
  • Qui doit être conforme également au métamodèle UML



Un langage est défini par un métamodèle
• Un langage possède une syntaxe
– Précise comment représenter chaque élément d'un modèle
– Elément d'un modèle = instance d'un métaélément
• Syntaxe textuelle
– Ensemble de mots-clés et de mots respectant des contraintes défini selon des règles précises : Notions de syntaxe et de grammaire dans les langages Notation graphique, chaque élément a une forme graphique particulière
• Exemple : associations entre classes/interfaces sur les diagrammes de classe UML
  – Trait normal : association
  – Flèche, trait pointillé : dépendance
  – Flèche en forme de triangle, trait en pointillé : implémentation
  – Flèche en forme de triangle, trait plein : spécialisation

Abstraite
– Les éléments et leurs relations sans une notation spécialisée
– Correspond à ce qui est défini au niveau du métamodèle, à des instances de méta-éléments
• Concrète
– Syntaxe graphique ou textuelle définie pour un type de modèle
– Plusieurs syntaxes concrètes possibles pour une même syntaxe abstraite
• Un modèle peut être défini via n'importe quelle syntaxe
– L'abstraite
– Une des concrètes
• MOF : langage pour définir des métamodèles
– Pas de syntaxe concrète définie

Questions :

1. **Faire le méta modèle du diagramme de cas d'utilisation.**
2. **Faire la correspondance entre diagramme de cas d'utilisation et son méta modèle.**
3. **Faire le méta modèle du diagramme de classe.**

6. L'utilité de MOF :

Le langage MOF fournit le standard de méta-modélisation et d'échange de constructions utilisées par MDA. Les autres modèles standards de l'OMG, comme UML et CWM, sont définis par des constructions MOF, ce qui permet de les relier entre elles. C'est également le mécanisme par lequel les modèles sont sérialisés en XMI. Le MOF est un exemple de méta-méta-modèle, ou de modèle du méta-modèle. Il définit les éléments essentiels, la syntaxe et la structure des méta-modèles utilisés pour construire des modèles orientés objet. **La spécification MOF fournit les points suivants** :

– Un modèle abstrait d'objets MOF génériques et leurs associations.

– Un ensemble de règles pour exprimer un méta-modèle MOF à l'aide d'interfaces IDL. Une implantation de ces interfaces pour un méta-modèle donné peut être utilisée pour manipuler une instance de celui-ci (un modèle).

– Un ensemble de règles sur le cycle de vie, la composition et la fermeture sémantique des éléments d'un méta-modèle MOF.

– Une hiérarchie d'interfaces réflexives permettant de découvrir et manipuler des modèles basés sur des méta-modèles MOF dont on ne connaît pas les interfaces.

Un intérêt du MOF est qu'il permet de faire interopérer des méta-modèles différents. Une application MOF peut manipuler un modèle à l'aide d'opérations génériques sans connaissances du domaine.

7. Les spécifications du MOF

Le MOF est une spécification volumineuse .Elle décrit les éléments du méta-méta-modèle, c'est-à-dire les concepts utilisés pour la méta-modélisation, les notations possibles et leurs restrictions, ainsi que des règles de projections (correspondance MOF-IDL, correspondances MOF-XML (XMI)). De plus, des règles de projection particulières ont été spécifiées pour les modes statique (interopérabilité pour échange de fichiers) et dynamique (interopérabilité via des interfaces : MOF-IDL de DCOM par exemple).

Le modèle MOF peut être considéré de plusieurs façons différentes telles qu'un formalisme de représentation de modèles, un outil de création de modèles, un outil de validation de modèles.

Le standard ne contraint pas l'utilisation d'une notation particulière pour la méta-modélisation à l'aide du MOF. Toutefois le standard propose deux notations : UML qui est graphique et MODL qui est textuel.

8. Eléments du MOF

La **figure ci-dessous** présente les principales méta-entités du MOF aussi que leurs principales relations. Le méta-méta-modèle MOF est orienté objet. Les méta-entités principales du MOF sont **Packages, Class, MOFAttribut, References et Operation**. Un paquetage (méta-entité Package) est destiné à contenir les entités (méta-entité Class) d'un méta-modèle et leurs relations (méta-entité Association). La méta-entité class dispose des mécanismes d'héritage. Une entité pourra donc être la spécialisation d'une autre entité. Une entité est définie par un ensemble d'attributs (méta-entité MOFAttribut), d'opérations (méta-entité Operation) et de références(méta-entité Reference). La méta-entité DataType permet de définir des types de valeur non-objet. La méta-entité exception permet de décrire les exceptions liées aux opérations. Enfin, les caractéristiques d'une référence sont définies par une association (méta-entité Association). Une association définit une relation entre deux entités. Une caractéristique importante d'une relation est la multiplicité de chacune de ces extrémités.

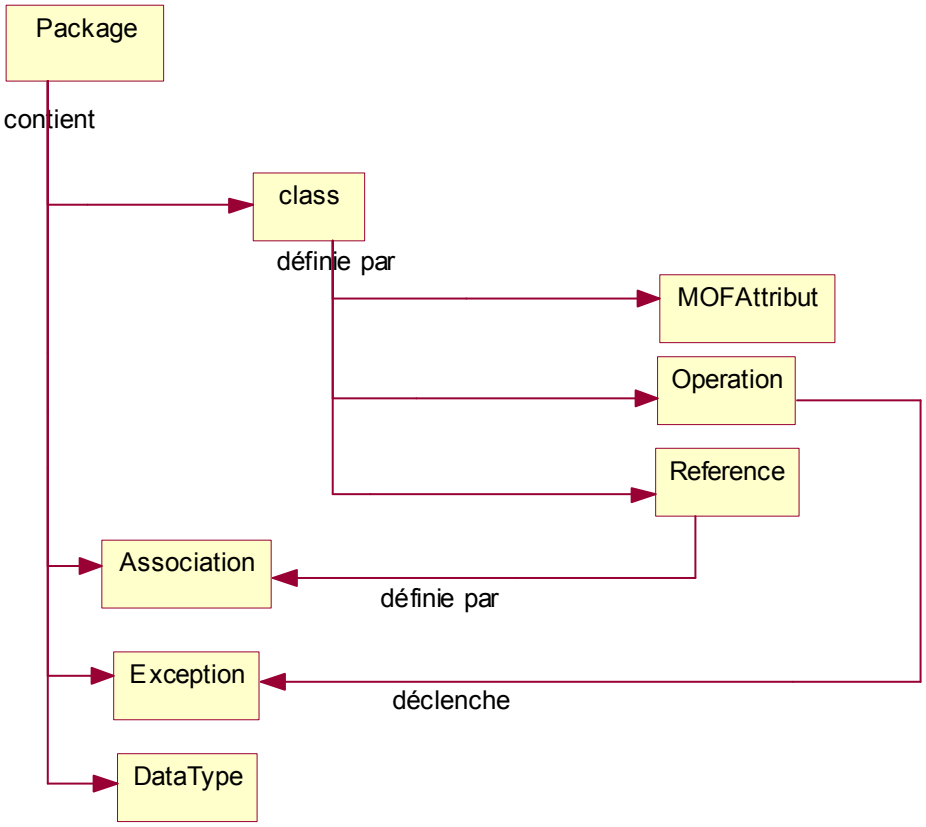


Figure : Principales méta-entités du MOF

Par exemple, l'entité Interface du méta-modèle OMG-IDL, peut être décrite par une classe MOF. De manière simplifiée, cette classe définit comme attributs, un nom, une liste d'attributs IDL et une liste d'opérations IDL. Elle définit ensuite une référence sur les interfaces dont elle hérite.

1. Les classes

Les classes sont des méta-objets MOF qui définissent les types des entités représentées via le MOF. Les classes sont principalement constituées de trois types d'entités – **attributs, opérations et références -** mais elles peuvent également contenir des **exceptions,** des **constantes**, **des contraintes**, des **types de données** ou encore **d'autres définitions de classes**.

Une classe dispose des attributs suivants :

- Un attribut **name** qui désigne le nom classe,
- Un attribut **annotation** qui décrit la classe,
- Un attribut **qualifiedName** qui désigne le nom global unique de la classe,
- Un attribut **isAbstract** qui détermine si la classe est abstraite, auquel cas il ne sera pas possible d'en créer des instances,
- Un attribut **isRoot** qui détermine si la classe est une racine d'un arbre d'héritage, auquel cas elle ne peut pas avoir de super-classes,
- Un attribut **isLeaf** qui détermine si la classe est une feuille d'un arbre d'héritage, auquel cas elle ne peut pas avoir de sous-classes,
- Un attribut **isSingleton** qui indique la classe ne peut pas disposer qu'une instance.

2. Les attributs

Un attribut est un méta-objet qui permet de rattacher des valeurs à une instance d'une classe MOF. La relation entre la valeur de l'attribut et l'instance aura une sémantique différente suivant que le type de l'attribut entre un classe MOF ou un type de données MOF.

Un attribut MOF dispose des attributs suivants :

- un attribut **name** qui désigne le nom de l'attribut MOF,
- un attribut **annotation** qui décrit le rôle de l'attribut MOF,
- un attribut **qualifiedName** qui désigne le nom global unique de l'attribut MOF,
- un attribut **scope** pouvant prendre pour valeur instance_level ou classifier_level et déterminant si la valeur de l'attribut MOF est portée par sa classe (et par conséquent commune à toutes les instances de la classe) ou par l'instance,
- un attribut **multiplicity** qui indique si la valeur de l'attribut MOF est optionnelle, simple ou multiple. Cet attribut détermine également dans le cas de valeurs multiples, si ces valeurs sont ordonnées et si elles sont uniques (toutes différentes deux à deux).

- un attribut **isChangable** qui détermine si la valeur de l'attribut MOF peut être modifiée,
- un attribut **isDerived** qui indique si la valeur de l'attribut est << déductible >> à partir d'autres attributs.

3. Les opérations

Les opérations permettent d'accéder au comportement dans une classe. Les opérations ne spécifient pas le comportement ou les méthodes qui vont implémenter ce comportement, elles spécifient simplement le nom et la signature par lesquels ce comportement peut être invoqué.

Une opération MOF dispose des attributs suivants :

- un attribut **name** qui désigne le nom de l'opération,
- un attribut **annotation** qui décrit le rôle de l'opération,
- un attribut **qualifiedName** qui désigne le nom global unique de l'opération,
- un attribut **scope** qui détermine si l'opération doit être invoquée indépendamment d'une instance particulière,
- un attribut **isQuery** qui indique que cette opération ne modifie pas l'état de la classe MOF sur lesquelles elle est invoquée.

4. Les associations

Les associations sont des méta-objets MOF qui permettent d'exprimer les relations dans un méta-modèle.

Une association dispose des attributs suivants :

- un attribut **name** qui désigne le nom de l'association,
- un attribut **annotation** qui décrit le rôle de l'association,
- un attribut **qualifiedName** qui désigne le nom global unique de l'association,

- un attribut **isAbstrac**t nécessairement à faux et sans intérêt pour une association MOF,
- un attribut **isLeaf** nécessairement à vrai pour assurer que l'héritage n'est pas applicable aux associations MOF,
- un attribut **isDerived** qui indique que cette association est dérivée.

5. Les extrémités d'associations

Une extrémité est un méta-objet MOF qui décrit l'extrémité d'un lien qui dispose des attributs suivants :

- un attribut **name** qui désigne le nom de cette extrémité de l' association,
- un attribut **annotation** qui décrit cette extrémité de l' association,
- un attribut **qualifiedName** qui désigne le nom global unique de cette extrémité,
- un attribut **multiplicity** qui indique le nombre d' instances de cette extrémité du lien pouvant être liés (via ce lien) à une instance de l'autre extrémité du lien,
- un attribut **agregation** pouvant prendre les valeurs none ou composite, et indiquant si l'objet connecté à cette extrémité du lien est un composé via cette relation, auquel cas cette relation est dite << composite >>,
- un attribut **isNavigable** qui détermine s'il est possible de définir une référence pour l' instance de cette extrémité du lien,
- un attribut **isChangeable** qui indique s'il est possible de modifier l'instance de cette extrémité du lien.

6. Les références

Une référence est un méta-objet MOF qui permet à une instance d'une classe MOF de connaître ses relations avec d'autres instances.

Une référence MOF dispose des attributs suivants :

- un attribut **name** qui désigne le nom de la référence MOF,

- un attribut **annotation** qui décrit le rôle de la référence MOF,
- un attribut **qualifiedName** qui désigne le nom global unique de la référence MOF,
- un attribut **scope** sans intérêt pour la référence MOF,
- un attribut **multiplicity** qui doit avoir la même valeur que l'attribut multiplicity de l'extrémité de l'association qui est référencée,
- un attribut **isChangeable** qui doit également avoir la même valeur que l'attribut isChangeable de l'extrémité de l'association qui est référencée.

7. Les paquetages

Les paquetages permettent de regrouper les éléments d'un méta-modèle.

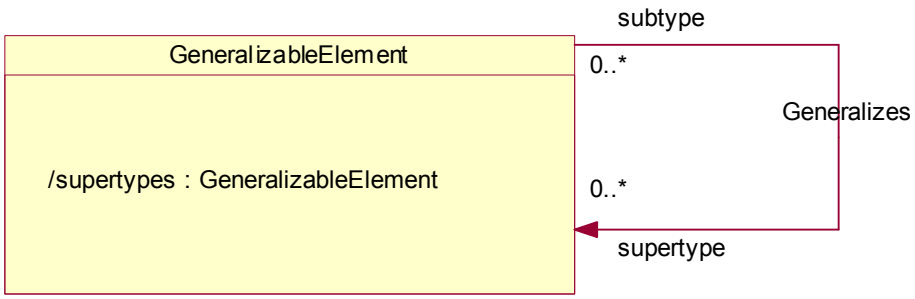
Un paquetage dispose des attributs suivants :

- un attribut **name** qui désigne le nom du paquetage,
- un attribut **annotation** qui décrit le paquetage,
- un attribut **isAbstract** nécessairement à faux et sans intérêt pour un paquetage MOF,
- un attribut **isRoot** qui détermine si le paquetage est une racine d'un arbre d'héritage,
- un attribut **isLeaf** qui détermine si le paquetage est une feuille d'un arbre d'héritage.

8. Descriptions des relations du MOF

1 La relation d'héritage(Generalizes)

Le MOF définit une relation d'héritage(Generalizes) applicable pour les éléments de type classes et pour les éléments de types paquetages. Cette relation lie donc un sous-type (le subtype) à son ou ses super-types (supertype). Bien que les types de données et les associations sont également des sous-types de GeneralizableElement(classe MOF définissant les éléments généralisables), le MOF ne permet pas l'héritage pour ces entités.

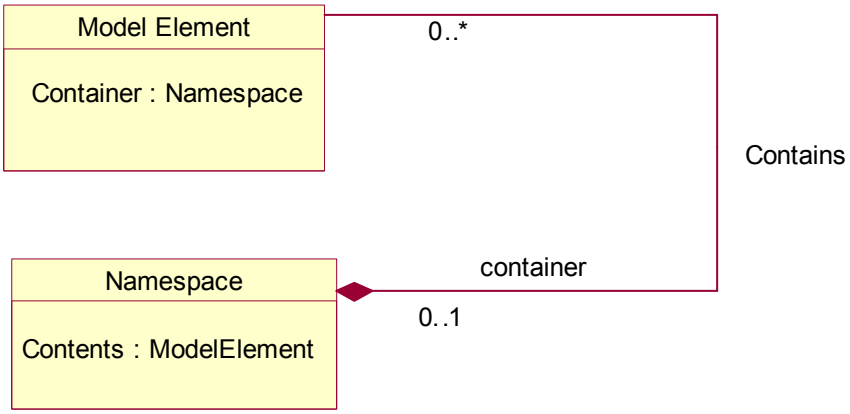


La relation Generalizes

Cette relation est navigable du sous-type vers le super-type. Ce qui permet la définition d'une référence supertypes de manière à accéder directement aux super-types d'un élément généralisable.

2 La relation de contenance (Contains)

Cette relation permet de rattacher les classes, associations et autre entités MOF pouvant être définies dans un paquetage à un paquetage. Mais, elle permet également de rattacher les attributs, et opérations à leur classe ou encore les paramètres à leur opération.

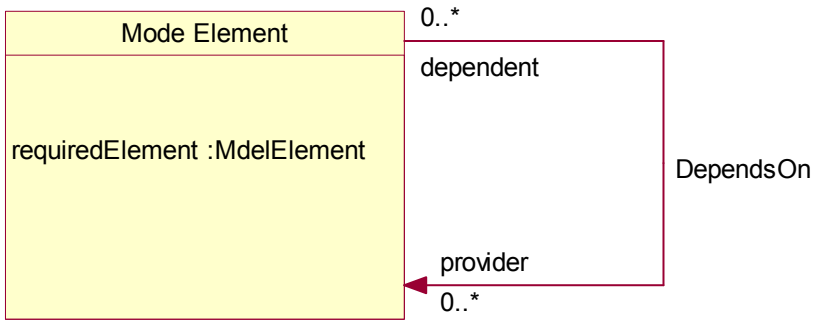


La relation Contains

Cette relation lie donc un élément (le contained Elément) à son conteneur (le Container). Elle est navigable dans les deux sens. Ce qui permet la définition d'une référence Container surtout les éléments de modélisation de manière à accéder directement à leur conteneur et d'une référence contents sur tous les conteneurs afin d'accéder directement à leur conteneurs.

3 La relation de dépendance (DependsOn)

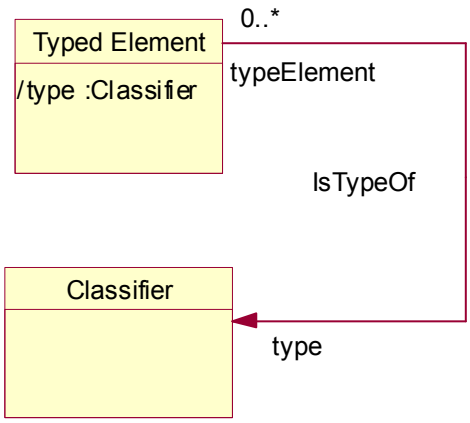
Cette relation permet de représenter les dépendances entre les éléments de modélisation. Elle lie un élément de modélisation (le dépendent) à tous les éléments dont il dépend (provider). Elle est navigable du dépendant vers le provider. Ce qui permet la définition d'une référence requiredElement sur tous les éléments de modélisation de manière à accéder directement à tous les éléments dont ils dépendent.



La relation dérivée DependsOn

4 La relation de typages(IsTypeOf)

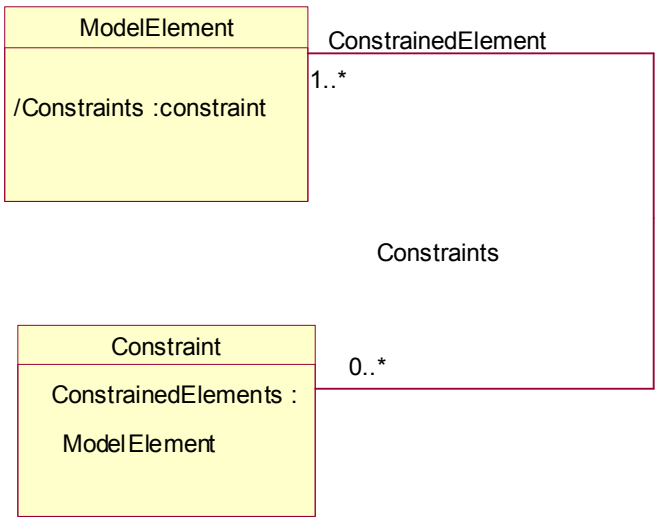
Cette relation permet de lier un élément de modélisation typé (un attribut, un paramètre, une constante) à son type (une classe ou un type de données). Elle est également utilisée pour lier les extrémités d'associations aux classes MOF (une extrémité d'association et un élément de modélisation typé). Cette relation lie donc un élément (le typedElement) à son type(le type). Elle est navigable du typedElement vers son type. Ce qui permet la définition d'une référence type sur tous les éléments de modélisation typés de manière à accéder directement à ce type.



La relation IsTypeOf

5. La relation entre les extrémités et leur contraintes (constraints)

Cette relation permet de lier un élément de modélisation aux contraintes qui le référence. Cette relation lie donc un élément de modélisation (le constrainedElement) à un ensemble de contraintes (les constraint). Elle est navigable dans les deux sens. Ce qui permet la définition d'une référence constraints sur tous les éléments de modélisation de manière à accéder directement à leurs contraintes et une référence constrainedElement sur toutes les contraintes afin d'accéder directement aux éléments contraints



La relation constraints

MOF peut manipuler un modèle à l'aide d'opérations génériques sans connaissances du domaine.

9. CONCLUSION

Nous avons élaboré le standard MOF de méta-méta-modélisation qui se situe au niveau le plus abstrait (M3) de l'approche MDA proposée par l'OMG , de plus nous avons vu ses spécifications et abordé son utilité dans cette approche.