

# ***Ingénierie des Modèles***

## ***Méta-modélisation***

Eric Cariou

*Université de Pau et des Pays de l'Adour*  
*Département Informatique*

Eric.Cariou@univ-pau.fr

# *Introduction/Plan*

- ◆ But de la méta-modélisation
  - ◆ Définir des langages de modélisation ou des langages de manière générale
- ◆ Architecture MOF
  - ◆ 4 niveaux de (méta)modélisation
  - ◆ Architecture 4 niveaux généralisable en dehors du MOF
- ◆ Syntaxes abstraite et concrète
- ◆ Profils UML
  - ◆ Spécialisation et définition de méta-modèles

# *Normes OMG de modélisation*

- ◆ MOF : Meta-Object Facilities
  - ◆ Langage de définition de méta-modèles
- ◆ UML : Unified Modelling Language
  - ◆ Langage de modélisation
- ◆ CWM : Common Warehouse Metamodel
  - ◆ Modélisation ressources, données, gestion d'une entreprise
- ◆ OCL : Object Constraint Language
  - ◆ Langage de contraintes sur modèles
- ◆ XMI : XML Metadata Interchange
  - ◆ Standard pour échanges de modèles et méta-modèles

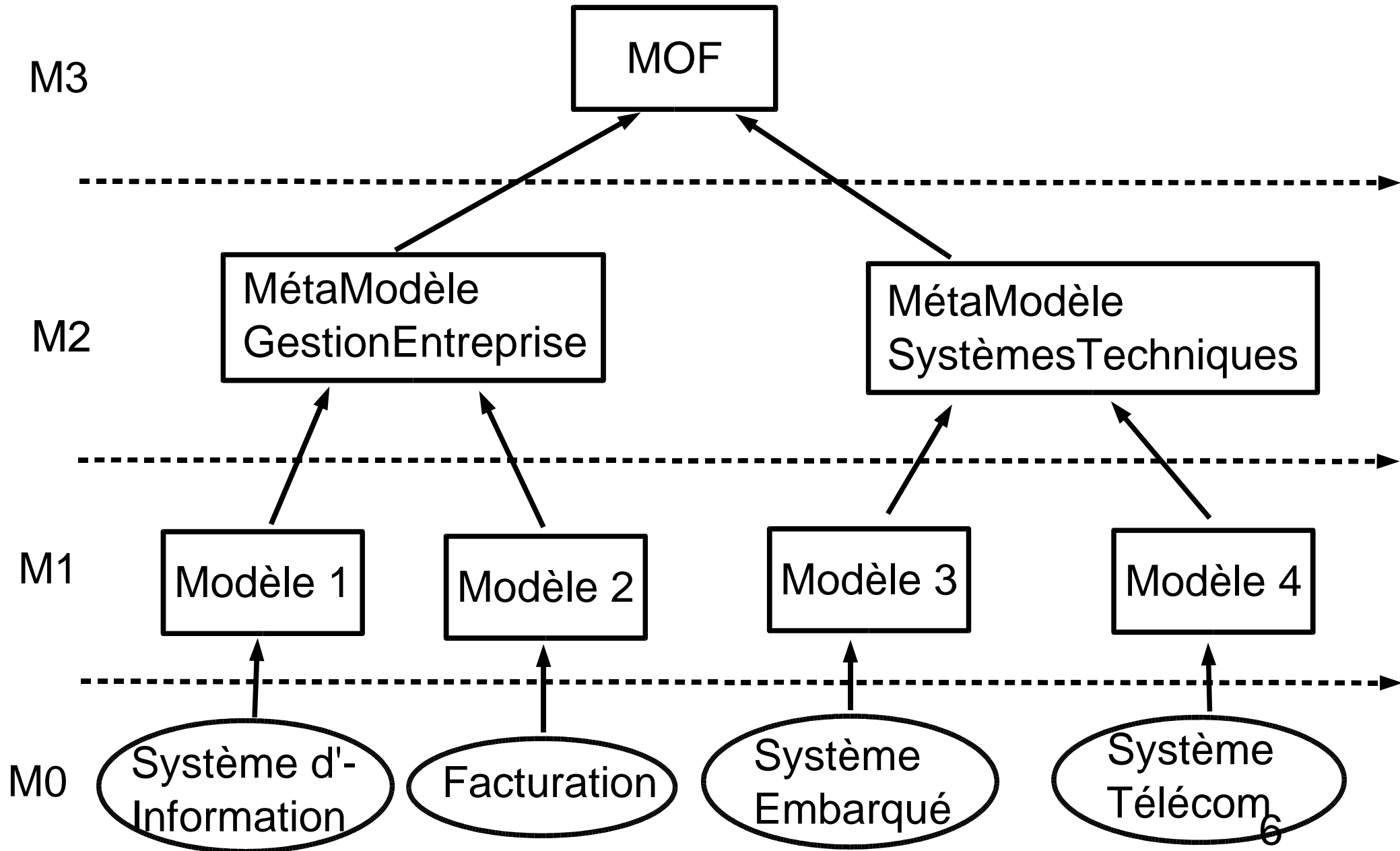
# *Normes OMG de modélisation*

- ◆ Plusieurs de ces normes concernent la définition et l'utilisation de méta-modèles
  - ◆ MOF : but de la norme
  - ◆ UML et CWM : peuvent être utilisés pour en définir
  - ◆ XMI : pour échange de (méta-)modèles entre outils
- ◆ MOF
  - ◆ C'est un méta-méta-modèle
    - ◆ Utiliser pour modéliser des méta-modèles
  - ◆ Définit les concepts de base (22)
    - ◆ Entité/classe, relation/association, type de données, référence, package ...
  - ◆ Le MOF peut définir le MOF

## *4 Niveaux du MOF*

- ◆ Le MOF définit 4 niveaux de modélisation
  - ◆ M0 : système réel, système modélisé
  - ◆ M1 : modèle du système réel défini dans un certain langage
  - ◆ M2 : méta-modèle définissant ce langage
  - ◆ M3 : méta-méta-modèle définissant le méta-modèle
    - ◆ Le niveau M3 est le MOF
    - ◆ Dernier niveau, il est méta-circulaire : il peut se définir lui même
- ◆ Le MOF est – pour l'OMG – le méta-méta-modèle unique servant de base à la définition de tous les méta-modèles

# 4 niveaux du MOF



# *Hiérarchie 4 Niveaux*

- ◆ On retrouve cette hiérarchie à 4 niveaux en dehors du MOF et d'UML, dans d'autres espaces technologiques que celui de l'OMG
- ◆ Langage de programmation
  - ◆ M0 : l'exécution d'un programme
  - ◆ M1 : le programme
  - ◆ M2 : la grammaire du langage dans lequel est écrit le programme
  - ◆ M3 : le concept de grammaire EBNF
- ◆ XML
  - ◆ M0 : données du système
  - ◆ M1 : données modélisées en XML
  - ◆ M2 : DTD XML
  - ◆ M3 : le langage XML

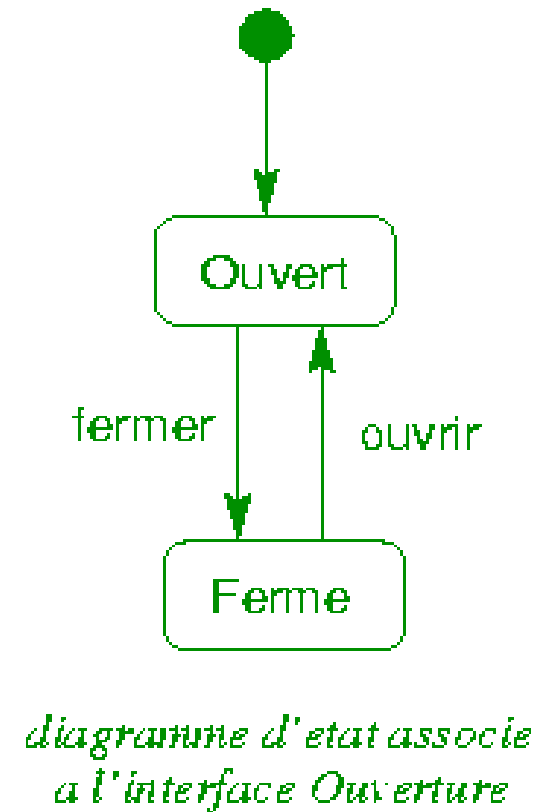
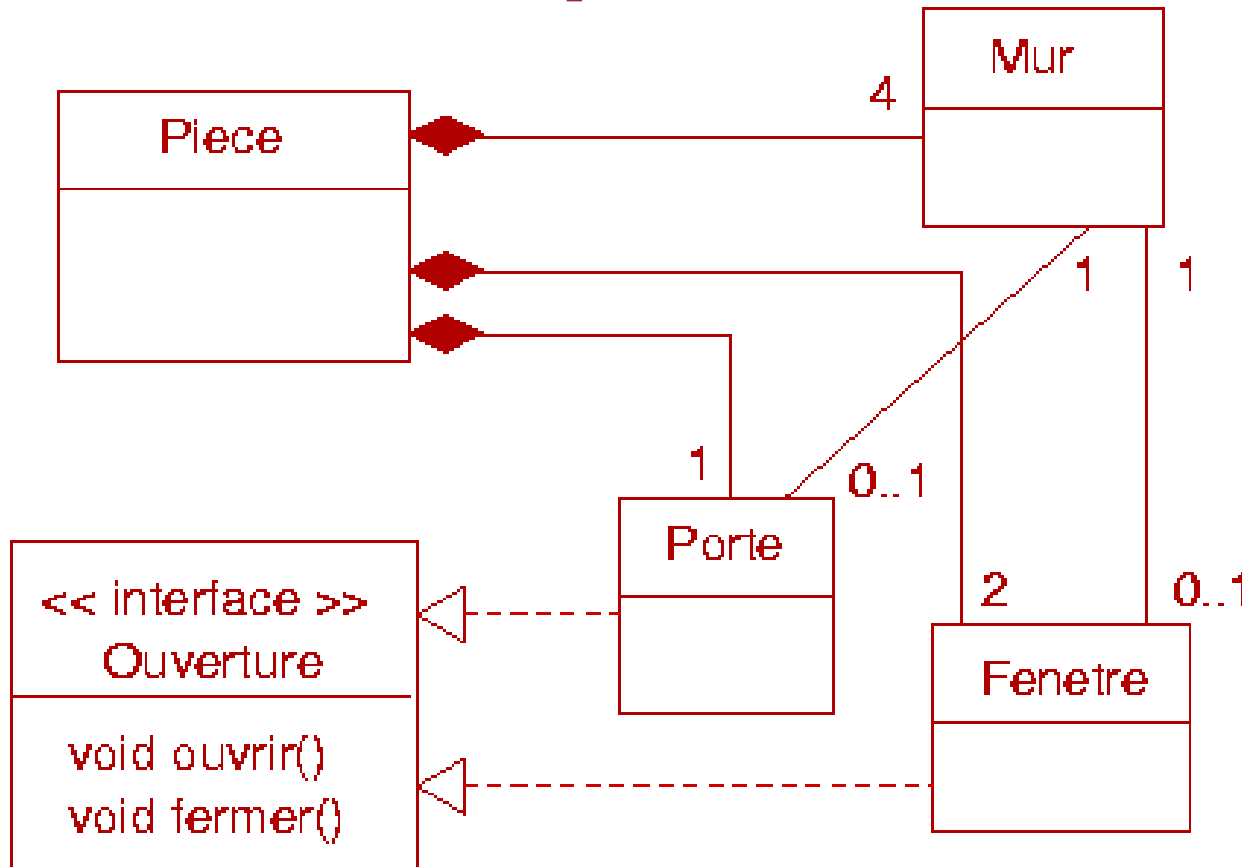
# *Méta-modélisation UML*

- ◆ Dans UML, on retrouve également les 4 niveaux
  - ◆ Mais avec le niveau M3 définissable en UML directement à la place du MOF
- ◆ Exemple de système 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

# ***Méta-modélisation UML***

- ◆ 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 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

# M1 : spécification du Système

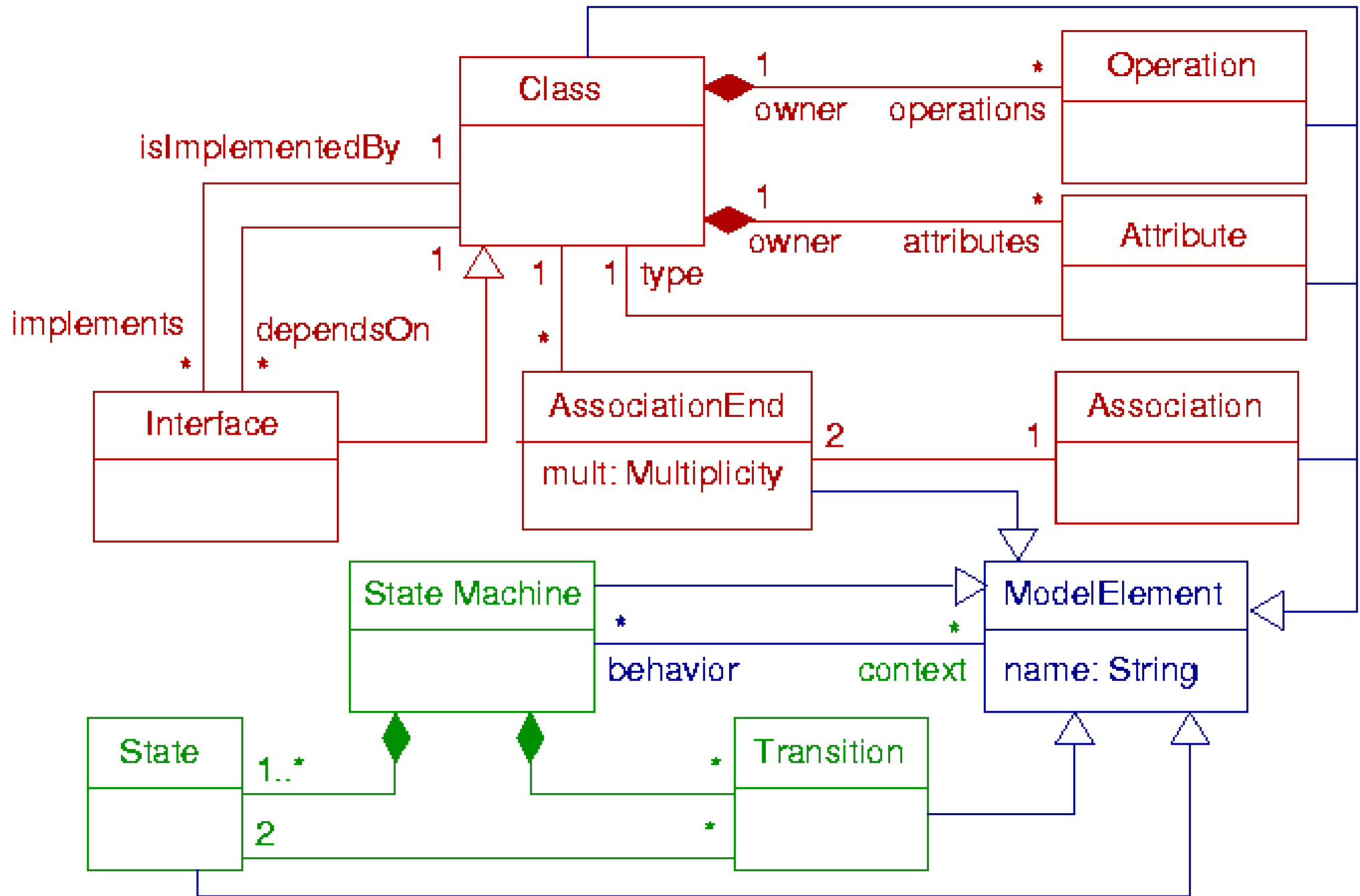


- ◆ **context** Mur inv: fenetre -> union(porte) -> size() <= 1  
-- un mur a soit une fenêtrre soit une porte (soit rien)
- ◆ **context** Piece inv:  
mur.fenetre -> size() = 2 -- 2 murs de la pièce ont une fenêtrre  
mur.porte -> size() = 1 -- 1 mur de la pièce a une porte 10

# ***Méta-modélisation UML***

- ◆ Les 2 diagrammes de ce modèle de niveau M1 sont des diagrammes UML valides
- ◆ Les contraintes sur les éléments des diagrammes UML et leurs relations sont définies
  - ◆ Dans le méta-modèle UML : niveau M2
  - ◆ Un diagramme UML (classe, état ...) doit être conforme au méta-modèle UML
- ◆ Méta-modèle UML
  - ◆ Diagramme de classe spécifiant la structure de tous types de diagrammes UML
  - ◆ Avec contraintes OCL pour spécification précise

# M2 : Méta-modèle UML (simplifié)



# *M2 : Méta-modèle UML (simplifié)*

- ◆ 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*

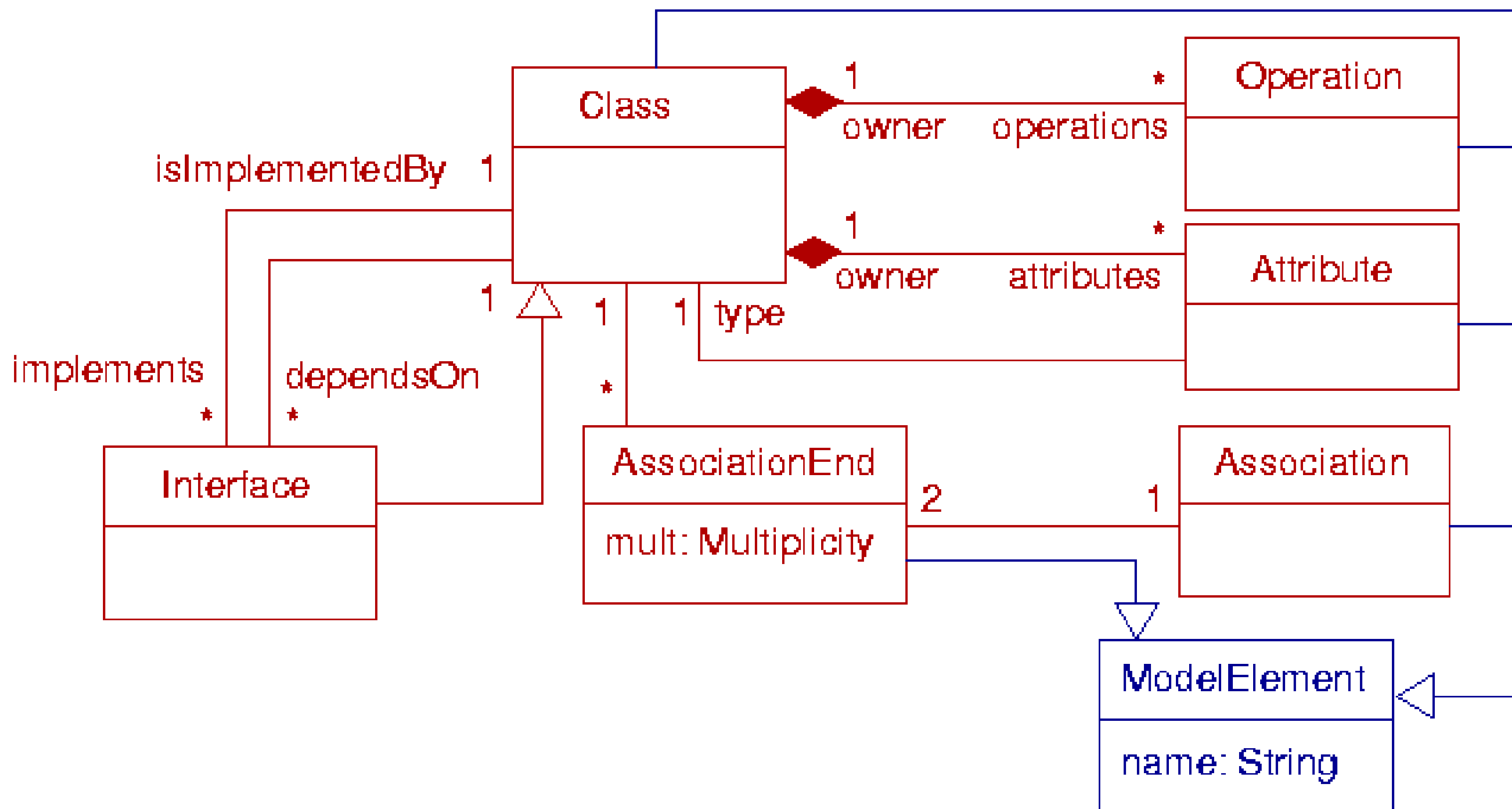
- ◆ **context** StateMachine **inv**: transition -> forAll ( t | self.state -> includesAll(t.state))

*Une transition d'un diagramme d'état connecte 2 états de ce diagramme d'état*

# ***Méta-modélisation UML***

- ◆ Le méta-modèle UML doit aussi être précisément défini
  - ◆ Il doit être conforme à un méta-modèle
  - ◆ C'est le méta-méta-modèle UML
- ◆ Qu'est ce que le méta-modèle UML ?
  - ◆ Un diagramme de classe UML (avec contraintes OCL)
- ◆ Comment spécifier les contraintes d'un diagramme de classe ?
  - ◆ Via le méta-modèle UML
  - ◆ Ou plus précisément : via la partie du méta-modèle UML spécifiant les diagrammes de classes
- ◆ Méta-méta-modèle UML = copie partielle du méta-modèle UML : niveau M3

# M3 : Méta-méta-modèle UML (simplifié)

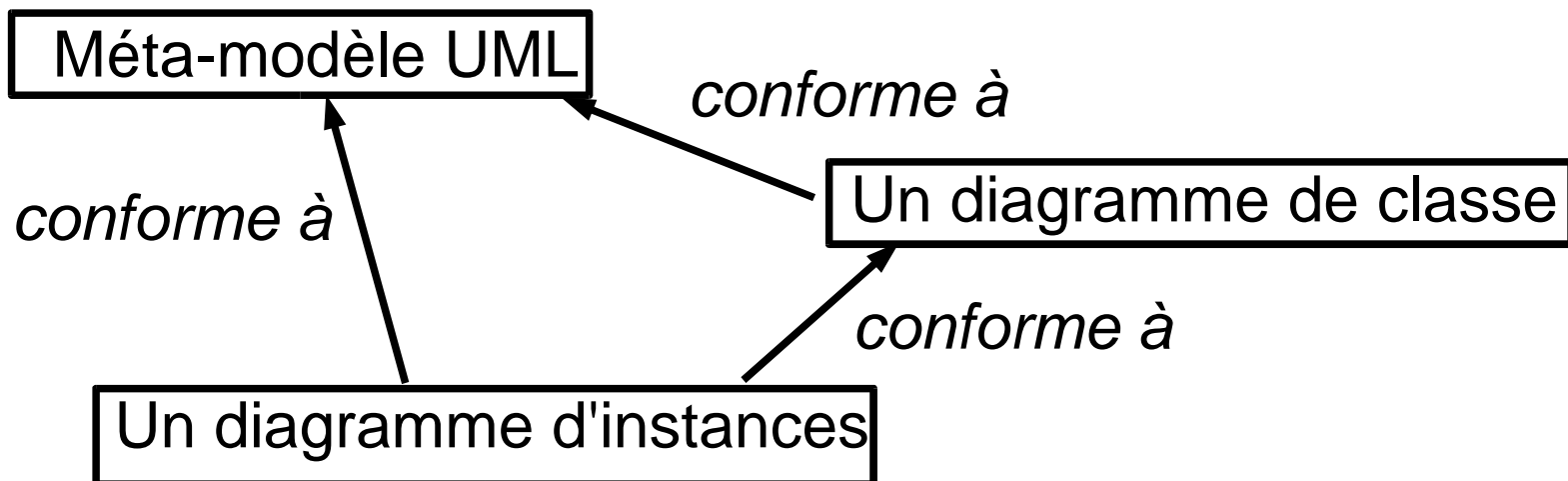


# *Meta-modélisation UML*

- ◆ Méta-méta-modèle UML doit aussi être clairement défini
  - ◆ Il doit être conforme à un méta-modèle
  - ◆ Qu'est ce que le méta-méta-modèle UML ?
    - ◆ Un diagramme de classe UML
  - ◆ Comment spécifier les contraintes d'un diagramme de classe ?
    - ◆ Via la partie du méta-modèle UML spécifiant les diagrammes de classe
      - ◆ Via le méta-méta-modèle UML
- ◆ Le méta-méta-modèle UML peut donc se définir lui même
  - ◆ Méta-circulaire
  - ◆ Pas besoin de niveau méta supplémentaire

# Diagrammes d'instances UML

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



# Syntaxe

- ◆ Un langage est défini par un méta-modèle
- ◆ Un langage possède une syntaxe respectant le méta-modèle
  - ◆ Syntaxe textuelle
  - ◆ Ensemble de mots-clé et de mots respectant des contraintes défini selon des règles précises
    - ◆ Notions de syntaxe et de grammaire dans les langages
    - ◆ Exemple pour langage Java  
*public class MaClasse implements MonInterface { ... }*
    - ◆ Grammaire Java pour déclaration de classe  
*class\_declaration ::= { modifier } "class" identifier [ "extends" class\_name ] [ "implements" interface\_name { "," interface\_name } ] "{" { field\_declaration } "}"*

# Syntaxe

- ◆ Syntaxe graphique

- ◆ 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



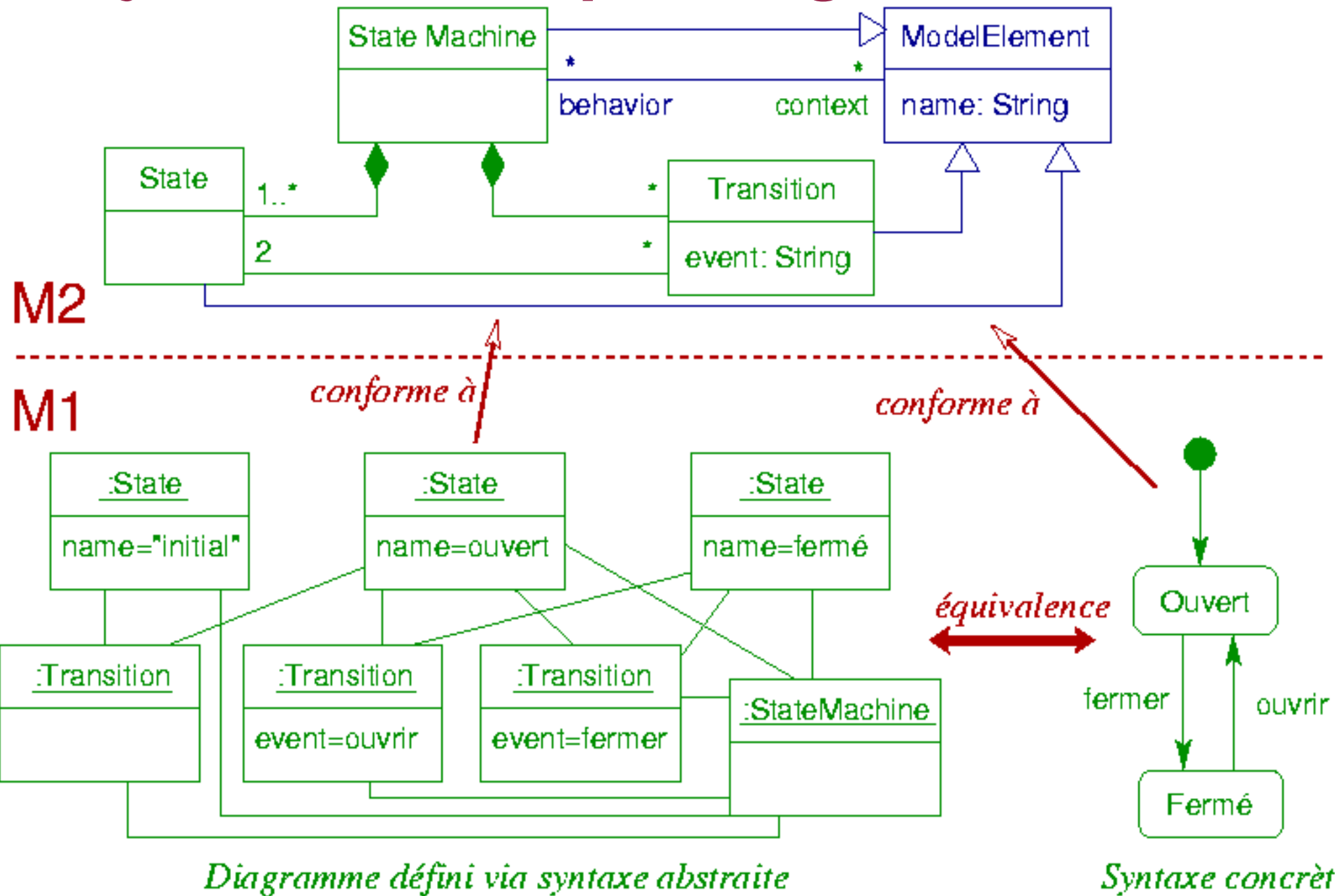
# Syntaxe

- ◆ Syntaxe abstraite/concrète
  - ◆ Abstraite
    - ◆ Les éléments et leurs relations sans une notation spécialisée
    - ◆ Correspond à ce qui est défini au niveau du méta-modèle
  - ◆ 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 méta-modèles
  - ◆ Pas de syntaxe concrète définie

# Syntaxe

- ◆ Exemple de la modélisation de la pièce
  - ◆ Syntaxe concrète
    - ◆ 2 diagrammes UML (classe et états) avec syntaxe spécifique à ces types de diagrammes
  - ◆ Via la syntaxe abstraite
    - ◆ Diagramme d'instances (conforme au méta-modèle) précisant les instances particulières de classes, d'associations, d'états...
- ◆ Pour la partie diagramme d'états
  - ◆ Diagramme défini via syntaxe concrète : diagramme d'états de l'exemple
  - ◆ Diagramme défini via syntaxe abstraite : diagramme d'instance conforme au méta-modèle UML

# Syntaxe : exemple diagramme état



# *Spécification de méta-modèles*

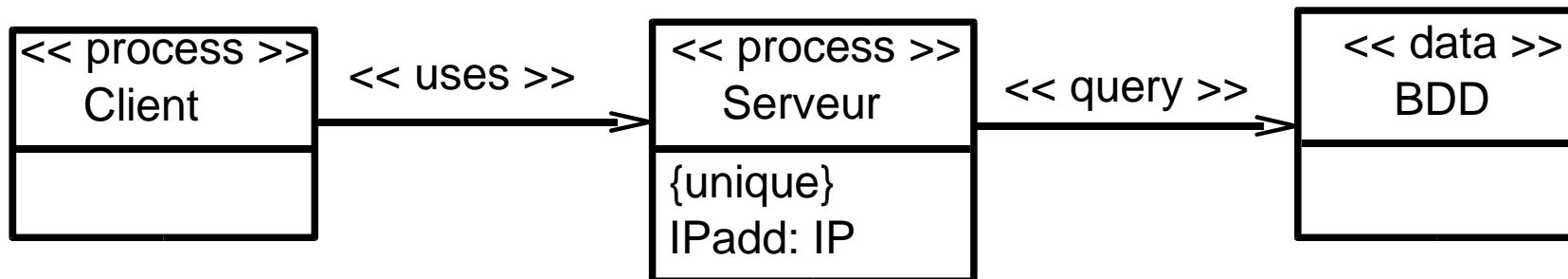
- ◆ But : définir un type de modèle avec tous ces types d'éléments et leurs contraintes
- ◆ Trois approches possibles
  - ◆ Définir un méta-modèle nouveau à partir de rien
  - ◆ Modifier un méta-modèle existant : ajout, suppression, modification d'éléments et des contraintes sur leurs relations
    - ◆ Correspond au MOF, décomposé en 2 parties
      - ◆ E-MOF : essential MOF, les méta-éléments de base réutilisés tel quel dans tous les méta-modèles
      - ◆ MOF : un méta-modèle particulier défini via E-MOF
  - ◆ Spécialiser un méta-modèle existant en rajoutant des éléments et des contraintes (sans en enlever)
    - ◆ Correspond aux profils UML

# *Profils UML*

- ◆ Un profil est une spécialisation du méta-modèle UML
  - ◆ Ajouts de nouveaux types d'éléments
    - ◆ Et des contraintes sur leurs relations entre eux et avec les éléments d'UML
  - ◆ Ajouts de contraintes sur éléments existants d'UML
  - ◆ Ajouts de contraintes sur relations existantes entre les éléments d'UML
  - ◆ Aucune suppression de contraintes ou d'éléments
- ◆ Profil : mécanisme d'extension d'UML pour l'adapter à un contexte métier ou technique particulier
  - ◆ Profil pour composants EJB
  - ◆ Profil pour gestion bancaire
  - ◆ Profil pour architecture logicielle

# Profils UML

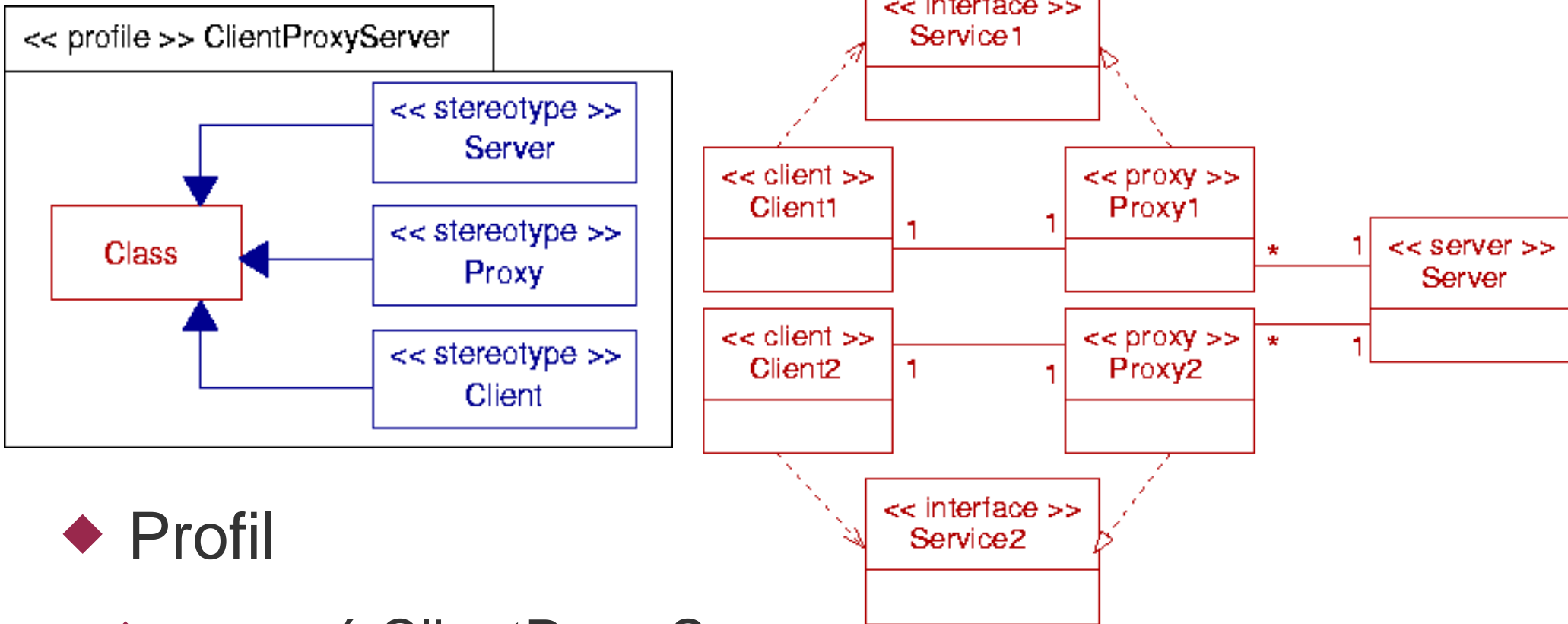
- ◆ Stéréotype : extension, spécialisation d'un élément du méta-modèle
  - ◆ Classe, association, attribut, opération ...
  - ◆ Le nom d'un stéréotype est marqué entre << ... >>
  - ◆ Il existe déjà des stéréotypes dans UML
    - ◆ << interface >> : une interface est une classe particulière (sans attribut)
- ◆ On peut marquer des attributs d'une classe pour préciser une contrainte ou un rôle particulier : tagged value
  - ◆ Exemple {unique} id: int



# *Profils UML*

- ◆ Profil UML est composé de 3 types d'éléments
  - ◆ Des stéréotypes
  - ◆ Des tagged value
  - ◆ Des contraintes (exprimables en OCL)
    - ◆ Sur ces stéréotypes, tagged value
    - ◆ Sur des éléments du méta-modèle existant
    - ◆ Sur les relations entre les éléments
- ◆ Un profil UML est défini sous la forme d'un package stéréotypé << profile >>
- ◆ Exemple de profil : architecture logicielle
  - ◆ Des composants clients et serveur
  - ◆ Un client est associé à un serveur via une interface de service par l'intermédiaire d'un proxy

# Exemple Profil



## ◆ Profil

- ◆ nommé `ClientProxyServer`

- ◆ Définit trois stéréotypes

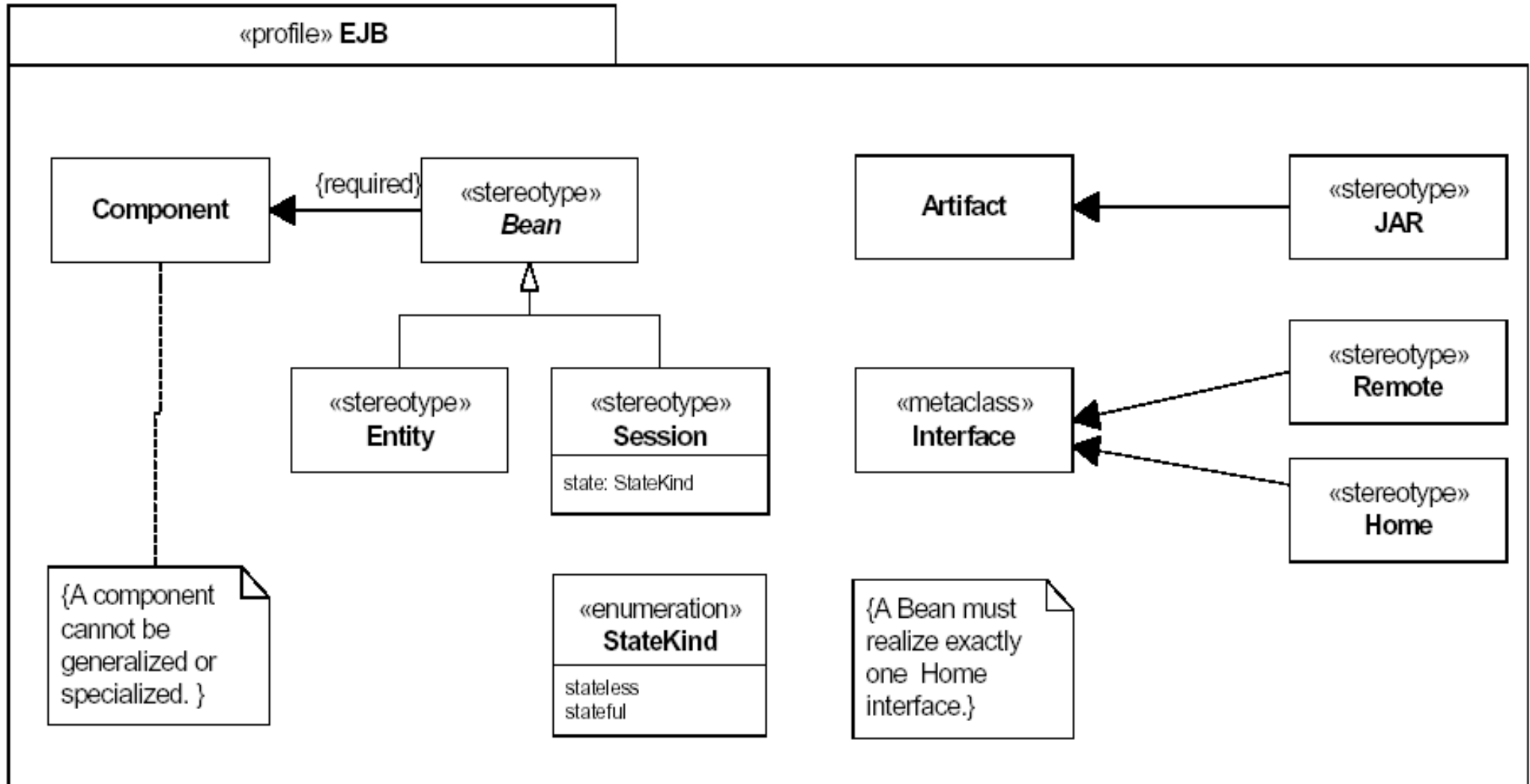
- ◆ Trois classes jouant un rôle particulier : extensions de la méta-class `Class` du méta-modèle UML

- ◆ `Server`, `Proxy`, `Client`

# Exemple Profil

- ◆ Pour compléter le profil, ajout de contraintes OCL
  - ◆ Navigation sur le méta-modèle (simplifié) en considérant que la méta-class Class a trois spécialisations (Server, Client, Proxy)
  - ◆ **context Client inv:**  
**let** proxies = self.associationEnd.association.associationEnd.-  
class -> select ( c | c.ocllsTypeOf(Proxy)) **in**  
**let** interfaces = self.dependsOn **in**  
interfaces -> forAll ( i | proxies.implements -> includes (i) **and**  
proxies -> forAll ( p | p.implements -> includes (i)  
**implies** p.hasClassRefWith(self)))
  - ◆ **context Class def:** hasClassRefWith(cl : Class) : Boolean=  
self.associationEnd.association.associationEnd.class  
-> exists ( c | c = cl )
  - ◆ Un proxy associé à un client doit implémenter une des interfaces dont dépend le client et un proxy implémentant une interface d'un client doit avoir une association avec ce client

# Autre exemple : profil EJB



Source : norme UML 2.0