UML : Les diagrammes

UML 2 propose 13 types de diagrammes

Modélisation des vues statiques d'un système:

- Le diagramme des cas d'utilisation
- Le diagramme de classes
- Le diagramme d'objets ...

Modéliser les vues dynamiques d'un système

- Le diagramme de séquence
- Le diagramme de collaboration, d'états/transition , ...

Diagramme de cas d'utilisation

- Diagramme de cas d'utilisation
  - Acteurs
  - Cas d'utilisation (use case)
  - Système
- Relations dans les diagrammes de cas d'utilisation
  - Relations entre Acteurs
  - Relations entre cas d'utilisation

Diagramme de cas d'utilisation

Les diagrammes de cas d'utilisation (use case models) sont créés pour :

- Présenter les concepts UML relatifs à la vue fonctionnelle
- Visualiser l'interaction du système avec le monde extérieur
- Modéliser très tôt les processus métiers et l'organisation de l'entreprise

Ils se basent sur des diagrammes où des acteurs interagissent avec le système de l'extérieur ou de l'intérieur.

Acteurs

Définition:

- Un acteur représente un rôle joué par une entité externe (utilisateur humain, dispositif matériel ou autre système) qui interagit directement avec le système étudié
- Un acteur peut consulter et/ou modifier directement l'état du système, en émettant et/ou en recevant des messages susceptibles d'être porteurs de données

Question: Comment les identifier ?

Acteurs (suite 1)

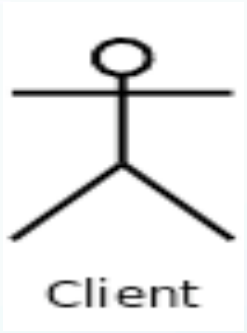
Les acteurs candidats sont systématiquement :

- Les utilisateurs humains directs : faites donc en sorte d'identifier tous les profils possibles, sans oublier l'administrateur, l'opérateur de maintenance, etc. ;

- Les autres systèmes connexes qui interagissent aussi directement avec le système étudié, souvent par le biais de protocoles bidirectionnels.

Acteurs (suite 2)

- La représentation graphique standard de l'acteur en UML est l'icône appelée *stick man*, avec le nom de l'acteur sous le dessin.
- On peut également figurer un acteur sous la forme rectangulaire d'une classe, avec le mot-clé <<actor>>.
- Une bonne recommandation consiste à faire prévaloir l'utilisation de la forme graphique du *stick man* pour les acteurs humains et une représentation rectangulaire pour les systèmes connectés.



Acteurs (suite 3)

Les types d'acteur :

- Les acteurs principaux : personnes qui utilisent les fonctions du système.
- Les acteurs secondaires : personnes qui effectuent des tâches administratives ou de maintenance.
- Les matériels extérieurs sauf la machine où se trouve l'application (tels que distributeur de billets).
- Les autres systèmes tels que par exemple le réseau des cartes bancaires

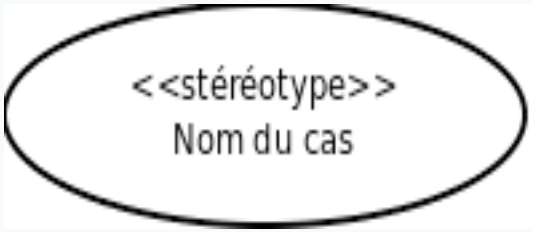
Cas d'utilisation (use case)

- Les cas d'utilisations (use-case) sont des séquences d'actions menées par le système qui doit donné un résultat observable pour un acteur.
- Il est recommandé que l'intitulé du cas d'utilisation respecte le pattern « verbe + compléments ». Le verbe de l'intitulé permet de spécifier la nature de la fonctionnalité offerte par l'application, tandis que les compléments permettent de spécifier les données d'entrée ou de sortie de la fonctionnalité.

Use case (suite 1)

Exemple: Le cas d'utilisation " calculer taux d'intérêt du prêt " permet de comprendre d'une certaine manière que l'application permet à ses utilisateurs de calculer le taux d'intérêt d'un prêt.

Un cas d'utilisation se représente par une ellipse contenant l'intitulé du cas d'utilisation.

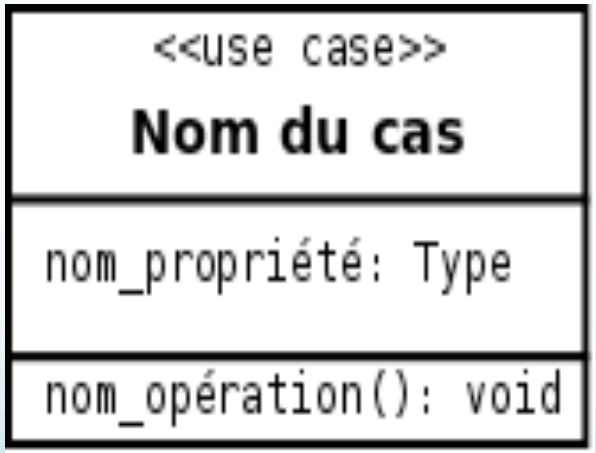


Dans le cas où l'on désire présenter les attributs ou les opérations du cas d'utilisation, il est préférable de le représenter sous la forme d'un classeur stéréotypé << use case >>.

Use case (suite 2)

Nous reviendrons sur les notions d'attributs ou d'opération lorsque nous aborderons les diagrammes de classes et d'objets .

Exemple: Représentation d'un cas d'utilisation sous la forme d'un classeur:



Système

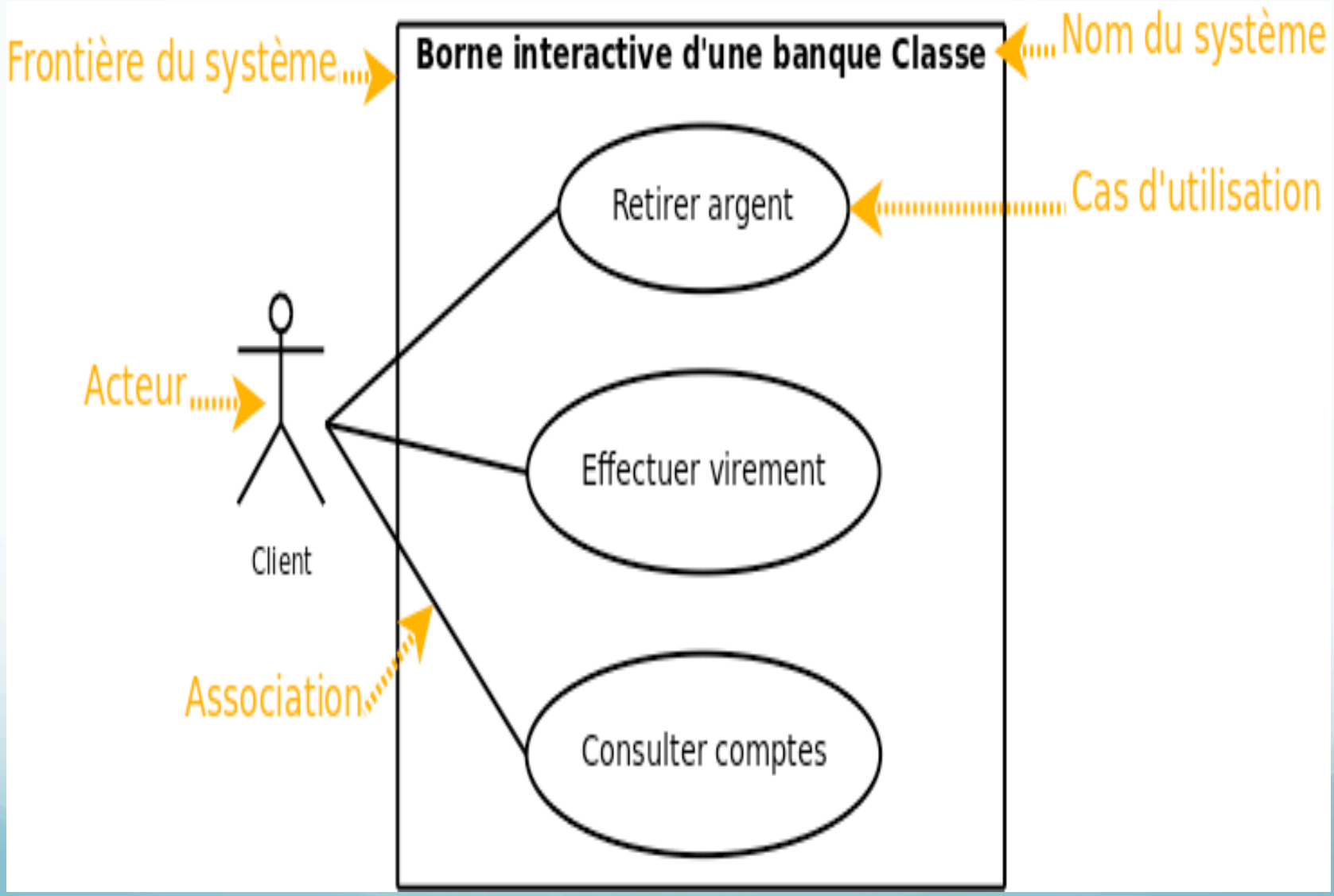
- Un système représente une application dans le modèle UML. Il est identifié par un nom et regroupe un ensemble de cas d'utilisation qui correspondent aux fonctionnalités offertes par l'application à son environnement
- L'environnement est spécifié sous forme d'acteurs liés aux cas d'utilisation
- Un système se représente par un rectangle contenant le nom du système et les cas d'utilisation de l'application.

Système (suite 1)

Les acteurs, extérieurs au système, sont représentés et reliés aux cas d'utilisation qui les concernent. L'ensemble correspond à un diagramme de cas d'utilisation.

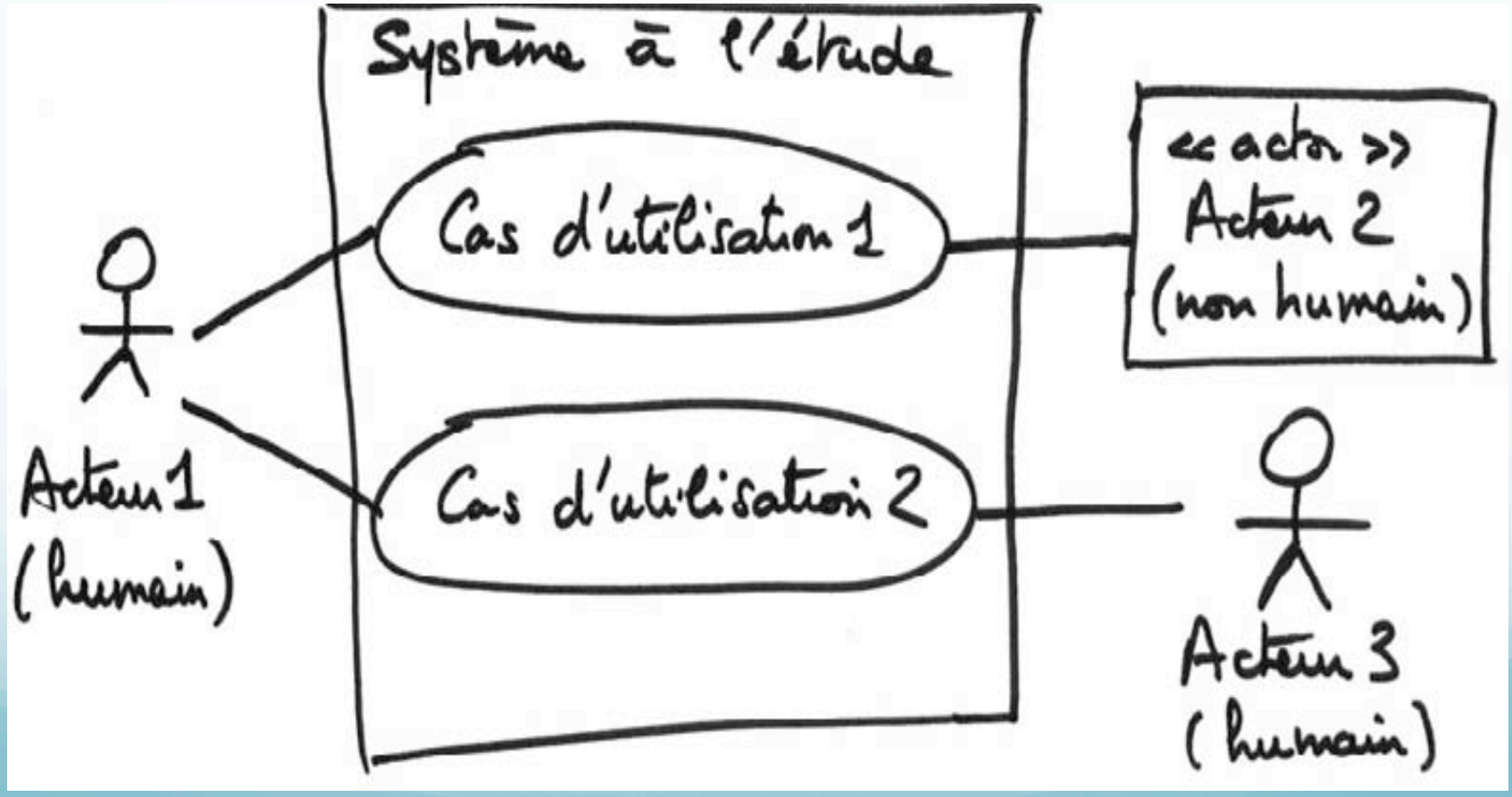
Exemple1: La figure suivante représente un exemple simplifié de diagramme de cas d'utilisation modélisant une borne d'accès à une banque avec les cas d'utilisation offert à l'acteur qui représente le client.

Système (suite 2)



Système (suite 3)

- **Exemple 2:** Interactions fonctionnelles entre les acteurs

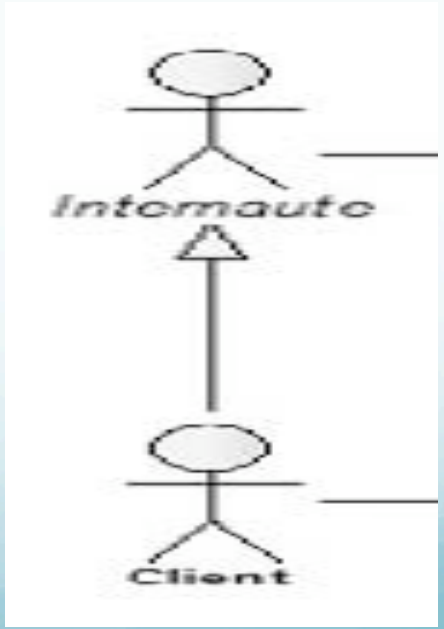


Relations dans les diagrammes de cas d'utilisation

Relations entre Acteurs

La seule relation possible entre deux acteurs est la généralisation : un acteur A est une généralisation d'un acteur B si l'acteur A peut être substitué par l'acteur B. Dans ce cas, tous les cas d'utilisation accessibles à A le sont aussi à B, mais l'inverse n'est pas vrai.

Le symbole utilisé pour généralisation entre acteurs est une flèche avec un trait plein dont la pointe est un triangle fermé désignant l'acteur le plus général



Relations entre cas d'utilisation

Il existe principalement deux types de relations :

- Les dépendances stéréotypées, qui sont explicitées par un stéréotype (les plus utilisés sont l'inclusion et l'extension).
- La généralisation/spécialisation.

Une dépendance se représente par une flèche avec un trait pointillé. Si le cas A inclut ou étend le cas B, la flèche est dirigée de A vers B.

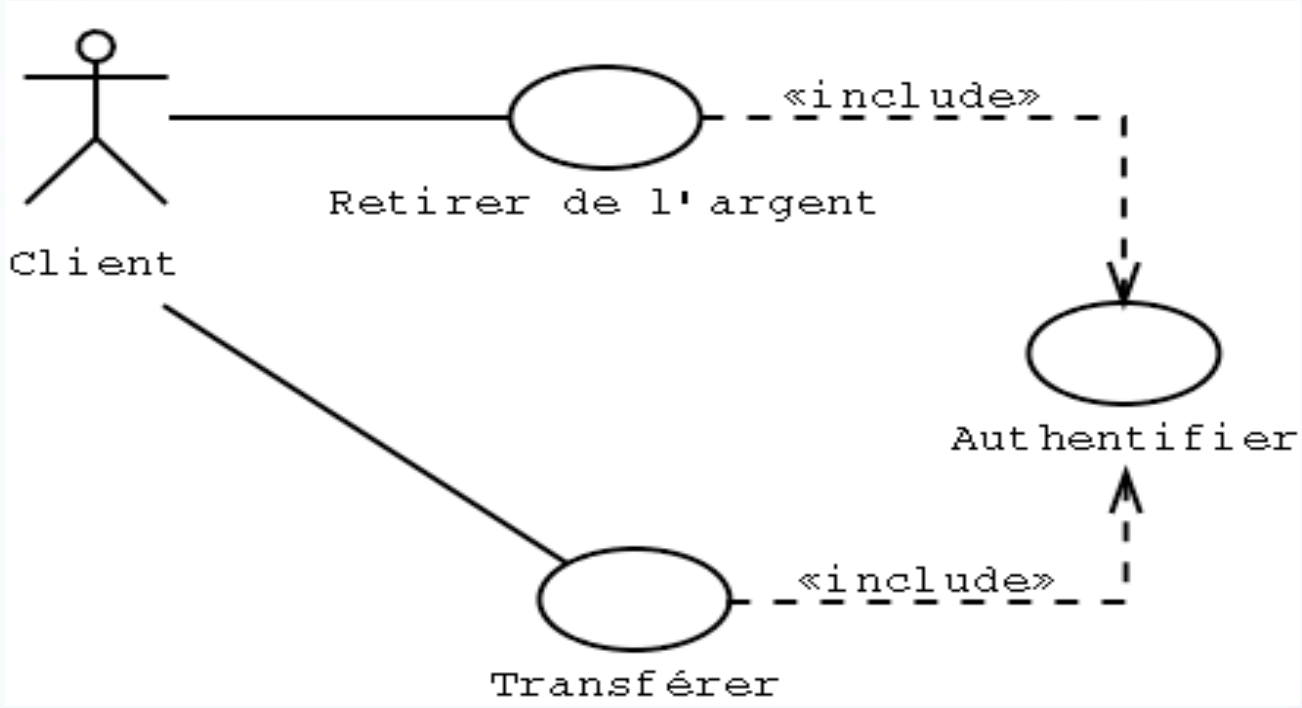
Le symbole utilisé pour la généralisation est un flèche avec un trait pleins dont la pointe est un triangle fermé désignant le cas le plus général

Relations d'inclusion

- Un cas A inclut un cas B si le comportement décrit par le cas A inclut le comportement du cas B : le cas A dépend de B. Lorsque A est sollicité, B l'est obligatoirement, comme une partie de A. Cette dépendance est symbolisée par le stéréotype << include >>.

Exemple: Une opération de retrait et une opération de transfert nécessitent toutes deux une opération de vérification de l'identité du client. Effectuer un retrait inclut nécessairement une phase d'authentification.

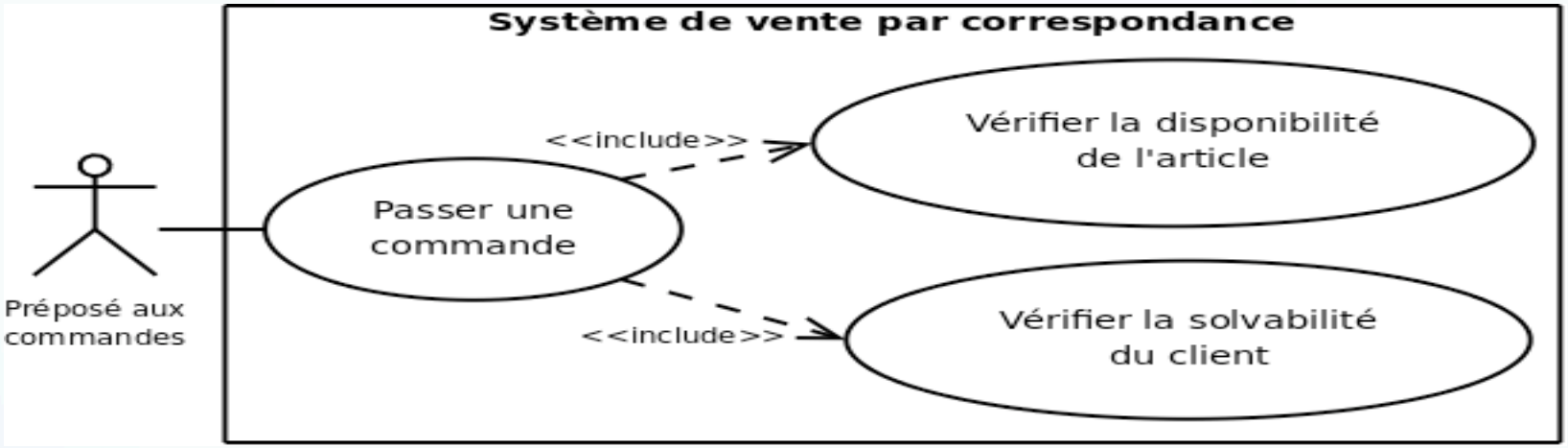
Relations d'inclusion (suite 1)



Les inclusions permettent essentiellement de factoriser une partie de la description d'un cas d'utilisation qui serait commune à d'autres cas d'utilisation (le cas S'authentifier)

Relations d'inclusion (suite 2)

Les inclusions permettent également de décomposer un cas complexe en sous-cas plus simples:



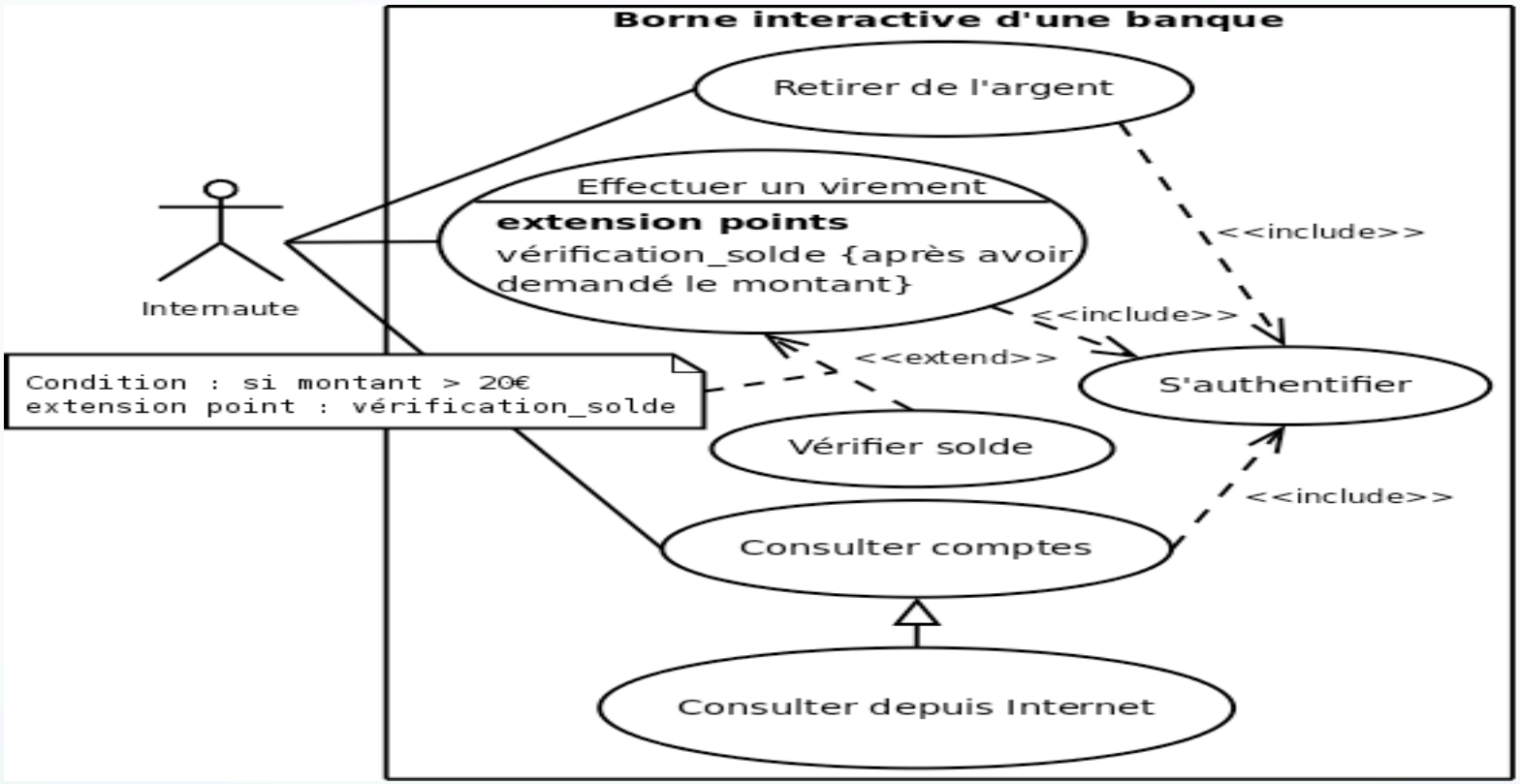
Cependant, il ne faut surtout pas abuser de ce type de décomposition : il faut éviter de réaliser du découpage fonctionnel d'un cas d'utilisation en plusieurs sous-cas d'utilisation pour ne pas retomber dans le travers de la décomposition fonctionnelle.

Relation d'extension

La relation d'extension est probablement la plus utile car elle a une sémantique qui a un sens du point de vue métier au contraire des deux autres qui sont plus des artifices d'informaticiens.

On dit qu'un cas d'utilisation A étend un cas d'utilisation B lorsque le cas d'utilisation A peut être appelé au cours de l'exécution du cas d'utilisation B. Exécuter B peut éventuellement entraîner l'exécution de A : contrairement à l'inclusion, l'extension est optionnelle. Cette dépendance est symbolisée par le stéréotype << extend >>

Relation d'extension (suite 1)



L'extension peut intervenir à un point précis du cas étendu. Ce point s'appelle le point d'extension. Une extension est souvent soumise à une condition. Graphiquement, la condition est exprimée sous la forme d'une note.

Dans l'exemple d'une banque où la vérification du solde du compte n'intervient que si la demande de retrait dépasse 20 euros.

Relation de généralisation

Définition: Un cas A est une généralisation d'un cas B si B est un cas particulier de A. Dans la figure précédente, la consultation d'un compte via Internet est un cas particulier de la consultation.

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.