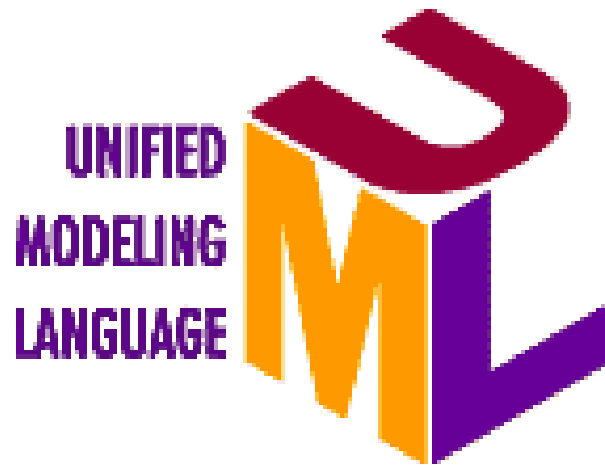


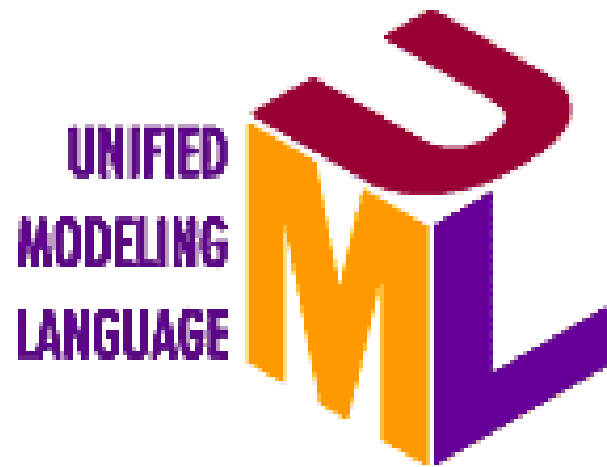
Notation UML



Sommaire

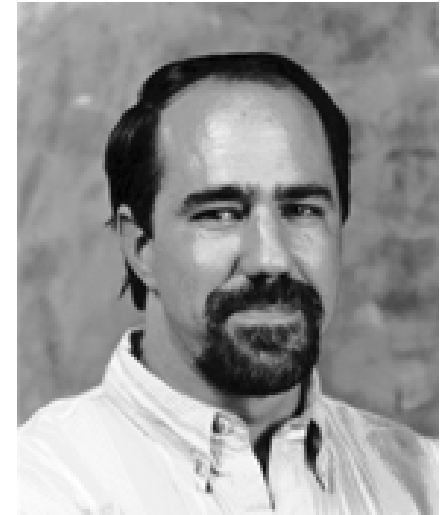
- Historique
- La Modélisation
- Axe Statique
- Axe Dynamique
- Références

Historique



BOOCH

- **Pionnier** de l'Objet-Orienté
 - Article en 1981: ' Object Oriented Development '
 - Au début, **méthode** pour le développement d'applications en **Ada** pour le ' Department of Défense '
 - Etendue au **C++**
- Distingue 2 niveaux:
 - **Logique**
 - Diagrammes de classes
 - Diagramme d'instance
 - Diagramme états/transitions
 - **Physique**
 - Diagrammes de modules
 - Diagramme de processus



Grady Booch

OMT

- **Object Modeling Technique**
 - Livre de James Rumbaugh (1991)
- 3 axes
 - **Statique**
 - **Dynamique**
 - **Fonctionnel**
- 4 phases de modélisation
 - Analyse
 - Conception Système
 - Conception Objet
 - Implantation
- Un article signé J. Rumbaugh paraît chaque mois dans JOOP



James Rumbaugh

OOSE

- **Object Oriented Software Engineering**
 - Souvent appelée **Objectory**
- 5 modèles
 - Besoins
 - Analyse
 - Conception
 - Implantation
 - Test
- 3 types d'objets (MVC en Design Paterns)
 - **entités**
 - **contrôles**
 - **interfaces**
- Notion de Cas d'Utilisation: **Use Cases**



Ivar Jacobson

Méthodes Objets

- En 1994, plus de **50 méthodes OO**
 - Fusion, Shlaer-Mellor, ROOM, Classe-Relation, Wirfs-Brock, Coad-Yourdon, MOSES, Syntropy, BOOM, OOSD, OSA, BON, Catalysis, COMMA, HOOD, Ooram, DOORS...
- Les **méta-modèles se ressemblent** de plus en plus
- Les **notations graphiques** sont toutes **différentes**
- L'industrie a besoin de **standards**

Naissance d'UML

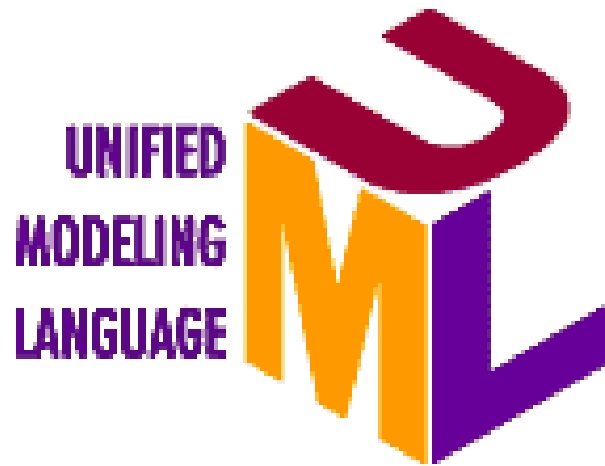
- 1993-1994: **Booch'93, OMT-2**
 - Les 2 méthodes sont **leaders** sur le marché
 - Elles sont de plus en plus **proches**
- Octobre 1994
 - **J. Rumbaugh** (OMT) rejoint **G. Booch** chez **Rational**
 - Annonce de l'unification des deux méthodes
- Octobre 1995: **Méthode Unifiée** v0.8
- Fin 1995: le fondateur d'Objectory, **Ivar Jacoson**, rejoint à son tour **Rational**
- Janvier 97 : **Soumission** à l'**OMG** de la version UML 1.0
 - OMG: Object Management Group
 - Organisme à but non lucratif fondé en 1989
 - Plus de 700 entreprises y adhèrent
 - Connu pour la norme CORBA
- Septembre 97 : **UML 1.1**



Conclusion

- **UML**: Prendre le **meilleur** de chacune des méthodes
 - **OOSE** (Jacobson): **Use Cases**
 - **OMT** (Rumbaugh): **Analyse**
 - **Booch**: **Conception, Architecture**
- UML est dans le **domaine public**
- Soutenu par le **marché**
 - Microsoft, HP, Oracle, IBM...

La Modélisation



UML ?

- Est une **notation, pas une méthode**
- Est un **langage** de modélisation objet
- Convient à **toutes** les méthodes objets
 - Unified Software Development Process
 - Catalysis
- Convient à **tous** les langages objets
 - C++ (Héritage multiple, Template)
 - Java (Interface)
 - SmallTalk

Pourquoi Modéliser

Propos tenus par Grady Booch



- « Dog House »
 - clous, planches, marteau, scie
- « House for your Family »
 - clous, planches, marteau, scie, plans, fondation, lumière, chauffage, plomberie, équipe
- « High-Rise Office Building »
 - maquette, planning, plusieurs équipes (communication), parallélisation des tâches, spécialistes, machines

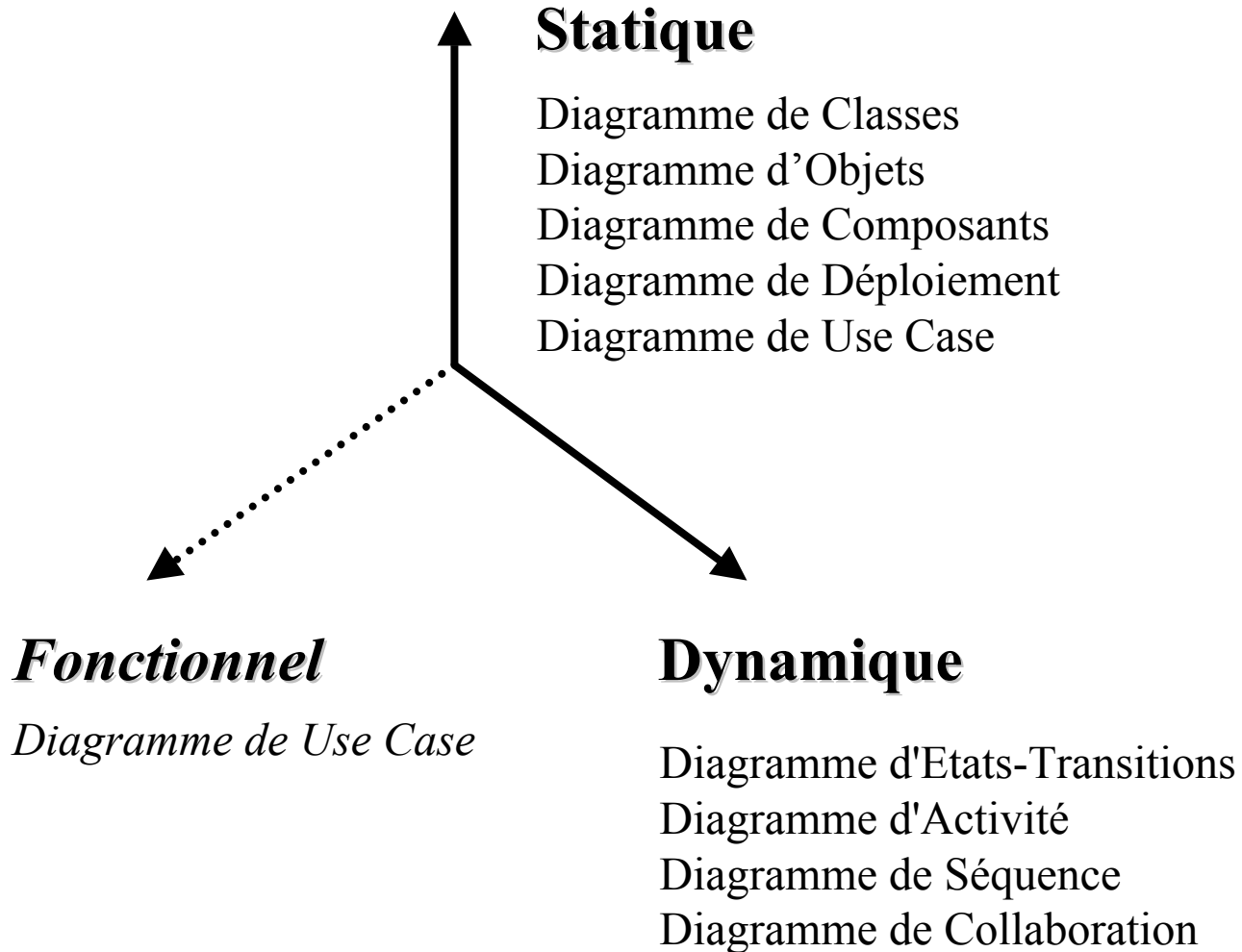
Processus de développement

- **UML ne définit pas de processus standard de développement**
 - Ce n'est le but ni d 'UML, ni de l 'OMG
- Ceux de Booch, OMT, et OOSE **restent applicables**
- Processus **itératif** et **incrémental**
 - Segmentation du travail
 - Les premières itérations sont des prototypes

NOUVEAU!

- The Unified Software Development Process (addison-Wesley, 1999)

Axe de Modélisation

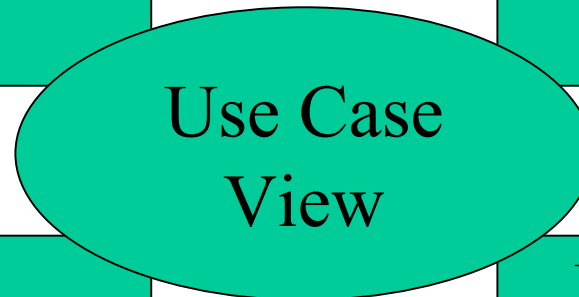


Les 4+1 Vues

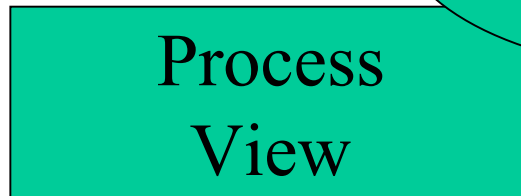
- Classes
 - Interfaces
 - Collaboration
- => Les services du systèmes



- Composant
 - Fichiers Source
- => Configuration du système



=> Comportement du système

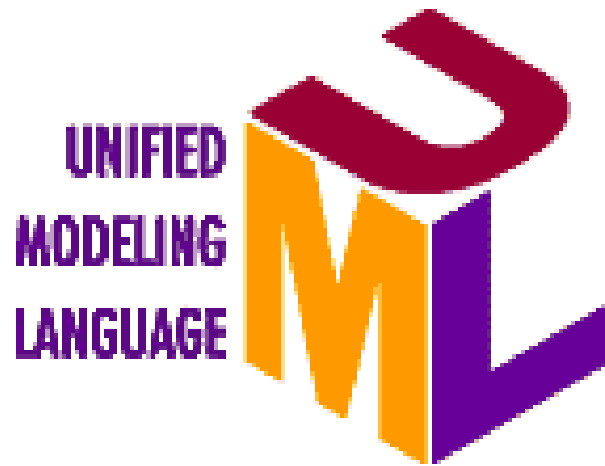


- Thread
 - Process
 - Concurrency
 - Synchronisation
- => Performance du système



- Architecture
 - Hardware
 - Distribution
- => Topologie du système

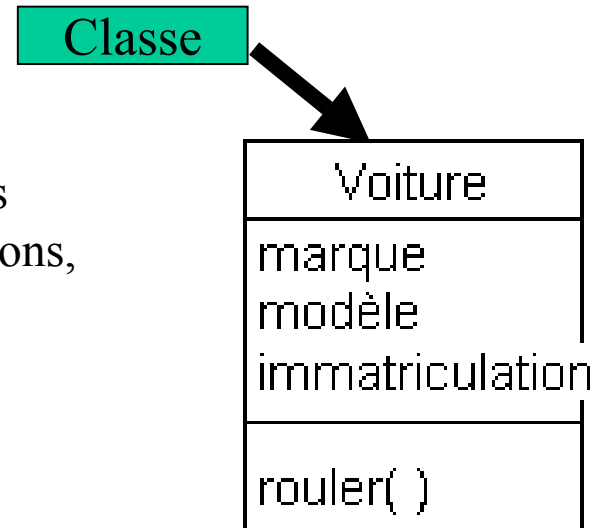
L 'Axe Statique



Notation de base

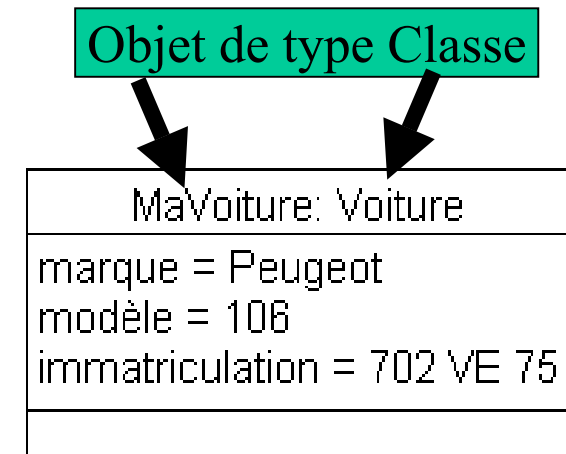
- **Classe**

- Une description d'un ensemble d'objets qui partage les mêmes attributs, opérations, méthodes, relations et contraintes



- **Objet**

- Une entité avec une limite et une identité bien définies qui encapsule de l'état et du comportement. L'état est représenté par des attributs et des relations, le comportement est représenté par des opérations et des méthodes. Un objet est une instance d'une classe.



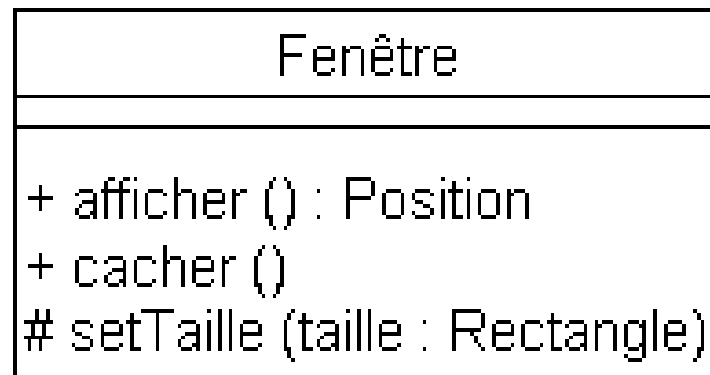
Attribut

- **Attribut** = **propriété** nommée d'une classe
- **Syntaxe**
 - *visibilité nom : type = valeur initiale*
- **Visibilité**
 - + public
 - # protégé
 - - privé
 - package
- **Attribut de classe**
 - la **portée standard** d'un attribut est limité à un **objet**
 - quand cette **portée** s'applique à la **classe** elle même, on parle d'**attribut de classe** (représenté par le symbole \$ ou **souligné**)
- **Attribut dérivé**
 - attribut qui peut être **déduit** d'un ou plusieurs **autres attributs** (représenté par le symbole /)

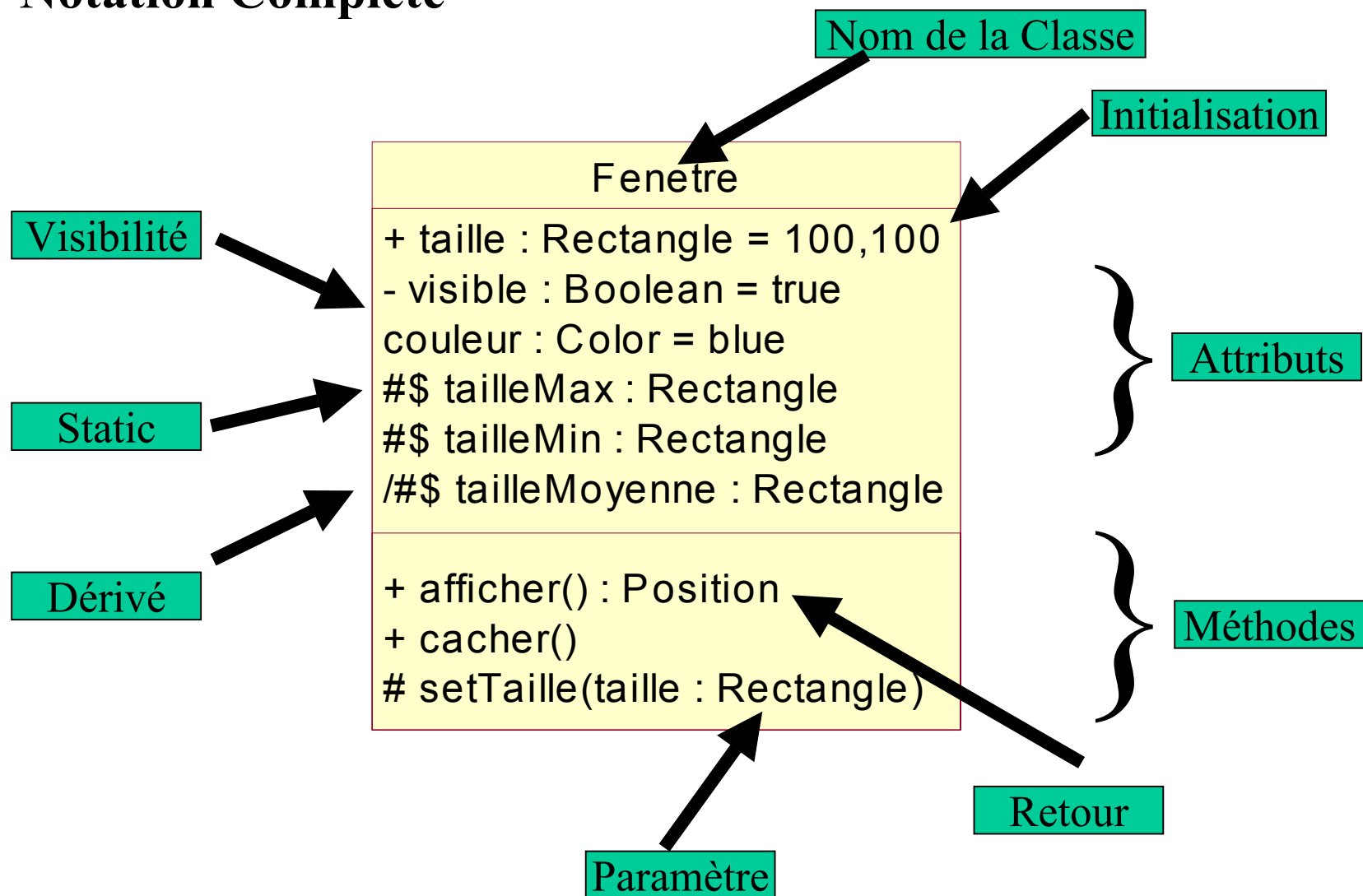
Fenêtre
+ taille : Rectangle = (100,100)
- visible : boolean = TRUE
- xptr : XWindow
#\$ tailleMax : Rectangle
#\$ tailleMin : Rectangle
/#\$ tailleMoyenne : Rectangle

Méthode

- **Méthode** = **service** que l'on peut demander à un objet pour réaliser un comportement
- **Syntaxe**
 - *visibilité nom (paramètres) : type retour*
- Mêmes notions que l'attribut
 - **visibilité**
 - **méthode de classe**
 - **méthode dérivée**



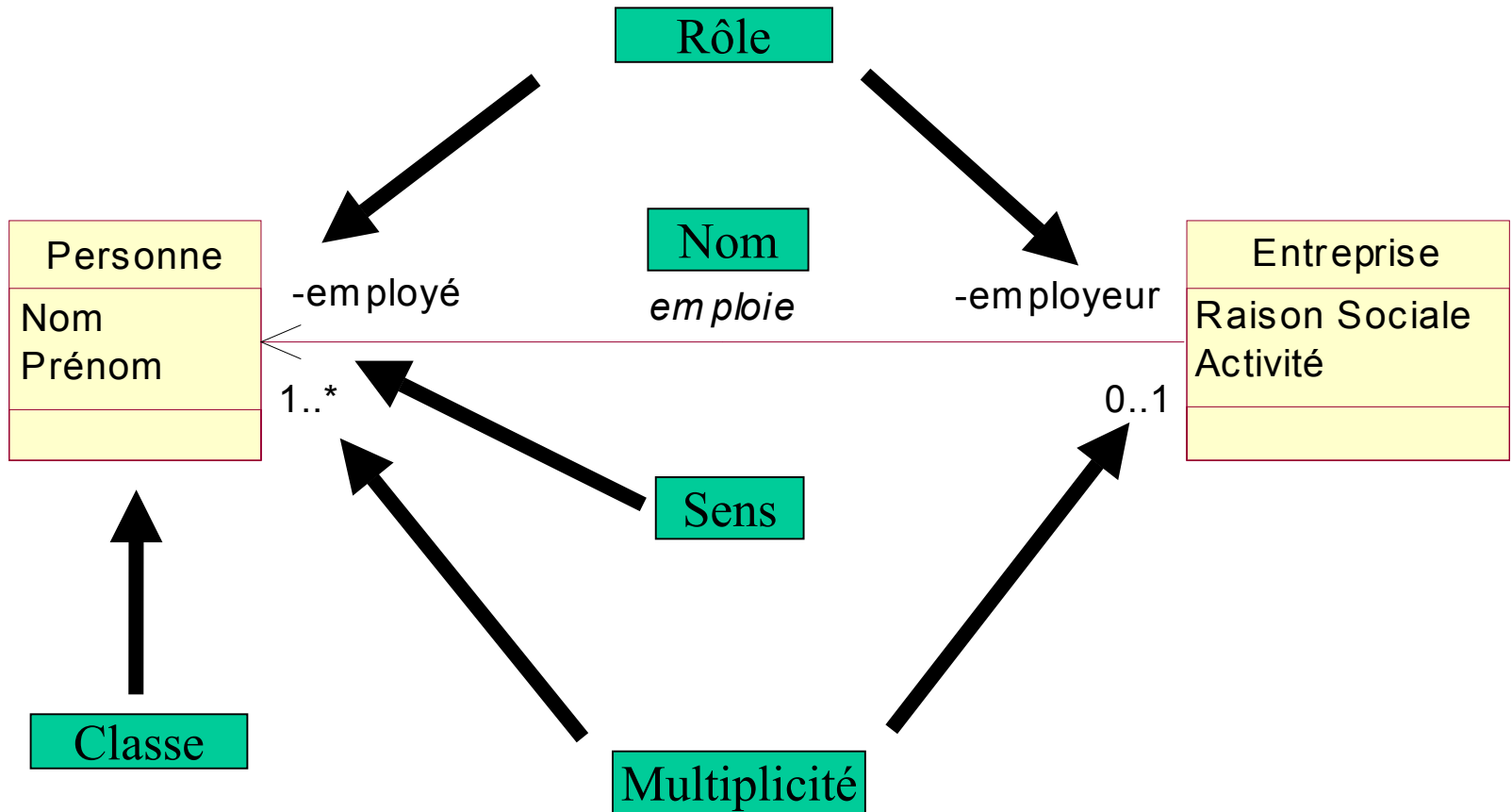
Notation Complète



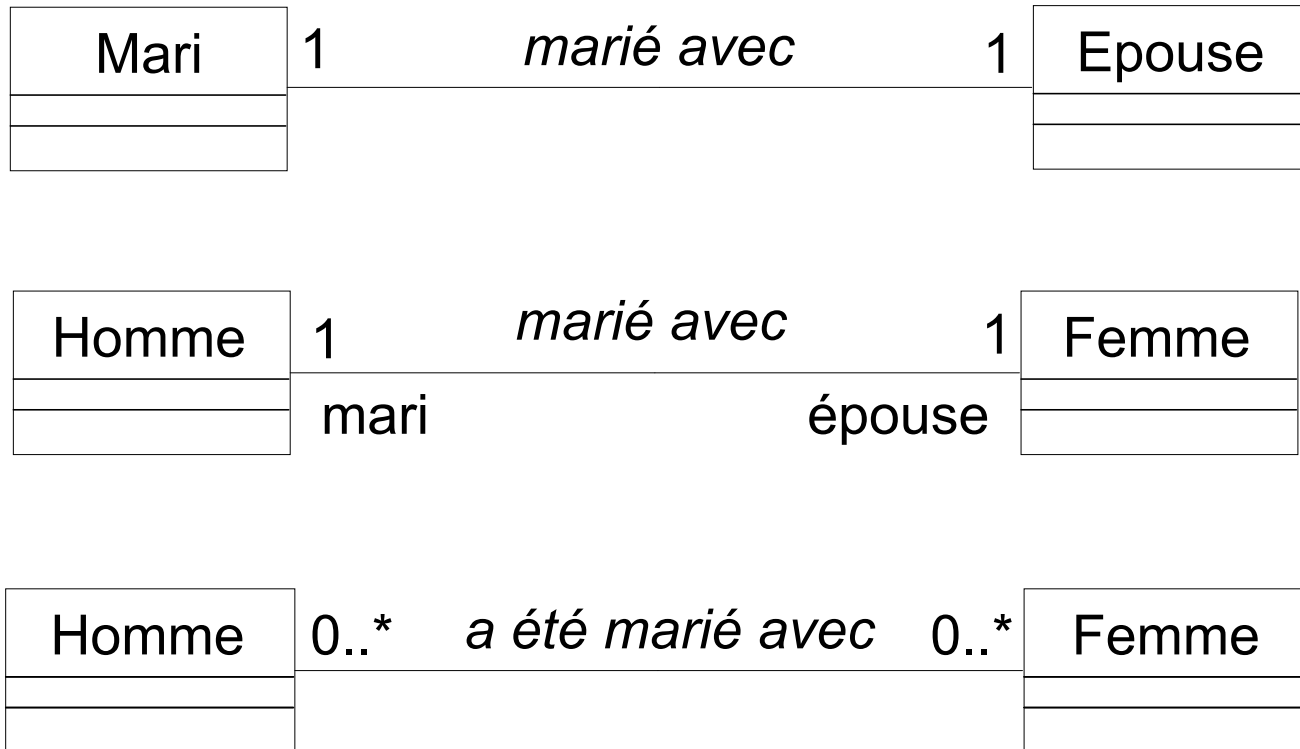
Définition

- **Association**
 - Exprime une **connexion** sémantique bi-directionnelle entre classes
 - Abstraction des liens qui existent entre objets
 - Le **sens** d'une association peut-être précisé par une **flèche**
- **Association binaire** = Association entre 2 classes. Cas particulier d'association **n-aire**
- **Rôle** = rôle joué par une classe dans une association
- **Multiplicité** = indique le **nombre** d'instances d'une classe qui peut être mise en relation avec une seul instance de la classe associée
 - 1 : obligatoire
 - 0..1 : optionnel
 - 0..* ou * : quelconque
 - 1..* : au moins 1
 - 1..5, 10 : entre 1 et 5, ou 10

Exemple

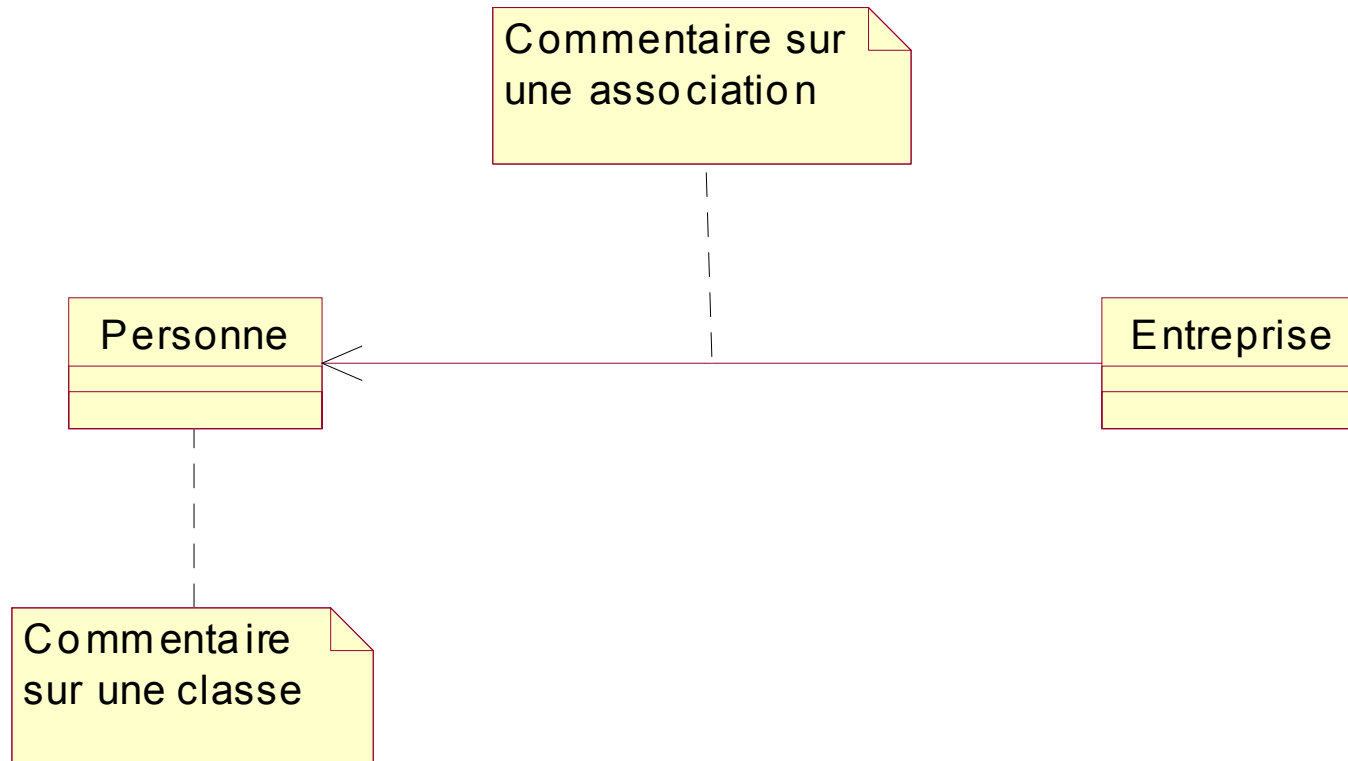


Sémantique



Note

- **Note** = Commentaire placé sur un diagramme



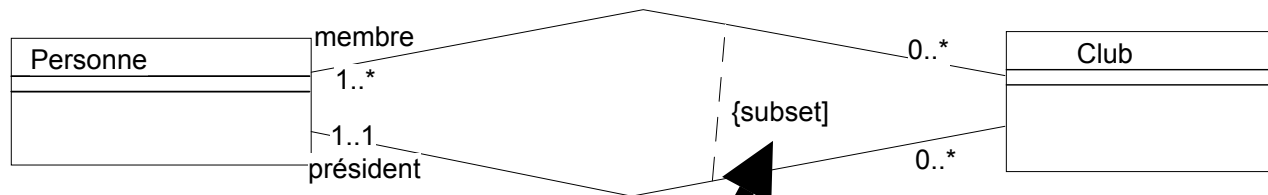
Contrainte

- **Contrainte** = relation sémantique entre éléments de modèle qui spécifie des conditions ou propositions devant rester **vraies pour que le modèle soit valide**

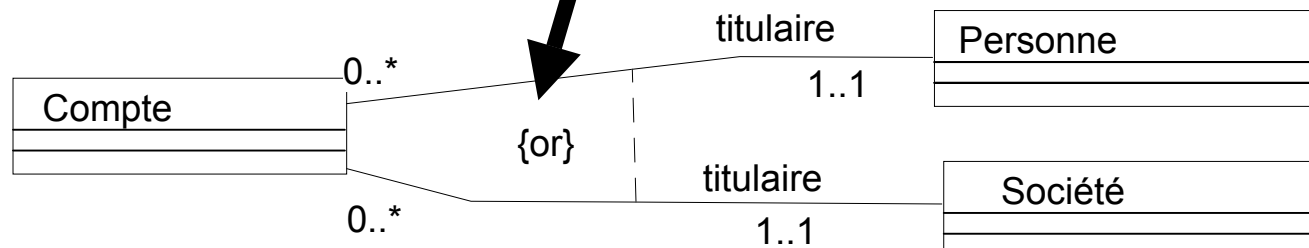
{ contrainte }

- Contraintes **Pré-Définies**

– Inclusion

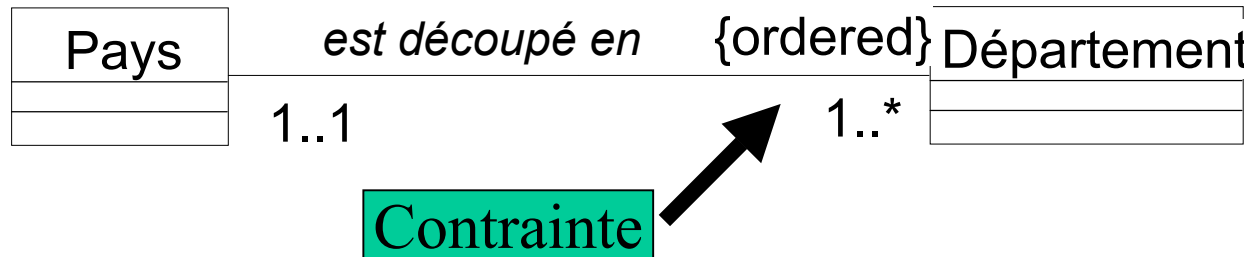


– Exclusion

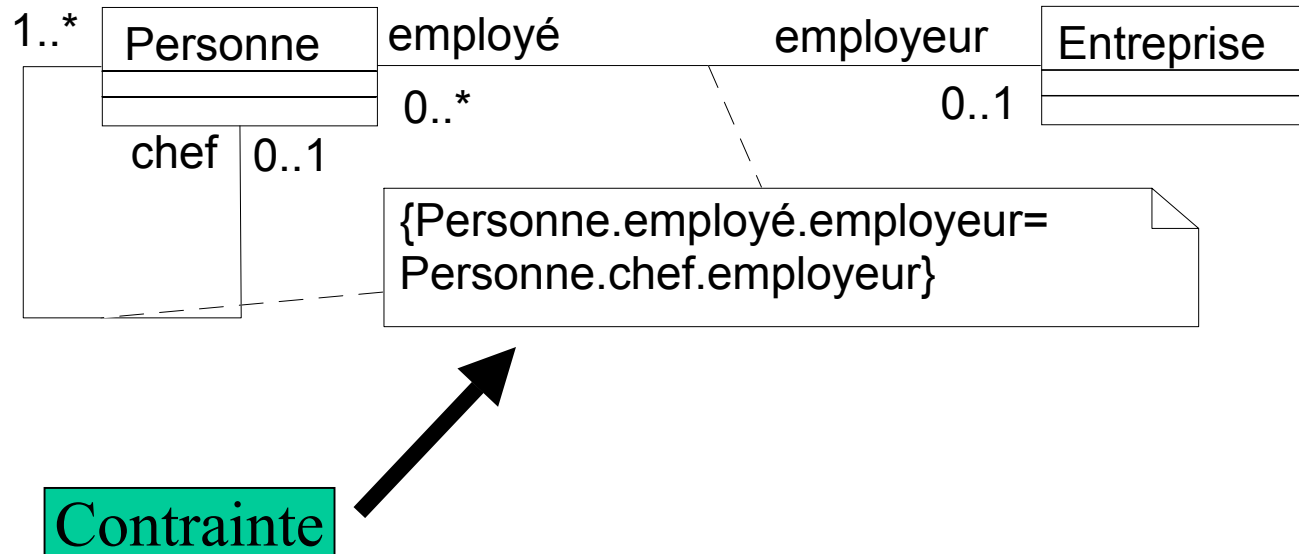


Contraintes Pré-définies

- **Ordre**

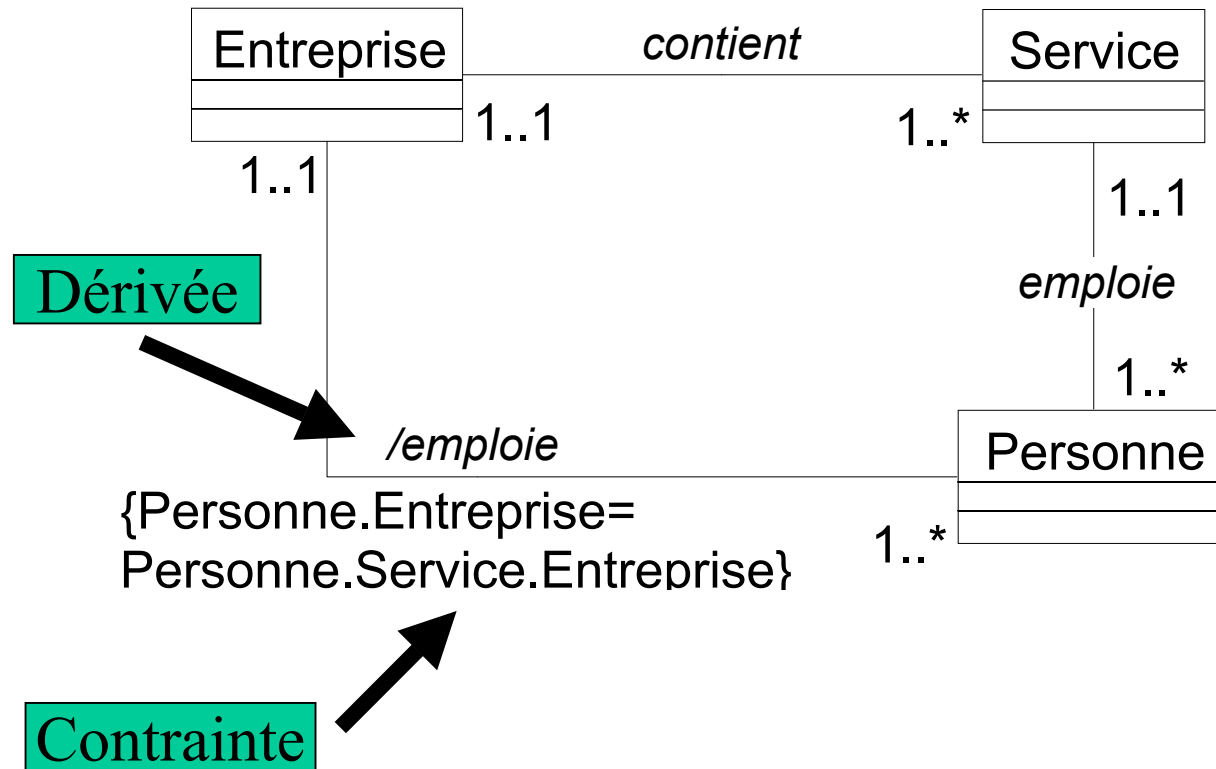


- **Chemins**



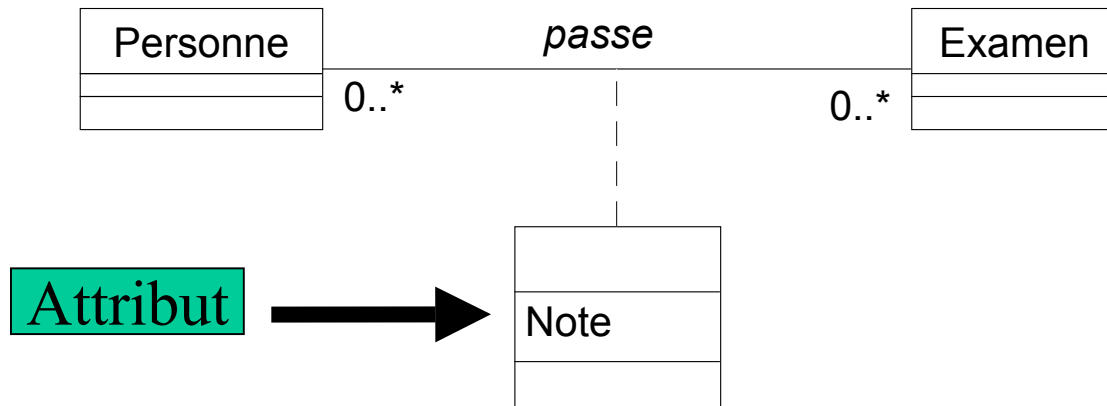
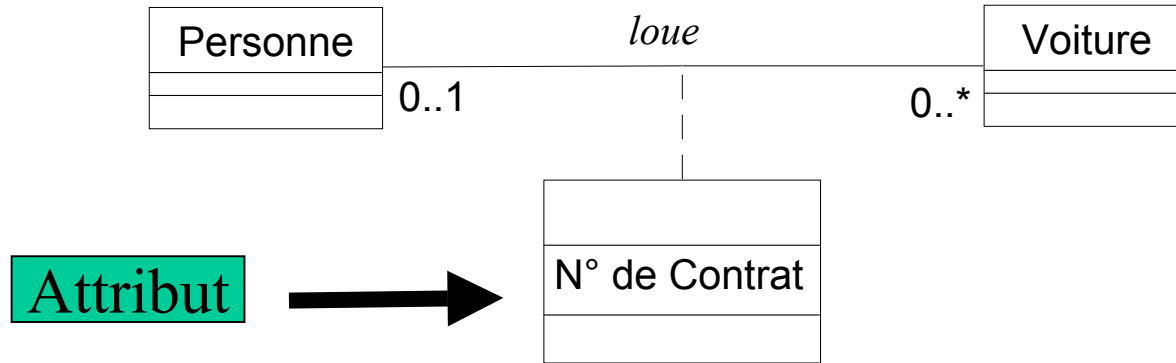
Associations dérivée

- Une **association dérivée** est une association qui peut se **déduire** des associations existantes



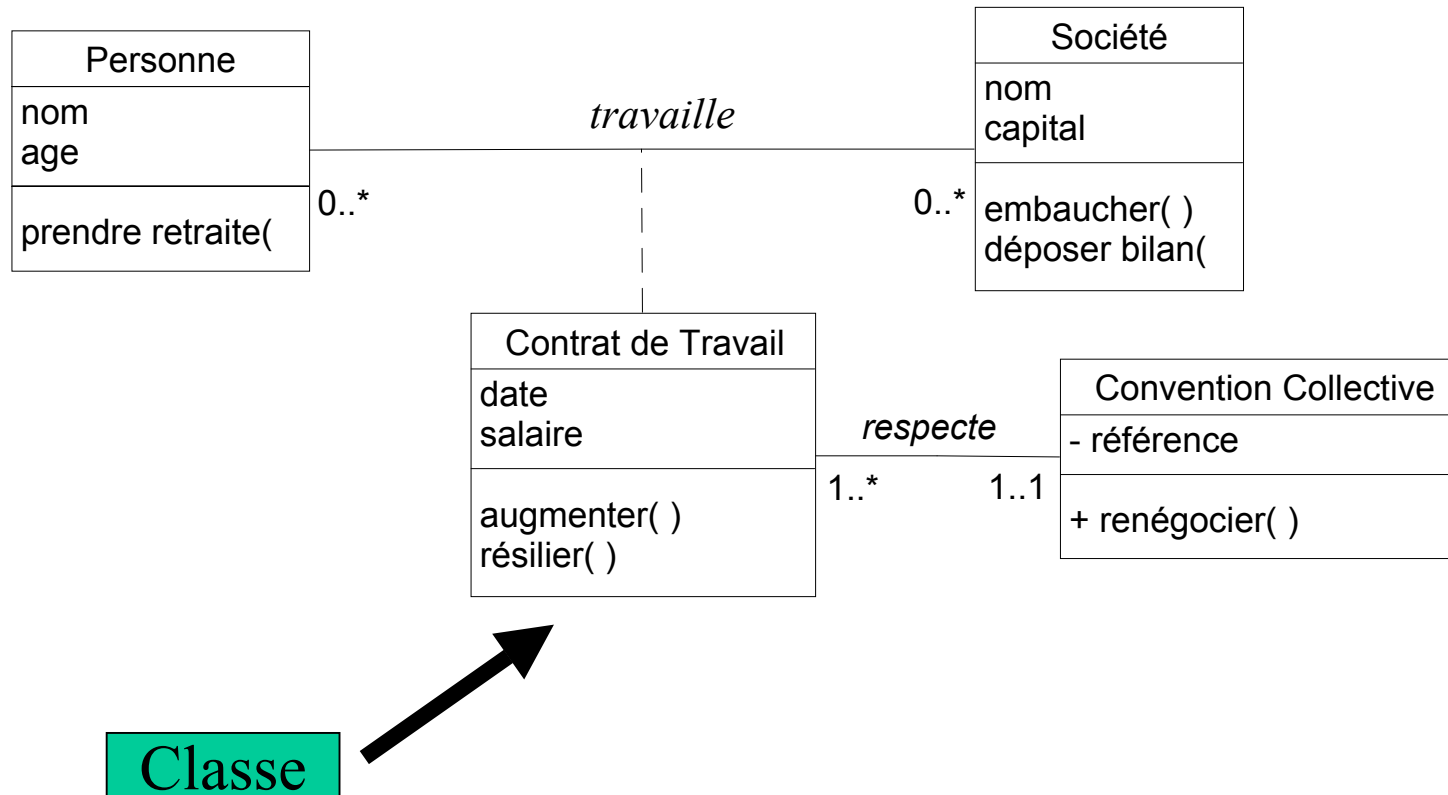
Attribut d'Association

- **Attribut d'association = propriété** du lien entre deux objets



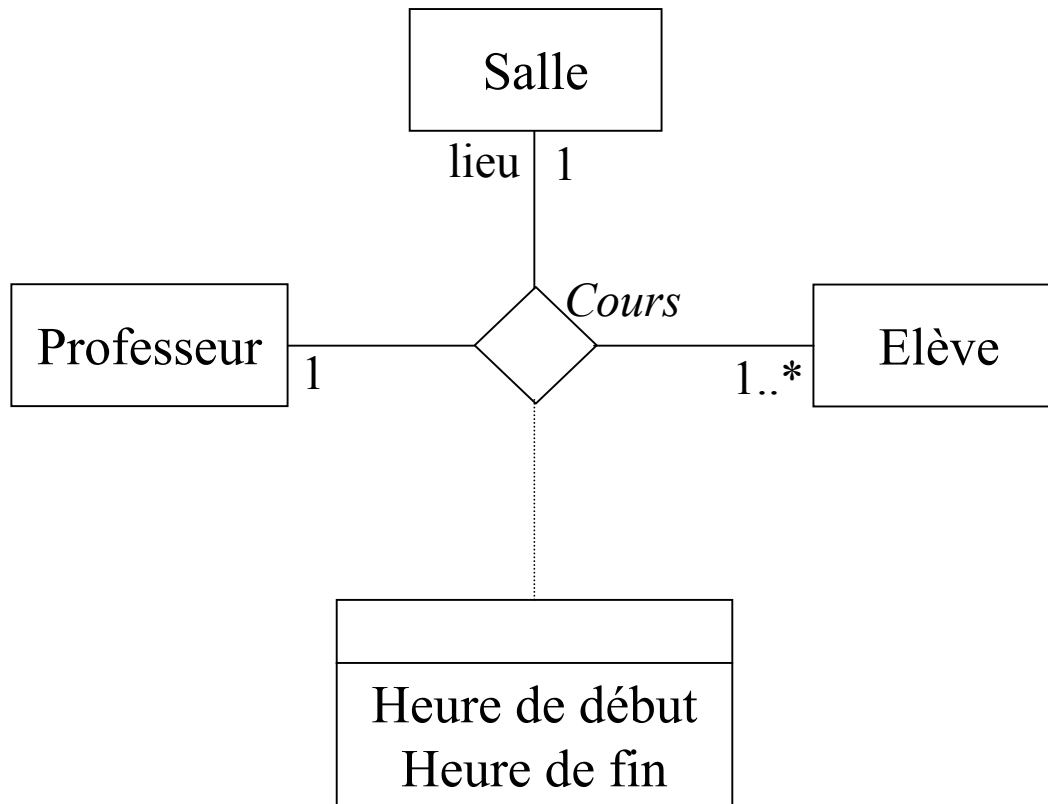
Classe d'Association

- **Classe d'association** = Élément ayant à la fois les propriétés d'une classe et d'une association



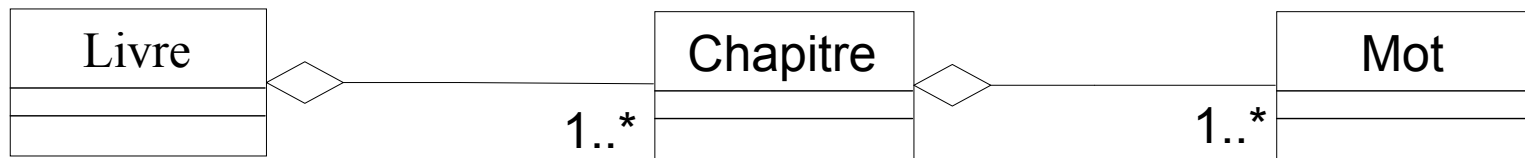
Association n-aire

- **Association n-aire** = Une association parmi 3 classes ou plus. Chaque instance de l'association est un n-tuple de valeurs des classes respectives.



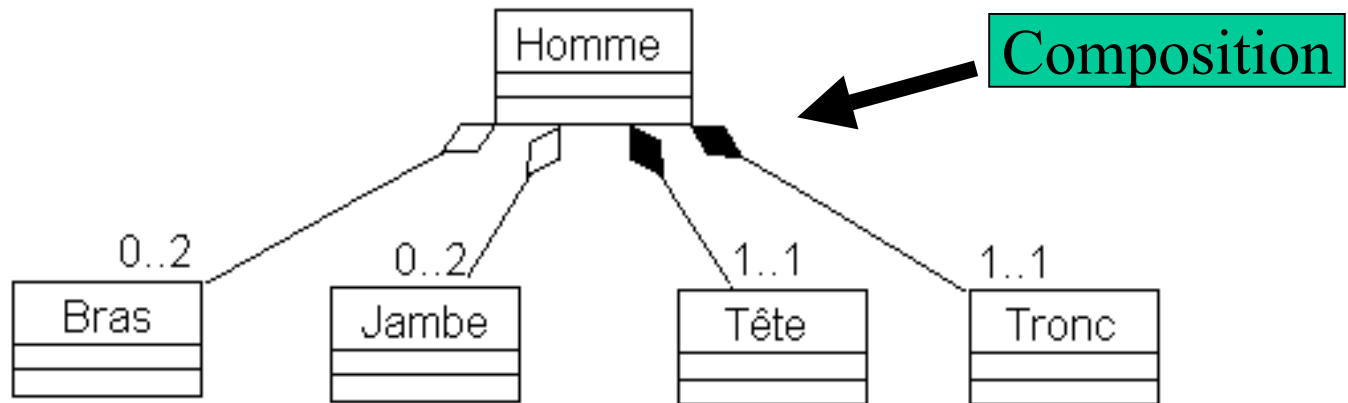
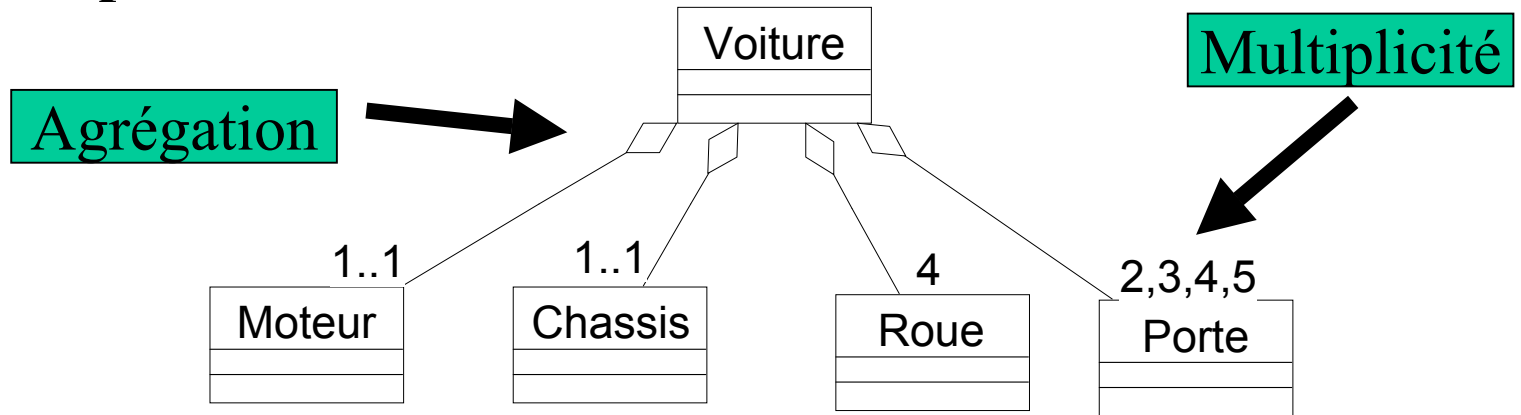
Définitions

- **Agrégation** = **association** particulière spécifiant une relation ‘tout - partie’ entre l’agrégat et un composant
 - **Inclusion**
 - **Propagation**



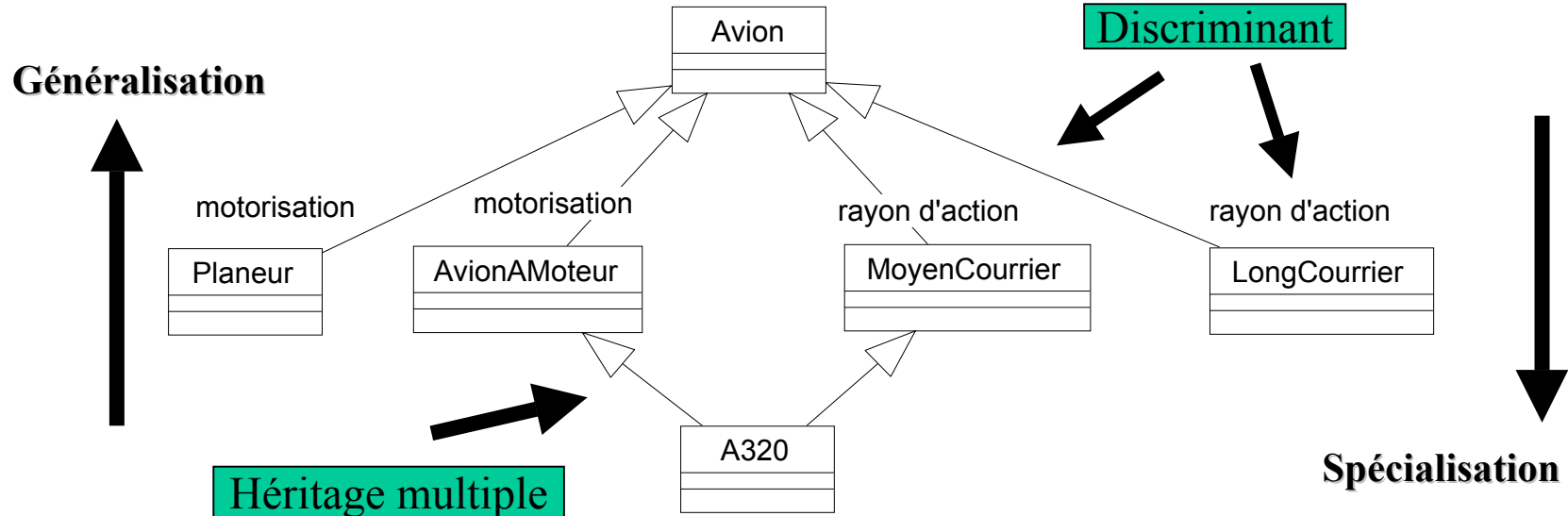
- **Composition** = forme **forte d’agrégation** avec un **cycle de vie** des parties lié à celui du composite

Exemples



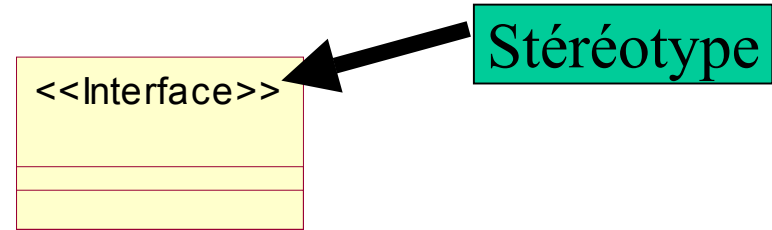
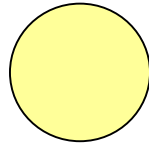
Définitions

- **Généralisation** = relation entre un élément plus général et un élément plus spécifique qui est entièrement conforme avec le premier élément, et qui ajoute de l'information supplémentaire
- **Spécialisation** = mécanisme par lequel des éléments plus spécifiques incorporent la structure et le comportement d'éléments plus généraux (notion d'**héritage**).

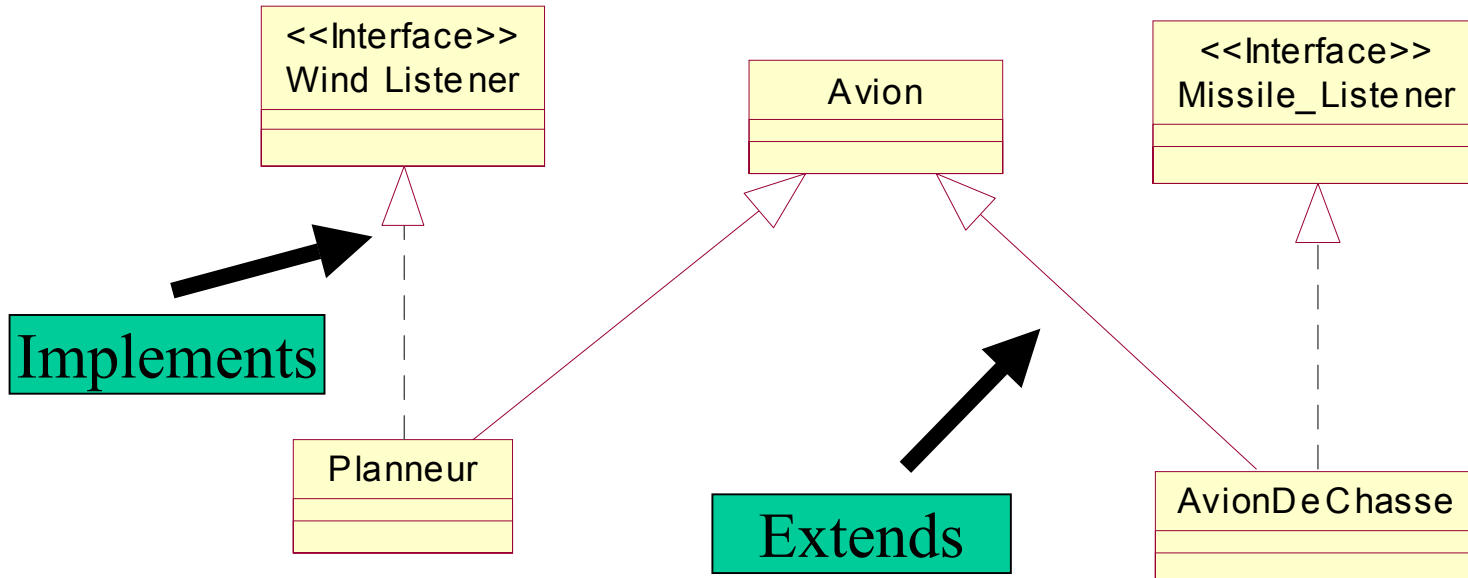


Interface

- Notations

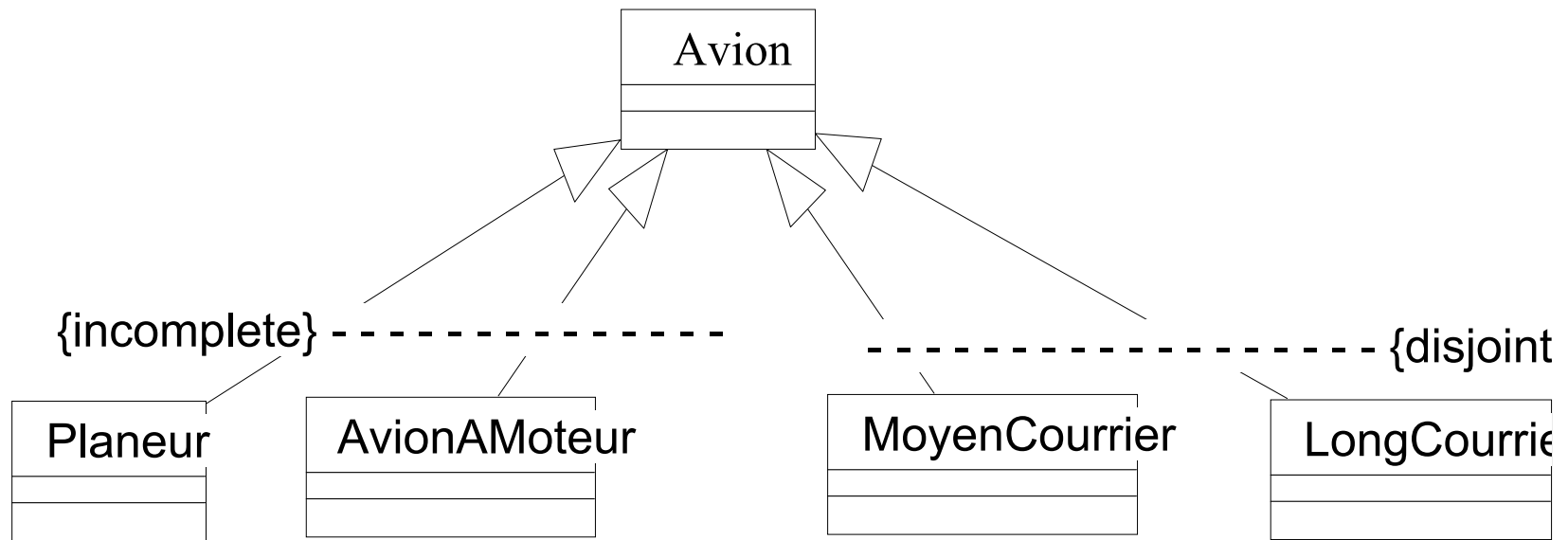


- Hériter d'une interface



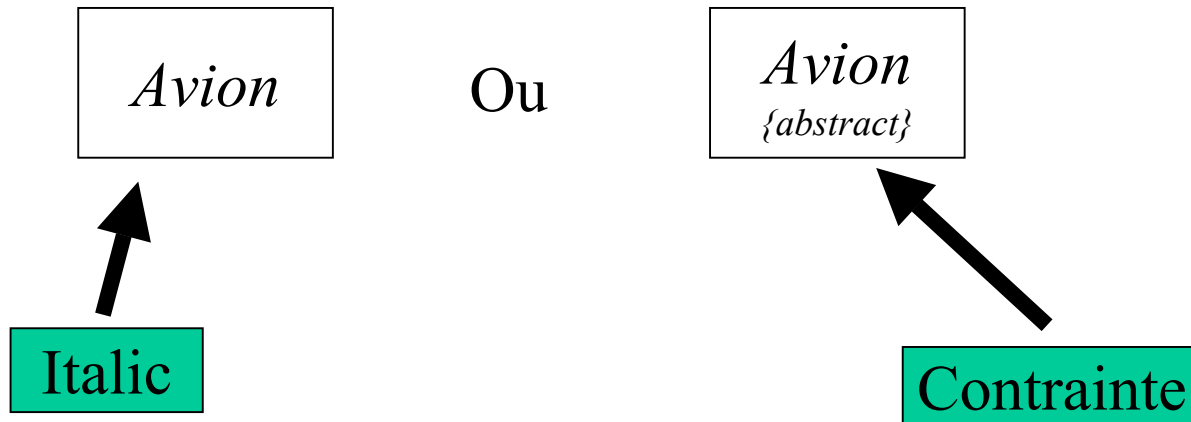
Contraintes

- Les seules **contraintes** pré-définies en UML pour la généralisation sont :
 - **disjoint** (un moyen courrier ne peut être long courrier) / **overlapping**
 - **complète** (liste exhaustive de classe) / **incomplète**



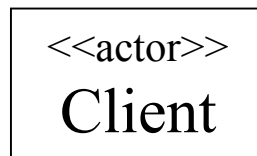
Classe Abstraite

- **Classe Abstraite** = classe que l'on ne peut pas **instancier**
- **Notation :**

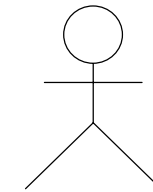


Stéréotype et Valeur Etiquetée

- Deux mécanismes d'extensibilité d'UML, applicables à tout élément de modèle :
 - **Stéréotypes** = nouveau type d'élément de modélisation qui étend la sémantique du méta-modèle (peut avoir une **représentation visuelle**) :



<< stéréotype >>

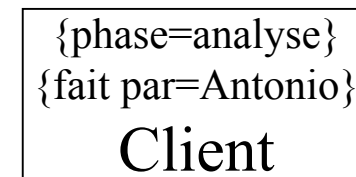


Client

Valeur Etiquetée = définition explicite d'une propriété sous la forme d'une contrainte :

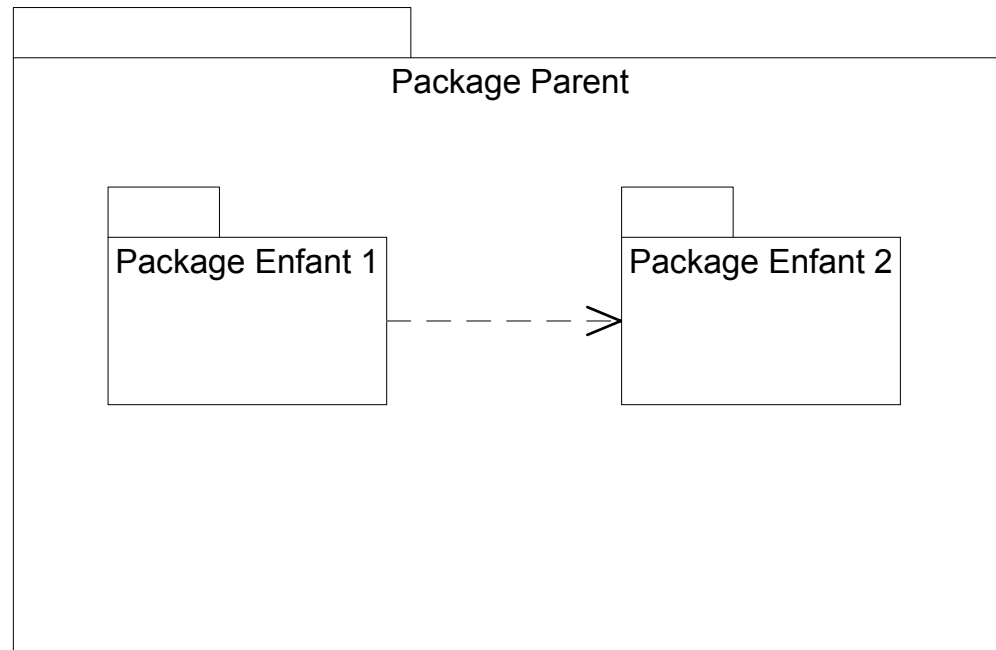


{étiquette = valeur}



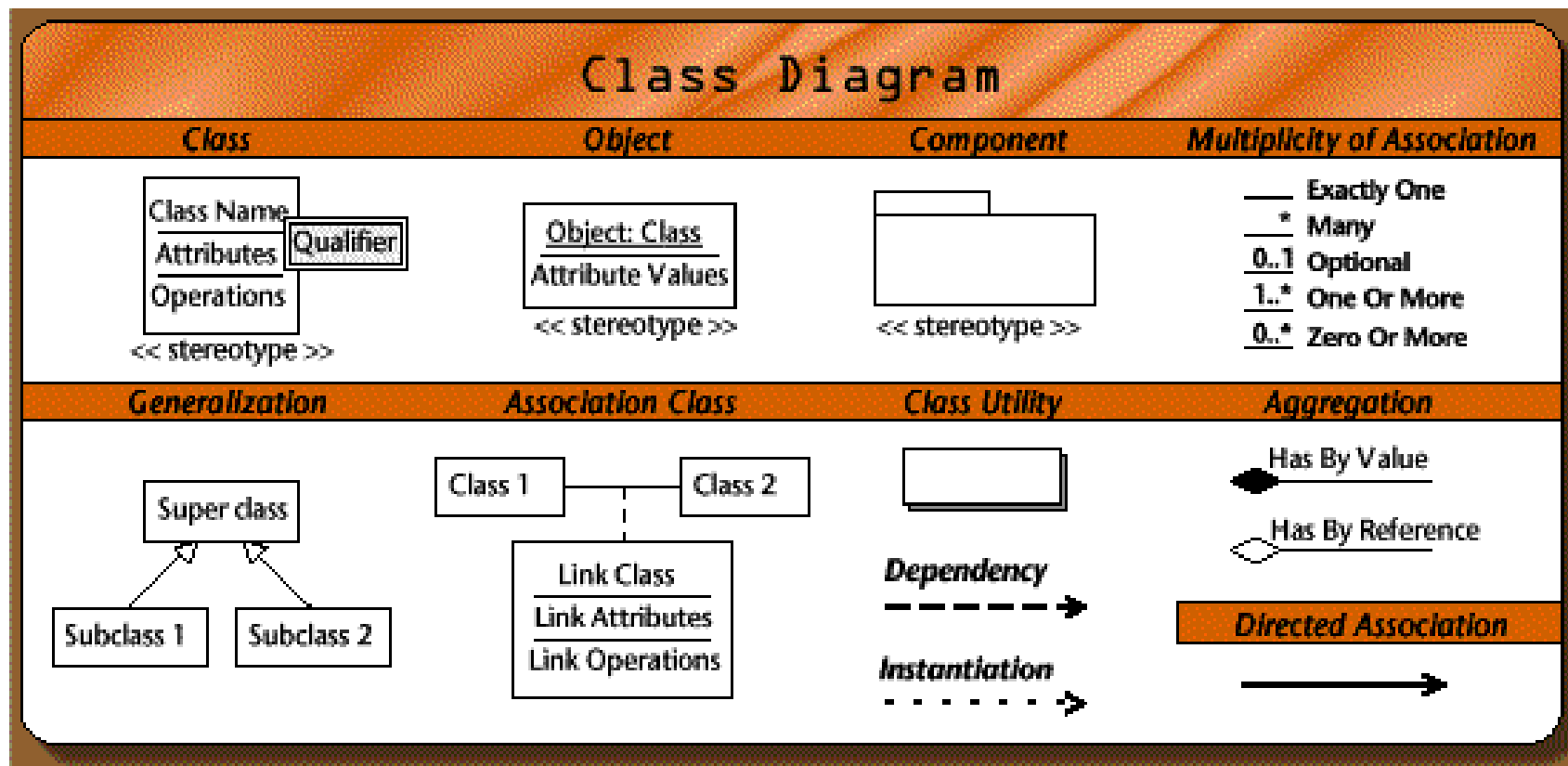
Pour Structurer

- **Package = Regroupement** d'éléments de modèle
- Les Packages divisent et organisent les modèles de la même manière que les répertoires organisent les systèmes de fichiers
- Les Packages eux-mêmes peuvent être **imbriqués** à l'intérieur d'autres Packages

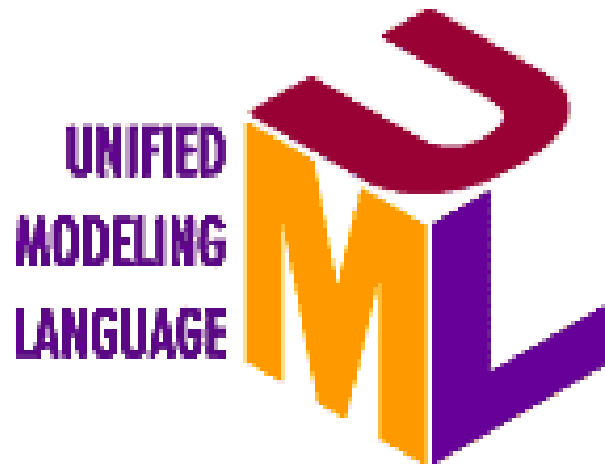


Notation

NOTATION UML



L 'Axe Dynamique



Que représente-t-on dans le modèle dynamique ?

- Le modèle dynamique représente les séquences **d'événements**, **d'états** et de **réactions** qui doivent survenir dans le système.
- Il est intimement lié au modèle objet et décrit les aspects de contrôle d'un système en prenant compte du **temps**, du **séquencement des opérations** et des **interactions** entre objets
- Deux diagrammes fondamentaux :
 - **Diagramme de Séquence**
 - **Diagramme Etats-Transitions**

Scénario

- Il y a autant de diagrammes de séquence qu'il y a de scénarios
- Un **Scénario** montre une séquence particulière d'interactions entre objets, dans un seul contexte d'exécution du système
- Un **scénario** peut être vu comme une des instances possibles des **Use Cases**.
- On y fait intervenir des **objets**, des **messages** et des **événements**

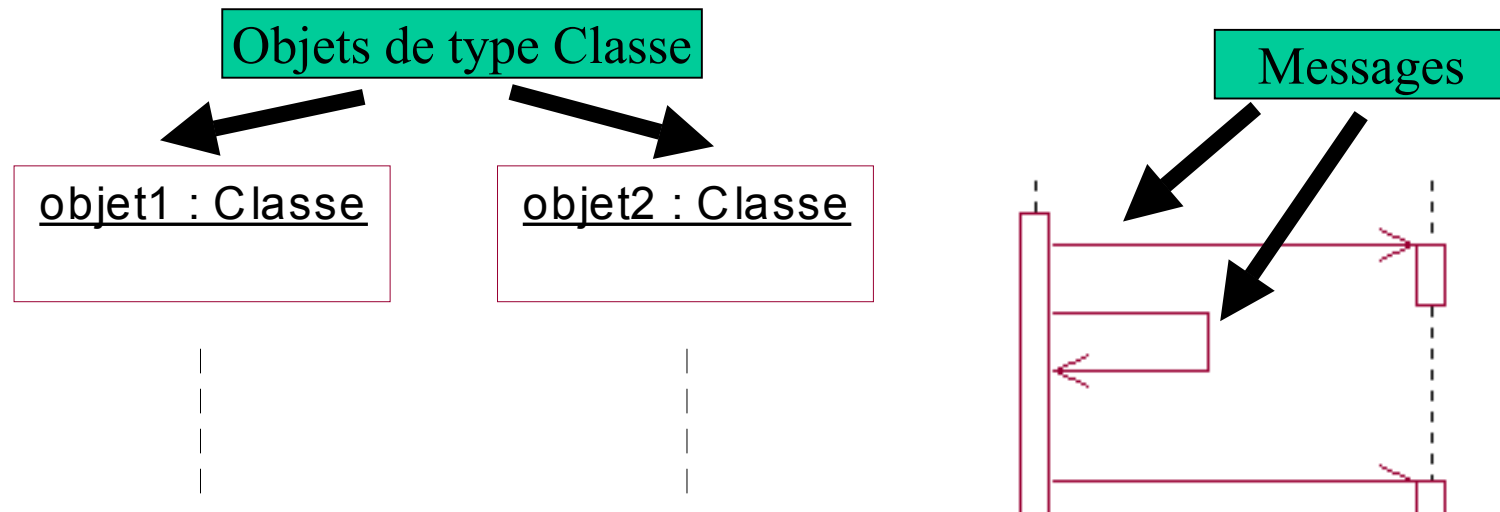
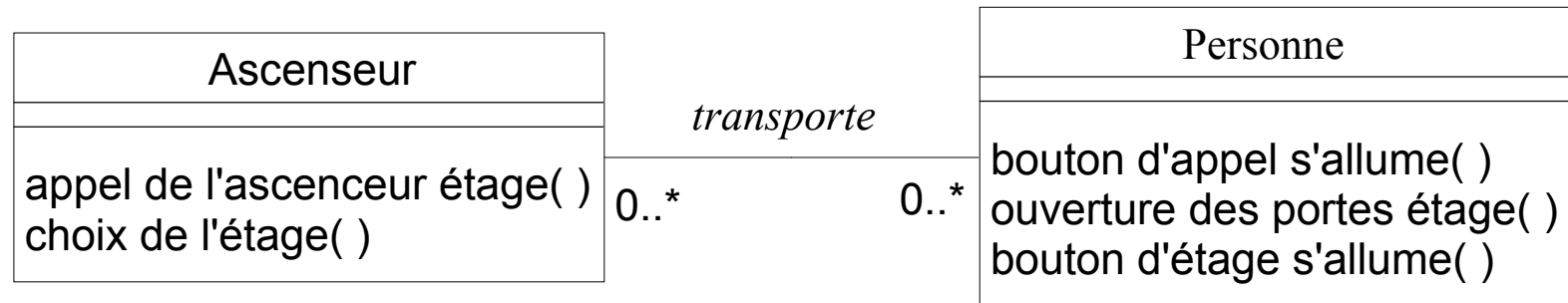


Diagramme de classes

- Exemple de l'ascenseur



- Scénario

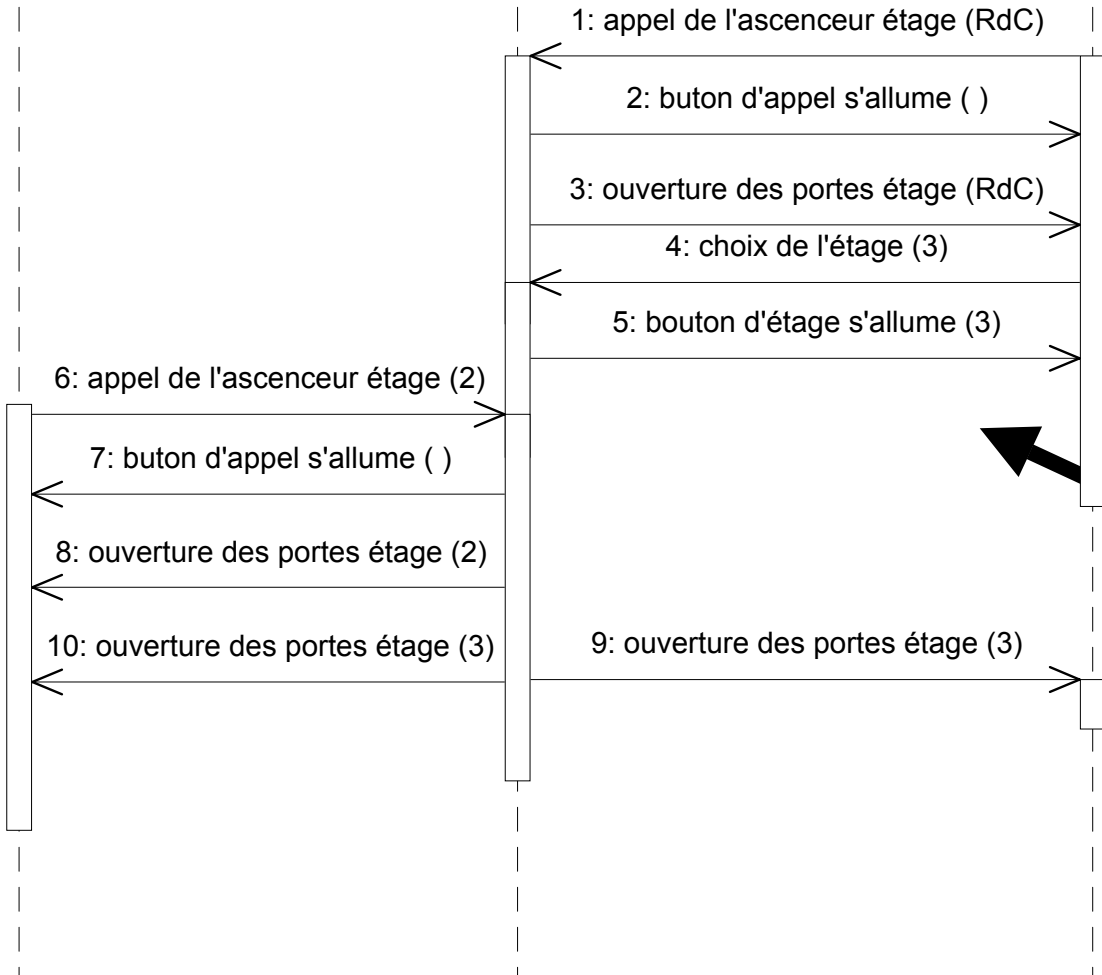
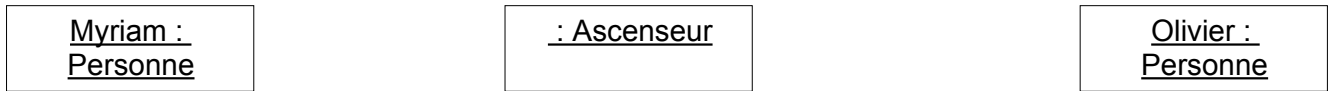
- **Deux** usagers situés à des **étages différents** empruntent le même ascenseur pour se rendre au **troisième** étage

≠

- **Trois** usagers situés au **même étage** empruntent le même ascenseur pour se rendre au **troisième et cinquième** étage

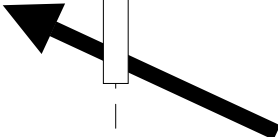
Notation Graphique

Objet



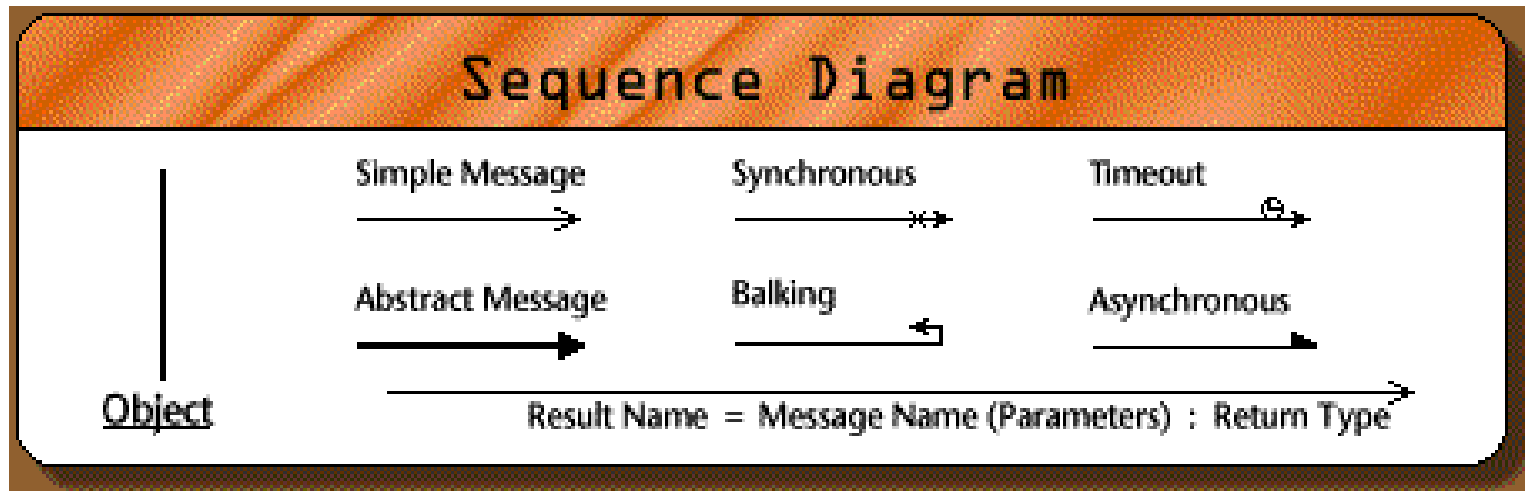
< 2 secondes

Méthode



NOTATION UML

Notation



Définition

- Un diagramme **Etats-Transitions** (ou **Automate**) :
 - décrit l'**évolution** au cours du **temps** d'une instance d'une classe en réponse aux interactions avec d'autres objets
 - est forcément associé à **une classe**, mais toutes les classes n'en ont pas besoin
 - est un graphe orienté d'**états** (nœuds) connectés par des **transitions** (arc orientés)
- Source: Les **Statecharts** de **David Harel**



David Harel

Etats

- Chaque objet est à un moment donné dans un **état** particulier :
 - **Etat Initial** : état d'une instance juste après sa création (un seul état initial)
 - **Etat Intermédiaire** : un objet est toujours dans un état donné pour un certain temps
 - **Etat Final** : état d'une instance juste avant sa destruction (un automate infini peut ne pas avoir d'état final)

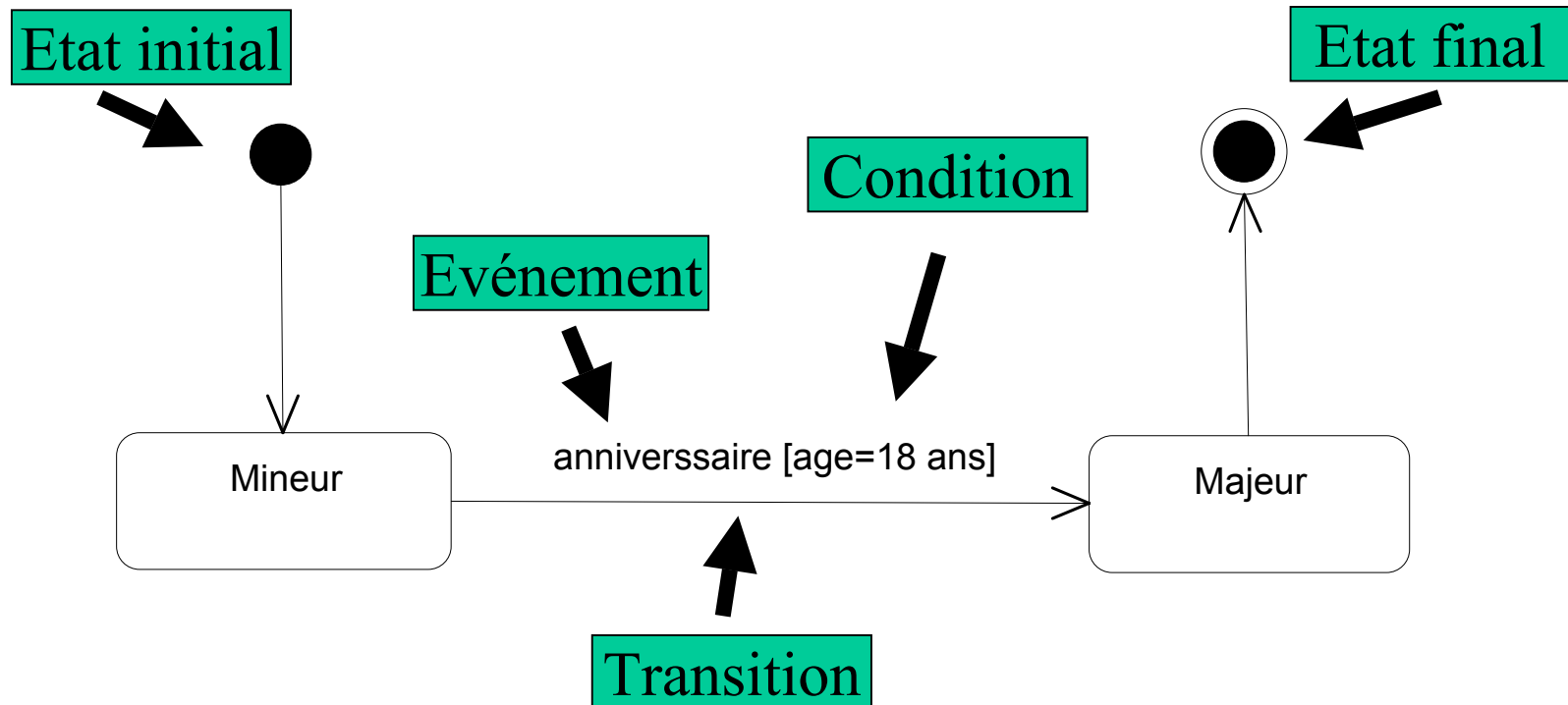
 état initial

 état intermédiaire

 état final

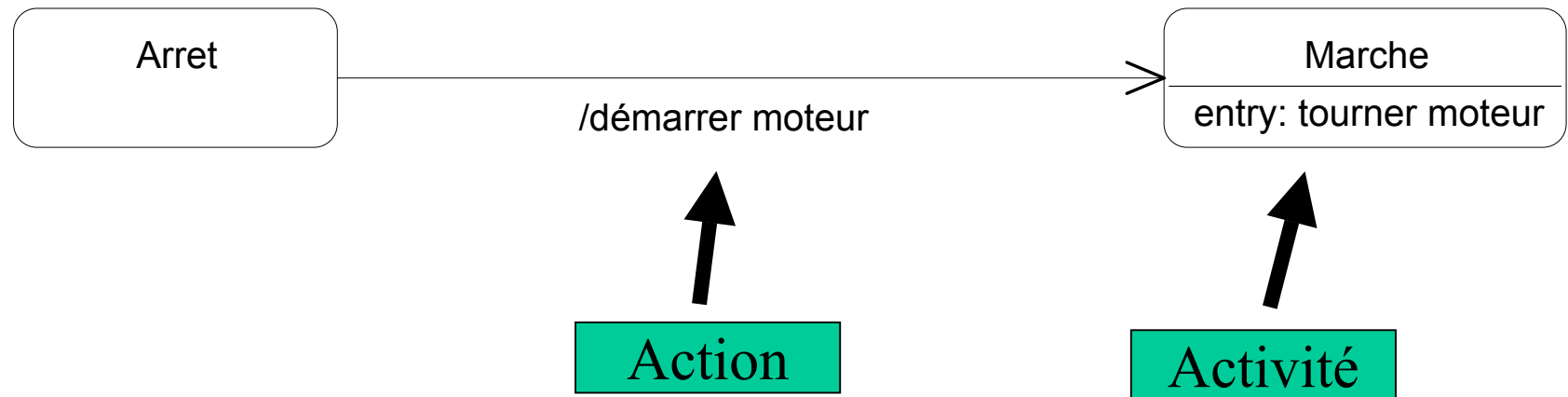
Transition, Condition

- **Transition** : relation entre 2 états indiquant qu'un objet dans le premier état va exécuter une **action** et entrer dans le deuxième état quand un **événement** apparaîtra
- **Condition** : expression booléenne devant être **vérifiée** pour permettre la transition



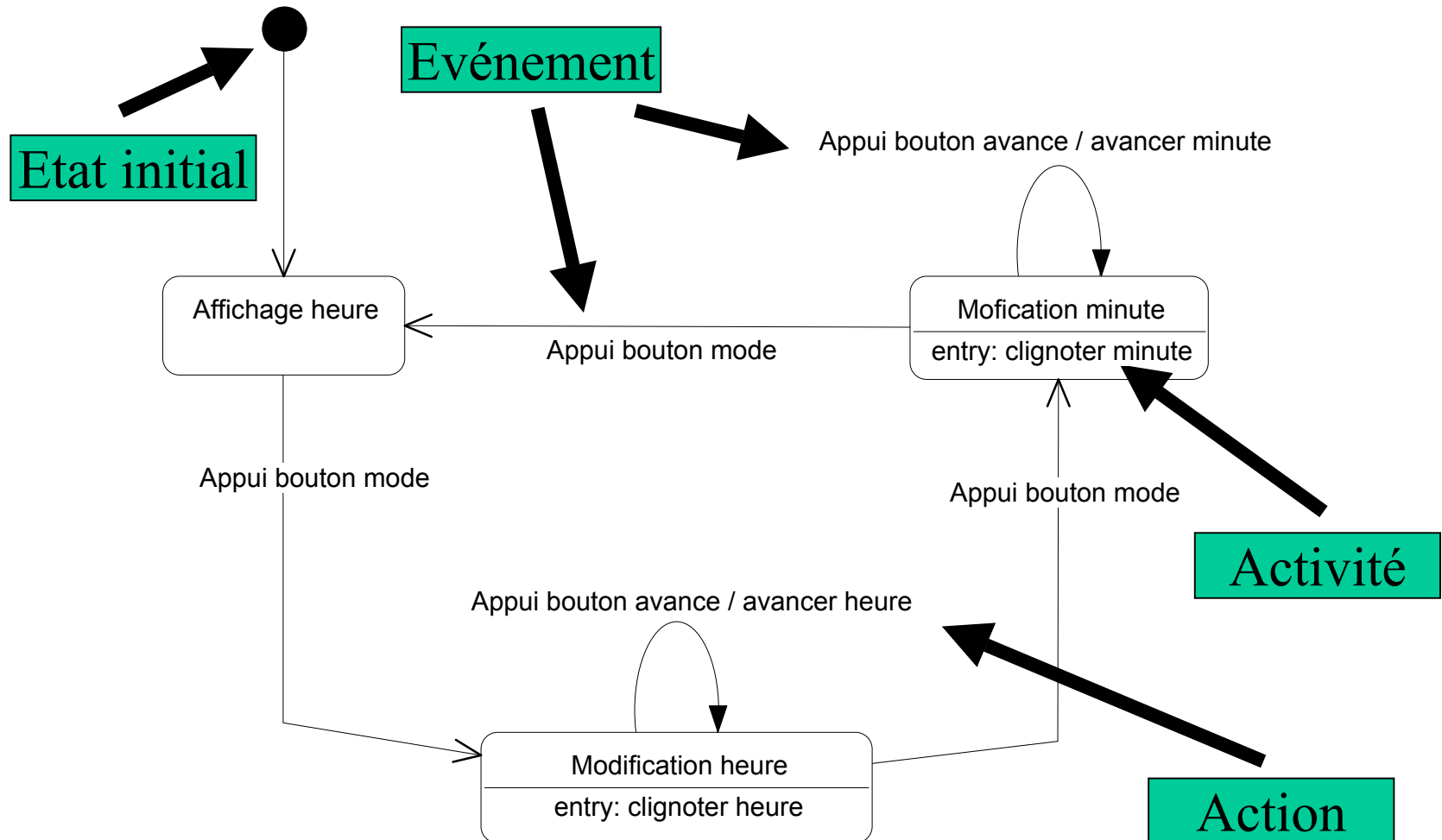
Action, Activité

- **Action** : opération **atomique** (non interruptible) déclenchée par une transition
- **Activité** : opération qui **dure** un certain temps (interruptible) dans un état particulier
 - **entry** : action exécutée chaque fois que l'on **rentre** dans l'état
 - **exit** : action exécutée chaque fois que l'on **quitte** l'état



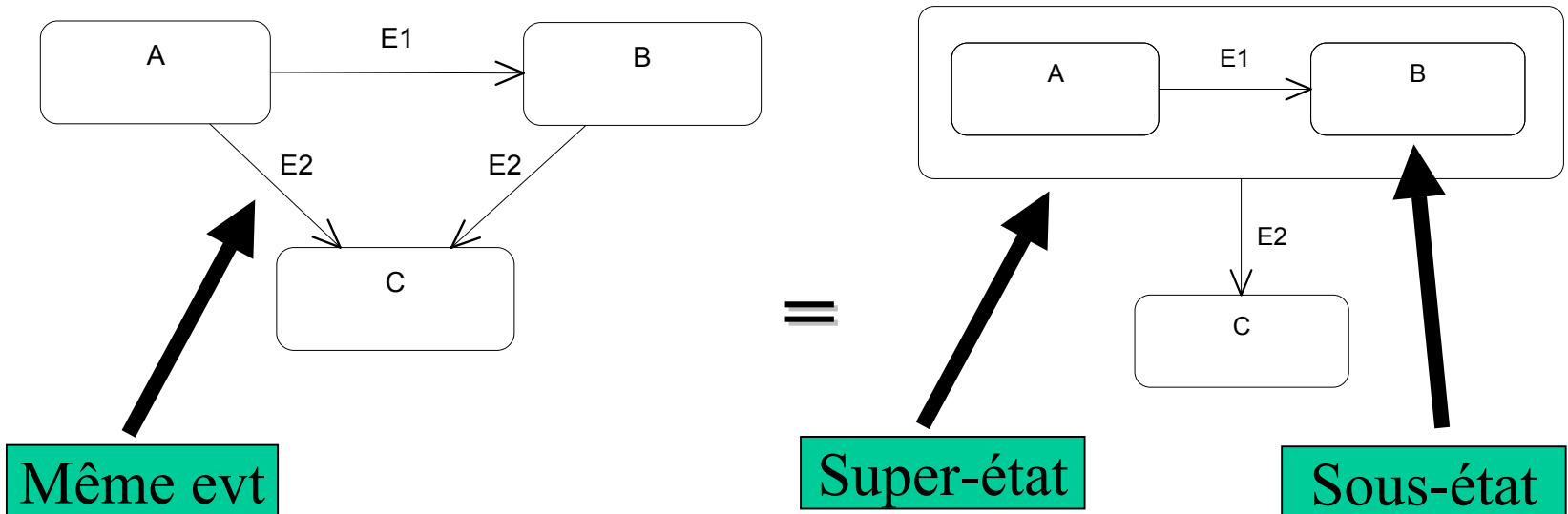
Notation Complète

- Exemple : fonctionnement d'une montre digitale



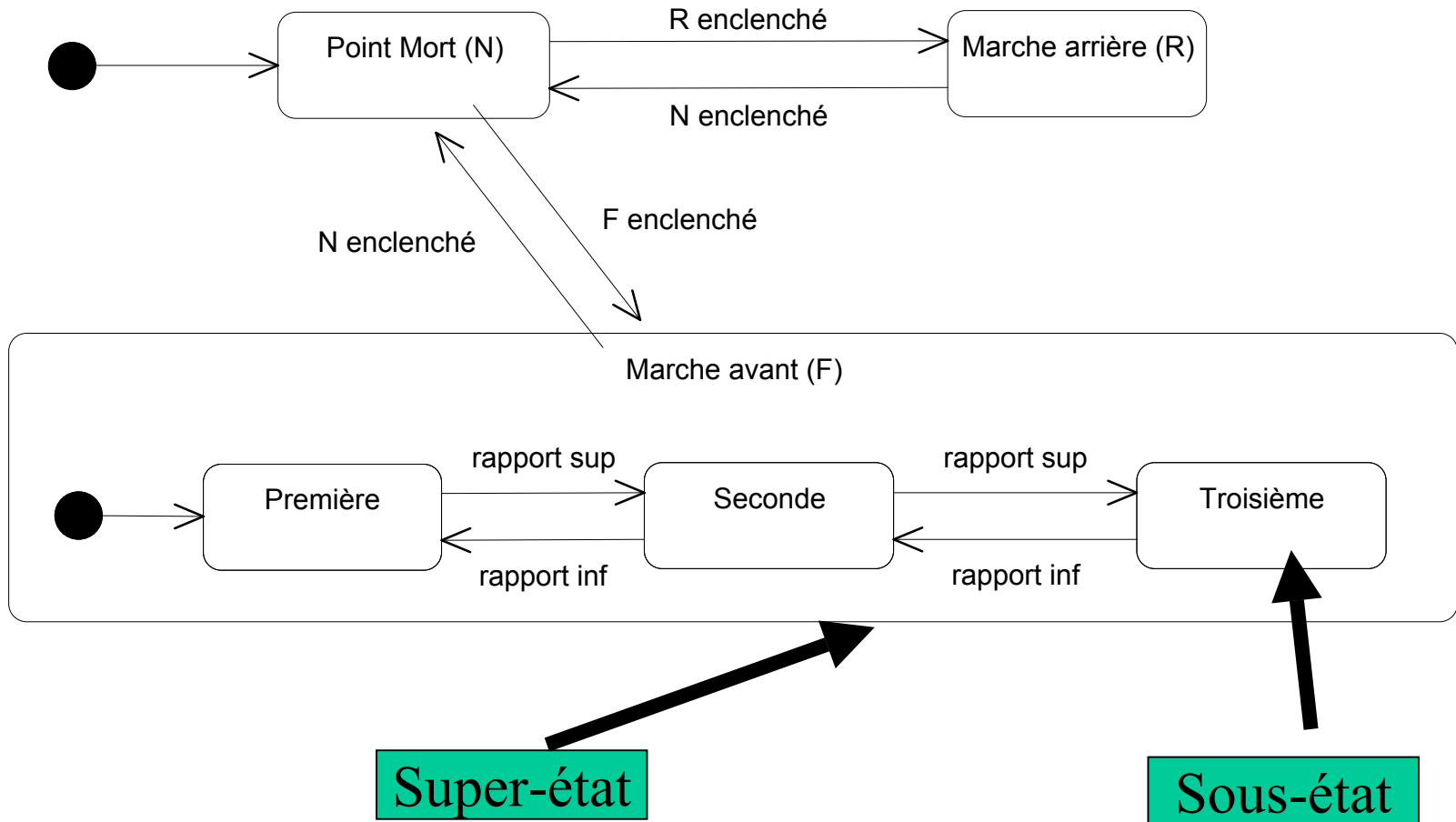
Généralisation d'états

- Dans le cas d'un comportement dynamique **complexe**, les diagrammes d'états sur **un** niveau deviennent rapidement **illisibles**
- Pour éviter ce problème, il est nécessaire de **structurer** les diagrammes d'états en:
 - **super-états** : états généraux
 - **sous-états** : héritent des caractéristiques des états généraux



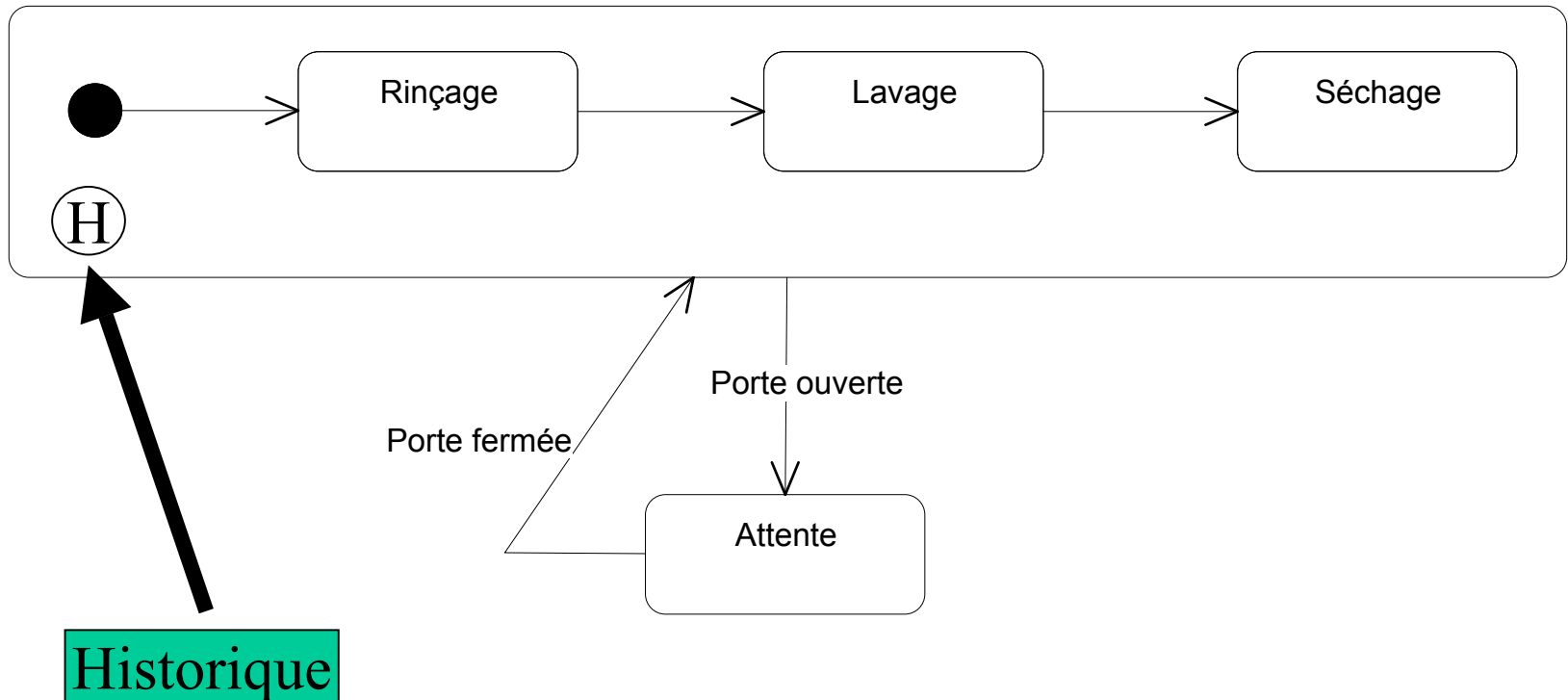
Notation Complète

- Exemple : transmission d'une automobile

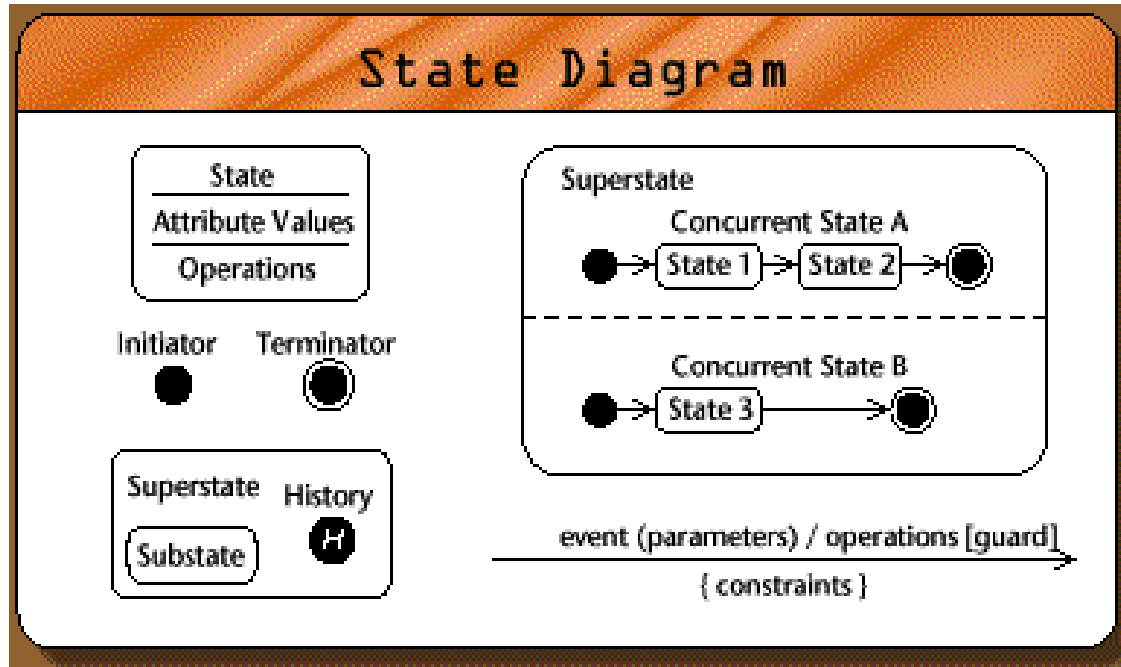


Historique

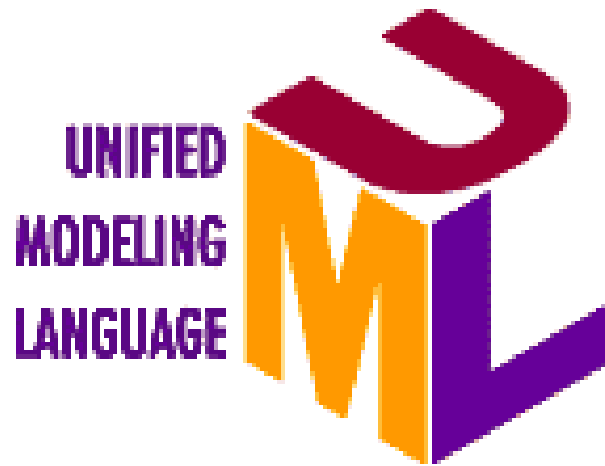
- Par défaut, un automate n'a **pas de mémoire**
- La notation **H** offre un **mécanisme pour mémoriser** le dernier sous-état qui l'englobe
- **Exemple** : cycle de lavage d'un lave vaisselle



Notation



Références



Livres

- « *The Unified Modeling Language User Guide* », **G. Booch, J. Rumbaugh, I. Jacobson**, 1999, Addison Wesley
- « *Object-Oriented Modeling and Design* », **J. Rumbaugh**, 1991, Prentice-Hall
- « *Object Solution* », **G. Booch**, 1996, Addison-Wesley
- « *Object-Oriented Software Engineering : A Use Case Driven Approach* », **I. Jacobson**, 1992, Addison-Wesley
- « *Modélisation Objet avec UML* », **P. A. Muller**, 1997, Eyrolles
- « *UML Distilled* », **M. Fowler**, 1997, Addison-Wesley
- « *UML La notation unifiée de modélisation objet* », **M. Lai**, Masson
- « *Designing Object Systems : Object-Oriented Modeling with Syntropy* », **S. Cook, J. Daniels**, 1994, Prentice-Hall

Articles

- « *Getting started : using use case to capture requirements* », **J. Rumbaugh**, Sept 1994, JOOP
- « *Formalizing use-case modeling* », **I. Jacobson**, Juin 1995, JOOP
- « *OMT : The object model* », **J. Rumbaugh**, Jan 1995, JOOP
- « *A search values : Attributes and associations* », **J. Rumbaugh**, Juin 1996, JOOP
- « *A matter : How to define subclasses* », **J. Rumbaugh**
- « *The life of an object model : How the object model changes during development* », **J. Rumbaugh**, Mars 1994, JOOP
- « *Statecharts : a visual Formalism for Complex Systems* », **D. Harel**, 1987, Science of Computer Programming vol 8
- « *Executable Object Modeling with Statecharts* », **D. Harel**, Juillet 1997, Computer
- « *OMT : The dynamic model* », **J. Rumbaugh**, Fev 1995, JOOP

Sur le Web

- Rational Software / UML Resource Software
 - <http://www.rational.com/uml>
- OMG
 - <http://www.omg.org>
- Groupe UML France
 - <http://www.essaim.univ-mulhouse.fr/uml/index.html>
- Carrefour Cetus : Orienté Objet
 - <http://www.csioo.com/cetusfr/software.html>
- Using Use Cases (Tutorial)
 - <http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/use>
- Object Constraint Language
 - <http://www.software.ibm.com/ad/ocl>
- David Harel 's Home Page
 - <http://www.wisdom.weizmann.ac.il/people/homepages/harel>
- I-Logix (société de Harel) : produit Rhapsody
 - <http://www.ilogix.com>