

# **Méthodologie de Développement Objet**

## **Troisième partie : Ingénierie des Modèles**

Christine Solnon

INSA de Lyon - 4IF

2014 - 2015

# Automatiser la production de logiciels ?

## Saint Graal du développement logiciel :

→ Générer automatiquement le code à partir d'une spécification

## Point clé : Comment spécifier un système ?

- Description informelle en langue naturelle ?
  - Facile à comprendre par l'utilisateur
  - Mais ambiguë et impossible à exploiter automatiquement
- Description à l'aide d'un langage formel ?
  - Difficile à comprendre par un utilisateur non initié
  - Mais non ambiguë et plus facile à exploiter automatiquement

En pratique : Mélange d'informel et de formel

# Petite parenthèse sur les langages formels

- Langage = ensemble (potentiellement infini) de "mots"
  - ↪ Langage Java = ensemble des programmes Java
  - ↪ Langage UML = ensemble des modèles UML
- Langage formel = langage décrit par une grammaire formelle
  - ↪ Ensemble de règles définissant les mots syntaxiquement corrects

## Exemple de grammaire formelle :

$$\begin{aligned} \langle \text{Expr} \rangle & ::= \langle \text{Expr} \rangle ' + ' \langle \text{Terme} \rangle \mid \langle \text{Expr} \rangle ' - ' \langle \text{Terme} \rangle \mid \langle \text{Terme} \rangle \\ \langle \text{Terme} \rangle & ::= \langle \text{Terme} \rangle ' * ' \langle \text{Fact} \rangle \mid \langle \text{Terme} \rangle ' / ' \langle \text{Fact} \rangle \mid \langle \text{Fact} \rangle \\ \langle \text{Fact} \rangle & ::= \langle \text{Const} \rangle \mid \langle \text{Var} \rangle \mid ' ( ' \langle \text{Expr} \rangle ' ) ' \\ \langle \text{Const} \rangle & ::= \dots \\ \langle \text{Var} \rangle & ::= \dots \end{aligned}$$

- Automate = procédure qui décide si un mot appartient au langage décrit par une grammaire ↪ Construction d'un arbre de syntaxe abstraite

## Questions :

- Peut-on décrire tous les langages comme ça ?
- Comment décrire le langage des grammaires formelles ?

↪ Plus de détails dans le cours "Grammaires et langages"

# Qu'est-ce que la transformation de modèles ?

## Procédure qui transforme un modèle d'un langage vers un autre

- Modèle = mot d'un langage décrit par une grammaire formelle  
     $\rightsquigarrow$  Ex. : Diagramme de classes, Contrainte OCL, Programme Java, ...
- Langage source = langage du modèle en entrée
- Langage cible = langage du modèle en sortie

## Reformulation : Langage cible = Langage source

- Optimisation  $\rightsquigarrow$  Améliorer les performances en temps ou en mémoire
- Refactoring  $\rightsquigarrow$  Améliorer le code sans changer le comportement
- Re-engineering  $\rightsquigarrow$  Restructuration forte

## Traduction : Langage cible $\neq$ Langage source

- Migration  $\rightsquigarrow$  cible et source sont des versions différentes d'un même langage
- Synthèse de programmes  $\rightsquigarrow$  source = spécification ; cible = lang. de prog.
- Compilation  $\rightsquigarrow$  source de haut niveau ; cible de plus bas niveau
- Reverse engineering  $\rightsquigarrow$  source de bas niveau ; cible de plus haut niveau

## Comment transformer automatiquement des modèles ?

- Une grammaire décrit les mots "syntaxiquement corrects" :  
 ~> Construction d'un arbre de syntaxe abstraite à partir d'un mot
- Une grammaire ne dit pas comment interpréter le mot / monde réel  
 ~> Besoin d'associer une sémantique aux arbres de syntaxe abstraite
- Pour transformer automatiquement un mot  $m_1 \in L_1$  en un mot  $m_2 \in L_2$ , il faut connaître la correspondance sémantique des arbres de syntaxe abstraite de  $L_1$  vers ceux de  $L_2$   
 ~> Règles de transformation

Interprétation sémantique  
de  $m_1$  dans  $R_1$



Arbre de syntaxe  
abstraite de  $m_1$



Mot  $m_1$  de  $L_1$



Application  
de règles de  
transformation



Interprétation sémantique  
de  $m_2$  dans  $R_2$



Arbre de syntaxe  
abstraite de  $m_2$



Mot  $m_2$  de  $L_2$

## Les modèles dans un processus de développement classique :

- Elaboration de modèles
- Éventuellement : Génération de squelettes de code à partir de modèles
- Ecriture et maintenance du code
  - Les modèles deviennent obsolètes
  - Éventuellement : Re-génération de certains modèles  
~> Reverse engineering

## Les modèles dans un processus d'ingénierie des modèles

- Ecriture et maintenance de modèles
  - Génération du code à partir des modèles
- ~> Les modèles sont au centre du processus

~> **Objectif ambitieux... utopiste ou réaliste ?**

## Pour en savoir plus

- Ingénierie dirigée par les modèles : des concepts à la pratique  
J.-M. Jézéquel, B. Combemale, D. Vojtisek
- EMF : Eclipse Modeling Framework  
D. Steinberg, F. Budinsky, M. Paternostr  
Thomas Stahl, Markus Völter
- MDA en action : Ingénierie logicielle guidée par les modèles  
Xavier Blanc
- MDA Guide Version 1.0.1, 2003  
Site de l'OMG  
<http://www.omg.org/cgi-bin/doc?omg/03-06-01>

# Plan du cours

## 1 Introduction

## 2 Méta-modélisation, description et manipulation de modèles

- Méta-modélisation et MOF
- Contraintes et OCL
- XMI/JMI et la sérialisation/manipulation de modèles
- Synthèse

## 3 Model-Driven Architecture

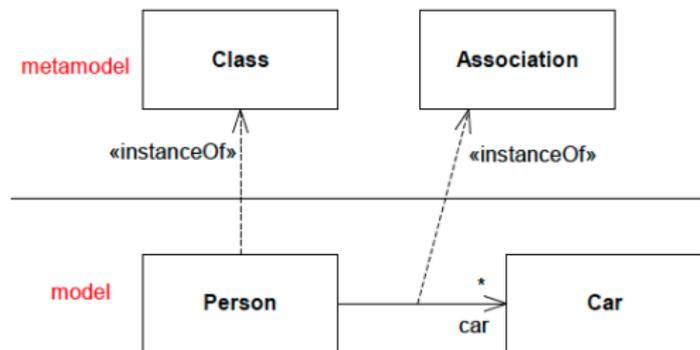
# Méta-modèles

## Qu'est-ce qu'un méta-modèle ?

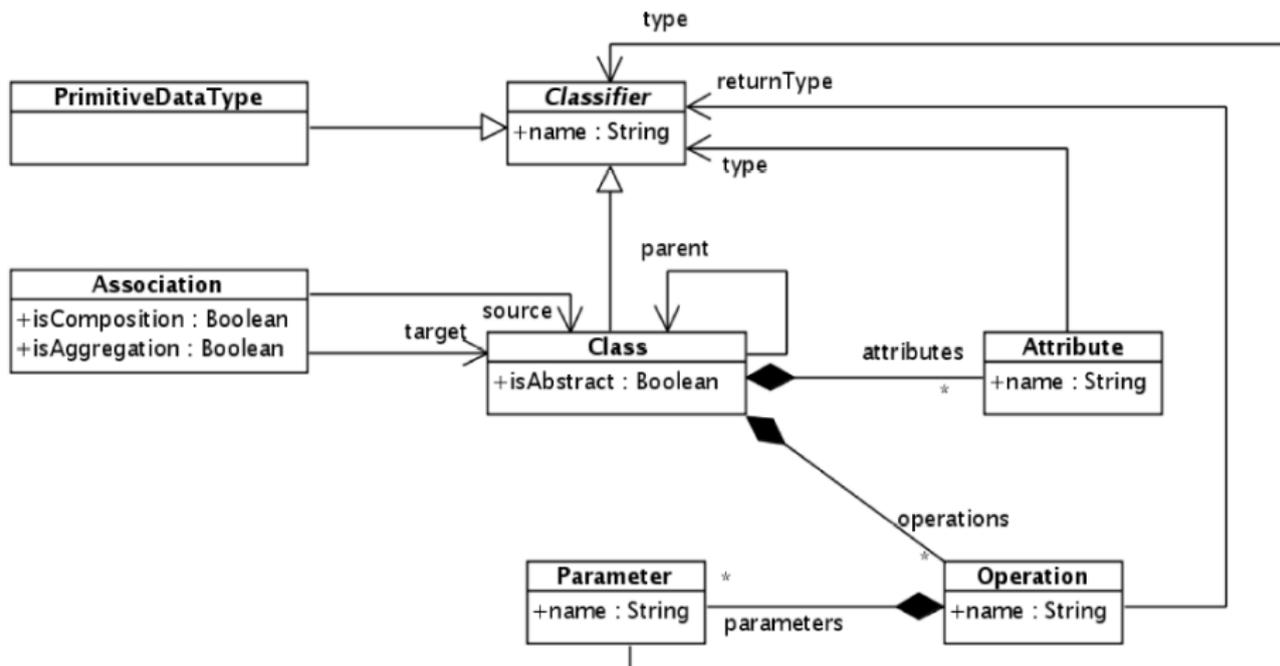
- Modèle d'un ensemble de modèles
  - ≈ Grammaire décrivant un langage de modélisation
  - ↪ Définit la syntaxe abstraite des modèles

## Pourquoi des méta-modèles ?

- Pour éditer et valider les modèles
- Pour transformer des modèles
  - ↪ Règles de transformation entre méta-modèles



## Exemple : Méta modèle des diagrammes de classes (simplifié)



~> Définit la syntaxe abstraite (relations entre éléments du modèle),  
mais pas la syntaxe concrète (représentation textuelle, graphique, ...)

# Méta-méta-modèles

## Qu'est-ce qu'un méta-méta-modèle ?

- Modèle d'un ensemble de méta-modèles
  - ≈ Grammaire décrivant les grammaires décrivant des lang. de modélisation
  - ↪ Définit la syntaxe abstraite des méta-modèles

## 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

M3 (MOF)

- M2 : Méta-modèles des modèles de M1

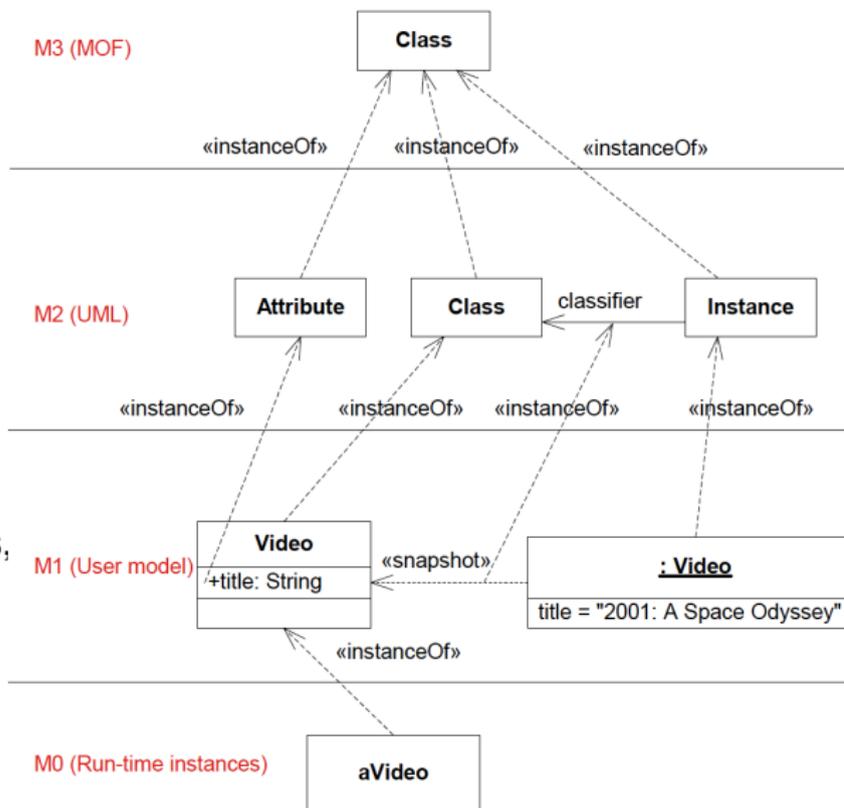
M2 (UML)

- M1 : Modèles (Diagrammes de classes, de séquence, ...)

M1 (User model)

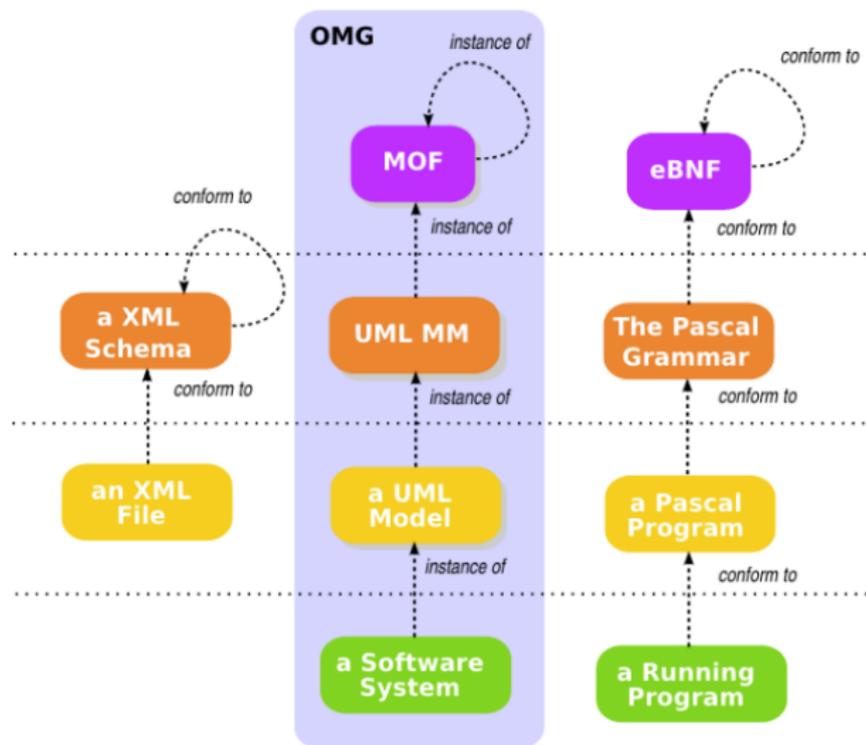
- M0 : Instances des modèles à l'exécution

M0 (Run-time instances)



[Image empruntée à www.omg.org]

# Méta-modélisation, XML Schemas, et Grammaires



# Le MOF (Meta Object Facilities)

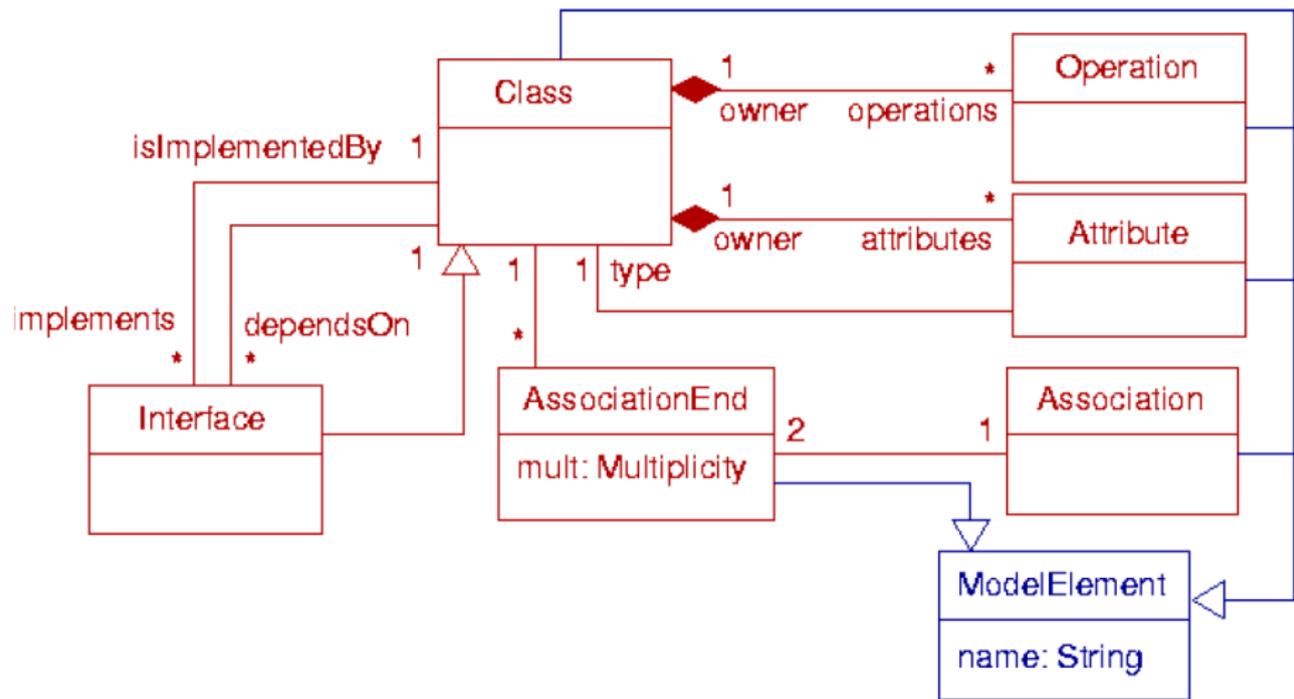
## Qu'est-ce que le MOF ?

- Méta-méta-modèle défini par l'OMG
  - Utilisé pour définir la syntaxe abstraite des méta-modèles UML
  - Définit les éléments des méta-modèles et leurs relations
- Méta-circulaire
  - Le MOF est défini en lui-même
- Correspond au niveau M3 de (méta-)modélisation selon l'OMG

## Pourquoi le MOF ?

- Standardisation des langages de méta-modélisation  
~> Outils génériques pour la transformation de modèles

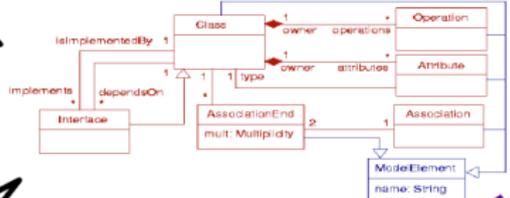
# Le MOF en version simplifiée



[Image empruntée à Eric Cariou]

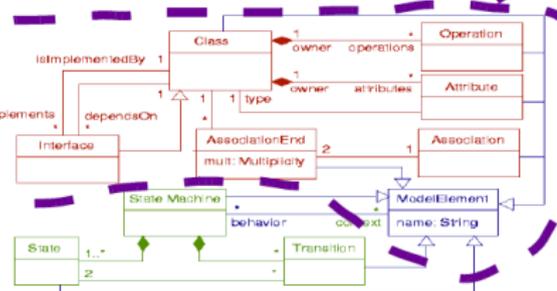
Niveau M3

conforme à



Niveau M2

conforme à



Niveau M1

conforme à

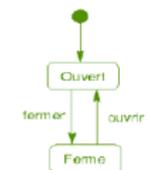
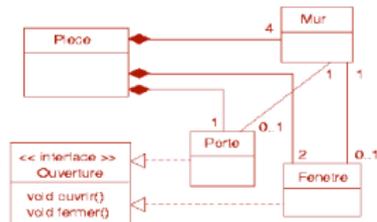
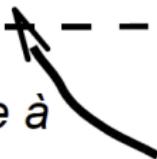


diagramme d'état associé à l'interface Ouverture

Niveau M0

conforme à



[Image empruntée à Eric Cariou]

# Description de UML2.0 et MOF2.0 par l'OMG

## Description d'UML2.0

- UML2.0 Infrastructure
  - Méta-modèle commun à UML et MOF
    - ↪ Méta-classes partagées : package, class, ...
  - Constitué d'une trentaine de packages "de base"
    - ↪ *Basic* : décrit diagrammes de classes sans association
    - ↪ *Constructs* : décrit diagrammes de classes avec associations
  - Utilisation de *merge* pour réutiliser les packages
- UML2.0 Superstructure
  - Méta modèle d'UML
  - Intégration de packages de l'infrastructure par *merge*

## Description de MOF2.0

- Essential MOF (EMOF) : décrit méta-modèles sans association
  - ↪ Intègre le package *Basic* de l'infrastructure
- Complete MOF (CMOF) : décrit méta-modèles avec associations
  - ↪ Intègre le package *Constructs* de l'infrastructure

# Plan du cours

## 1 Introduction

## 2 Méta-modélisation, description et manipulation de modèles

- Méta-modélisation et MOF
- Contraintes et OCL
- XMI/JMI et la sérialisation/manipulation de modèles
- Synthèse

## 3 Model-Driven Architecture

## Qu'est-ce qu'une contrainte ?

- Une contrainte est une relation entre des objets qui doit être satisfaite  
Elle peut être définie :
  - En extension, en énumérant tous les tuples de la relation  
(ex :  $(x, y) \in \{(1, 4), (2, 3), \dots\}$ )
  - En intention, à l'aide d'un langage formel (ex :  $x < y + 2 * z$ )
  - En langue naturelle (ex :  $x$  est au dessus de  $y$ )
- Une contrainte n'a **pas d'effet de bord**
  - ↪ Ne modifie pas l'état des objets
  - ↪ Décrit le "quoi", pas le "comment"

## Pourquoi des contraintes ?

- Description de règles métier
- Spécification de la sémantique des méthodes
- ... et plein d'autres choses que l'on ne peut exprimer dans un diag. UML

## La vérification de contraintes est un problème difficile...

- La difficulté dépend du langage utilisé pour exprimer les contraintes
- Généralement  $\mathcal{NP}$ -complet... parfois indécidable !

# Object Constraint Language (OCL)

## Qu'est-ce que OCL ?

Langage formel normalisé par l'OMG pour définir des contraintes  
~> Peut être utilisé pour compléter des diagrammes UML ou MOF

## Qu'est-ce qu'une contrainte OCL ?

- Expression booléenne : Relation qui doit être vérifiée
- 3 types de contraintes :
  - `pre` : Précondition vérifiée à l'appel d'une méthode
  - `post` : Postcondition vérifiée au retour d'une méthode
    - ~> `result` = Objet retourné par la méthode
    - ~> `x@pre` = État de l'objet `x` avant l'appel de la méthode
  - `inv` : Invariant vérifié à tout moment
- Contexte d'une contrainte : Classe ou Opération
- Syntaxe : `<typeContrainte> <nomContrainte> : <exprBool>`

## OCL et la programmation par contrat

- Spécifier ce que doit faire la classe... mais pas comment le faire !
- Utilisation (automatique ?) pour concevoir les tests unitaires

## Exemple 1 d'utilisation d'OCL : Modèle d'une pile

```
context Pile::depile(): Object
  pre nonVide : not estVide()
  post sommetRet : result = self@pre.sommet()
  post tailleDec : taille() = self@pre.taille()-1
```

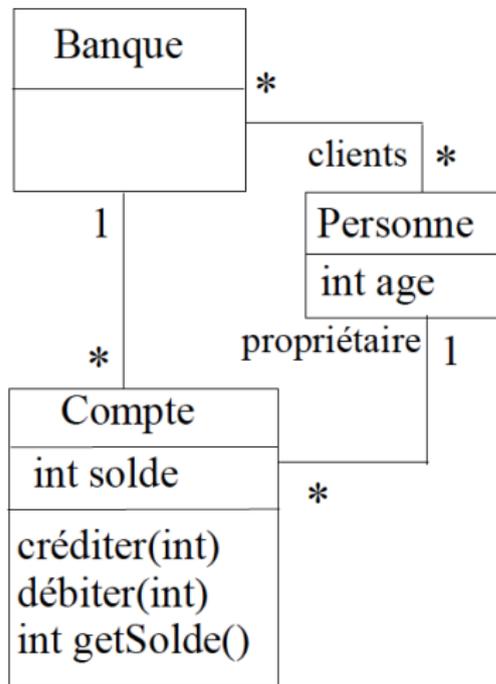
```
context Pile::sommet(): Object
  pre nonVide : not estVide()
```

```
context Pile::empile(unObjet: Object): void
  post LIFO : sommet() = unObjet
```

```
context Pile
  inv : taille() >= 0
  inv : taille() = 0 implies estVide()
  inv : estVide() implies taille()=0
```

...

## Exemple 2 d'utilisation d'OCL



**context** Compte

**inv:** solde > 0

**context** Compte : débiter(somme : Integer)

**pre:** somme > 0

**post:** solde = solde@ pre - somme

**context** Compte

**inv:** banque.clients -> includes (propriétaire)

[Image empruntée à Eric Cariou]

Commentaire ?

## Exemples extraits de UML Superstructure Specification

### Utilisation d'OCL pour spécifier les méta modèles :

- Contraintes sur *Interface* :

- Les opérations doivent être publiques

```
inv: self.operation->forall(f|f.visibility=#public)
```

- Une interface n'a pas d'attribut

```
inv: self.attributes->isEmpty()
```

- Contraintes sur *Classifier* : les hiérarchies d'héritage doivent être acycliques

```
inv: not self.allParents()->includes(self)
```

- Contrainte sur *Constraint* : une contrainte ne peut être appliquée à elle-même

```
inv: not self.constrainedElement->includes(self)
```

### Certaines contraintes ne peuvent être exprimées avec OCL :

- L'évaluation d'une contrainte ne doit pas avoir d'effet de bord
- L'évaluation d'une contrainte doit rendre une valeur booléenne
- ...

# Plan du cours

## 1 Introduction

## 2 Méta-modélisation, description et manipulation de modèles

- Méta-modélisation et MOF
- Contraintes et OCL
- XMI/JMI et la sérialisation/manipulation de modèles
- Synthèse

## 3 Model-Driven Architecture

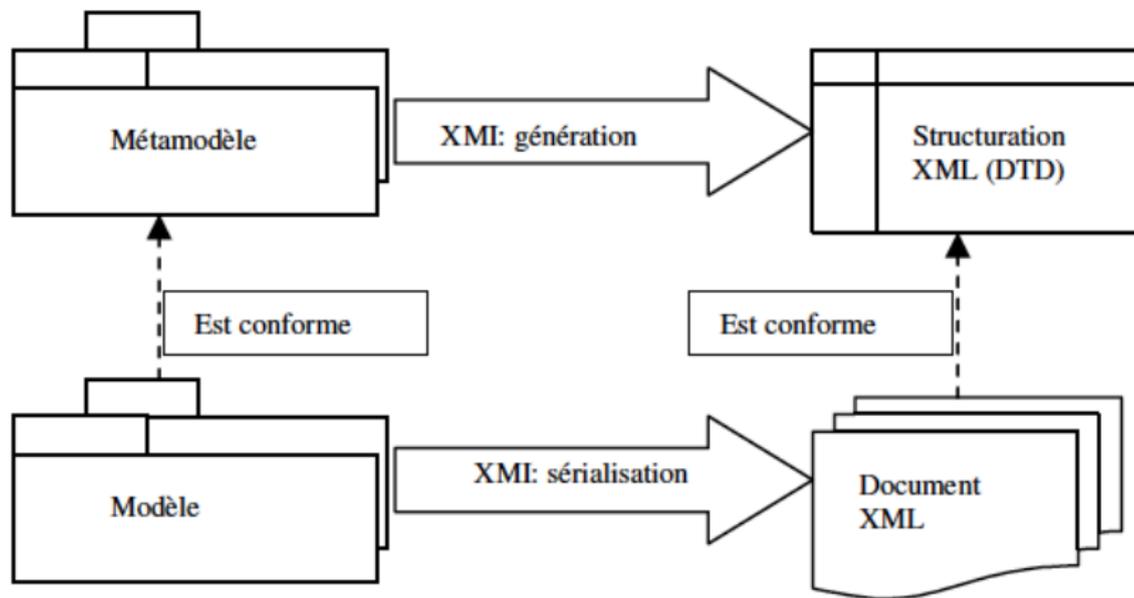
# Sérialisation de modèles et XMI

## Objectifs :

Stocker et échanger des modèles (syntaxe abstraite et non concrète !)

## Moyen : description de modèles avec XML

- XML (eXtensible Markup Language) du W3C :
  - ~> Format pour la représentation textuelle de données struct.
  - ~> Utilisation de balises `< xxx > ... < /xxx >`
- Grammaire définissant la validité d'un document XML :
  - DTD (Document Type Definition) :
    - ~> Relations d'inclusion entre balises
  - XML Schema :
    - ~> Doc XML définissant la structuration de documents XML
- XMI (XML Metadata Interchange) de l'OMG :
  - Standard pour représenter des (meta-)modèles au format XML
    - ~> Sérialisation de modèles
  - Utilisation des métamodèles pour définir les DTD / Schema
    - ~> Génération automatique des DTD / Schema



[Image empruntée à Xavier Blanc]

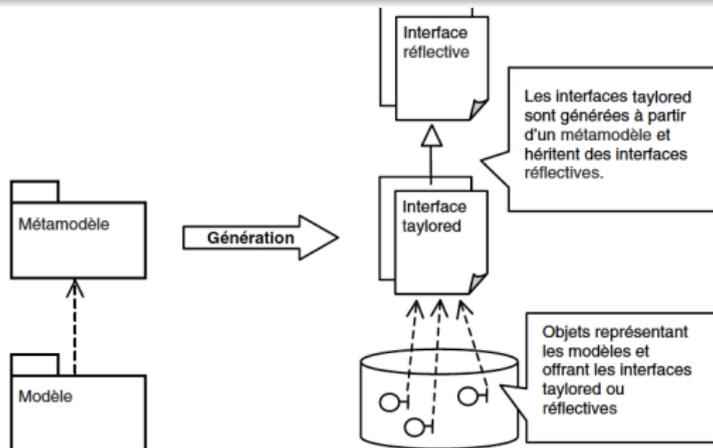
# Manipulation de modèles et JMI

## Objectif :

Manipuler (transformer, valider, ...) les modèles avec un langage orienté objet

## Moyen : API Java JMI (Java Metadata Interface)

- Interfaces Java dédiées à un méta-modèle (tayloré) : Manipulation d'éléments des modèles (conformes au métamodèle)
- Interfaces Java réflexives : Utilisables sur tout type de modèle



# Plan du cours

## 1 Introduction

## 2 Méta-modélisation, description et manipulation de modèles

- Méta-modélisation et MOF
- Contraintes et OCL
- XMI/JMI et la sérialisation/manipulation de modèles
- Synthèse

## 3 Model-Driven Architecture

# Synthèse des standards pour la description et la manipulation de (méta-)modèles

## Standards de l'OMG :

- UML (Unified Modeling Language) :  
~> Langage pour décrire les modèles
- MOF (Meta Object Facility) :  
~> Langage pour décrire les méta-modèles
- XMI (XML Metadata Interchange) :  
~> Règles pour représenter tout (méta-)modèle au format XML
- OCL (Object Constraint Language) :  
~> Expressions pour modéliser des propriétés (pré/post/inv)

## Standard du Java Community Process :

- JMI (Java Metadata Interface) :  
~> API pour manipuler des modèles en Java

# Plan du cours

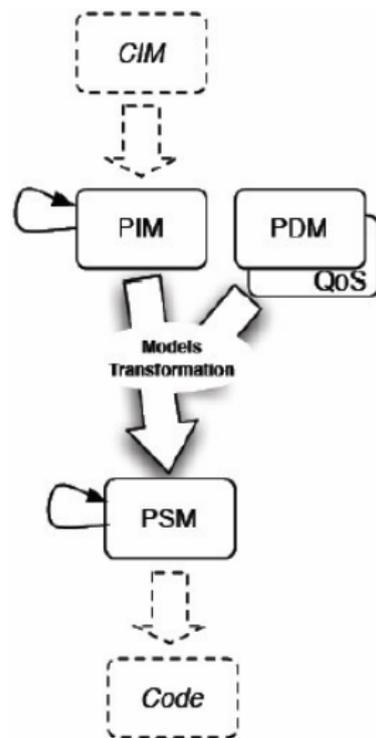
## 1 Introduction

## 2 Méta-modélisation, description et manipulation de modèles

## 3 Model-Driven Architecture

- Présentation générale
- MDA et les profils UML
- Eclipse Modeling Framework

# Le processus de développement en Y relooké par le MDA

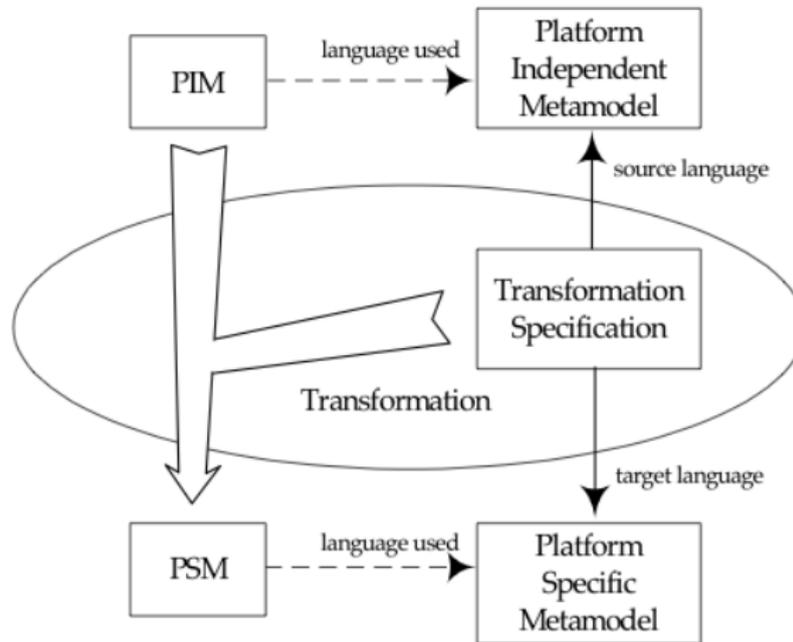


## Processus centré sur les modèles

- CIM (Computation Independent Model) : Modèle des exigences  $\leadsto$  Modèle pérenne du domaine
- PIM (Platform Independent Model) : Modèle métier  $\leadsto$  Modèle pérenne du système indépendant de plateformes techniques
- PDM (Platform Description Model) :  $\leadsto$  Modèle de plateforme technique (ex. : EJB, PHP, .NET, CORBA, ...)
- PSM (Platform Specific Model) : Modèle technique  $\leadsto$  Modèle du système dépendant d'une plateforme technique

# Transformation de modèles selon l'OMG

~ Point déterminant et critique de l'approche MDA !



## Exemples de mise-en-œuvre :

- Les profils UML
- EMF (Eclipse Modeling Framework)

# Plan du cours

## 1 Introduction

## 2 Méta-modélisation, description et manipulation de modèles

## 3 Model-Driven Architecture

- Présentation générale
- MDA et les profils UML
- Eclipse Modeling Framework

# Profils UML

## Qu'est-ce qu'un profil UML ?

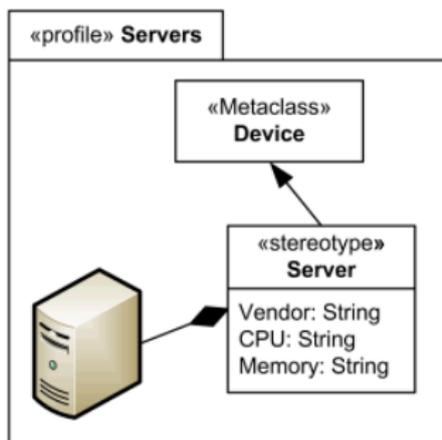
Moyen simple pour définir de nouveaux méta-modèles par extension de méta-modèles ou de profils existants

## Pourquoi des profils UML ?

- Définir des méta-modèles spécialisés pour :
  - Un domaine d'application particulier (finance, santé, tps réel, ...)  
    ~> Définition de PIM
  - Une plateforme technique particulière (.NET, J2EE, CORBA, ...)  
    ~> Définition de PSM
- ~> Domain Specific Modeling Languages (DSL)
- Permettre la transformation de modèles et la génération de code  
    ~> Interprétation sémantique par rapport à l'application/plateforme

# Comment définir un profil UML ? (1/2)

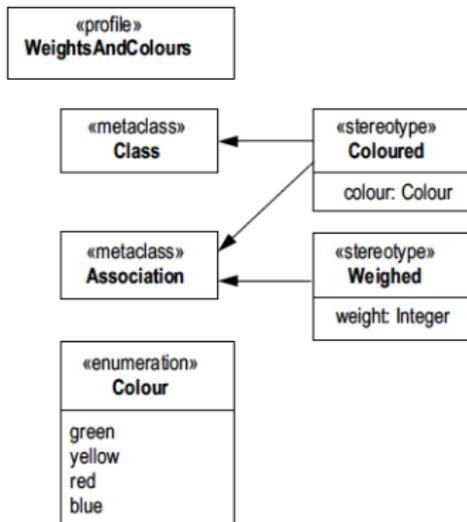
- Un profil est un package portant le stéréotype «profile»
- Un profil contient la définition de nouveaux stéréotypes :
  - Définition par extension d'une méta-classe (→)
  - ou par spécialisation d'un autre stéréotype (→▷)
  - Possibilité de spécifier une représentation graphique (→◆)
  - Possibilité de spécifier des attributs (tagged values)



## Comment définir un profil UML ? (2/2)

- Un stéréotype peut étendre plusieurs méta-classes
- Un profil peut contenir des contraintes (OCL ou informelles)
  - ↪ Conditions d'emploi des stéréotypes et tagged values

### Exemple :



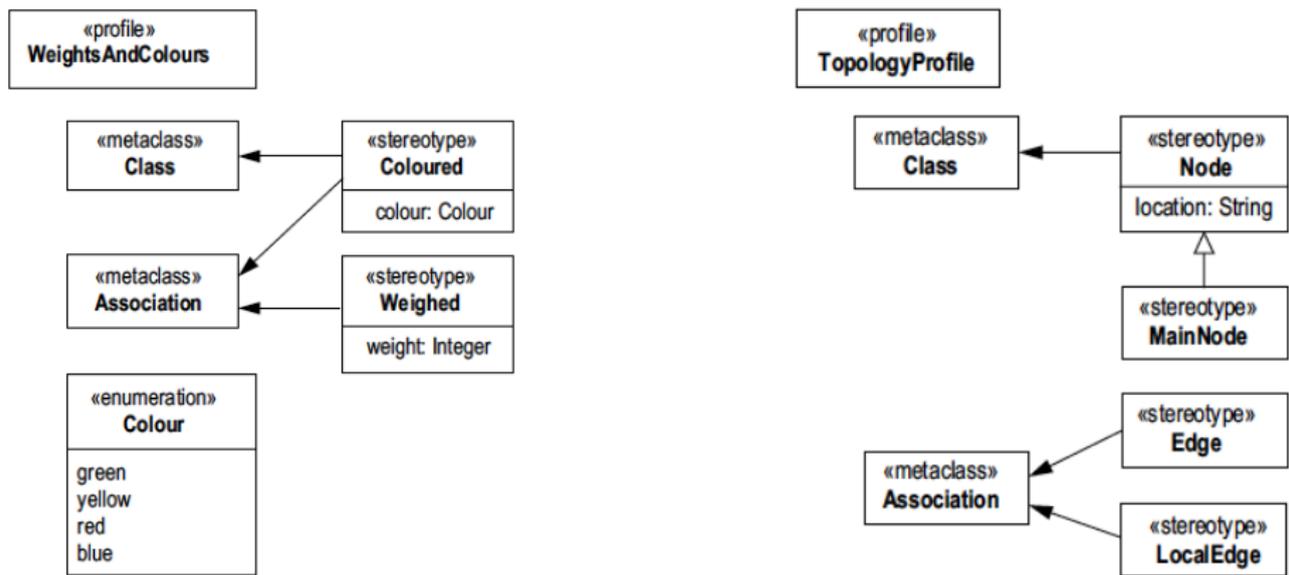
- Contrainte en français :  
*Une association Coloured ne peut lier que des classes Coloured de même couleur que l'association*

- Contrainte en OCL :

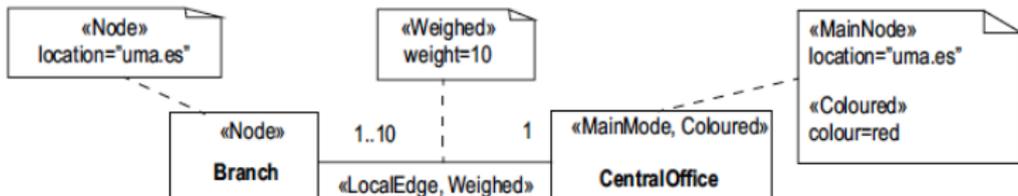
```

context UML::InfrastructureLibrary::Core::
  Constructs::Association
  inv : self.isStereotyped("Coloured") implies
    self.connection->forAll
      (isStereotyped("Coloured")
        implies color=self.color)
  
```

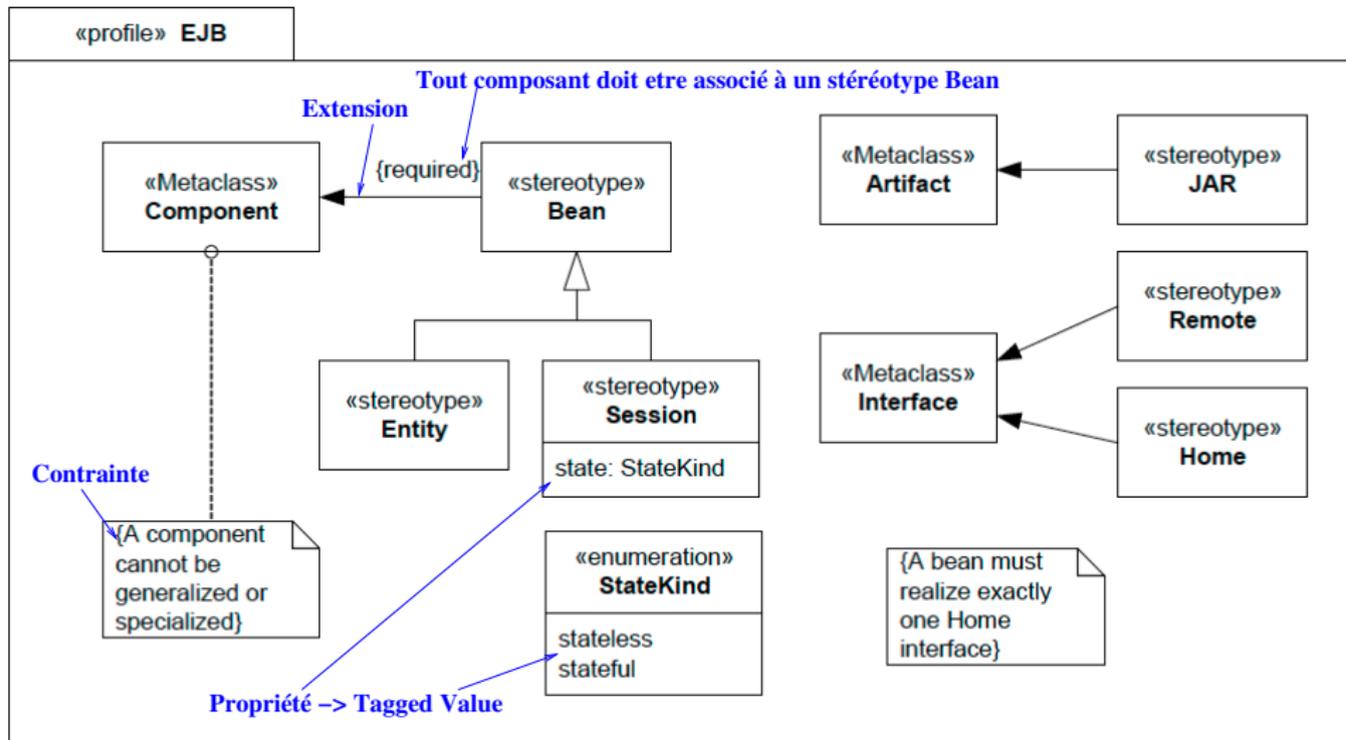
## Ex. de définition de profils [L. Fuentes-Fernández, A. Vallecillo-Moreno 2004]



## Ex. d'utilisation de profils [L. Fuentes-Fernández, A. Vallecillo-Moreno 2004]



# Exemple "réel" : Profil EJB (Enterprise JavaBean)



[Image empruntée à [www.omg.org](http://www.omg.org)]

# Plan du cours

## 1 Introduction

## 2 Méta-modélisation, description et manipulation de modèles

## 3 Model-Driven Architecture

- Présentation générale
- MDA et les profils UML
- Eclipse Modeling Framework

# Eclipse ?

## Une plateforme de développement “open source” :

- Cadre extensible (plug-in) pour construire et maintenir des logiciels
- Environnement de développement (IDE) construit à l'aide de ce cadre

## Gérée par la “Eclipse Foundation” :

- Organisation indépendante à but non lucratif créée en 2004
- Regroupant plus de 100 entreprises (Ericsson, HP, IBM, Intel, SAP, ...) et de nombreux développeurs indépendants

## Structurée en projets :

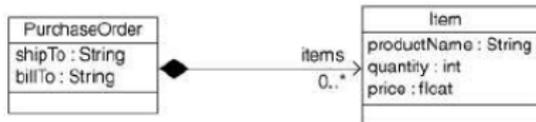
- JDT : Java development tools
- PDE : Plugin Development Environment
- GMF : Graphical Modeling Framework
- ...
- EMF : Eclipse Modeling Framework

*The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model.*

# UML, Java, et XML...

3 syntaxes concrètes différentes pour une même syntaxe abstraite :

- UML :



- Java :

```

public interface PurchaseOrder {
    String getShipTo();
    void setShipTo(String value);
    String getBillTo();
    void setBillTo(String value);
    List getItem(); // List of Item
}

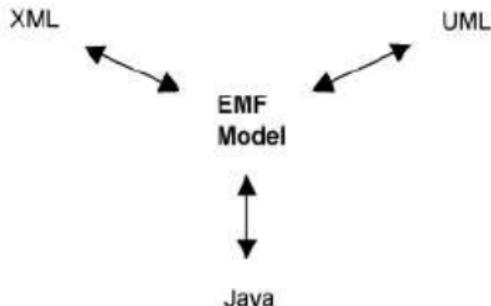
public interface Item {
    String getProductName();
    void setProductName(String value);
    int getQuantity();
    void setQuantity(int value);
    float getPrice();
    void setPrice(float value);
}
  
```

- XML :

```

<xsd:complexType name="PurchaseOrder">
  <xsd:sequence>
    <xsd:element name="shipTo" type="xsd:string"/>
    <xsd:element name="billTo" type="xsd:string"/>
    <xsd:element name="items" type="PO:Item"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Item">
  <xsd:sequence>
    <xsd:element name="productName" type="xsd:string"/>
    <xsd:element name="quantity" type="xsd:int"/>
    <xsd:element name="price" type="xsd:float"/>
  </xsd:sequence>
</xsd:complexType>
  
```

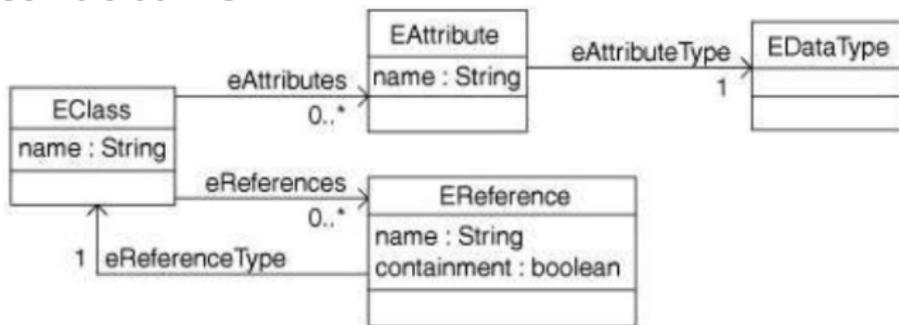


# Le Méta-modèle Ecore

## Méta-modèle des modèles EMF

Choix délibéré d'un langage de modélisation très simple

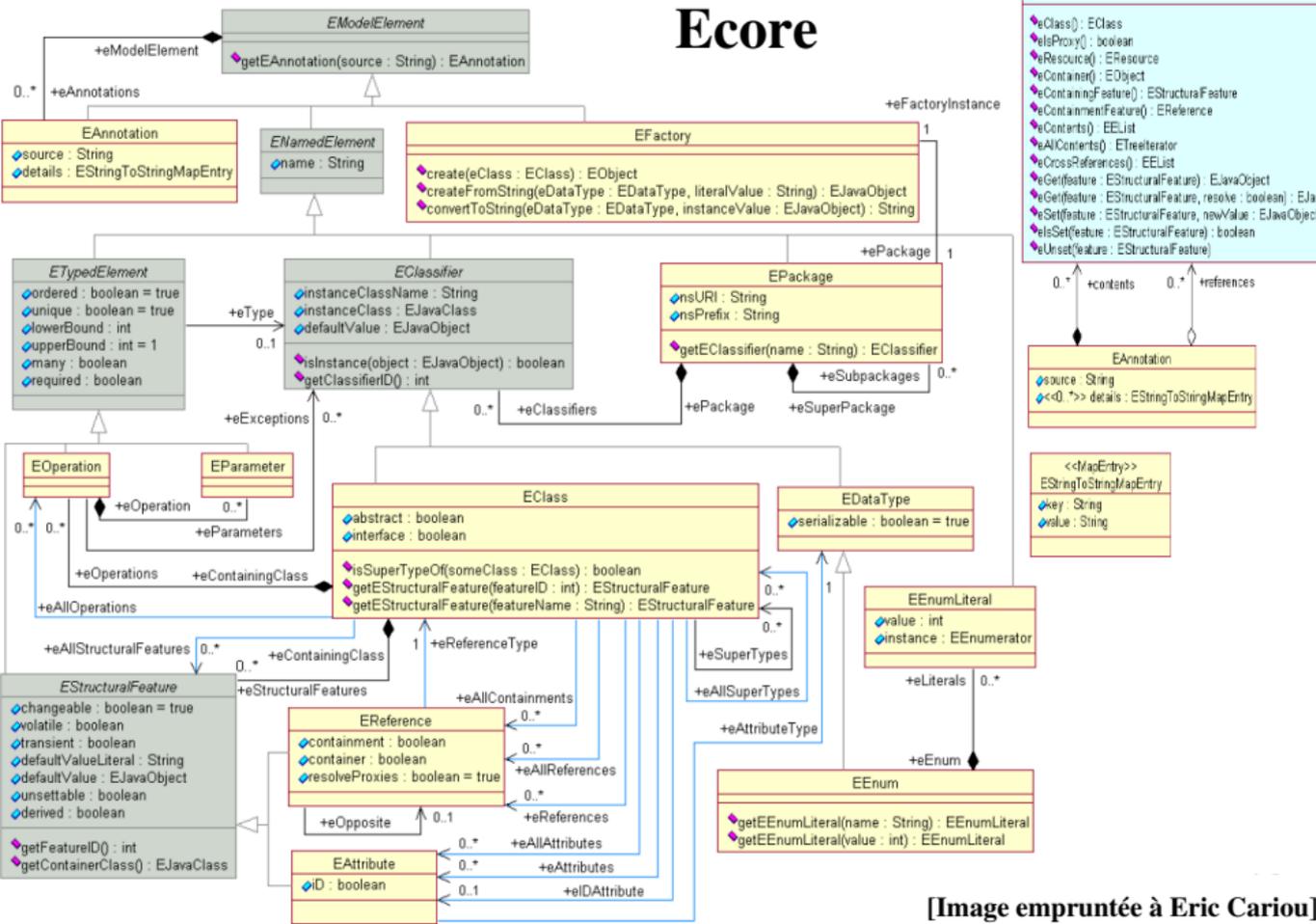
→ Sous-ensemble du MOF



## Correspondances avec UML, Java et XML

Ecore	UML	Java	XML
EClass	Classe	Interface	Type complexe
EAttribute	Attribut	Getter/Setter	Éléments imbriqués
EReference	Extrémité d'assoc.	Getter/Setter	Type complexe imbriqué

# Ecore



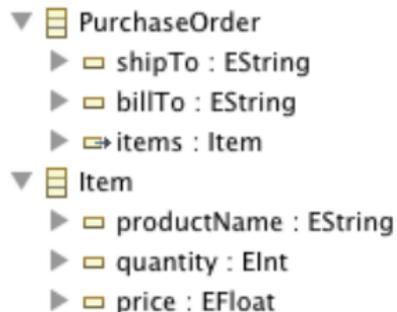
[Image empruntée à Eric Cariou]

# Construction d'un modèle Ecore

- Avec l'éditeur Ecore fourni dans Eclipse (construit en utilisant EMF !)
- Ou à partir d'interfaces Java annotées
- Ou à partir d'un diagramme de classes UML
- Ou à partir d'un XML Schema

## Exemple :

### • Modèle Ecore :



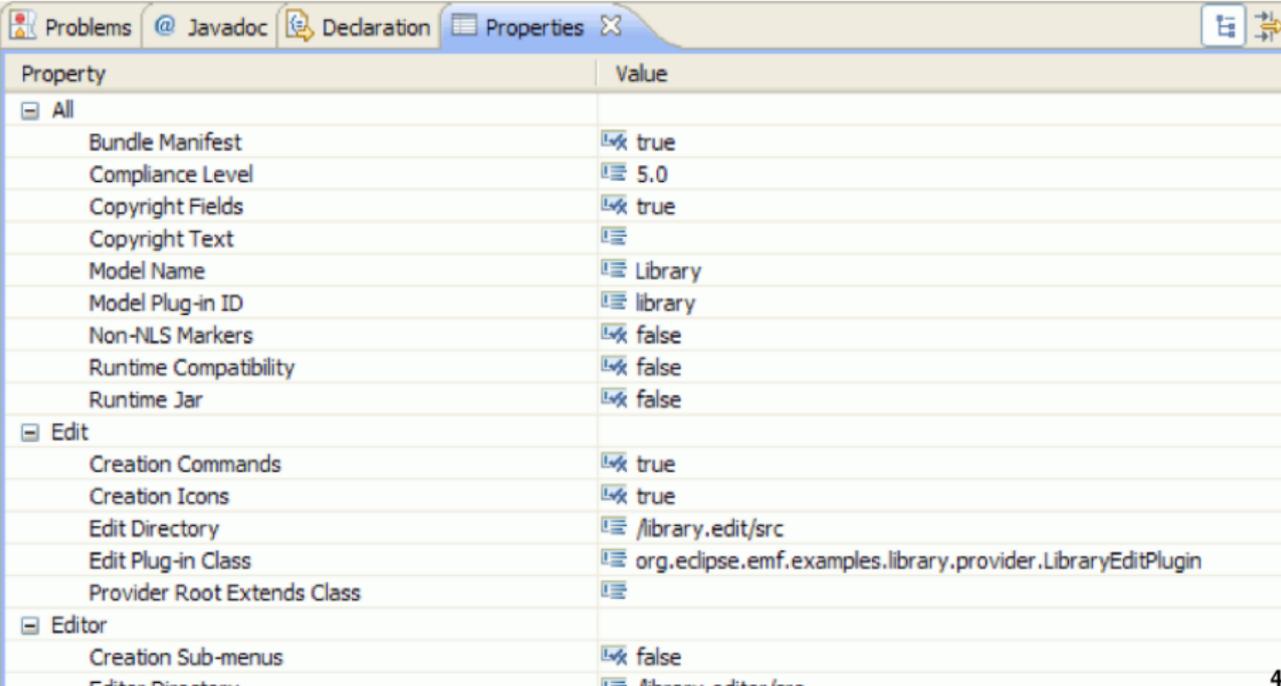
### • Modèle UML correspondant :



# Génération de code

## Modèle de génération (genModel)

- Décoration du modèle Ecore par des propriétés / génération de code  
    ~> Utilisation du design pattern Decorator

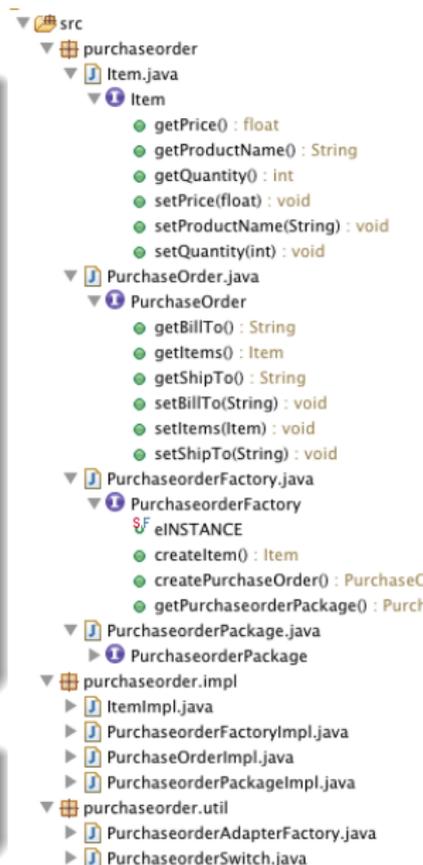


Property	Value
All	
Bundle Manifest	<input checked="" type="checkbox"/> true
Compliance Level	5.0
Copyright Fields	<input checked="" type="checkbox"/> true
Copyright Text	
Model Name	Library
Model Plug-in ID	library
Non-NLS Markers	<input checked="" type="checkbox"/> false
Runtime Compatibility	<input checked="" type="checkbox"/> false
Runtime Jar	<input checked="" type="checkbox"/> false
Edit	
Creation Commands	<input checked="" type="checkbox"/> true
Creation Icons	<input checked="" type="checkbox"/> true
Edit Directory	/library.edit/src
Edit Plug-in Class	org.eclipse.emf.examples.library.provider.LibraryEditPlugin
Provider Root Extends Class	
Editor	
Creation Sub-menus	<input checked="" type="checkbox"/> false
Edit Directory	/library.edit/src

# Génération de code (1/2)

## Code généré pour le modèle :

- Pour chaque EClass X :
  - Interface X
    - ↳ étend EObject, qui étend Notifier
  - Classe d'implémentation XImpl
- Classe Singleton Factory
  - ↳ createX pour chaque classe X
- Classe Singleton AdapterFactory : Fabrique d'adaptateurs pour observer les classes du modèle
- Interface Ressource : Gestion de la persistance
- ...



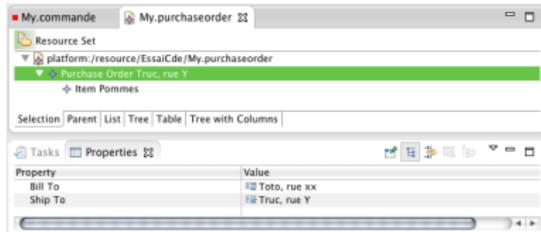
## Conservation des modifications de code :

Suppression de l'annotation @generated

# Génération de code (2/2)

## Code généré pour l'édition et la manipulation :

- Package .Edit : Classes Java pour construire des éditeurs/vues du modèle
  - Utilise JFace pour l'affichage de modèles structurés
  - Design pattern Command  $\rightsquigarrow$  execute/undo/redo
  - Design Pattern Observer  $\rightsquigarrow$  les vues observent le modèle
- Package .Editor : plug-in d'un éditeur complet
  - $\rightsquigarrow$  Visualisation/Modification des instances du modèle



## Code généré pour les tests :

Génération de classes de tests JUnit