Introduction à UML

Modélisation Orientée Objet de Systèmes Logiciels

Alae El Alami

Ecole supérieure de technologie Meknès UMI

Licence Informatique –

Plan

Matériel et logiciel

- Systèmes informatiques :
  - **80 % de logiciel** ;
  - 20 % de matériel.
- Depuis quelques années, la fabrication du matériel est assurée par quelques fabricants seulement.
  - Le matériel est relativement fiable.
  - Le marché est standardisé.

Les problèmes liés à l'informatique sont essentiellement des problèmes de logiciel.

La « crise du logiciel »

- Étude sur 8 380 projets (Standish Group, 95) :
  - Succès : 16 % ;
  - Problématique : 53 % (budget ou délais non respectés, défaut de fonctionnalités) ;
  - Échec : 31 % (abandonné).

Le taux de succès décroît avec la taille des projets et la taille des entreprises.

- **Génie Logiciel** (Software Engineering) :
  - Comment faire des logiciels de qualité ?
  - Qu'attend-on d'un logiciel ? Quels sont les critères de qualité ?

Critères de qualité d'un logiciel

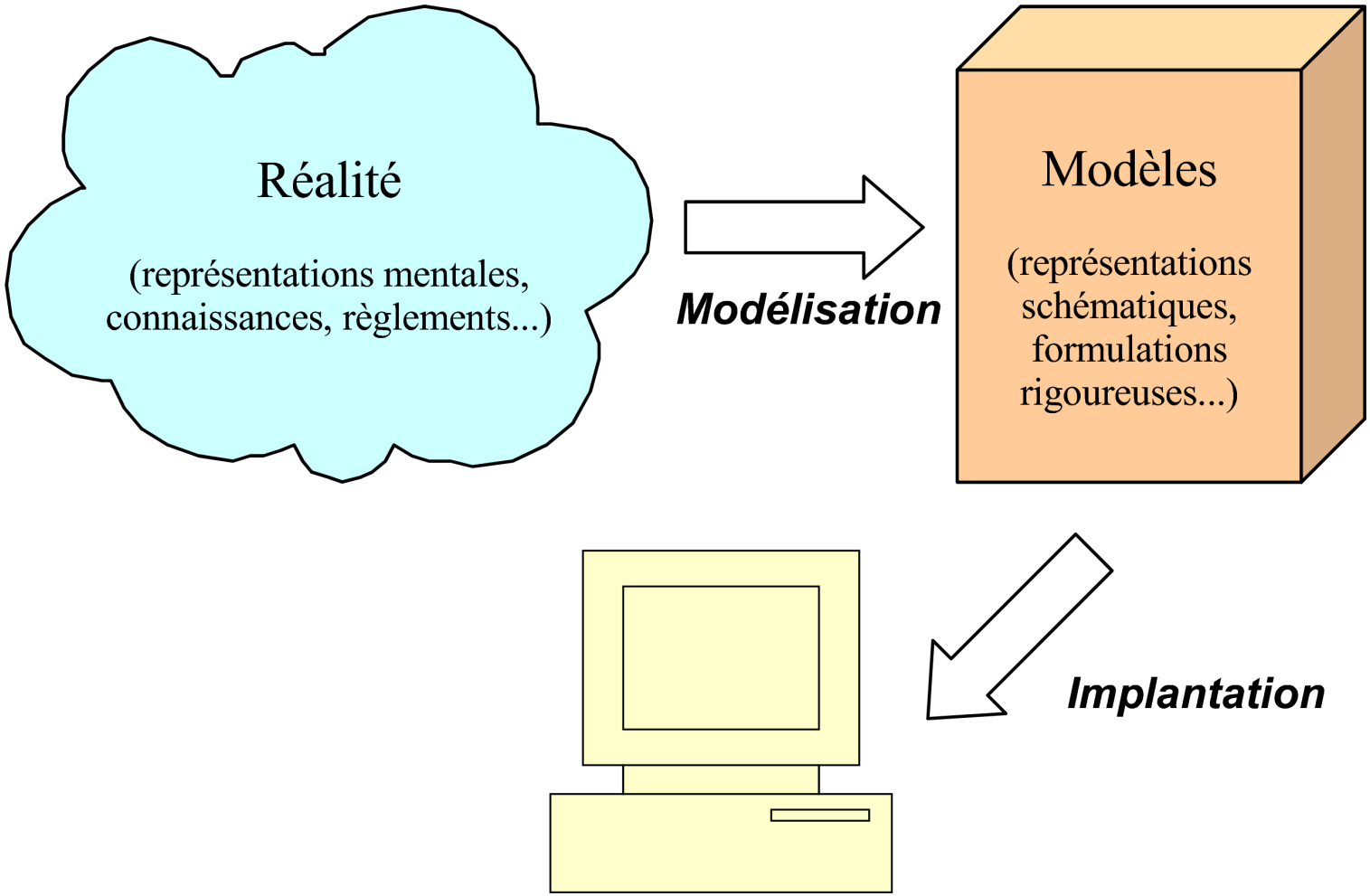
- **Utilité**
  - Adéquation entre le logiciel et les besoins des utilisateurs ;
- **Utilisabilité**
- **Fiabilité**
- **Interopérabilité**
  - Interactions avec d'autres logiciels ;
- **Performance**
- **Portabilité**
- **Réutilisabilité**
- **Facilité de maintenance**
  - Un logiciel ne s'use pas pourtant, la maintenance absorbe une très grosse partie des efforts de développement.

Etapes du développement

- **Étude de faisabilité**
- **Spécification**
  - Déterminer les fonctionnalités du logiciel.
- **Conception**
  - Déterminer la façon dont dont le logiciel fournit les différentes fonctionnalités recherchées.
- **Implantation**
- **Tests**
  - Essayer le logiciel sur des données d'exemple pour s'assurer qu'il fonctionne correctement.
- **Maintenance**

Le coup de la maintenance absorbe un grande partie du coût de développement.

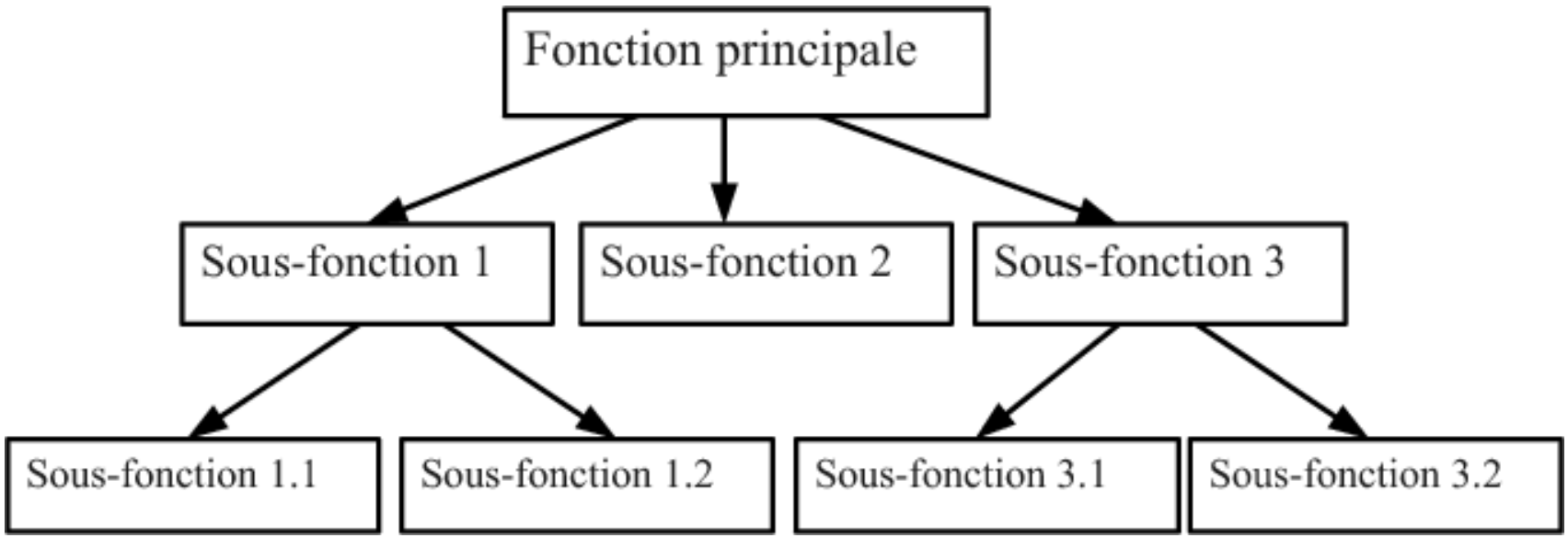
Modélisation



Modélisation par décomposition fonctionnelle

- Approche **descendante** :
  - Décomposer la fonction globale jusqu'à obtenir des fonctions simples à appréhender et donc à programmer.

C'est la **fonction** qui donne la **forme** du système.



Modélisation orientée objets

- La **Conception Orientée Objet** (COO) est la méthode qui conduit à des architectures logicielles fondées sur les objets du système, plutôt que sur une décomposition fonctionelle.

C'est la **structure** du système qui lui donne sa forme.

- On peut partir des objets du domaine (briques de base) et remonter vers le système global : **approche ascendante**.

Attention, l'approche objet n'est pas seulement ascendante.

Unified Modeling Language

- Au milieu des années 90, les auteurs de Booch, OOSE et OMT ont décidé de créer un langage de modélisation unifié avec pour objectifs :
  - Modéliser un système **des concepts à l'exécutable**, en utilisant les techniques orientée objet ;
  - **Réduire la complexité de la modélisation** ;
  - Utilisable par l'**homme comme la machine** :
    - Représentations graphiques mais disposant de qualités formelles suffisantes pour être **traduites automatiquement en code source** ;
    - Ces représentations ne disposent cependant pas de qualités formelles suffisantes pour justifier d'aussi bonnes propriétés mathématiquesque des langeges de spécification formelle (Z, VDM...).
- Officiellement UML est né en 1994.

UML v2.0 date de 2005. Il s'agit d'une **version majeure** apportant des innovations radicales et étendant largement le champ d'application d'UML.

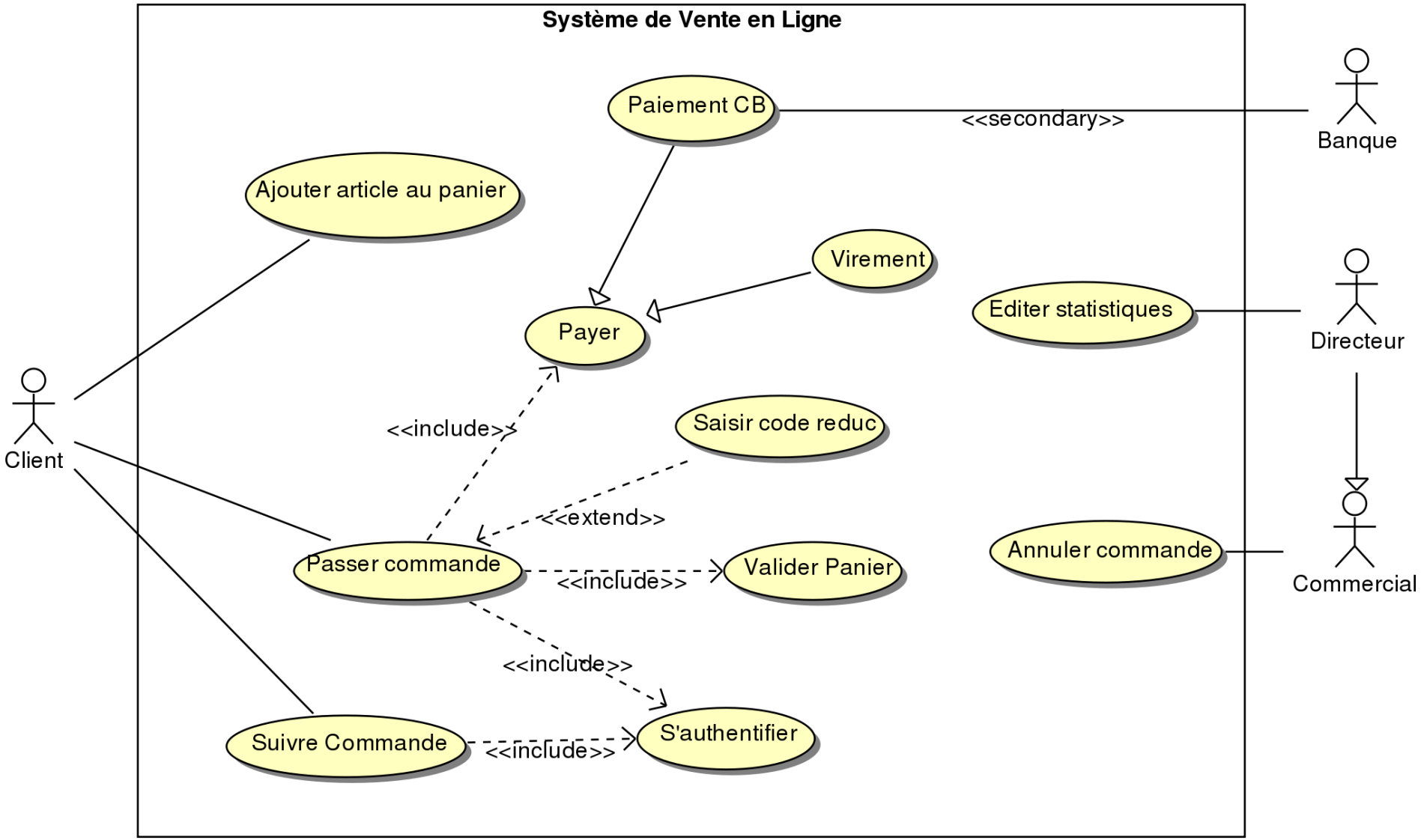
Plan

Modélisation des besoins

Avant de développer un système, il faut savoir **précisément** à *QUOI* il devra servir, *cad* à quels besoins il devra répondre.

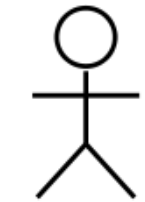
- **Modéliser les besoins** permet de :
  - Faire l'inventaire des fonctionnalités attendues ;
  - Organiser les besoins entre eux, de manière à faire apparaître des relations (réutilsations possibles, ...).
- Avec UML, on modélise les besoins au moyen de **diagrammes de cas d'utilisation**.

Exemple de diagramme de cas d'utilisation



Cas d'utilisation

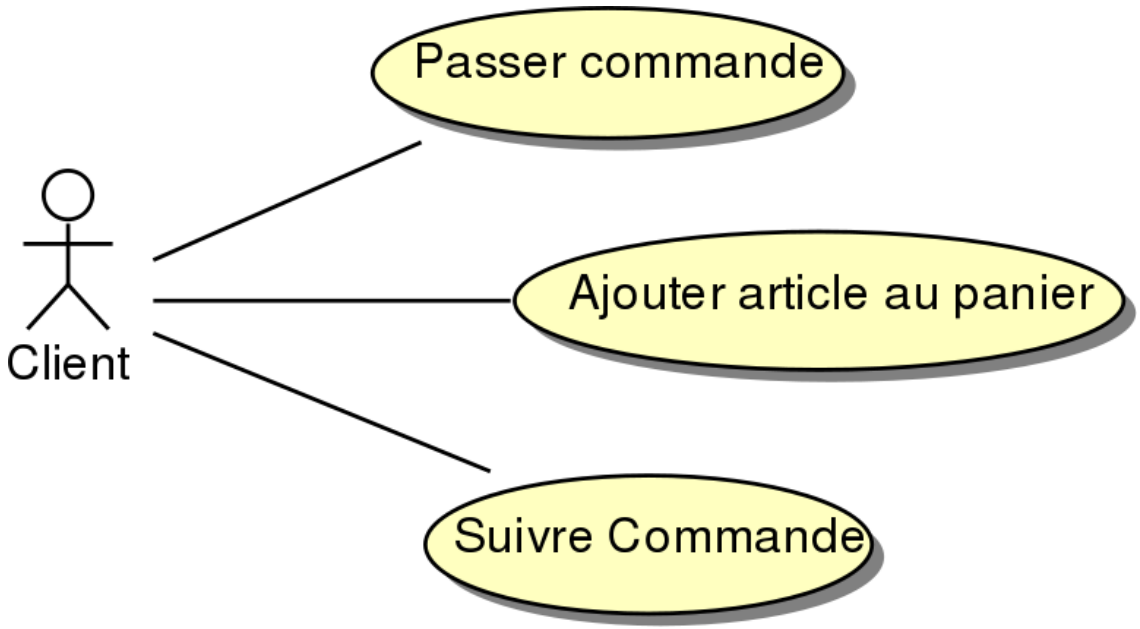
- Un **acteur** est une entité extérieure au système modélisé, et qui interagit directement avec lui.



- Un **cas d'utilisation** est un service rendu à un acteur, il nécessite une série d'actions plus élémentaires.

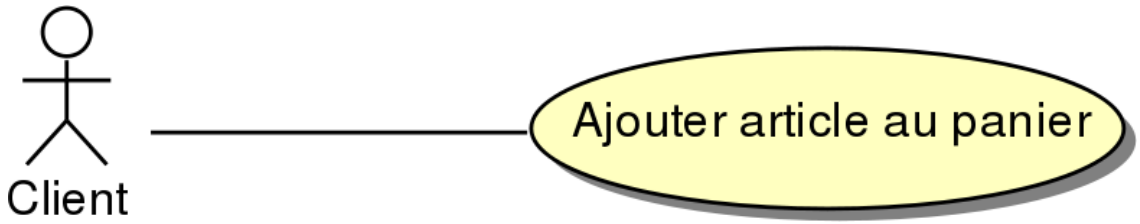
Un cas d'utilisation est l'expression d'un service réalisé de bout en bout, avec un déclenchement, un déroulement et une fin, pour l'acteur qui l'initie.

Acteurs et cas d'utilisation



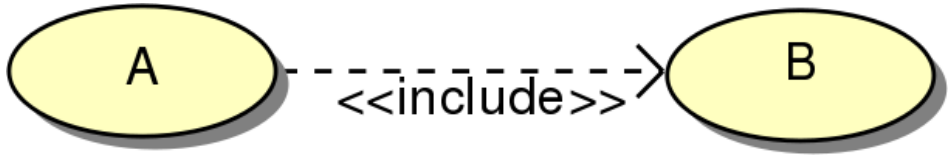
Relations entre cas d'utilisation en acteurs

- Les acteurs impliqués dans un cas d'utilisation lui sont liés par une **association**.
- Un acteur peut utiliser plusieurs fois le même cas d'utilisation.

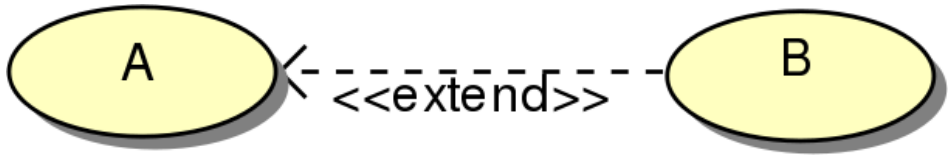


Relations entre cas d'utilisation

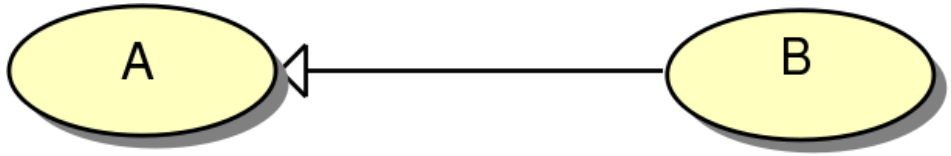
- **Inclusion** : le cas A inclut le cas B (B est une partie *obligatoire* de A).



- **Extension** : le cas B étend le cas A (B est une partie *optionelle* de A).



- **Généralisation** : le cas A est une généralisation du cas B (B est une sorte de A).



Dépendances d'inclusion et d'extension

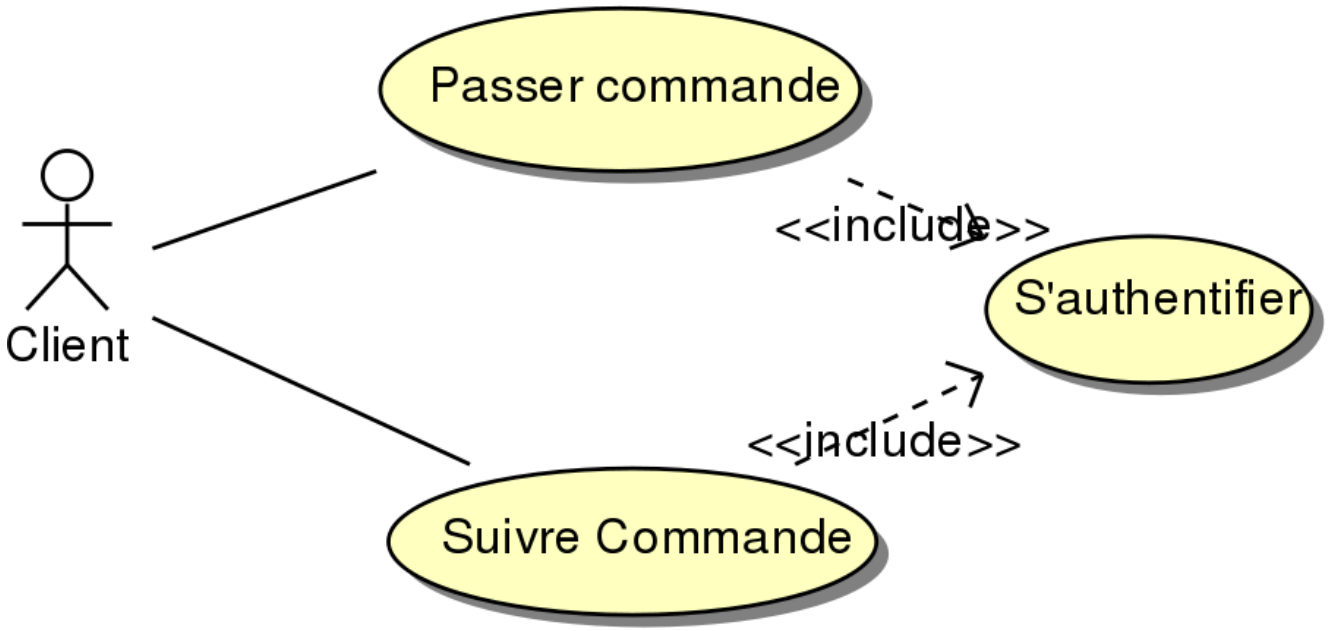
Attention

Le sens des flèches pointillées indique une dépendance, pas le sens de la relation d'inclusion.

- Les inclusions et les extensions sont toutes deux des **dépendances.**
  - Lorsqu'un cas B inclut un cas A, B dépend de A.
  - Lorsqu'un cas B étend un cas A, B dépend aussi de A.
  - On note toujours la dépendance par une flèche pointillée B $\dashrightarrow$ A qui se lit « B dépend de A ».
- Losqu'un élément B dépend d'un élément A, toute modification de A sera susceptible d'avoir un impact sur B.
- Les « incude » et les « extend » sont des **stéréotypes** (entre guillements) des relations de dépendance.

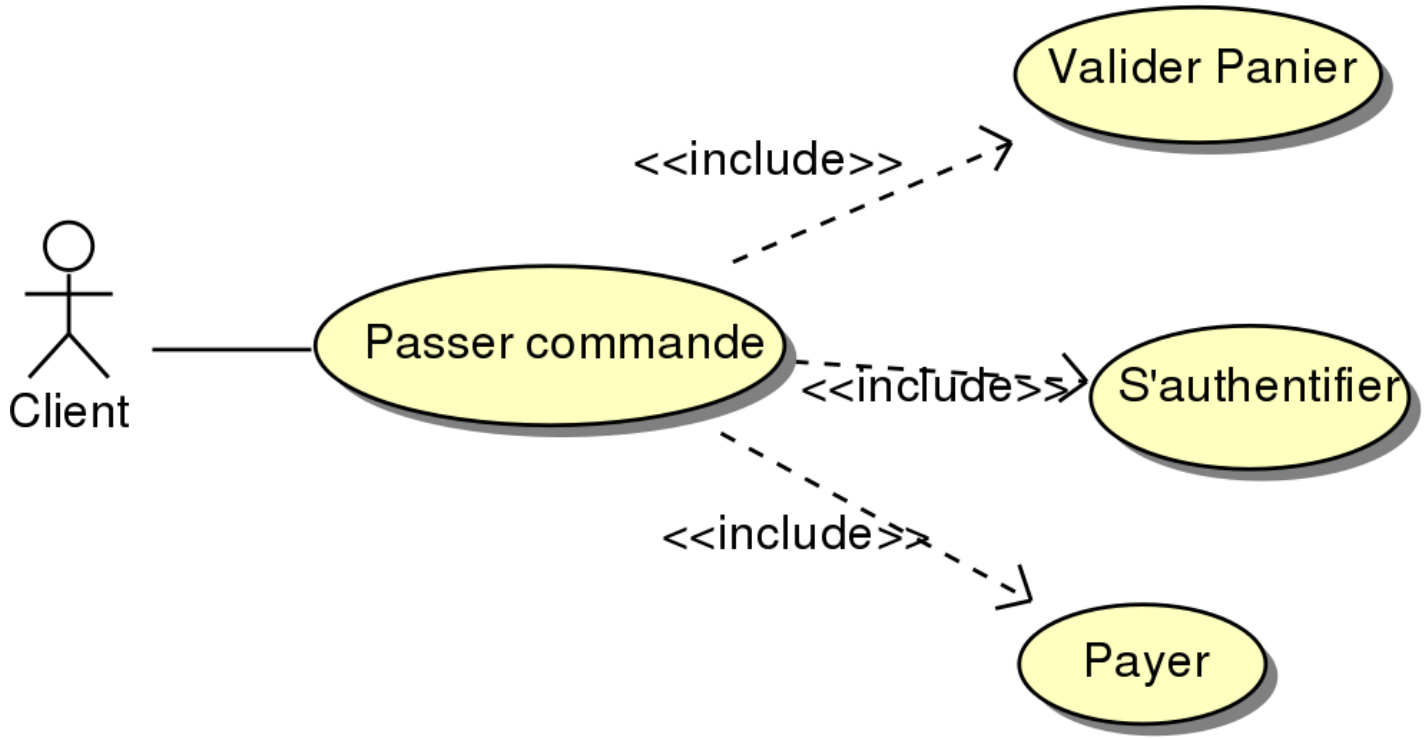
Réutilisabilité avec les inclusions et les extensions

- Les relations entre cas permettent la **réutilisabilité** du cas « s'authentifier » : il sera inutile de développer plusieurs fois un module d'authentification.

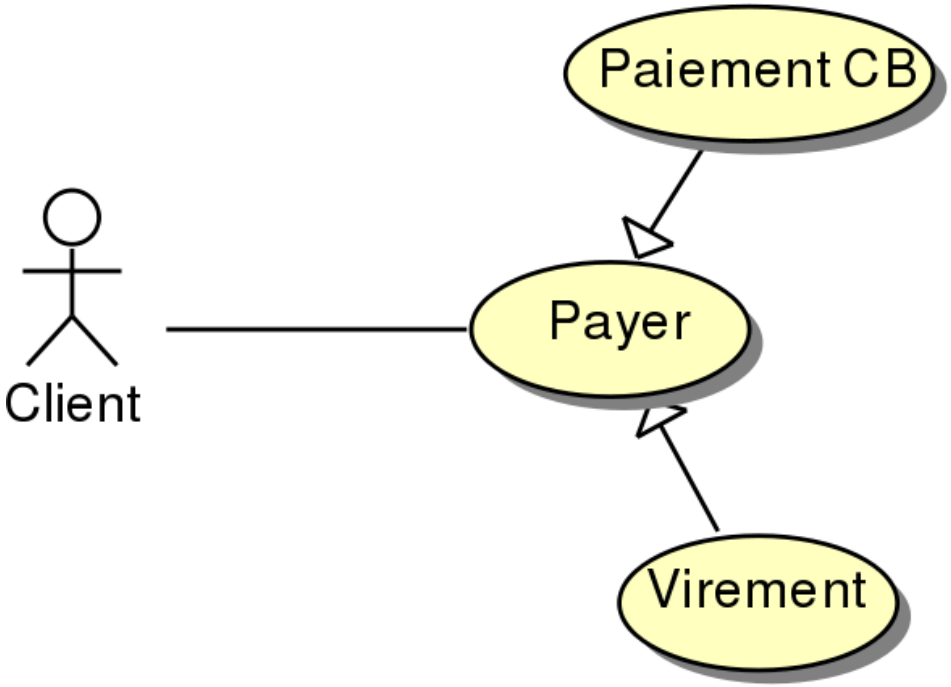


Décomposition grâce aux inclusions et aux extensions

- Quand un cas est trop complexe (faisant intervenir un trop grand nombre d'actions élémentaires), on peut procéder à sa **décomposition** en cas plus simples.



Généralisation



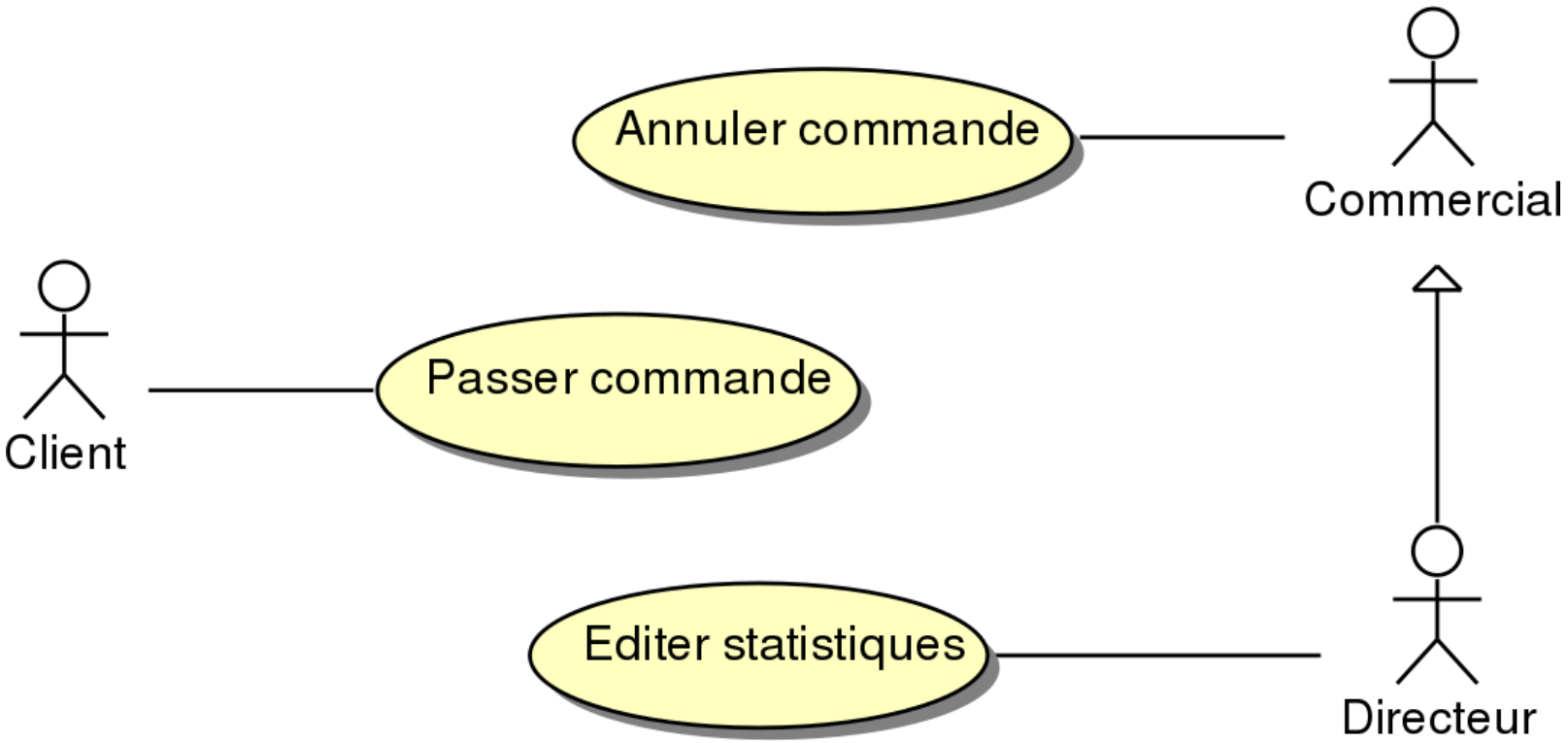
- Un virement est est un cas particulier de paiement.

Un virement **est une sorte de** paiement.

- La flèche pointe vers l'élément général.
- Cette relation de généralisation/spécialisation est présente dans la plupart des diagrammes UML et se traduit par le concept d'**héritage** dans les langages orientés objet.

Relations entre acteurs

- Une seule relation possible : la **généralisation**.



Identification des acteurs

- Les principaux acteurs sont les utilisateurs du système.

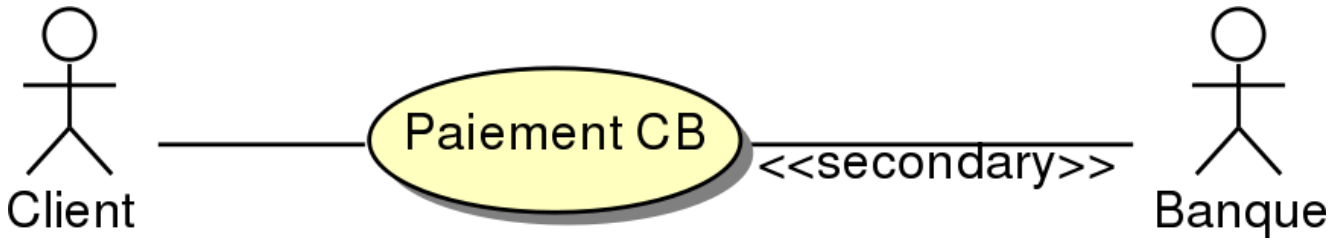
Attention

Un acteur correspond à un **rôle**, pas à une personne physique.

  - Une même personne physique peut être représentée par plusieurs acteurs si elle a plusieurs rôles.
  - Si plusieurs personnes jouent le même rôle vis-à-vis du système, elles seront représentées par un seul acteur.
- En plus des utilisateurs, les acteurs peuvent être :
  - Des périphériques manipulés par le système (imprimantes...) ;
  - Des logiciels déjà disponibles à intégrer dans le projet ;
  - Des systèmes informatiques externes au système mais qui interagissent avec lui, etc.
- Pour faciliter la recherche des acteurs, on se fonde sur les **frontières** du système.

Acteurs principaux et secondaires

- L'acteur est dit **principal** pour un cas d'utilisation lorsque l'acteur est à l'initiative des échanges nécessaires pour réaliser le cas d'utilisation.



- Les acteurs **secondaires** sont sollicités par le système alors que le plus souvent, les acteurs principaux ont l'initiative des interactions.
  - Le plus souvent, les acteurs secondaires sont d'autres systèmes informatiques avec lesquels le système développé est inter-connecté.

Recenser les cas d'utilisation

- Il n'y a pas une manière mécanique et totalement objective de repérer les cas d'utilisation.
  - Il faut *se placer du point de vue de chaque acteur* et déterminer comment il se sert du système, dans quels cas il l'utilise, et à quelles fonctionnalités il doit avoir accès.
  - Il faut *éviter les redondances* et *limiter le nombre de cas* en se situant au bon niveau d'abstraction (par exemple, ne pas réduire un cas à une seule action).
  - Il ne faut pas faire apparaître les détails des cas d'utilisation, mais il faut rester au niveau des grandes fonctions du système.

Trouver le bon niveau de détail pour les cas d'utilisation est un problème difficile qui nécessite de l'expérience.

Description des cas d'utilisation

- Le diagramme de cas d'utilisation décrit les grandes fonctions d'un système du point de vue des acteurs, mais n'expose pas de façon détaillée le dialogue entre les acteurs et les cas d'utilisation.
- Un simple nom est tout à fait insuffisant pour décrire un cas d'utilisation.

Chaque cas d'utilisation doit être documenté pour qu'il n'y ait aucune ambiguïté concernant son déroulement et ce qu'il recouvre précisément.

Description textuelle

- Identification :
  - **Nom du cas** : Payer CB
  - **Objectif** : Détailler les étapes permettant au client de payer par carte bancaire
  - **Acteurs** : Client, Banque (secondaire)
  - **Date** : 18/09
  - **Responsables** : Toto
  - **Version** : 1.0

Description textuelle

- **Séquencements :**
  - Le cas d'utilisation commence lorsqu'un client demande le paiement par carte bancaire
  - **Pré-conditions**
    - Le client a validé sa commande
  - **Enchaînement nominal**
    1. Le client saisit les informations de sa carte bancaire
    2. Le système vérifie que le numéro de CB est correct
    3. Le système vérifie la carte auprès du système bancaire
    4. Le système demande au système bancaire de débiter le client
    5. Le système notifie le client du bon déroulement de la transaction
  - **Enchaînements alternatifs**
    1. En (2) : si le numéro est incorrect, le client est averti de l'erreur, et invité à recommencer
    2. En (3) : si les informations sont erronées, elles sont re-demandées au client
  - **Post-conditions**
    - La commande est validée
    - Le compte de l'entreprise est crédité

Description textuelle

- Rubriques optionnelles
  - **Contraintes non fonctionnelles :**
    - Fiabilité : les accès doivent être sécurisés
    - Confidentialité : les informations concernant le client ne doivent pas être divulgués
  - **Contraintes liées à l'interface homme-machine :**
    - Toujours demander la validation des opérations bancaires