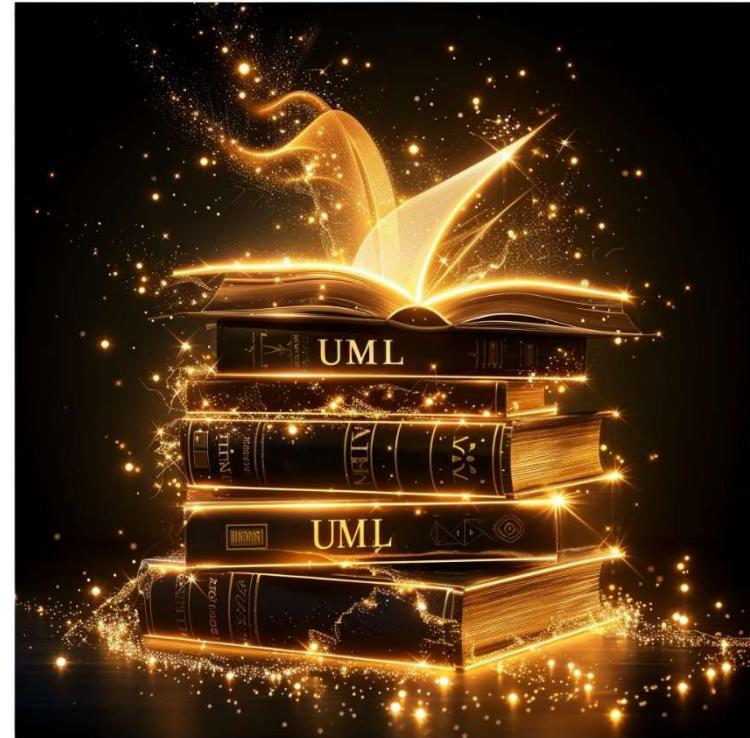


# Programmation – Concepts avancées

- Théorie



## Quelques outils

- <https://online.visual-paradigm.com/drive/>
- <https://yuml.me/diagram/scruffy/class/draw>

OMT  
*(Object Modeling  
Technique - 1991)*  
de James Rumbaugh

## Historique

OOSE  
de Ivar Jacobson

Notation  
de Grady Booch

## UML - *Unified Modeling Language*

Standard de modélisation objet  
adopté en 1997 par l'*Object Management Group* (OMG)

- Révision des spécifications initiales en 2001 – UML 1
- Approbation de la version **UML 2.0** en 2004
- Depuis 2017 : **UML 2.5.1**

- Rappel du cours de POO

- **Orienté objet** : organisation d'un logiciel sous la forme d'une collection d'objets indépendants incorporant structure de données et comportement
- **Objet**
  - Entité discrète et distinguable, concrète ou abstraite
  - Ex. Ce *slide*; l'*enseignant Mr. Desmet*; la *stratégie pédagogique de cette mise à niveau*
  - Identité intrinsèque - **Identifiant unique**
  - *Deux objets sont distincts, même s'ils ont des valeurs d'attributs identiques*

- **Classification** : regroupement des objets ayant même structure de données (**attributs**) et même comportement (**opérations**)

Ex. *SlideDeCours; Enseignant*

- **Classe** : abstraction décrivant un ensemble d'objets potentiellement infini

### Objets Bulles et Légendes



### Classe *BulleEtLégende*

#### Attributs

forme

couleurDeTrait

couleurDeRemplissage

#### Opérations

dessiner

effacer

déplacer

- **Instance d'une classe** : objet appartenant à la classe
- **Héritage** : partage de **propriétés** entre classes sur la base d'une relation hiérarchique
  - **Super-classe** (classe mère)
  - **Sous-classe** (classe fille) : spécialisation de la super-classe
- **Polymorphisme** : possibilité de comportements différents d'une même opération dans différentes classes
  - **Opération** : action exécutée par un objet ou transformation subie par un objet
  - **Méthode** : implémentation d'une opération par une classe

*Plusieurs méthodes pour une même opération*

*Une méthode par classe pour une opération donnée*

## Vocabulaire de méthodologie POO

- **Spécification initiale** : collaboration entre les analystes métier et les utilisateurs pour la genèse de l'application
- **Analyse**
  - Étude et re-formulation des besoins : collaboration avec le client pour comprendre le problème
  - **Modèle d'analyse** : abstraction concise et précise de l'objectif du système à développer (pas de la façon de le construire)
    - **Modèle de domaine** : description des objets du monde réel manipulés par le système
    - **Modèle de l'application** : parties du système visibles par l'utilisateur
- **Conception** : stratégie de haut niveau (**architecture du système**) pour résoudre le problème posé
  - Établissement des stratégies de conception générales
  - Allocations prévisionnelles des ressources
- **Conception des classes** : concentration sur les structures de données et algorithmes de chaque classe
- **Implémentation et Tests**

# Grands Thèmes de la POO

- **Abstraction**

Concentration sur ce que représente un objet et sur son comportement avant de décider de la façon de l'implémenter

- **Encapsulation**

Masquage de l'information : séparation des aspects externes d'un objet accessibles aux autres objets, des détails de l'implémentation cachés aux autres objets

- **Regroupement des données et du comportement**

**Polymorphisme** → transfert de la décision de quelle méthode utiliser à la hiérarchie de classes

- **Partage**

**Héritage** → possibilité de partager des portions de code communes et clarté conceptuelle (mise en évidence de traitement commun)

- **Mise en évidence de la nature intrinsèque des objets** : sur ce qu'est un objet et non sur la façon dont il est utilisé



## Chap 1: A la découverte de l'objet

- Les bases de l'UML
  - Esperanto
  - Dessins
  - Histoire
- But et Polyvalence de l'UML
  - Pas juste le plan
  - Quoi mais aussi Comment !
  - Outil de communication avec des techniciens ou autres

## Pourquoi voir l'UML

- UML et POO sont « IN LOVE »
  - Un stylo à besoin de papier
  - Concepts POO sont en UML visuel
  - Conception facilité
  - Compréhension
  - Communication de système complexe

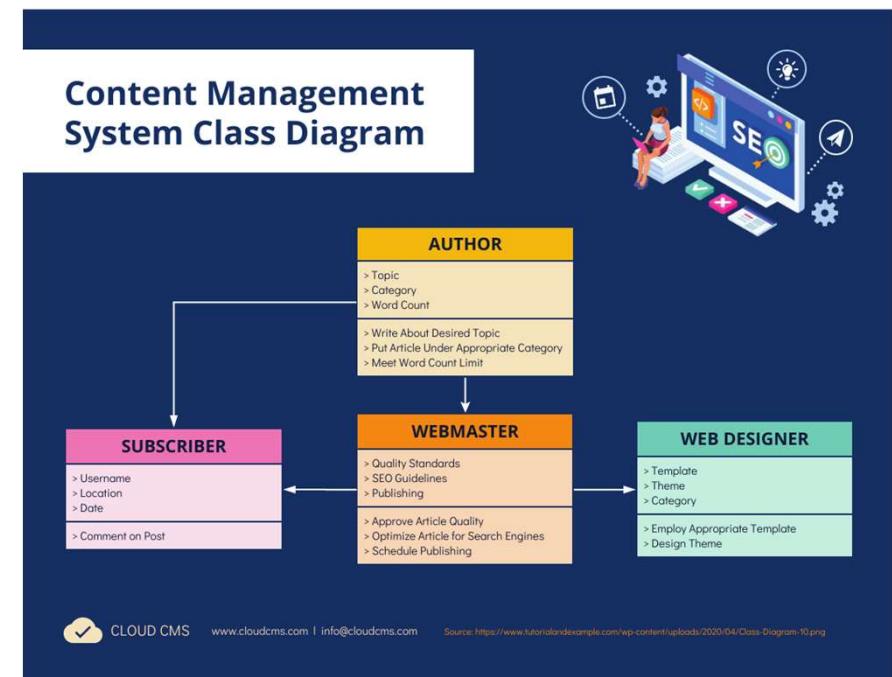
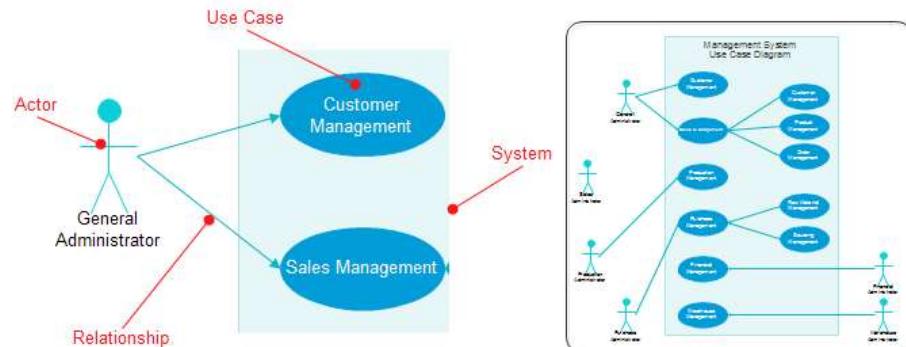


## Uml qui grandit

- Il évolue de la version 1 à la version 2.5
  - Plus de richesse
  - Plus d'expression
  - Plus facile de raconter son histoire



- Création de diagrammes
  - Etre un artiste
  - Manipulation de diagrammes



## Allons un peu plus loin

- Plus qu'un Langage, une Méthodologie
  - Holistique
    - Structure et Comportements
- Les fondements de l'UML :
  - Modélisation basé sur les objets
  - Abstraction et Encapsulation

## Le processus de développement

- De l'idée à la réalisation
- Un langage Universel
- Adaptabilité
- Extensions des stéréotypes

## Soyons critiques

- Complexité
  - Très vite complexe si mal réfléchis
  - Beaucoup, beaucoup, beaucoup de possibilités
- Mise à Jour
  - S'adapte tout le temps
  - Le monde va trop vite



CRITIQUE

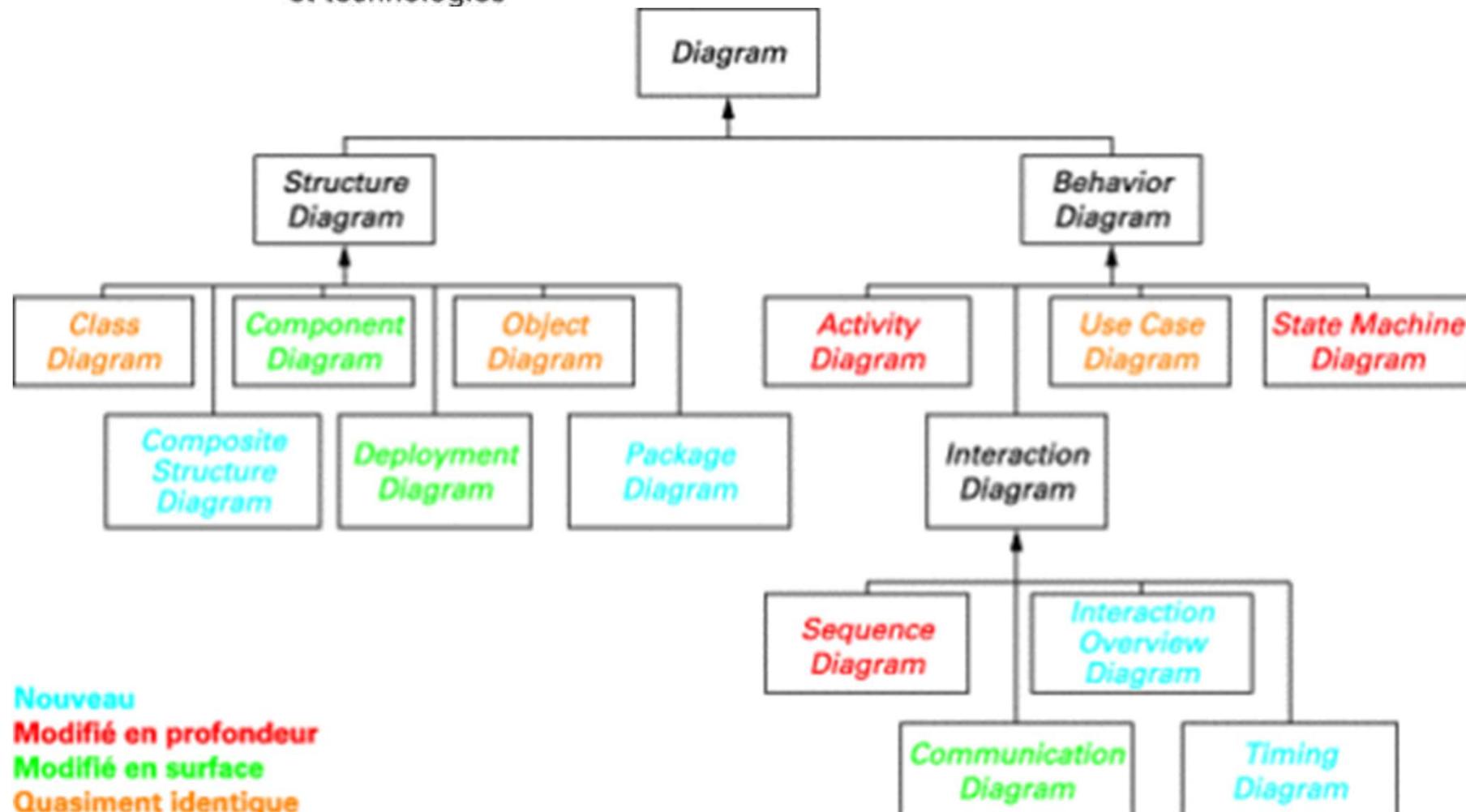
## Les diagrammes

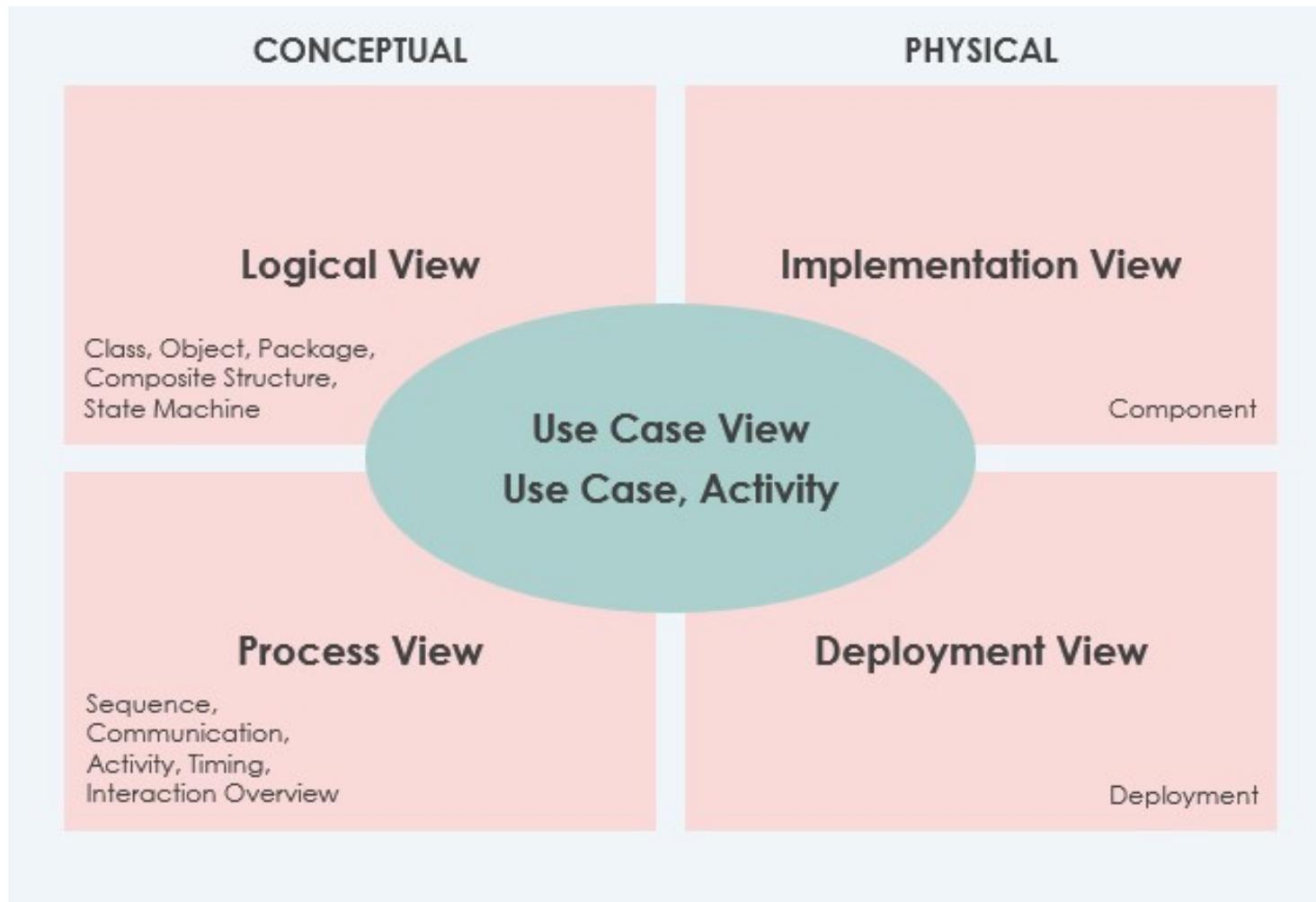
- **Diagramme de Classe** : Représente les classes et leurs relations, soulignant la structure et la conception du système.
- **Diagramme d'Objet** : Montre des instances de classes à un moment donné, illustrant la situation réelle de l'application.
- **Diagramme de Paquet** : Groupe les éléments associés, tels que les classes, dans des paquets, facilitant l'organisation du modèle.

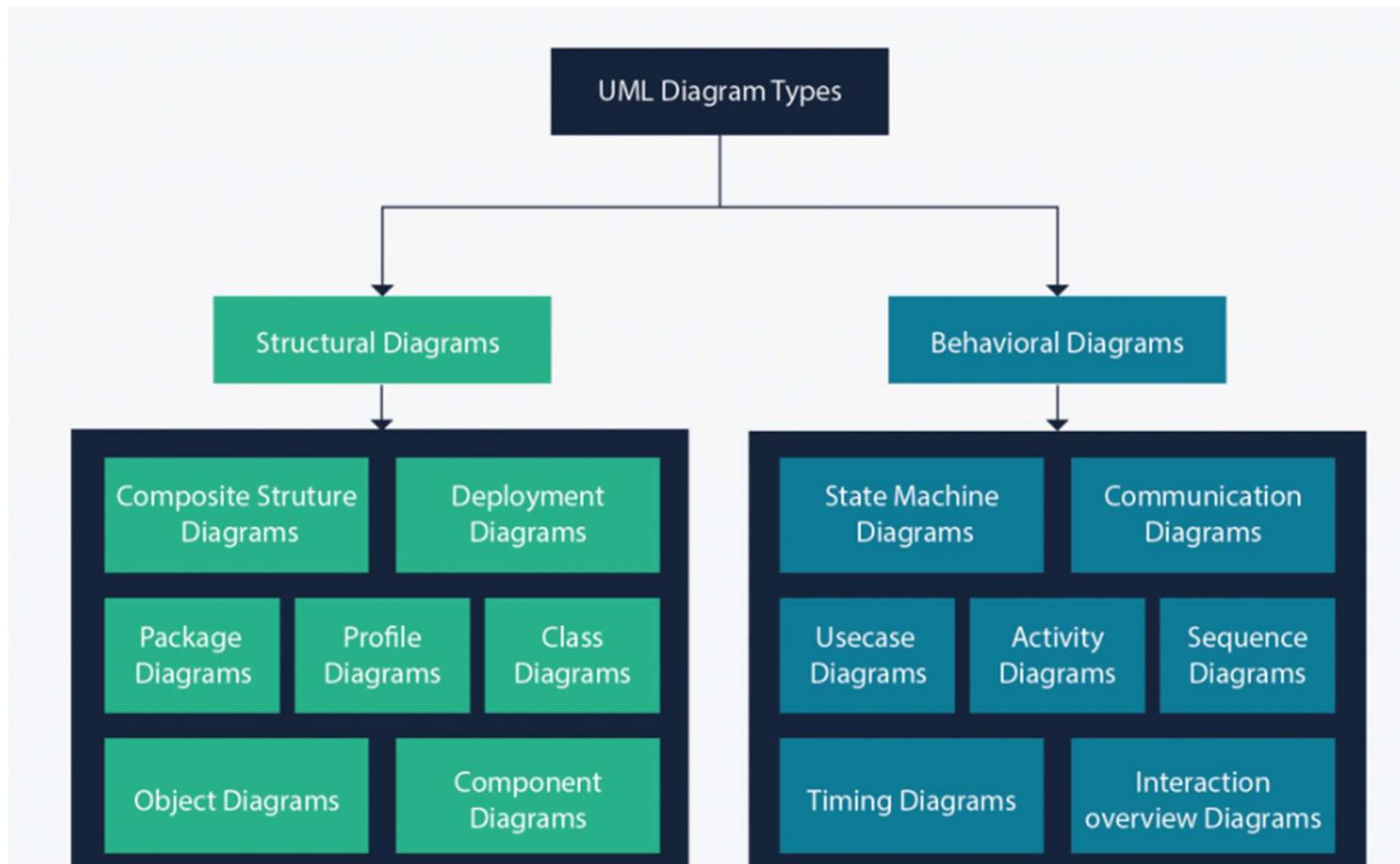
- **Diagramme de Composant** : Décrit les composants physiques ou logiciels et leurs interactions, utile pour la modélisation de l'architecture logicielle.
- **Diagramme de Déploiement** : Illustre la configuration physique des artefacts logiciels (logiciels, matériels) dans le système.
- **Diagramme de Cas d'Utilisation** : Présente les fonctionnalités du système et leurs interactions avec les utilisateurs ou autres systèmes.

- **Diagramme d'Activité** : Décrit le flux de travail ou les activités, souvent utilisé pour modéliser les processus d'affaires.
- **Diagramme d'État** : Représente les états d'un objet et les transitions entre ces états, idéal pour les systèmes dynamiques.
- **Diagramme de Séquence** : Montre comment les objets interagissent dans une séquence temporelle, mettant en évidence la collaboration entre objets.

- **Diagramme de Communication** : Similaire au diagramme de séquence mais se concentre sur la relation et l'interaction entre les objets.
- **Diagramme de Vue d'Ensemble des Interactions** : Offre une vue d'ensemble des interactions, combinant des diagrammes de séquence et d'activité.
- **Diagramme de Timing** : Spécialisé pour montrer les interactions des objets dans le temps, utile pour modéliser les systèmes temps réel.







## Un peu de vocabulaire

- **Système**
- **Modèle**
- **UML**
- **OMG**
- **Structurels ou comportements**

- **Système**

- est composé de différentes parties ayant des relations. Le tout est constitué de la sorte pour atteindre un but bien précis. Un système peut par exemple être un logiciel, du matériel, un processus de traitement de l'information, une entreprise, un processus métier, etc.

## Un peu de vocabulaire

- **Système**
- **Modèle**
- **UML**
- **OMG**
- **Structurels ou comportements**

- Modèle

- peut représenter un système à n'importe quel niveau d'abstraction, allant de l'architecture à l'implémentation technologique. Le modèle possède une sémantique qui permet de le manipuler et l'exploiter, cette dernière est liée au langage de modélisation. Les modèles peuvent aussi se présenter sous forme d'architectures types, à plusieurs niveaux qui permettent de guider la phase d'analyse : modèle en couche ou vertical par exemple. Le modèle d'un système représente le point de vue du développeur pendant la phase d'analyse du projet et à ce titre, il n'existe pas vraiment de modèle unique, mais plutôt un ensemble de modèles qui convergent vers une solution. La notion de modèle est tout à fait subjective.

## Un peu de vocabulaire

- **Système**
- **Modèle**
- **UML**
- **OMG**
- **Structurels ou comportements**

- Les diagrammes de structure sont statiques, ils permettent de définir l'architecture du système, c'est en quelque sorte le plan d'un logiciel ou d'un système. Ces diagrammes ne décrivent ni le comportement ni le fonctionnement dynamique du système dans le temps.
- Les diagrammes de comportement sont dynamiques. Ils décrivent le comportement du système et son fonctionnement dans le temps.

Analyse des processus métiers



Architecture fonctionnelle



Architecture applicative



Architecture technique

## Modélisation et génération de code

- Analyse du processus métiers et de l'architecture fonctionnelle
- Depuis classe ou diagramme de classe
- Outils de GL

## Cycle de vie d'un projet

- **Analyse des besoins et faisabilité**
- **Spécifications ou conception générale**
- **Conception détaillée**
- **Codage (Implémentation ou programmation)**
- **Tests unitaires**

- Intégration
- Qualification (ou recette)
- Documentation
- Mise en production
- Maintenance



## Diagramme de cas d'utilisation

- Compréhension du besoin client
- Limiter l'afflux d'informations
- Dialogue entre tous
- Fonctionnalités

## Définition simple

Un diagramme de cas d'utilisation est un type de diagramme comportemental en UML qui représente les fonctionnalités d'un système du point de vue des utilisateurs. Il illustre comment les utilisateurs extérieurs (acteurs) interagissent avec le système pour réaliser un objectif.



## Mais c'est quoi donc ?

- Représentent les fonctions ou les services fournis par le système.
- Chaque cas d'utilisation est une séquence d'actions que le système effectue pour fournir un résultat utile à un acteur.
- Habituellement décrits par un verbe et un nom (par exemple, "Gérer les commandes").

## 1<sup>er</sup> étape de la modélisation

- Définition des périmètres du système
- En ressortir les apports extérieurs : ACTEUR
- Acteur = Type stéréotypé qui représente un rôle joué par une personne ou une chose qui interagit avec le système

Une même personne physique = plusieurs rôles



## Descriptif d'un acteur

- Représentent les utilisateurs ou les entités externes qui interagissent avec le système.
- Peuvent être des personnes, d'autres systèmes, ou des entités organisationnelles.
- Illustrés par des figures humaines ou des icônes représentant des systèmes externes.

## Catégorie d'acteurs

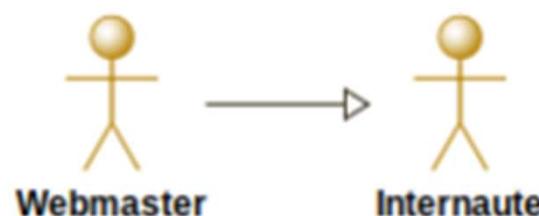
- **les acteurs principaux** : comme le nom de cette catégorie l'indique, ce sont les personnes qui vont employer les fonctions centrales ou principales du système ;
- **les acteurs secondaires** : ce sont les personnes qui effectuent des tâches dites secondaires : tâches de maintenance, tâches administratives, etc. ;
- **le matériel externe** : il s'agit des dispositifs matériels qui doivent être utilisés. Ils font partie intégrante du domaine de l'application, et en ce sens, sont incontournables ;
- **les autres systèmes** : les systèmes avec lesquels il y a interaction du système courant.

- Un acteur est représenté par un petit bonhomme stylisé dont le nom est indiqué en dessous, comme le montre la figure 1.

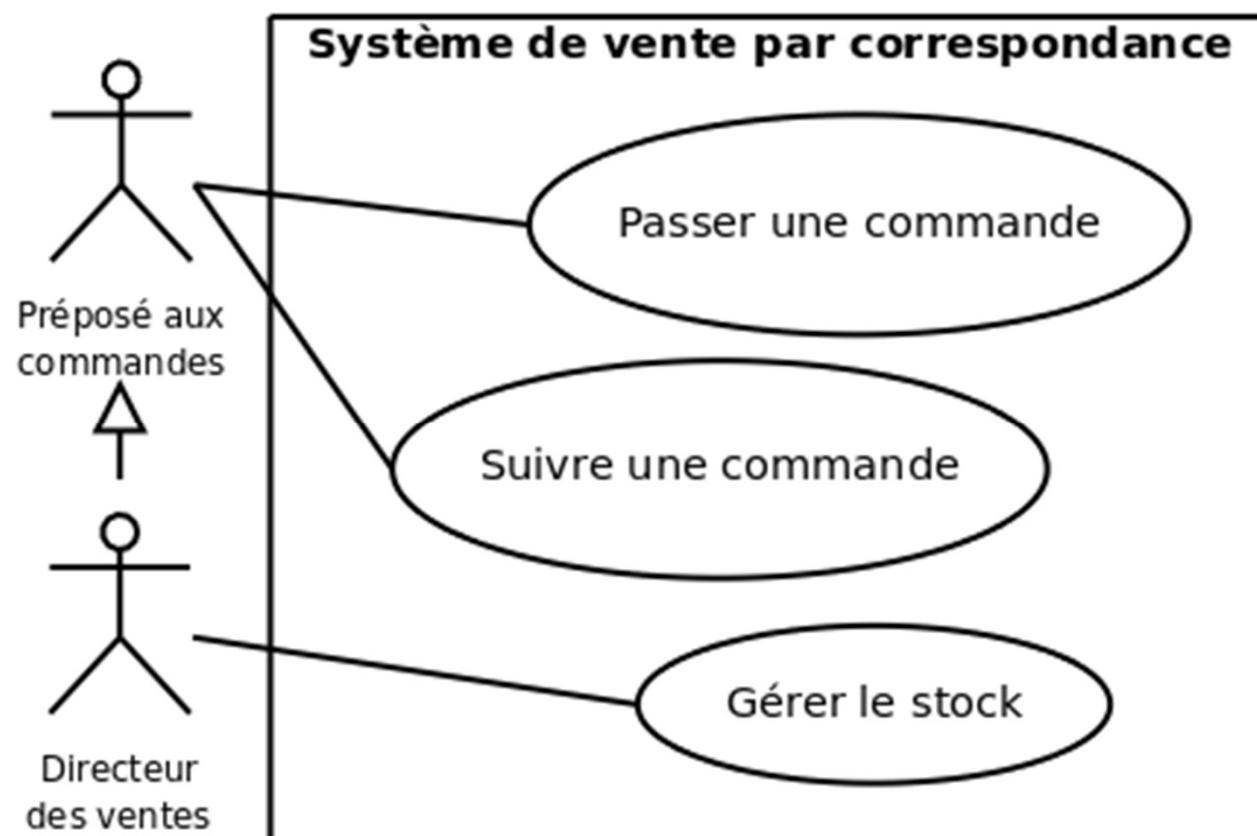


*Fig. 1 : Représentation d'un acteur (Modelio).*

- Généralisation (spécialisation) : Héritage

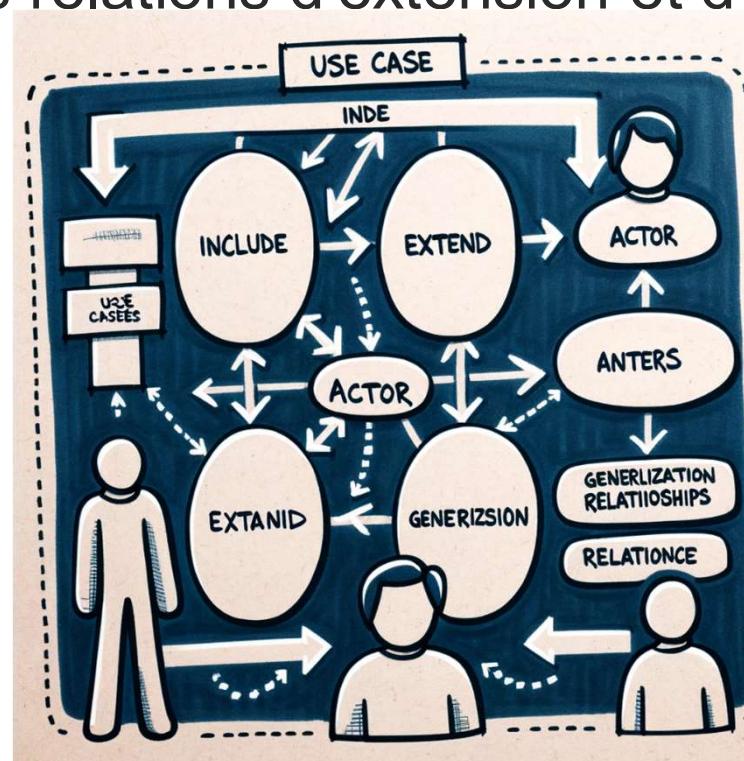


*Fig. 2 : Notion d'héritage.*



## Les liaisons

- Les lignes reliant les acteurs aux cas d'utilisation indiquent qu'ils sont impliqués ou intéressés par ces fonctionnalités.
- Peuvent inclure des relations de généralisation entre acteurs ou des relations d'extension et d'inclusion entre cas d'utilisation.



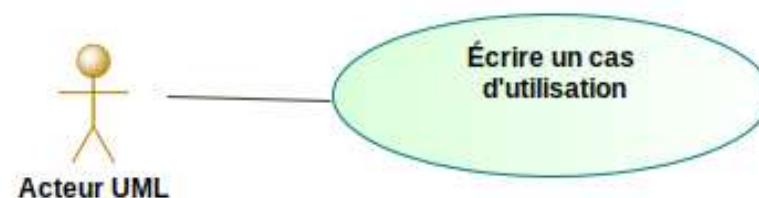
## Cas d'utilisation

- Une unité cohérente représentant une fonctionnalité visible de l'extérieur. Il réalise un service de bout en bout, avec un déclenchement, un déroulement et une fin, pour l'acteur qui l'initie. Un cas d'utilisation modélise donc un service rendu par le système, sans imposer le mode de réalisation de ce service.



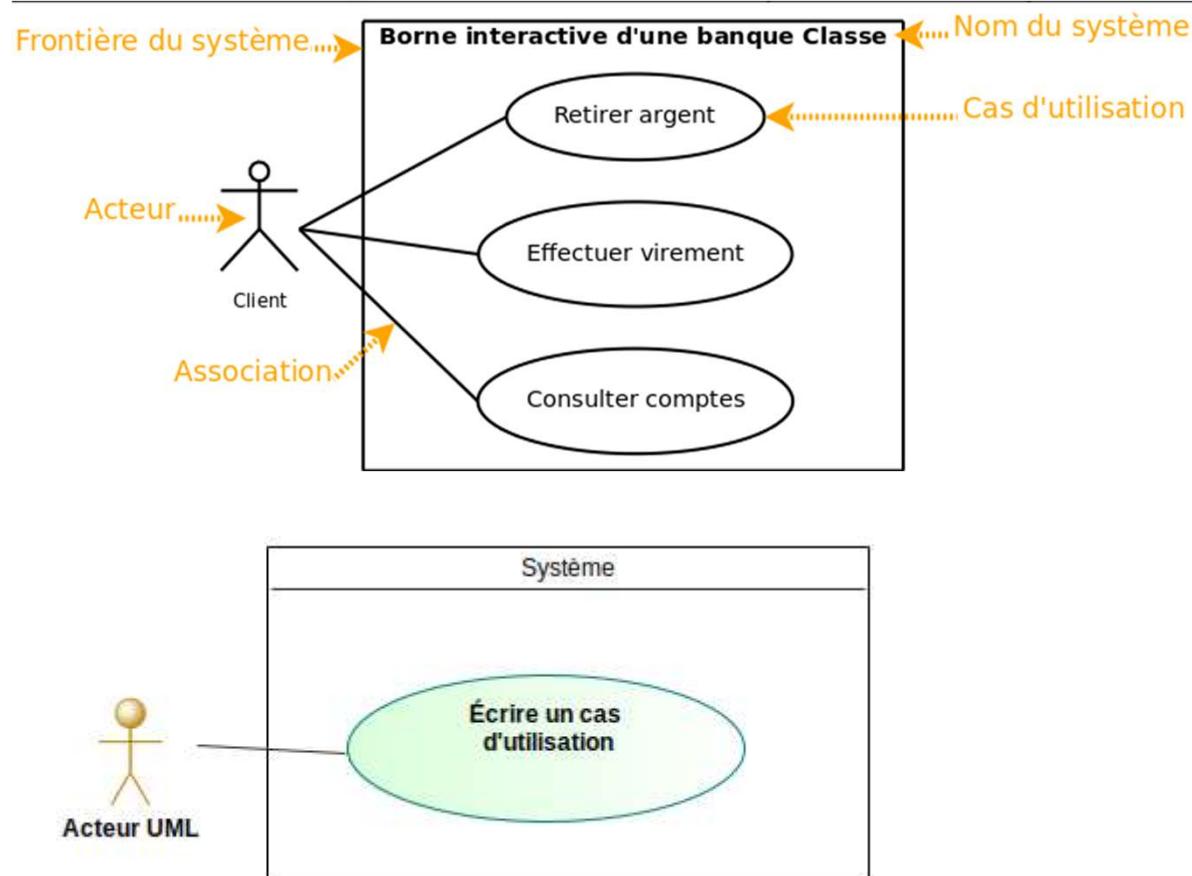
*Fig. 3 : Représentation d'un cas d'utilisation.*

## Liaison entre acteur et cas



# Le système

- Frontière
- Nom
- Ca(S)
- Acteur(s)
- Liaison(s)



## Relation : Multiplicité

- Le cas d' cardinalité chevaux acheteur haras na instance des chevaux à l'extrême



1..\*

Acheteur



1

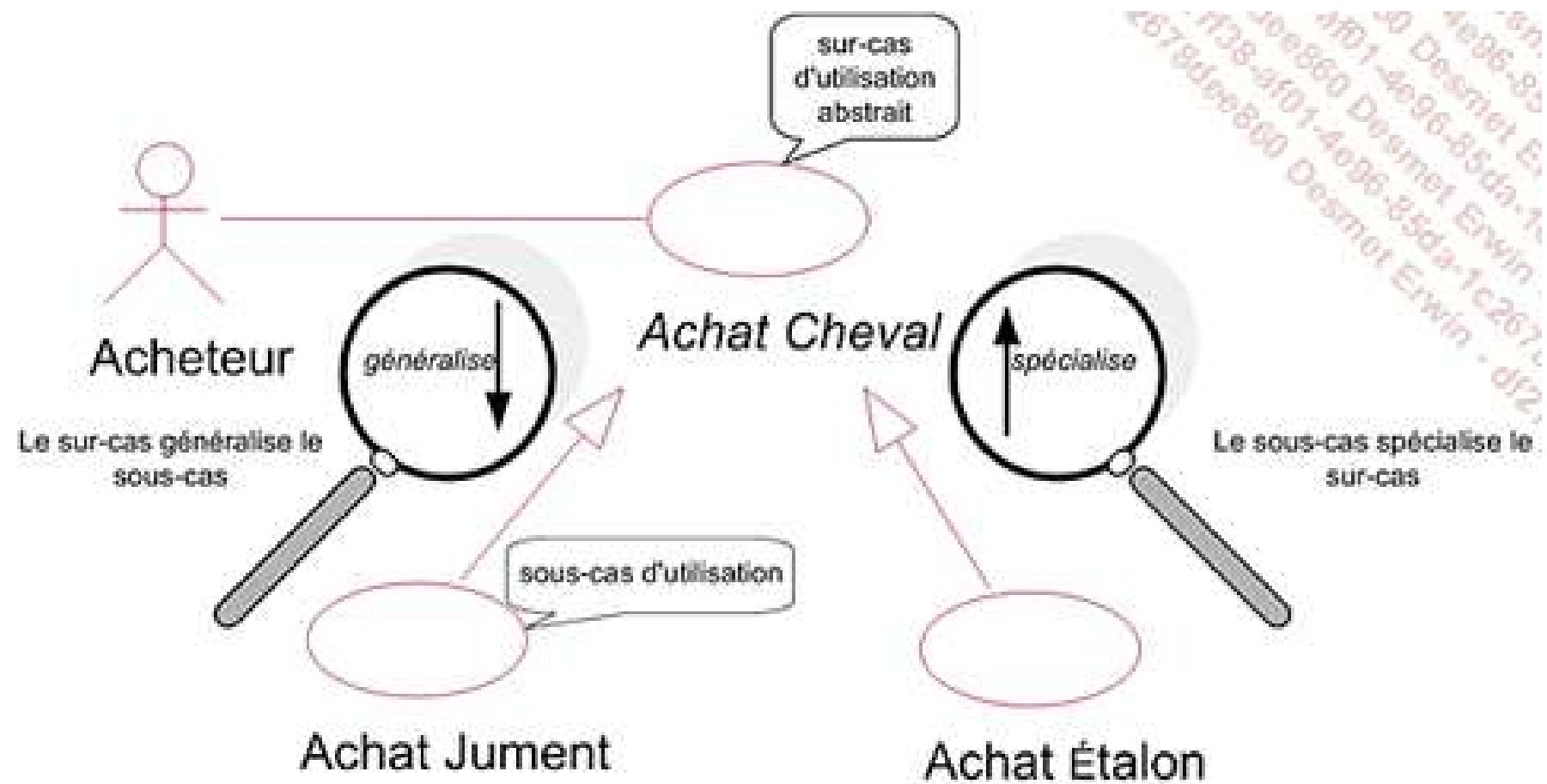
Haras Nationaux

s urs res par les vente

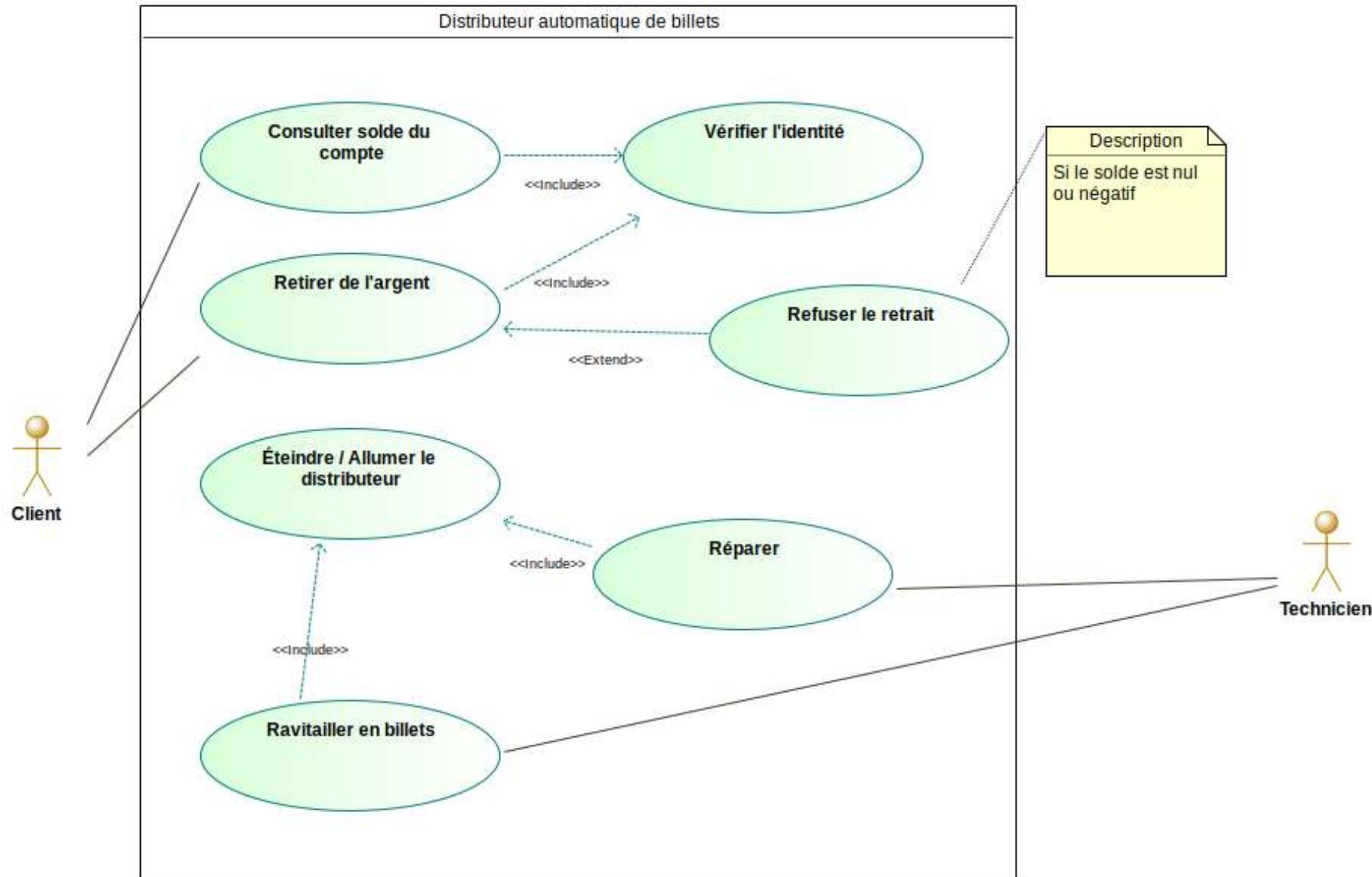
# Modélisation des exigences

Spécification	Cardinalités
0 .. 1	zéro ou une fois
1	une et une seule fois
*	de zéro à plusieurs fois
1 .. *	de une à plusieurs fois
M .. N	entre M et N fois
N	N fois

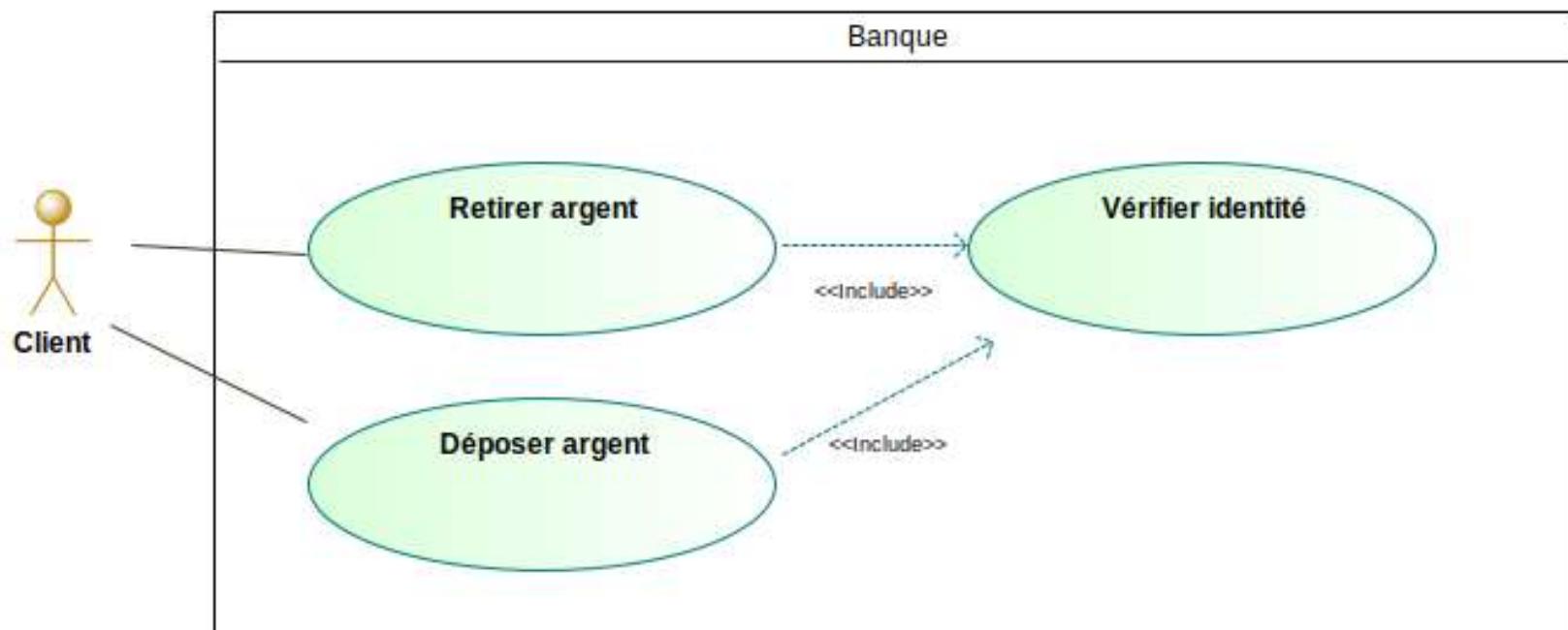
# Modélisation des exigences



# Exemple : Borne d'une banque

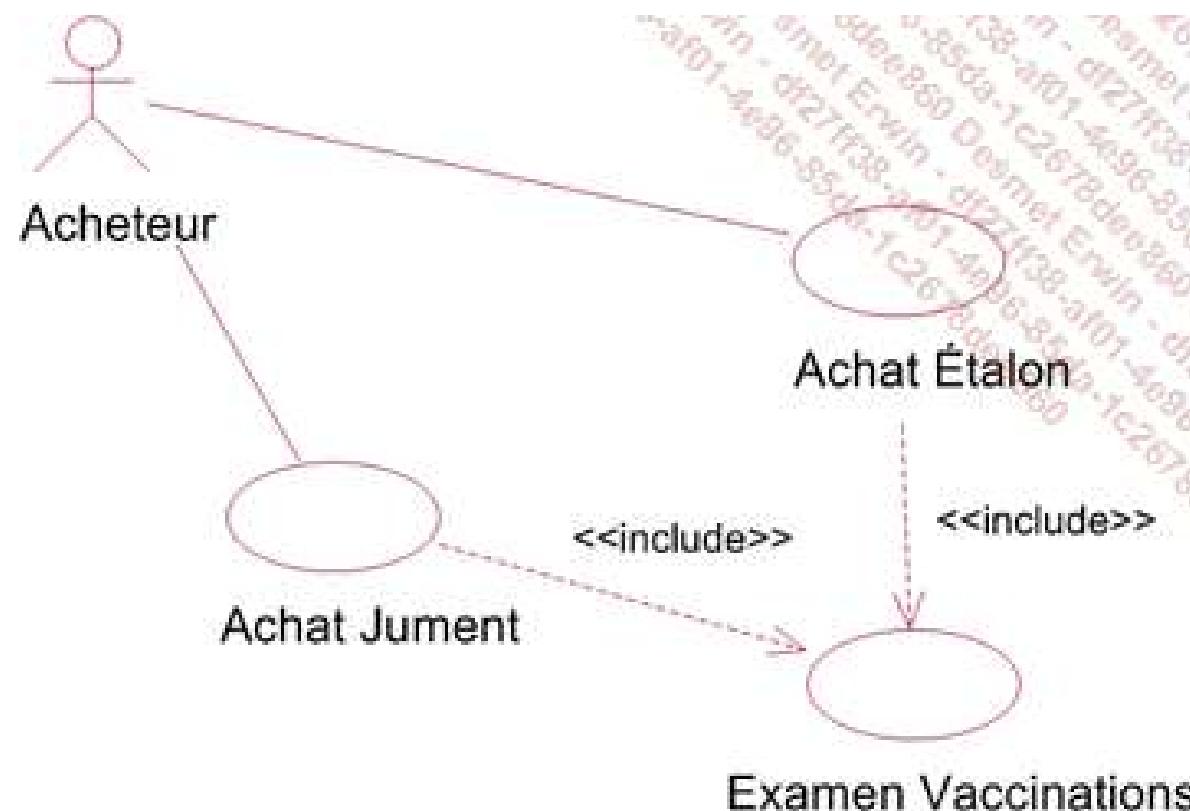


## Relation particulière



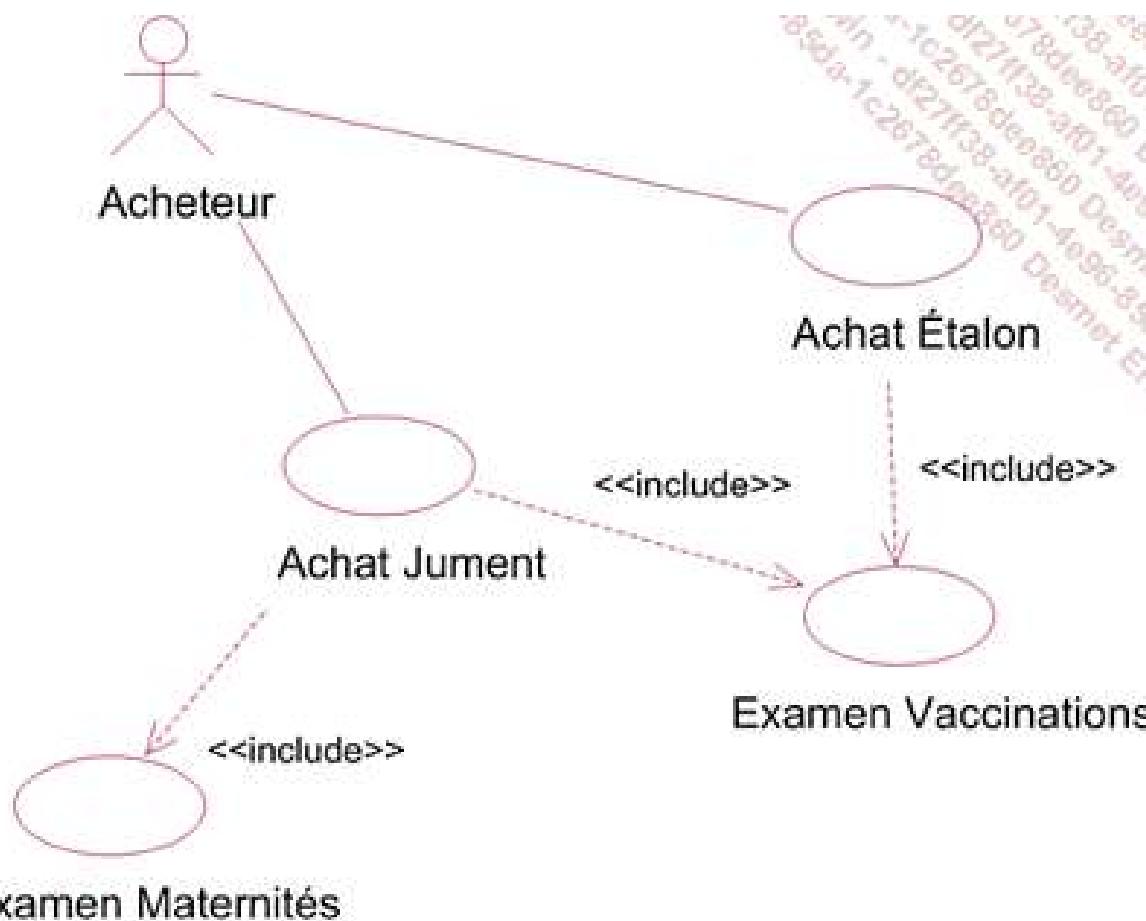
## Modélisation des exigences : relations entre les cas d'utilisation

- Exemple avec mise en commun du sous cas

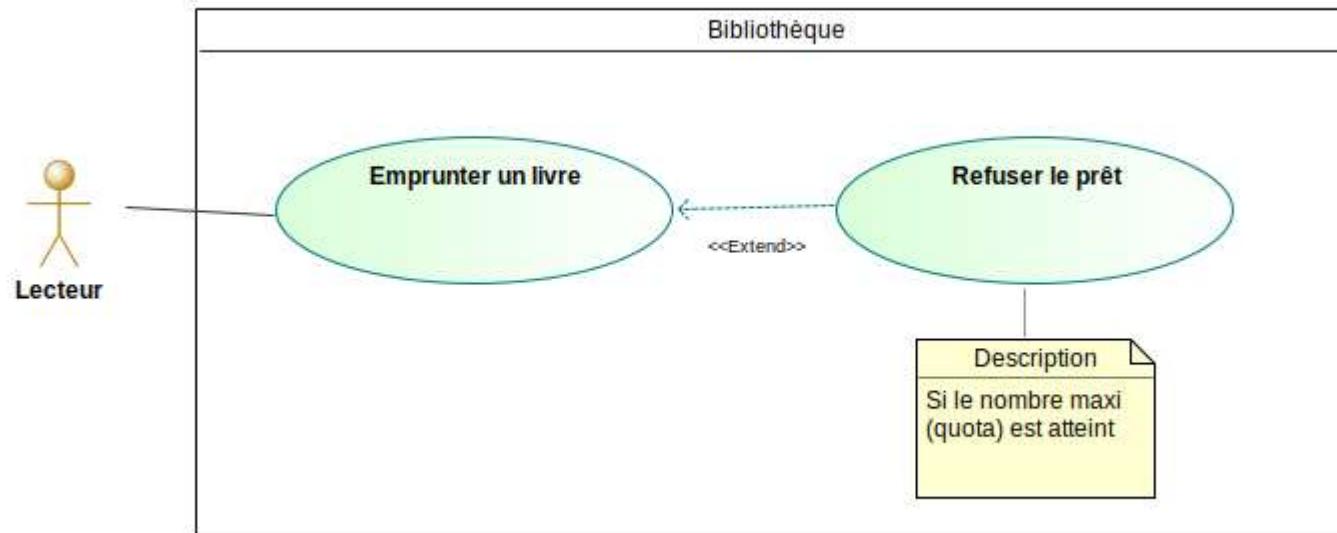


## Modélisation des exigences : relations entre les cas d'utilisation

- Sous cas-non partagé, on parle de décomposition



## Relation particulières



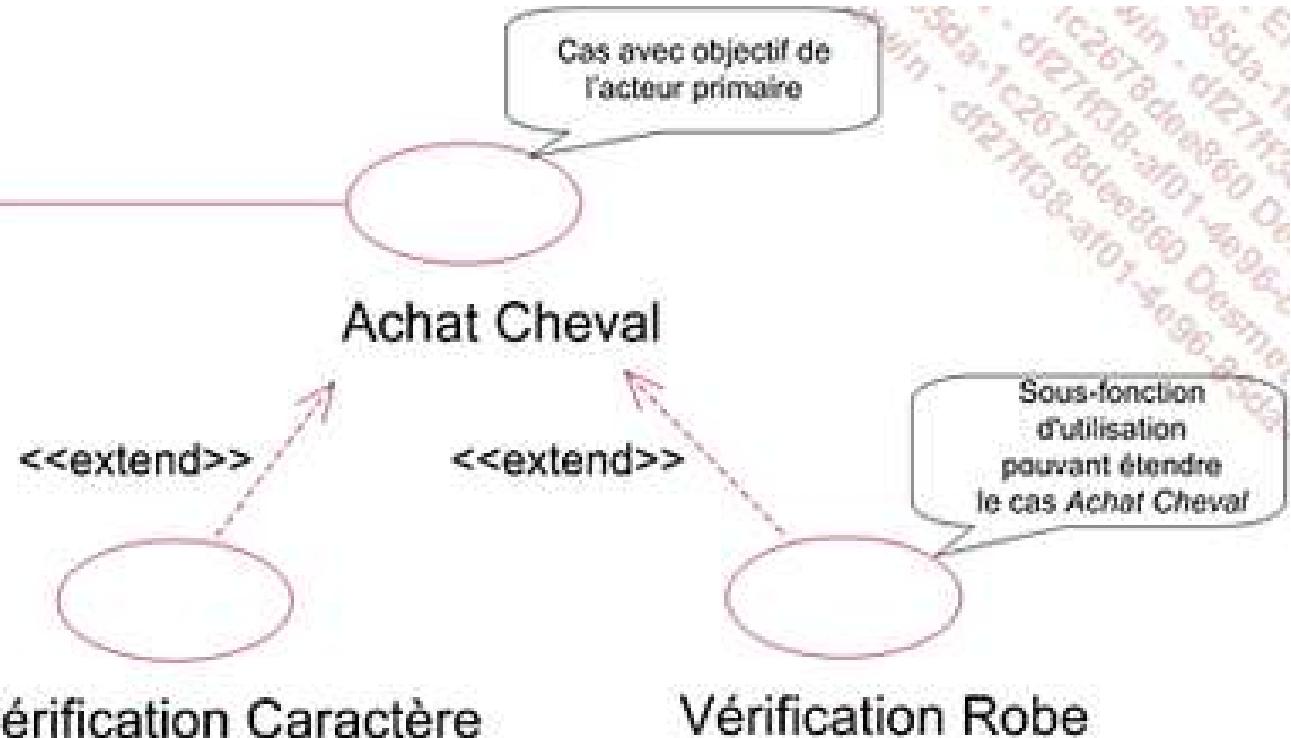
## Modélisation des exigences : relations entre les cas d'utilisation

### L'extension

— · · ·

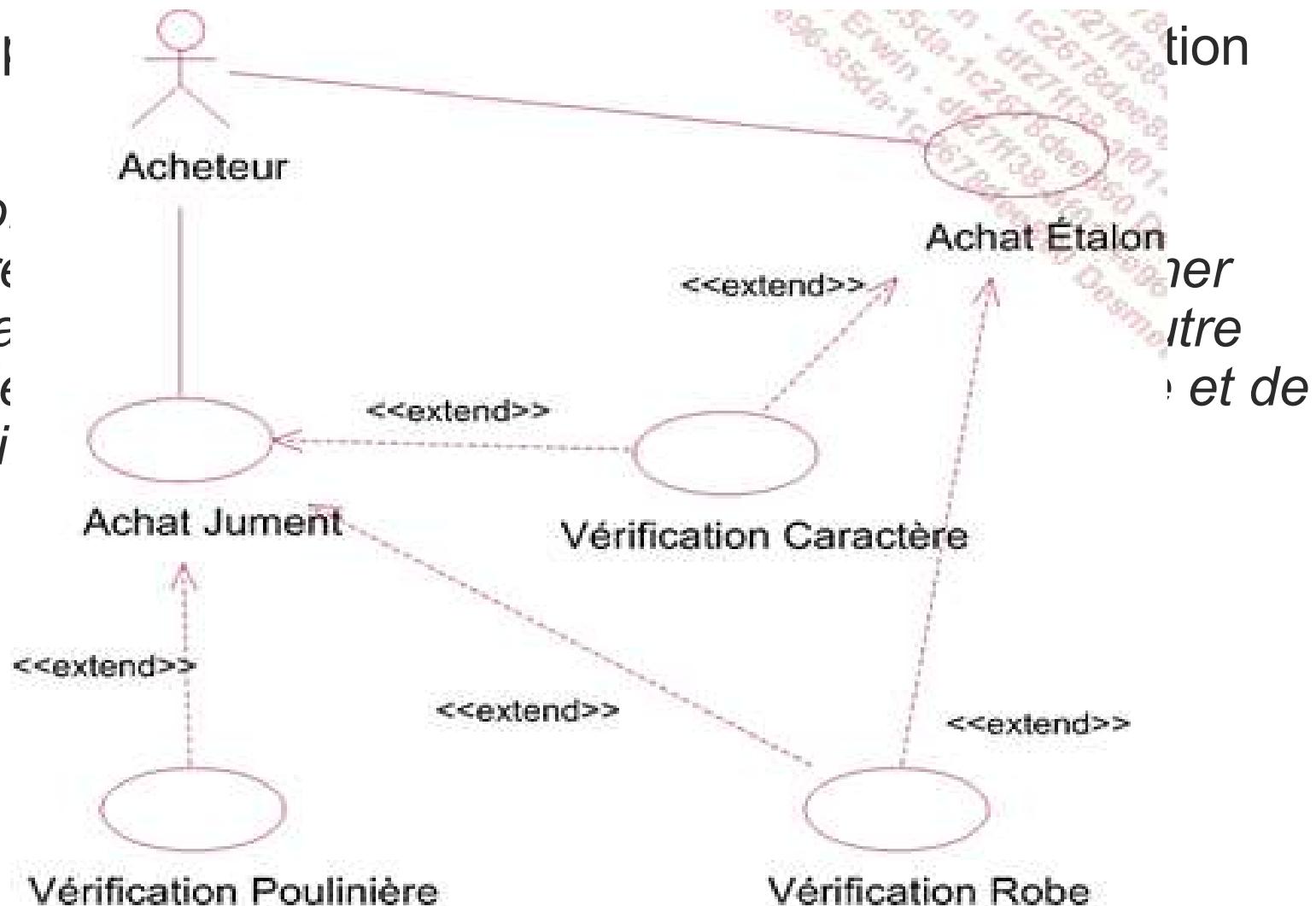


Acheteur



# Modélisation des exigences : relations entre les cas d'utilisation

- Exemples
  - *Preno sépare naissance part, le la vérité*



## Comment faire un exercice ?

- **Identifier les Acteurs:** Déterminez qui utilisera le système ou sera affecté par lui.
- **Définir les Cas d'Utilisation:** Listez les principales fonctionnalités que le système doit fournir aux acteurs.
- **Établir les Relations:** Connectez les acteurs aux cas d'utilisation appropriés et définissez les relations entre les cas d'utilisation.
- **Vérifier et Valider:** Assurez-vous que le diagramme couvre tous les aspects importants des interactions du système avec ses utilisateurs.

## Conseils pour la Réalisation

- **Simplicité:** Gardez le diagramme simple et compréhensible. Évitez les détails superflus.
- **Focalisation sur l'Utilisateur:** Pensez du point de vue de l'utilisateur pour identifier les cas d'utilisation pertinents.
- **Collaboration:** Impliquez les parties prenantes pour valider et affiner le diagramme.

## Comment identifier un acteur ?

- Qui utilise le système ?
- Qui installe le système ?
- Qui démarre le système ?
- Qui entretient le système ?
- Qui éteint le système ?
- Quels autres systèmes utilisent ce système ?
- Qui obtient des informations de ce système ?
- Qui fournit des informations au système ?
- Est-ce que quelque chose se passe automatiquement à l'heure actuelle ?

## Comment identifier un cas ?

- Quelles fonctionnalités l'acteur voudra-t-il du système ?
- Le système stocke-t-il des informations ?
- Quels acteurs créeront, liront, mettront à jour ou supprimeront ces informations ?
- Le système doit-il notifier un acteur des changements dans l'état interne ?
- Y a-t-il des événements externes que le système doit connaître ?
- Quel acteur informe le système de ces événements ?

## Tips

- Toujours structurer et organiser le diagramme de cas d'utilisation du point de vue des acteurs.
- Les cas d'utilisation devraient commencer de manière simple et avec la vue la plus élevée possible. Seulement alors peuvent-ils être affinés et détaillés davantage.
- Les diagrammes de cas d'utilisation sont basés sur la fonctionnalité et doivent donc se concentrer sur le "quoi" et non sur le "comment".