

# *Langage de modélisation U.M.L*

*Préparé par : Tarek BEN MENA*

tarek.benmena@gmail.com

Ecole Supérieur Privée d'Ingénierie et de Technologie  
Informatique, 3ème année

---

# *Introduction à U.M.L*

# Références

---

- « ***UML 2 par la pratique*** »
  - ↳ Pascal Roque
  - ↳ Editions EYROLLES (2006)
- **UML 2.0, guide de référence**
  - ↳ James Rumbaugh, Ivar Jacobson, Grady Booch
  - ↳ Editions Campus Press (2005)
- « ***UML 2 de l'apprentissage à la pratique*** »
  - ↳ Laurent Audibert,
  - ↳ Editions ELLYPSE (2009).

# Hardware et/ou Software

---

- **Systèmes informatiques :**

- ↳ 20 % de matériel

- ↳ **80 % de logiciel**

- **Depuis peu, 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**.

---

## **Qu'est ce qu'un Logiciel?**

# Qu'est qu'un Logiciel ?

---

- **il est fait pour des utilisateurs**

- ↳ dialoguer métier, produire des documents

- **il est complexe et de très grande**

- ↳ 500 000 Inst. exp. physique des particules CERN

- ↳ 1 000 000 Inst. central téléphonique

- ↳ 50 000 000 M Inst. contrôle sol + vol navette spatiale

- **il fait intervenir plusieurs participants**

- ↳ travail en équipe(s), organisation, planification

- **il est long et coûteux à développer**

- ↳ risques nombreux et important : délais, coût

---

**Quelles sont ses caractéristiques ?**

# Un Logiciel

---

## ● Le logiciel (*software*)

↳ « est défini comme une création intellectuelle rassemblant des programmes, des procédures, des règles et de la documentation utilisé pour faire fonctionner un système informatique » [ISO]

↳ Il est **immatériel** et **invisible**

- ↳ la qualité n'est pas vraiment apparente
- ↳ difficile d'estimer l'effort de développement

↳ Il est **difficile à automatiser**

- ↳ Beaucoup de main d'œuvre

↳ Il ne s'use pas, mais il **vieillit**

- ↳ Détérioration suite aux changements ou Evolution du matériel
- ↳ Mal conçu au départ

↳ Il a deux catégories : **sur mesure** et **générique**



---

**Des exemples ? Plutôt des familles de Logiciels**

---

## **Qu'est ce qu'un Système d'information (SI) ?**

# Un système d'information

---

- **est un ensemble organisé de ressources :**
  - ↳ matériels,
  - ↳ logiciels,
  - ↳ humaines,
  - ↳ données et procédures)
- **qui permet de collecter, regrouper, classifier, traiter et diffuser de l'information sur un environnement donné**
- **NTIC : moyens informatiques, électroniques et de procédés de télécommunication permet**
  - ↳ d'accompagner, d'automatiser et de dématérialiser quasiment toutes les opérations incluses dans les activités ou procédures d'entreprise.

---

**Et qu'est ce qu'un BON Système d'information ?**

## **Le « BON » Système d'information**

---

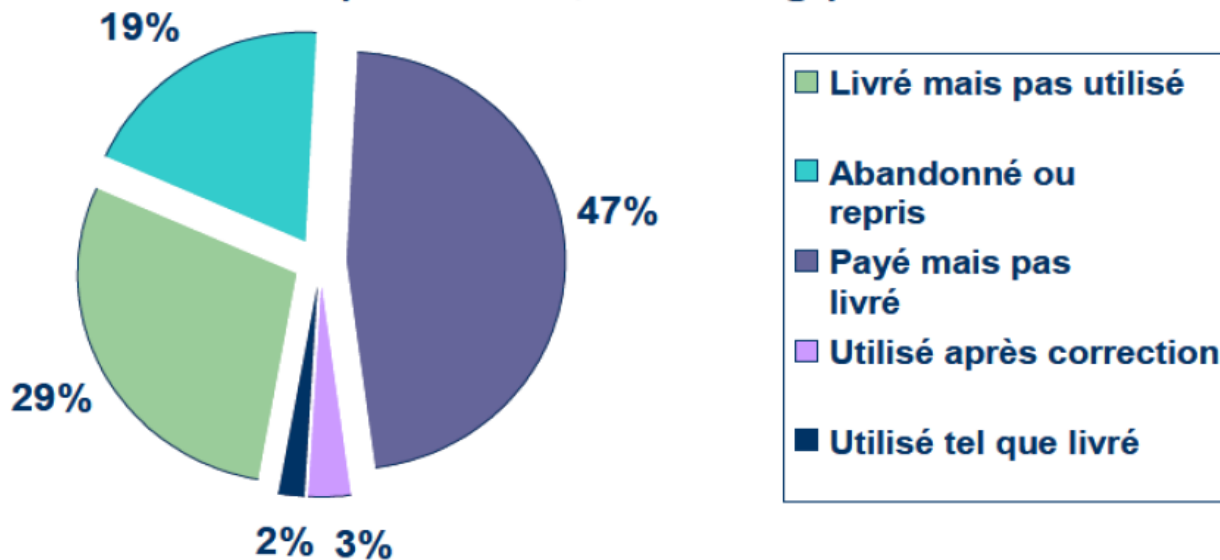
- **conforme aux besoins de l'utilisateur (client)**
- **fiable : ne doit pas tomber en panne, plus qu'il n'est autorisé**
- **efficace pas de gaspillage de ressources**
- **maintenable : changement pas coûteux**
- **avec interface adaptée aux utilisateurs**

---

**Et le problème dans tout cela ?**

# La « Crise Logiciel »

Sources: US Gov. Accounting Report,  
(in B. Cox, OO Prog.)



Rapport du Congrès Américain en 1979 sur 487 projets

Étude sur 8 380 projets  
(Standish Group, 1995) :

- 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 ?

---

**Comment obtenir cela?**



# Cycle de Vie

---

La qualité du **processus** de fabrication est garante de la qualité du **produit**.

- **Pour obtenir un logiciel de qualité, il faut en maîtriser le processus d'élaboration.**

↳ La vie d'un logiciel est composée de différentes **étapes**.

↳ La succession de ces **étapes** forme le cycle de vie du logiciel.

↳ Il faut contrôler la succession de ces différentes **étapes**.

---

**Quelles sont ses étapes**

# Les étapes de développement

---

- **Étude de faisabilité**

- **Spécification**

  - ↳ Déterminer les fonctionnalités du logiciel.

- **Conception**

  - ↳ Déterminer la façon dont le logiciel fournit les différentes fonctionnalités recherchées.

  - ↳ **Modéliser** la solution

- **Codage**

- **Tests**

  - ↳ Essayer le logiciel sur des données d'exemple pour s'assurer qu'il fonctionne correctement.

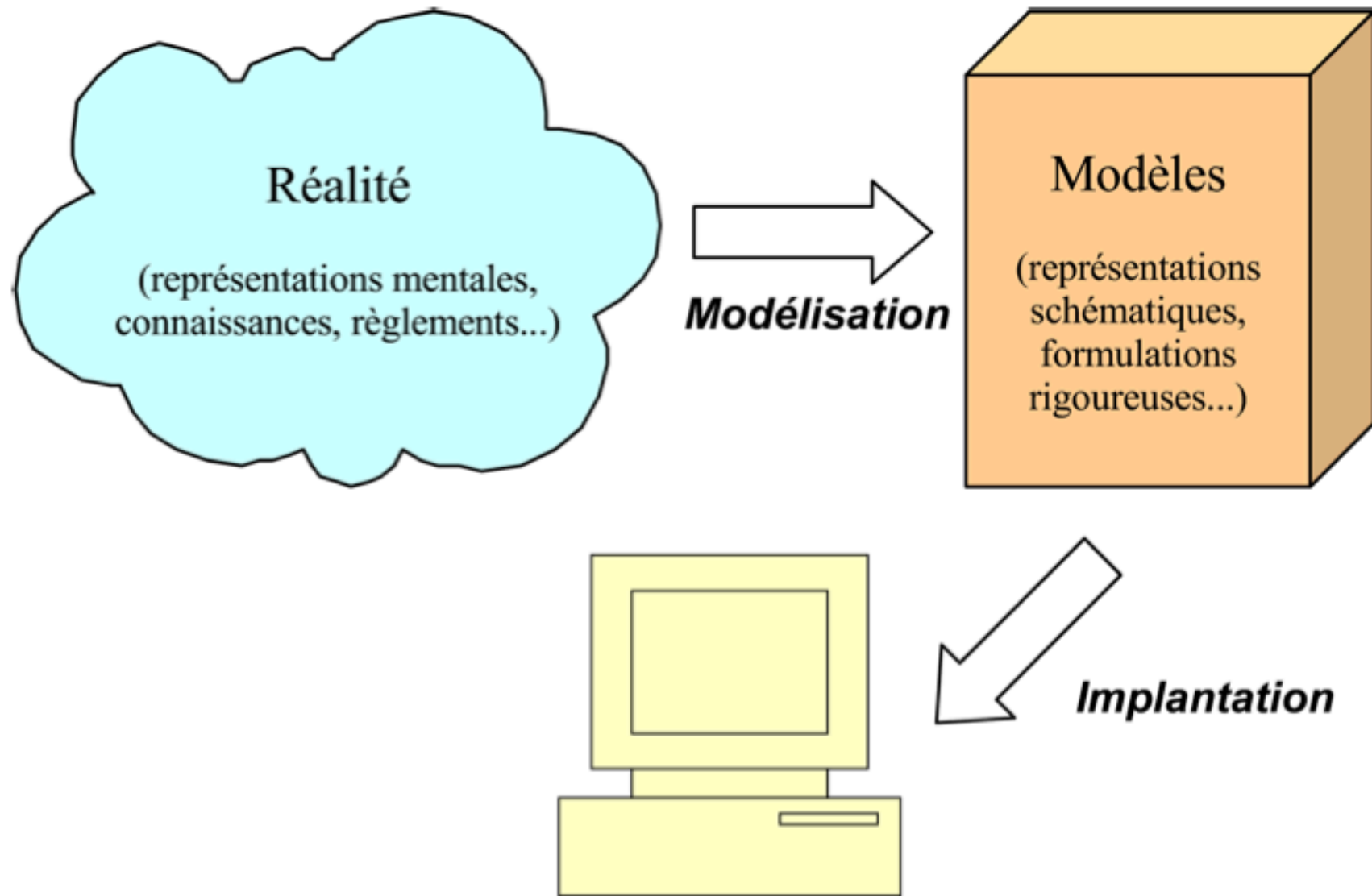
- **Maintenance**

---

## **Qu'est ce que la modélisation ?**

# Modélisation

---



## Modélisation : définition d'un modèle

---

- Un modèle est une représentation **abstraite** de la réalité qui exclut certains détails du monde réel.
  - ↳ Il permet de **réduire la complexité** d'un phénomène en éliminant les détails qui **n'influencent pas** son comportement de manière significative.
  - ↳ Il reflète ce que le concepteur croit **important / pertinent** pour la compréhension et la prédiction du phénomène modélisé,
  - ↳ les limites du phénomène modélisé dépendent des objectifs du modèle.

---

**Qu'est ce qu'un langage de modélisation ?**

# Un langage de modélisation

---

- **Un langage de modélisation doit définir :**

- ↳ La sémantique des concepts ;
- ↳ Une notation pour la représentation de concepts ;
- ↳ Des règles de construction et d'utilisation des concepts.

- **Niveaux de formalisation**

- ↳ **Langages formels** (Z,B,VDM) : le plus souvent mathématiques, au grand pouvoir d'expression
- ↳ **Langages semi-formels** (MERISE, UML...) : le plus souvent graphiques, au pouvoir d'expression moindre mais plus faciles d'emploi.

- **L'industrie du logiciel dispose de LM :**

- ↳ Adaptés aux systèmes procéduraux (MERISE...) ;
- ↳ Adaptés aux systèmes temps réel (ROOM, SADT...) ;
- ↳ Adaptés aux systèmes à objets (OMT, Booch, UML...).

- **Le rôle des outils (Ateliers Génie Logiciel) est**

- ↳ primordial pour l'utilisabilité en pratique des langages de modélisation.



# UML : Unified Modeling Language

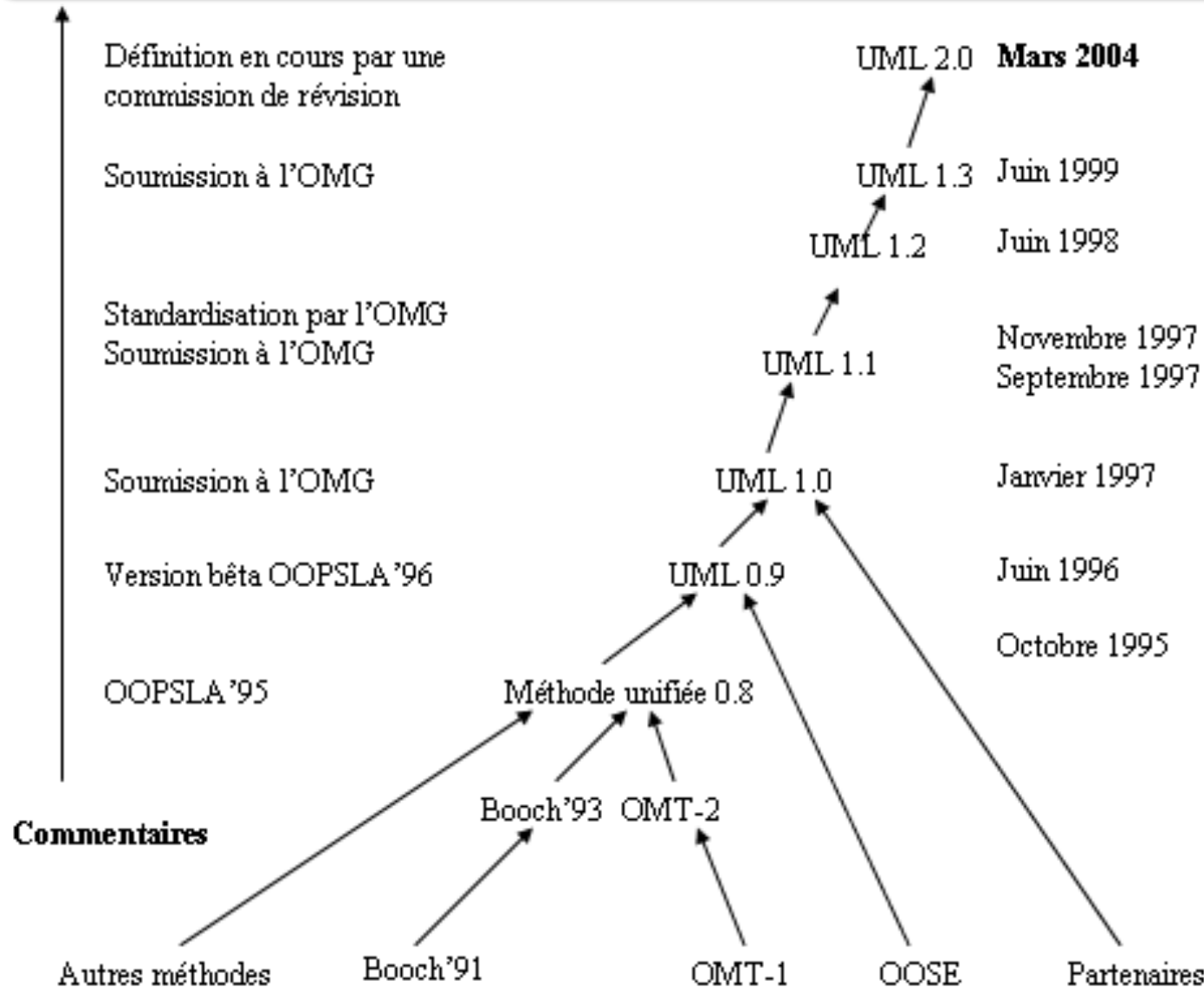
---

- **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 (Semi-formelle) pour être traduites automatiquement en code source ;
  - ↳ Ces représentations ne disposent cependant pas de qualités formelles suffisantes pour justifier d'aussi bonnes propriétés mathématiques que des langages de spécification formelle (Z, VDM...).

# Historique

**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.



## UML 2.0

---

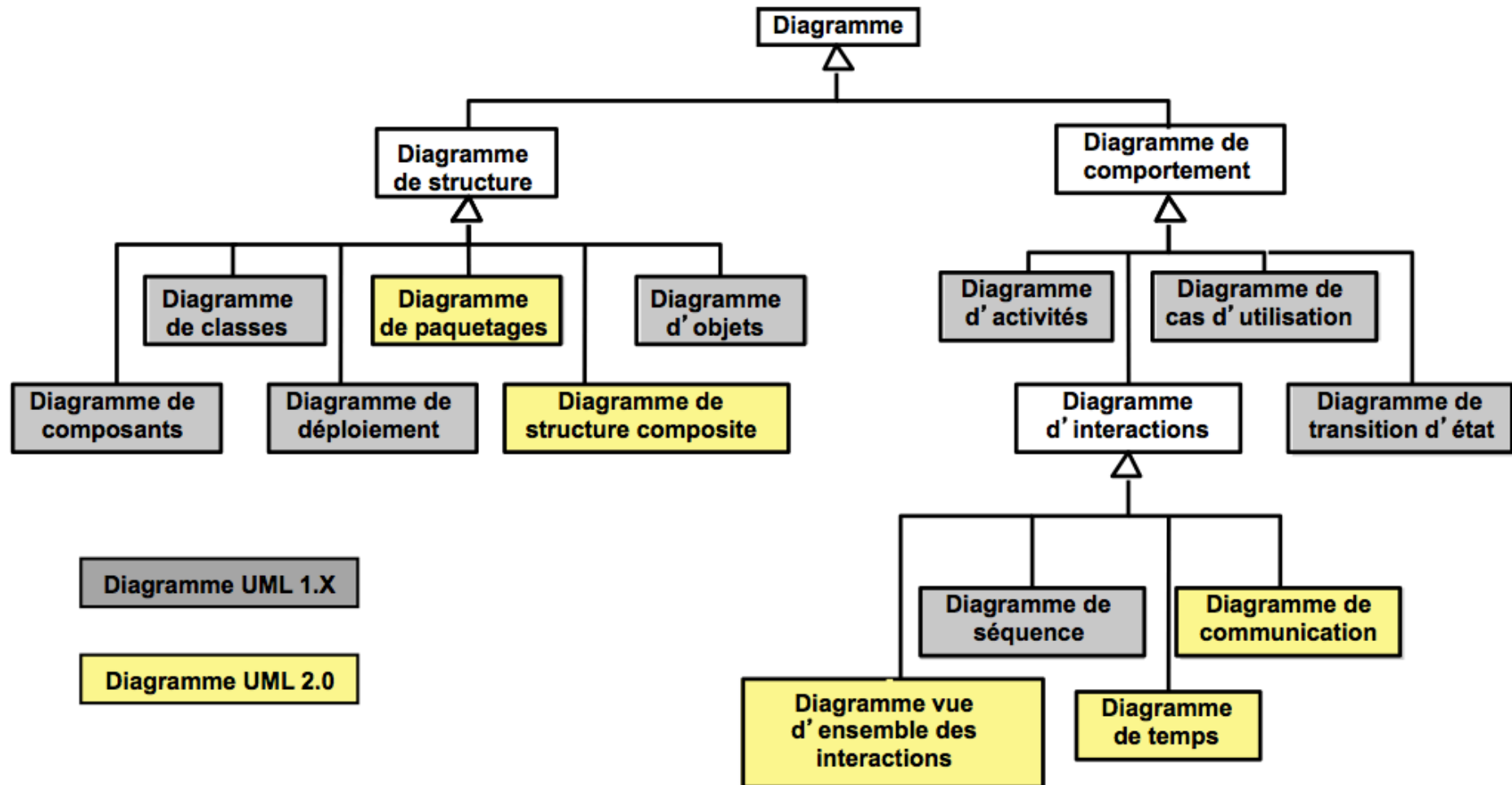
- **La version 2 de UML a été finalisée par l'OMG en Juillet 2005**
- **Ajouts d'un ensemble de nouvelles fonctionnalités en partie issues des "manques" de la version 1.x pour :**
  - ↳ Rendre plus "exécutable » le langage
  - ↳ Fournir des mécanismes plus robuste pour la modélisation des workflows et des actions
  - ↳ Créer un standard pour la communication entre outils
  - ↳ Fournir un cadre standard de modélisation
- **UML 2.0 utilise 13 types de diagrammes, contre 9 en UML 1.X**
  - ↳ Source : "UML 2.0", Martin Fowler, Pearson Education (2004)

## UML 2.0 vs. UML 1.x

---

- **Les diagrammes ont été revus pour répondre aux nouveaux besoins (abstraction, automatisatisation...)**
- **Le diagramme de collaboration d'UML 1.X devient le diagramme de communication**
- **Principaux changements dans les diagrammes de :**
  - ↳ Classes
  - ↳ Séquences
  - ↳ De machines-états
  - ↳ D'activités

# Les Diagrammes



# Diagrammes : de Structures ou Statiques

---

- **Diagramme de classes**

- ↳ Représente les classes intervenant dans le système

- **Diagramme d'objets**

- ↳ Représente les instances de classes (objets) utilisées dans le système

- **Diagramme de composants**

- ↳ Représente les composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, BD...)

- **Diagramme de déploiement**

- ↳ Représente les éléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage...) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent avec eux

- **Diagramme des paquetages**

- ↳ Représente l'hiérarchie des paquetages du projet, et leurs interdépendances

- **Diagramme de structures composites**

- ↳ décrit la structure interne d'un objet complexe lors de son exécution dont ses points d'interaction avec le reste du système

- **Diagramme de profiles (UML 2.2)**

# Diagrammes : Comportementaux

---

- **Diagramme des cas d'utilisation :**

- ↳ il décrit les possibilités d'interaction entre le système et les acteurs, c'est-à-dire toutes les fonctionnalités que doit fournir le système

- **Diagramme états-transitions :**

- ↳ il montre la manière dont l'état du système (ou de sous-parties) est modifié en fonction des événements du système

- **Diagramme d'activité :**

- ↳ variante du diagramme d'états-transitions, il permet de représenter le déclenchement d'événements en fonction des états du système et de modéliser des comportements parallélisables (multi- threads ou multi-processus)

# Diagramme : d'interaction ou dynamique

---

- **Diagramme de séquence :**

- ↳ la représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou des acteurs

- **Diagramme de communication :**

- ↳ La représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets

- **Diagramme global d'interaction (Interaction Overview Diagram):**

- ↳ variante du diagramme d'activité où les noeuds sont des interactions, permet d'associer les notations du diagramme de séquence à celle du diagramme d'activité, ce qui permet de décrire une méthode complexe

- **Diagramme de temps (Timing Diagram)!:**

- ↳ la représentation des interactions où l'aspect temporel est mis en valeur; il permet de modéliser les contraintes d'interaction entre plusieurs objets, comme le changement d'état en réponse à un événement extérieur

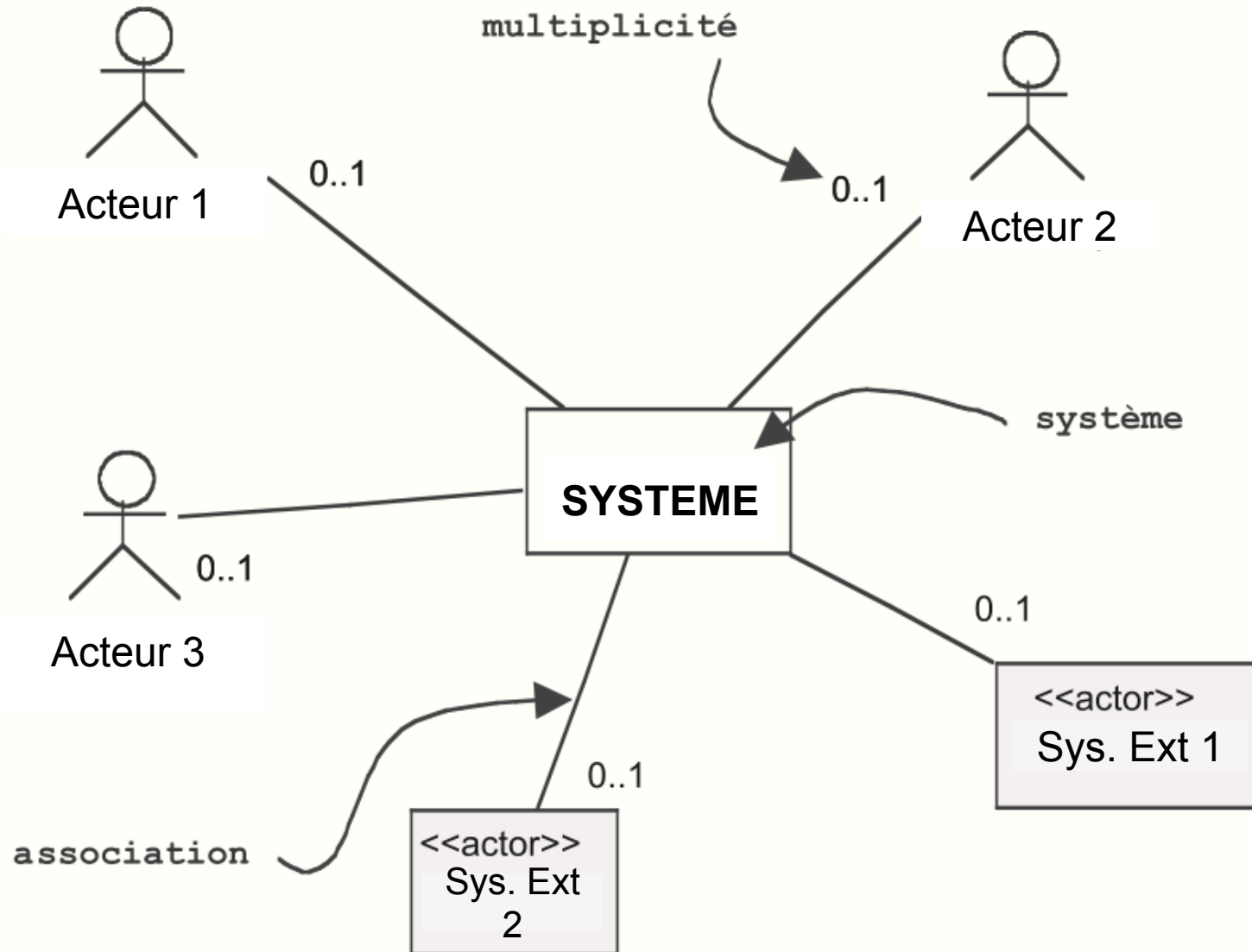
- ↳ UML 2.5



---

# *Diagramme de Contexte*

# Exemple



# Modélisation des besoins

---

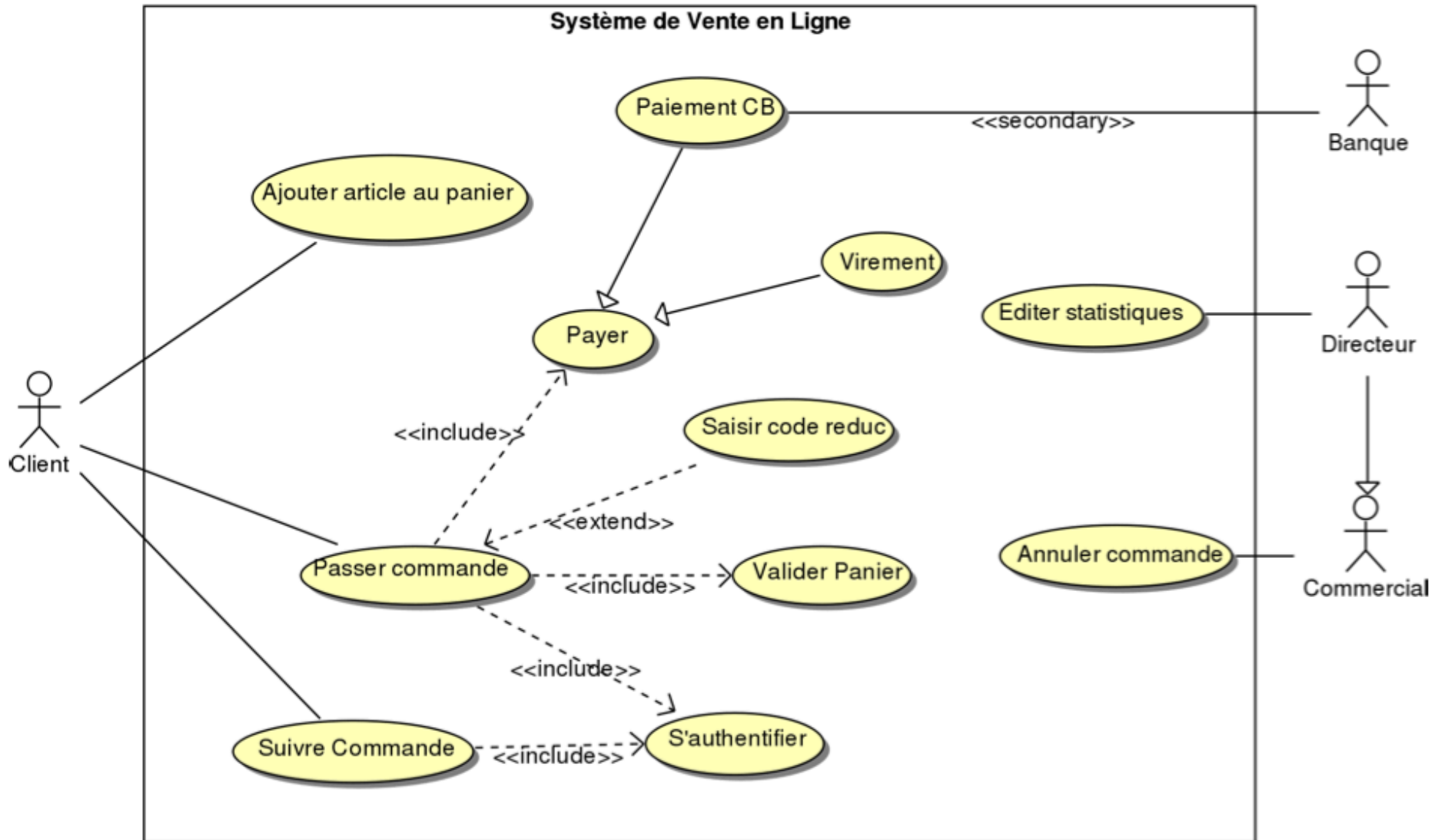
*Avant de développer un système, il faut savoir précisément à QUOI il devra servir, c-à-d à quels besoins il devra répondre.*

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

---

## *Diagramme de Cas d'utilisation*

# Exemple

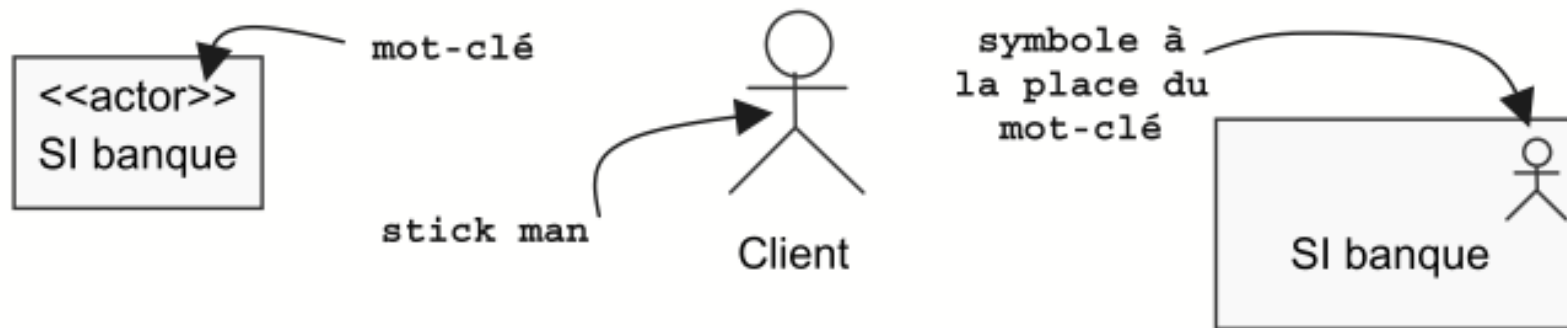


# Cas d'utilisation

- Un cas d'utilisation est un service rendu à l'utilisateur, il implique des séries **d'actions plus élémentaires**.



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

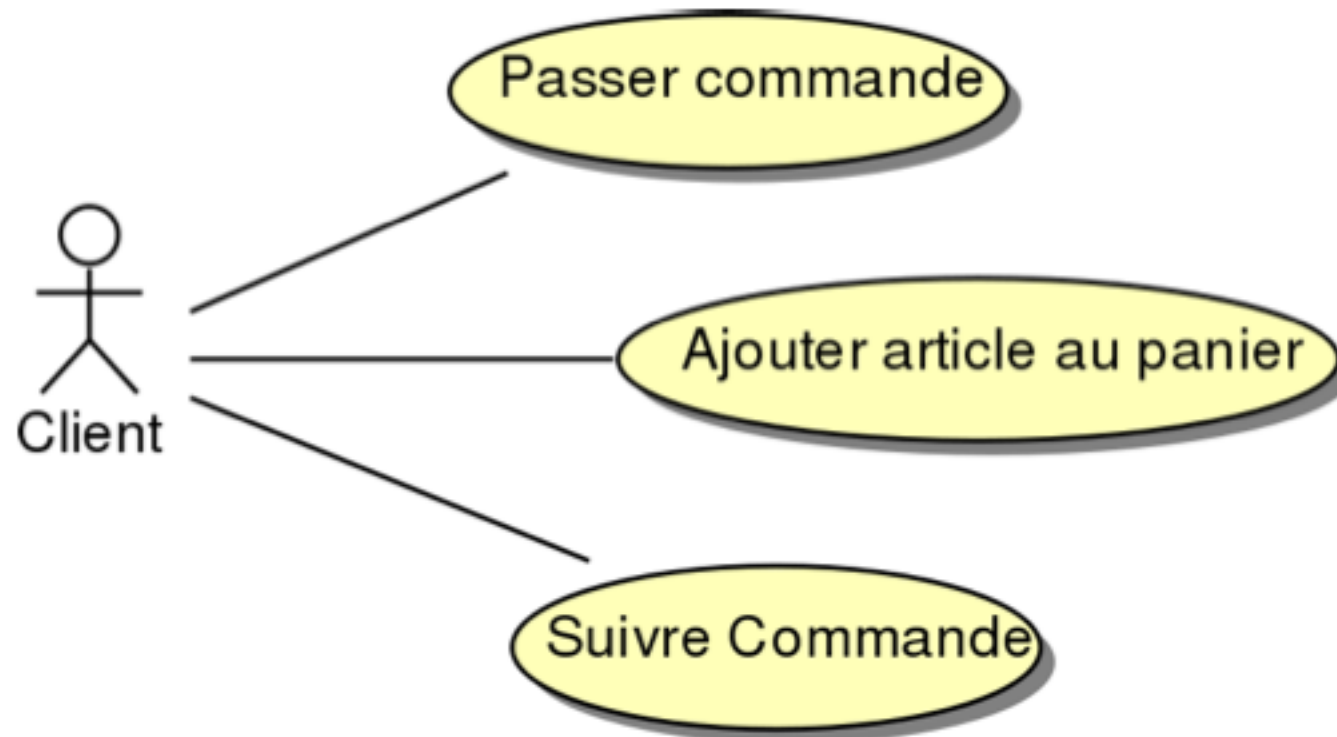


*Un cas d'utilisation est l'expression d'un service réalisé de **bout en bout**, avec un déclenchement, un déroulement et une fin, pour l'acteur qui l'initie.*

# Acteur et cas d'utilisation

---

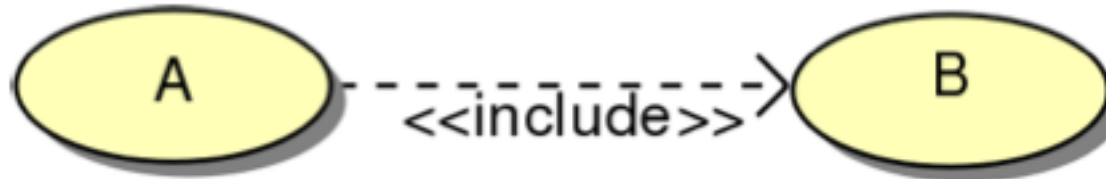
- **Acteur** : un **nom** représentant un **rôle**
- **Cas d'utilisation** : **verbe à l'infinitif** + **complément**
  - ↳ Les acteurs impliqués dans un cas d'utilisation lui sont liés par une association.
  - ↳ Un acteur peut utiliser plusieurs fois le même cas d'utilisation.



## Relations entre cas d'utilisation

---

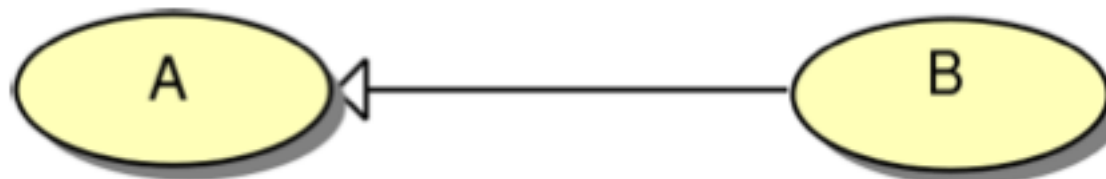
- **Inclusion** : le cas A inclut le cas B (B est une partie obligatoire de A).



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



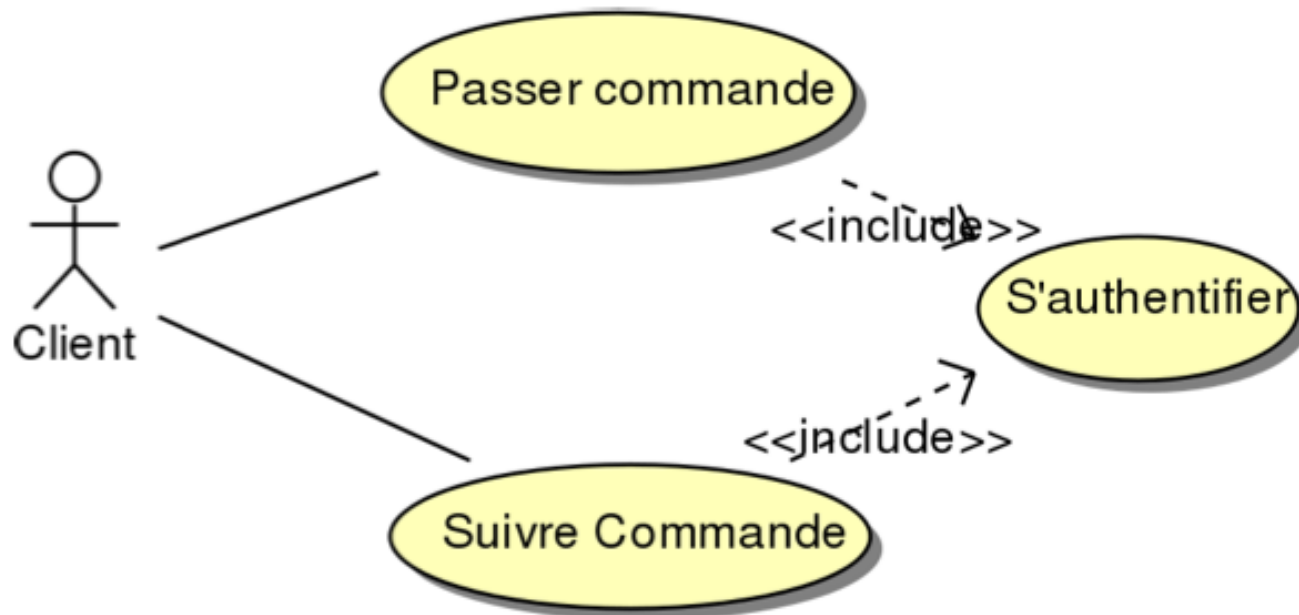
- **Généralisation** : le cas A est une généralisation du cas du cas B (B est une sorte de A).





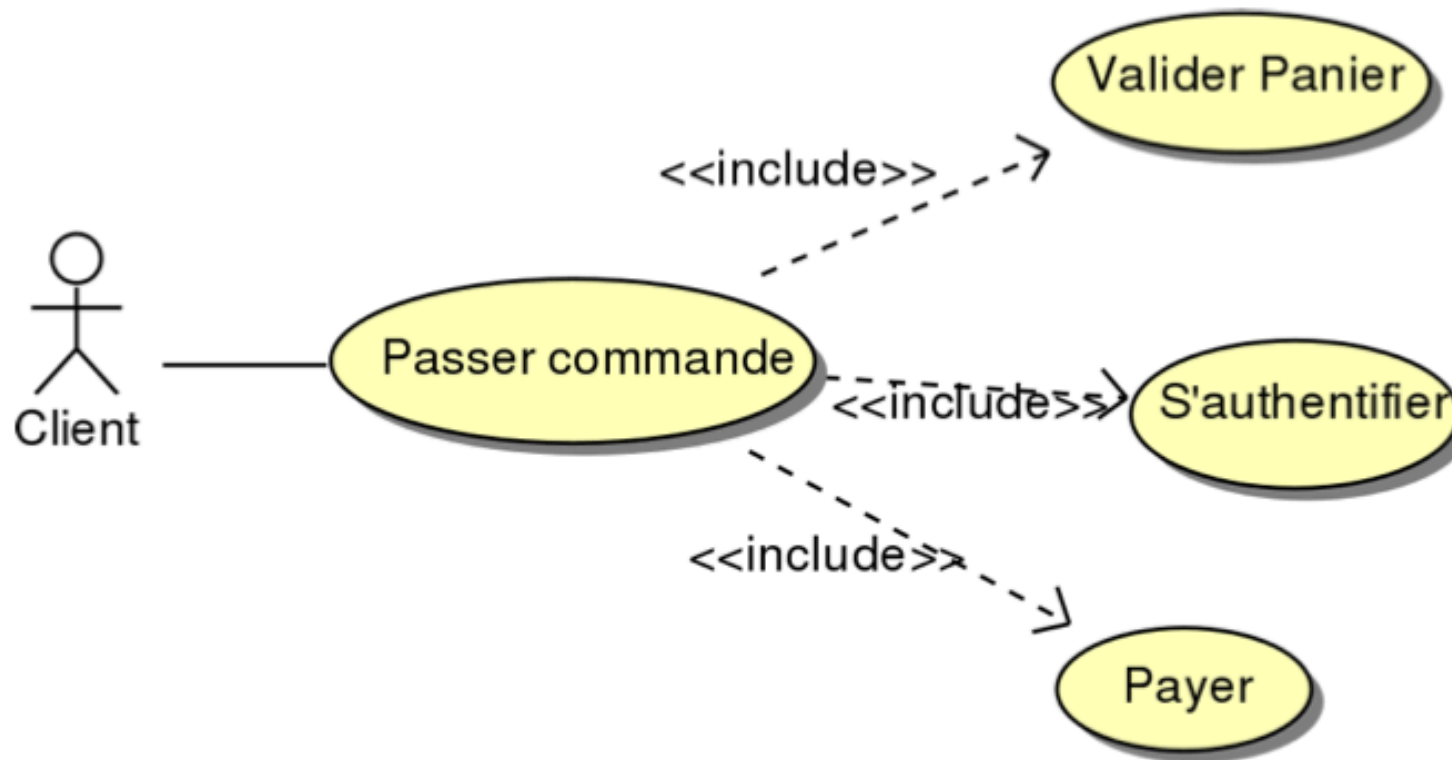
## Réutilisabilité avec les inclusions et les extensions

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



## Décomposition grâce au inclusion et au extension

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

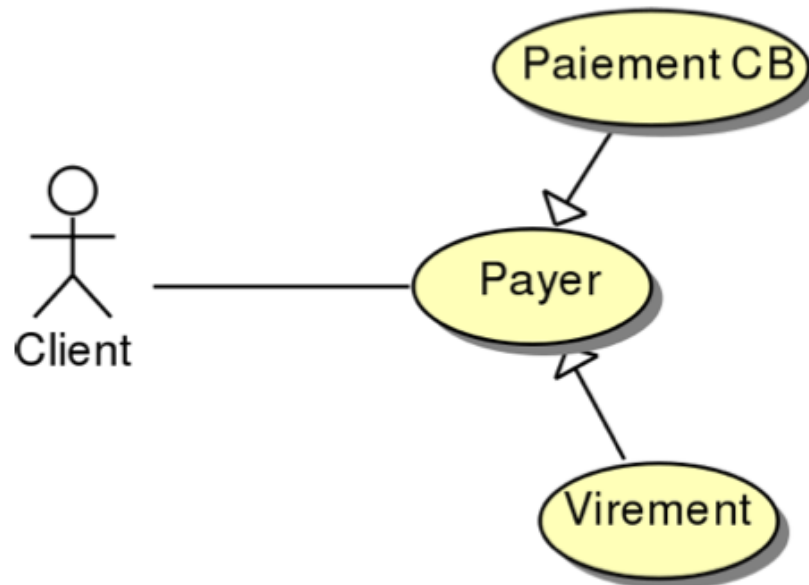


# Généralisation

---

- Un virement **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
- ↳ se traduit par le concept d'**héritage** dans les langages orientés objet.

# Identification des acteurs

---

- **Les principaux acteurs sont les utilisateurs du système.**

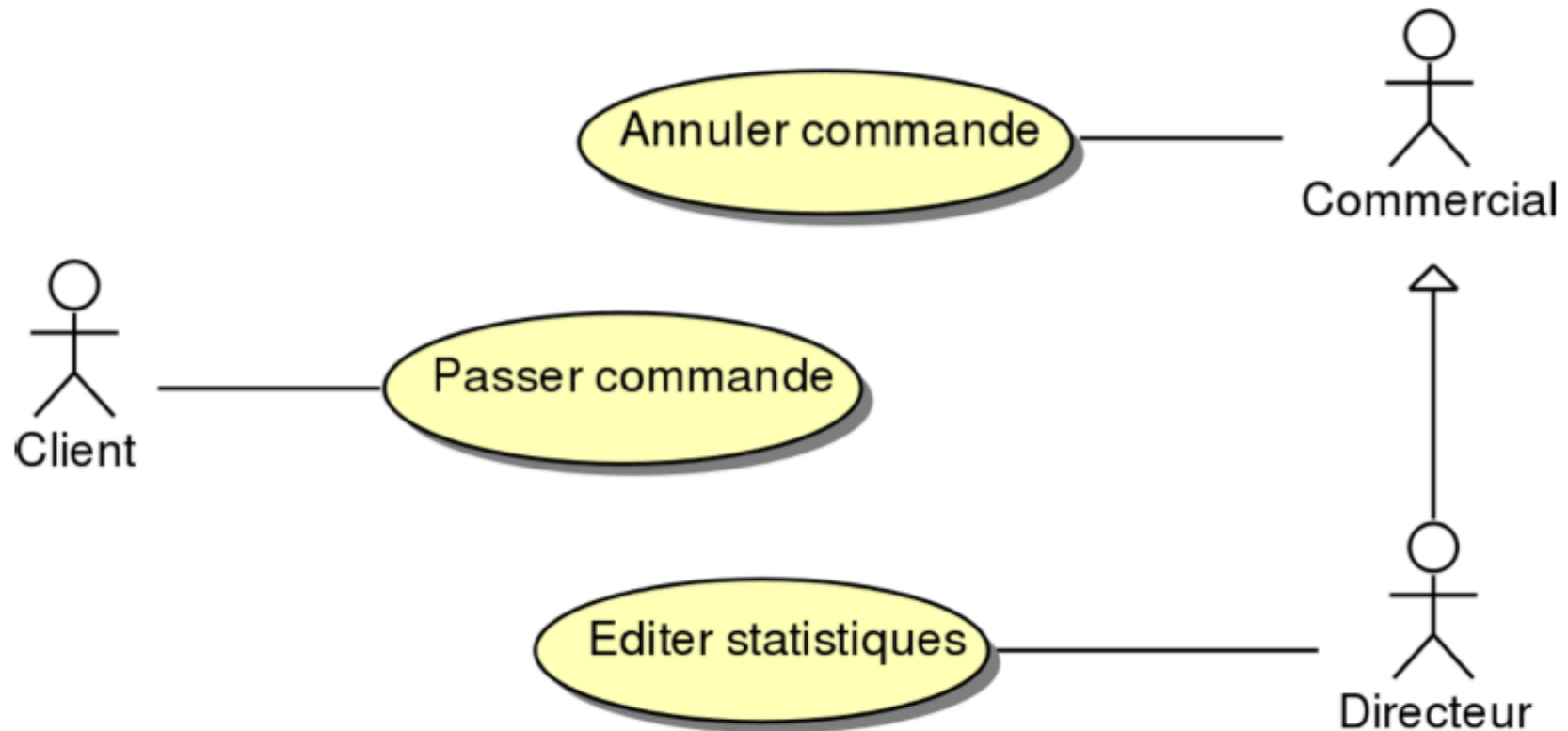
## Attention

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.**

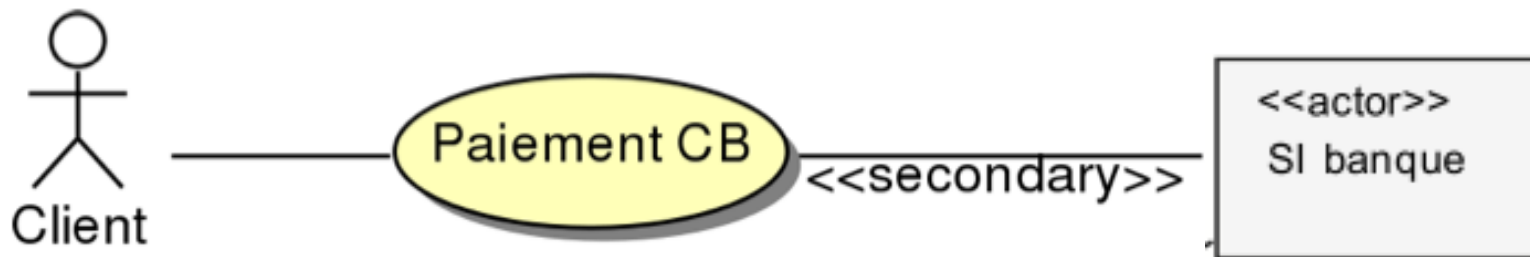
## Relations entre acteurs

- Une seule relation possible : la généralisation.



## 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.

### Remarque

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 (Source d'ambiguïté)**

### Remarque

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



## Exemple : Description textuelle (1/3)

---

### ● Identification :

- ↳ Nom du cas : Payer CB
- ↳ Objectif : Détailler les étapes permettant à client de payer par carte bancaire
- ↳ Acteurs : Client, Banque (secondaire)
- ↳ Date de création : 18/09/13
- ↳ Date de mise à jour : 27/09/13
- ↳ Version : 1.1
- ↳ Responsables : M. Toto

## Exemple : Description textuelle (2/3)

---

### ● 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



TBM 2013-14 ☞ Le compte de l'entreprise est créditée

## Exemple : Description textuelle (3/3)






---

### ● Rubriques optionnelles

#### **Contraintes non fonctionnelles :**

-  Fiabilité : les accès doivent être sécurisés
-  Confidentialité : les informations concernant le client ne doivent pas être divulgués

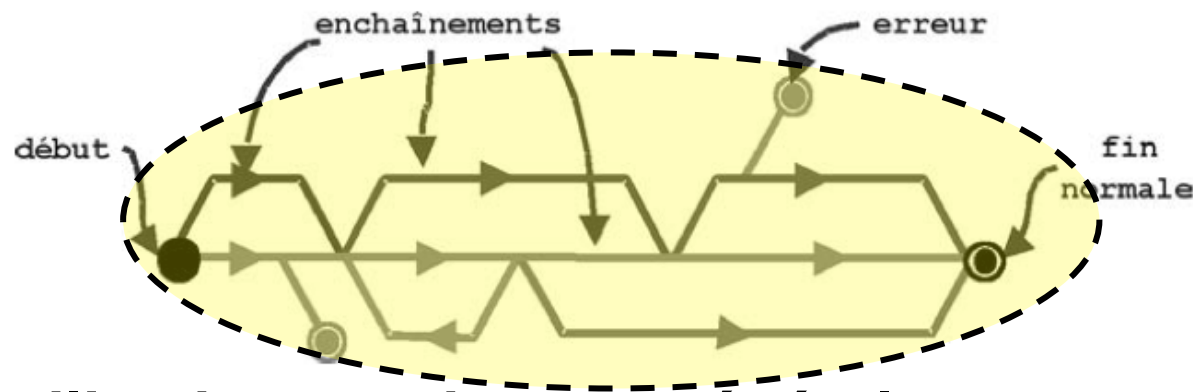
#### **Contraintes liées à l'Interface Homme-Machine :**

-  Toujours demander la validation des opérations bancaires
-  Un clavier numérique (pour saisir son code), avec des touches « validation », « correction » et « annulation ».
-  Un écran pour l'affichage des messages.
-  Des touches autour de l'écran pour sélectionner un montant de retrait parmi ceux qui sont proposés.
-  Un distributeur de tickets.

# Les enchaînements sont appelés des scénarios

## ● Un scénario représente

- ↪ une succession particulière d'enchaînements, s'exécutant du début à la fin du cas d'utilisation,
- ↪ un enchaînement étant l'unité de description de séquences d'actions.



## ● Un cas d'utilisation contient en général

- ↪ un scénario nominal et
- ↪ plusieurs scénarios alternatifs (qui se terminent de façon normale) ou d'erreur (qui se terminent en échec).

### Remarque

*un cas d'utilisation: « ensemble de scénarios d'utilisation d'un système reliés par un but commun du point de vue d'un acteur ».*

## Autre présentation du scénario [Larman 97]

### Exemple : GAB

Action coté l'Acteur	Réaction du Système
1- Le Porteur de carte introduit sa carte dans le lecteur de cartes du GAB.	2- Le GAB vérifie que la carte introduite est bien une carte bancaire.  3- Le GAB demande au Porteur de carte de saisir son code d'indentification.
4- Le Porteur de carte saisit son code d'identification.	5- Le GAB compare le code d'identification avec celui qui est codé sur la puce de la carte.  6- Le GAB demande une autorisation au Système d'autorisation.
7- Le Système d'autorisation donne son accord et indique le solde hebdomadaire.	8- Le GAB demande au Porteur de carte de saisir le montant désiré du retrait.
...	...

# Modélisation des Scénarios

---

- **Pour documenter les cas d'utilisation, la description textuelle est indispensable,**
  - ↳ communiquer facilement avec les utilisateurs
  - ↳ s'entendre sur la terminologie métier employée.
- **Le texte présente des désavantages**
  - ↳ il est difficile de montrer comment les enchaînements se succèdent,
  - ↳ Difficile de montrer l'intervention des acteurs secondaires.
  - ↳ la maintenance des évolutions devient souvent fastidieuse.

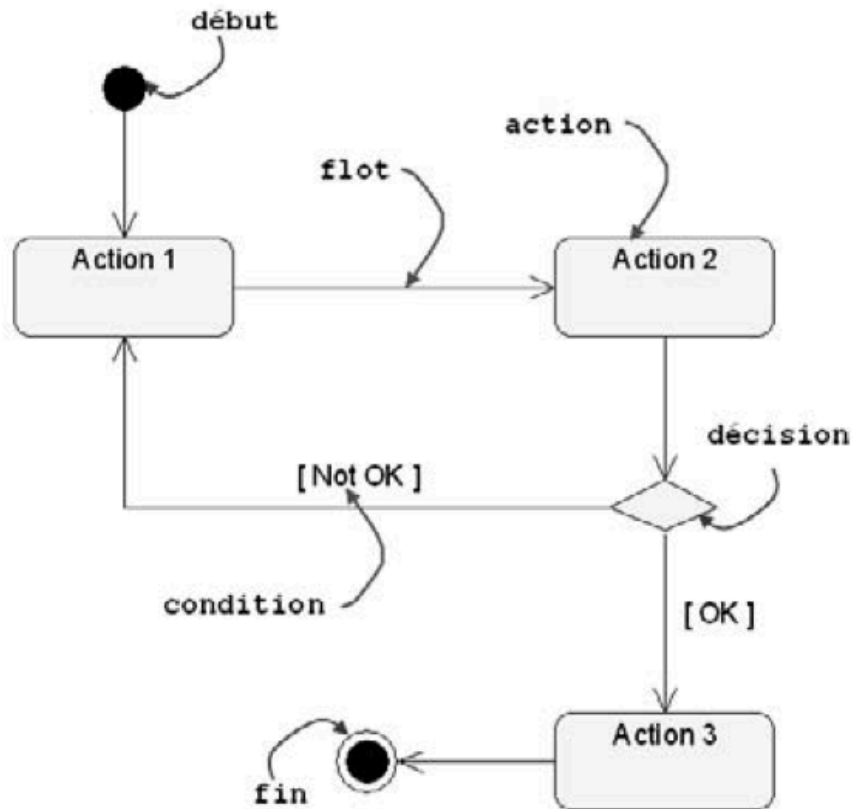
## Remarque

Il est donc recommandé de compléter la description textuelle par un ou plusieurs diagrammes dynamiques UML.

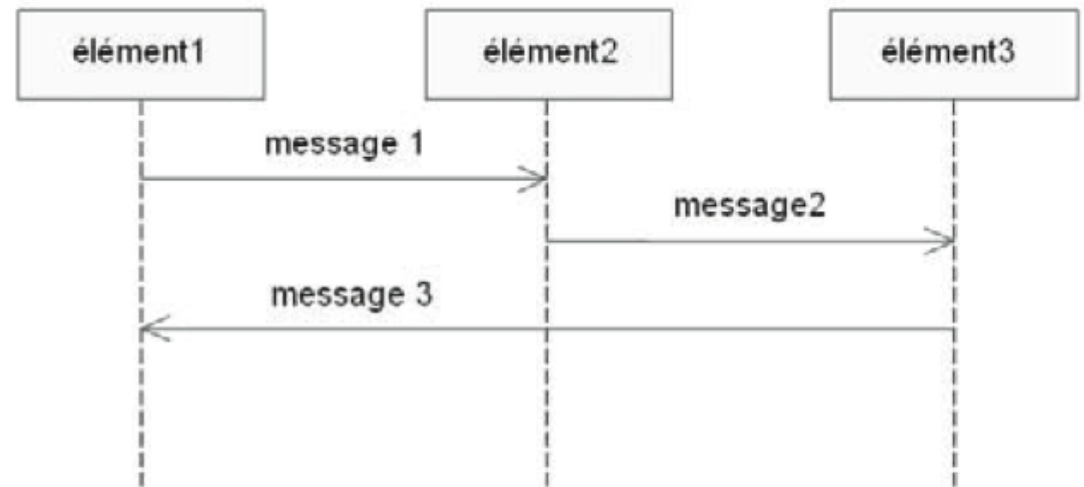
- **Les quels ?**
  - ↳ Diagramme d'activité
  - ↳ Diagramme de séquence (Système)
  - ↳ UML 2 : « Interaction Overview Diagram »

# Exemples

- Diagramme d'activité

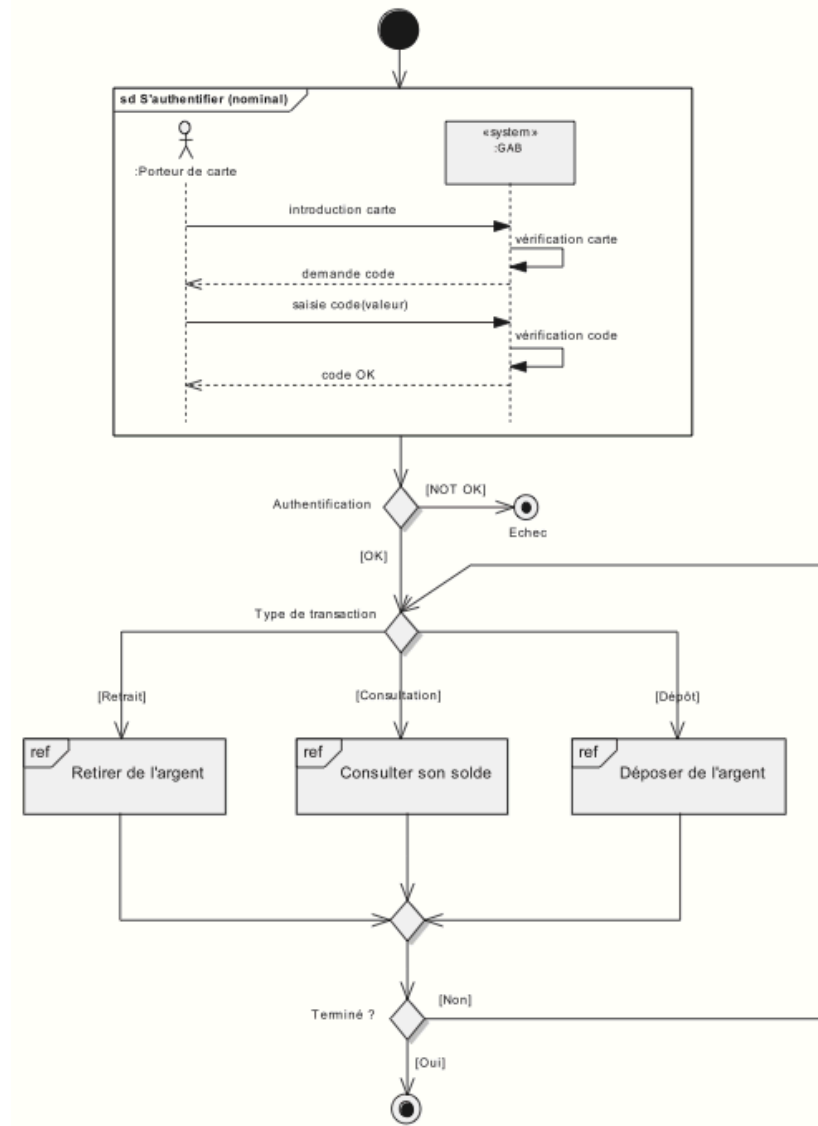


- Diagramme de Séquence



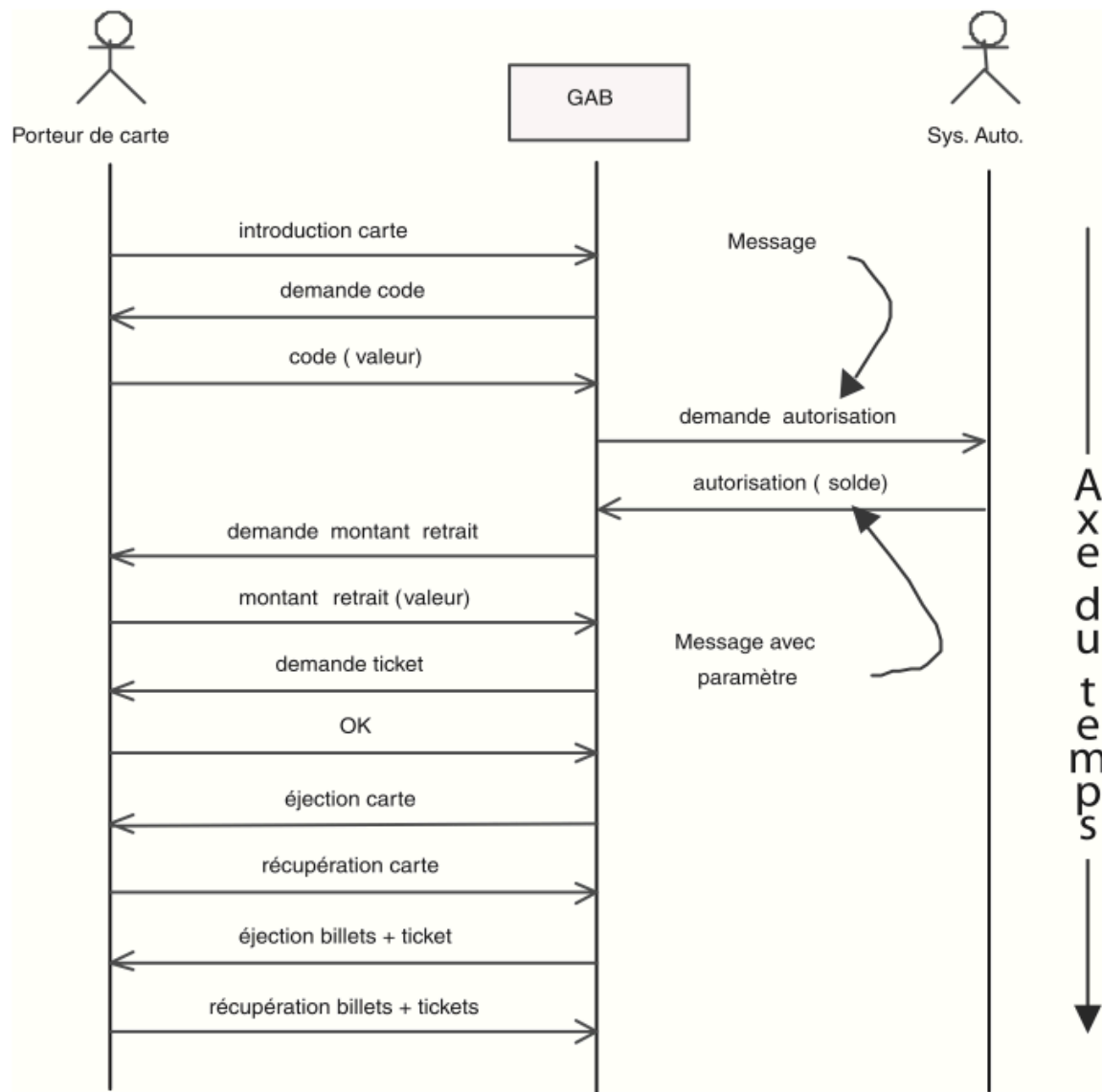
# Exemples

## ● Diagramme « Interaction Overview Diagram »





## Diagramme de séquence système du scénario nominal de Retirer Argent d'un DAB



## Exemple : *Diagramme d'activité de Retirer de l'argent d'un DAB*

