- **Semestre 2** R2.01 Développement orienté objets : UML
- **Semestre 3** R3.03 Analyse : UML

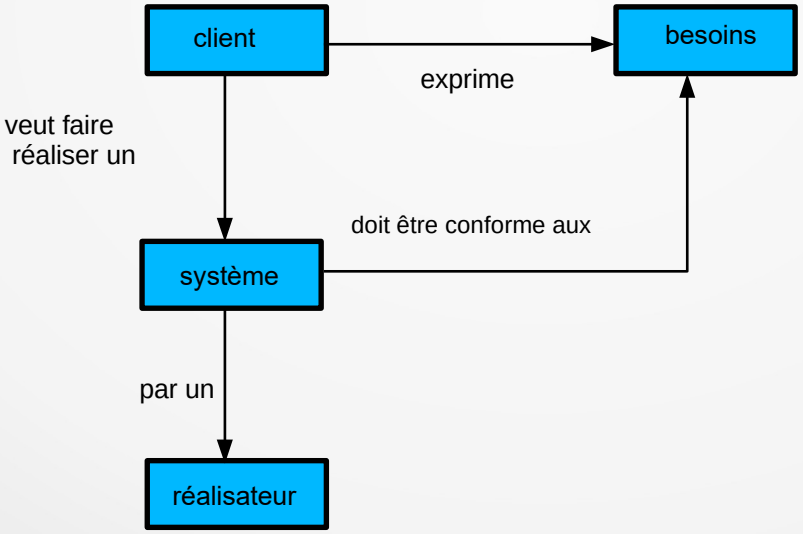
CM N°1	**Système et Modèle**	**UML**	**Cas d'utilisation**	
CM N°2	Déploiement	Classes		
CM N°3	Dépendances	Associations		
CM N°4	Code et dépendances	Code et associations		
CM N°5	Héritage	Stéréotypes	Abstraites	Interfaces
CM N°6	Rappels: Cas d'utilisation, classes, associations			
CM N°7	Rappels : Héritage, abstraites, interfaces			
CM N°8	Objets	Communication	Séquences	
CM N°9	Fragments	Scénarios	Séquences et code	
CM N°10	États-Transitions	Activités	Paquets	

R2.01 Développement orienté objets
UML Cours N°1
Sommaire

E.Porcq : R2.01 UML
Département : IUT Caen Informatique
Année universitaire : 2024-2025
Sources bibliographiques :
-Cours "M2104 Approche qualité" de P.Brutus
- Cours de Laurent AUDIBERT

1) Systèmes et modélisation

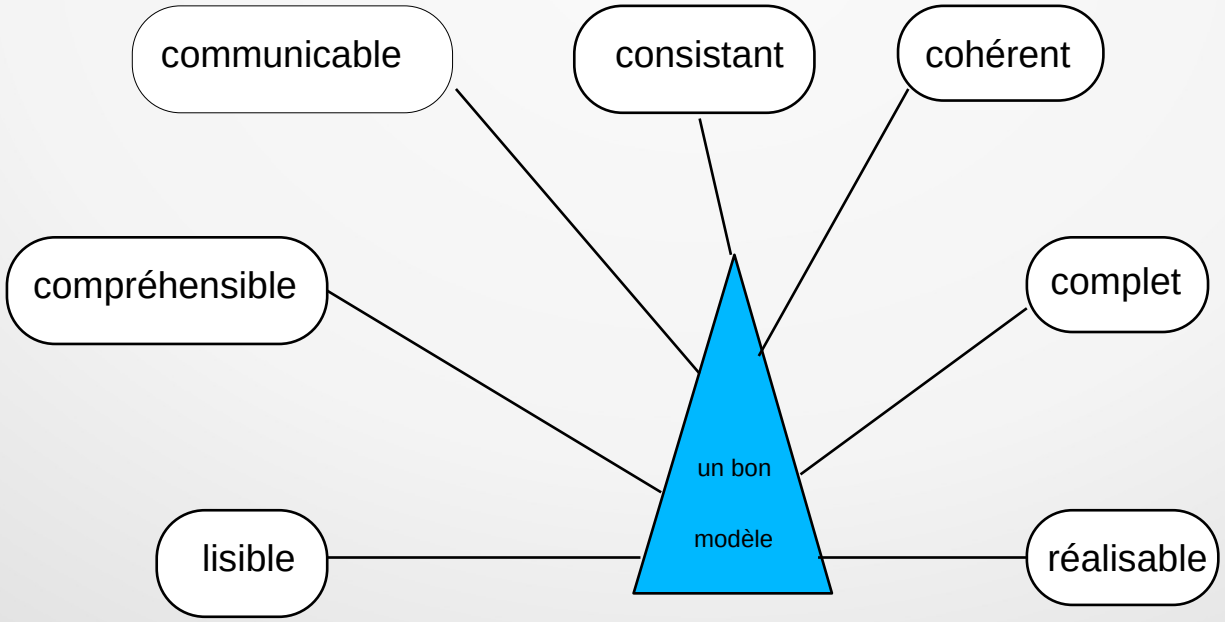
- La réalisation d'un projet industriel nécessite l'intervention de plusieurs personnes (un client, un utilisateur, un prescripteur, un demandeur, un commanditaire, un donneur d'ordre, un maître d'ouvrage, un chef de projet).
- Pour simplifier, on parlera de client, de réalisateur et de chef de projet



- La demande du client est (doit être) formalisée par écrit. Il exprime des besoins.

2) La notion de modèle

- Un modèle est une image de la réalité, expression d'un besoin que l'on cherche à appréhender.
- Il est une représentation du réel et doit posséder plusieurs propriétés. Il doit être lisible, compréhensible, communicable. Il doit être porteur de sens
- Il est souvent préférable d'utiliser un langage graphique



3) Phases de développement

- Le développement du système comporte les phases de spécification , de conception , de construction , d'intégration et de validation . L'ensemble de ces phases porte le nom de **cycle de développement du système**.
- Il existe plusieurs types d'organisation du développement
  - Le cycle en cascade
  - Le cycle en V
  - Les méthodes agiles
- (Voir cours de gestion de projet)

4) Les méthodes de conception

- Dans les années 70, on utilisait des méthodes d'analyse et de conception qui venaient en amont du développement
- Les plus connues sont
  - SA ou SADT
  - MERISE (France)
- Elles obéissaient au principe de cycle de développement linéaire (ex cycle en V)
- Avec l'avènement de la programmation objet, d'autres modèles ont vu le jour
  - OMT (object modeling technique)
  - Booch
  - OOSE
- Ces 3 notations fusionnent en 1997 pour donner UML

5) UML

- UML propose une notation standard
  - Normalisation de la sémantique des concepts de l'approche par objets
  - Expression visuelle d'une solution (à objets) qui facilite la comparaison et l'évaluation
  - Indépendance aux langages de programmation
- UML est une notation, pas une méthode
- UML propose différents points de vue sur le S.I avec différents diagrammes.
  - chacun est libre d'utiliser les diagrammes qu'il souhaite, dans l'ordre qu'il veut, à condition qu'ils soient cohérents les uns par rapport aux autres
  - des diagrammes de structure : aspects liés à l'architecture
    - Quels seront les différents composants à utiliser (matériels ou logiciels) ?
    - Sur quel matériel chacun des composants sera installé ?
  - des diagrammes de comportement
    - Qui utilisera le système et pour quoi faire ?
    - Comment les actions devront-elles se dérouler ?
    - Quelles informations seront utilisées pour cela ?

6) Les différents diagrammes

- Diagrammes de structure
  - Diagramme de déploiement (à voir)
  - Diagramme de composant (à voir)
  - Diagramme de classes (à voir)
  - Diagramme d'objets (à voir)
  - Diagramme de paquetages (à voir)
  - Diagramme de structure composite
- Diagrammes de comportement
  - Diagramme de cas d'utilisation (à voir lors de ce cours)
  - Diagramme d'activités (à voir en partie en TD PL/SQL)
  - Diagramme états-transitions (à voir)
  - Diagramme d'interaction
    - Diagramme de séquences (à voir)
    - Diagramme global d'interaction
    - Diagramme de communication (à voir)
    - Diagramme de temps

7) Les diagrammes de cas d'utilisation

- Les cas d'utilisation (use cases) ont été formalisés par Ivar Jacobson. Ils viennent combler un manque dans OMT-1 et Booch 91.
- Ils décrivent le système du point de vue de **l'utilisateur**.
- Ils permettent de définir les **limites** du **système** et les relations entre le système et **l'environnement**.
- Les diagrammes de cas d'utilisation (dcu) doivent être formalisés autour du langage **naturel**;
- Un cas d'utilisation est une manière spécifique d'utiliser un système.
- Les diagrammes de cas d'utilisations montrent les interactions entre les **acteurs** et les **cas d'utilisation** du système.
- Il doit être lisible par tous (sans formation importante); par analystes, les développeurs, les clients.
- Ils tentent d'éviter aux développeurs des dérives par rapport à l'expression des **besoins**.
- C'est un diagramme de comportement

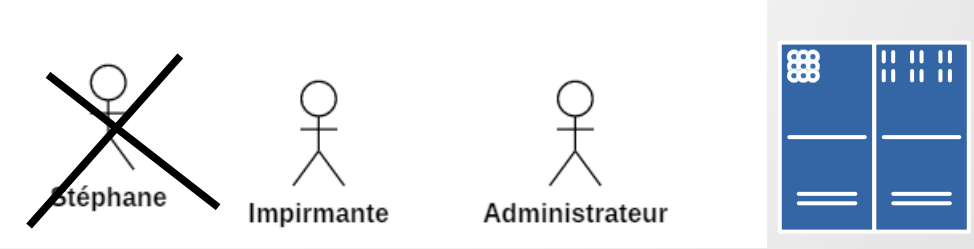
R2.01 Développement orienté objets

Cours N°1 : UML : Les diagrammes de cas d'utilisation

Sommaire

7-1) Les acteurs

- Ce sont des **rôles**, pas des **objets**
- Ils sont les **utilisateurs** du système (clients, pupitreurs,opérateurs, imprimante, routeur...),
- ils ne **font pas partie** du système (sauf exception d'acteurs internes mais cette notion est discutable),
- ils **interagissent** avec le système,
- le même objet peut jouer plusieurs **rôles**,
- ils sont symbolisés graphiquement par petit bonhomme.
- Ils peuvent être eux-mêmes un autre système d'où on pourrait réaliser un dcu.



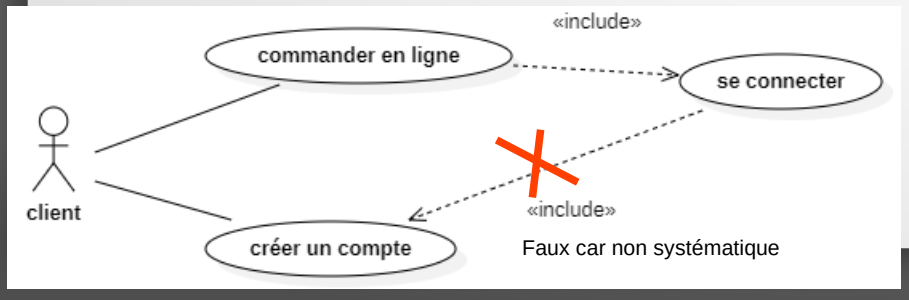
- On peut considérer qu'il existe 2 catégories d'acteurs :
  - les acteurs **principaux**. Ils utilisent les fonctions principales du système
  - les acteurs **secondaires**. Ils utilisent les fonctions secondaires du système (tâches administratives,maintenance, options ...)

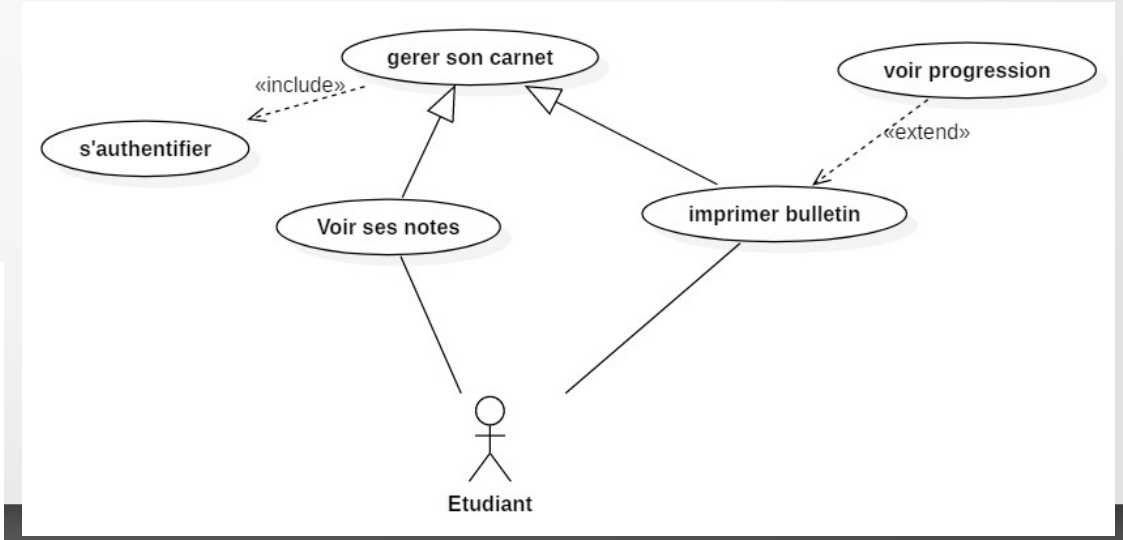
7-2) Les cas d'utilisation

- ils interagissent avec les acteurs ou d'autres cas d'utilisations
- ils regroupent une **famille de scénarios**
- Il faut voir les cas d'utilisations comme des classes dont les instances sont les scénarios.
- Les cas d'utilisation se décrivent se que le système **peut faire** (mais pas comment).
- Les cas contiennent des verbes d'action suivi d'un complément. On évitera donc : être, avoir, rester ...
- **On ne peut trouver 2 cas identiques dans un même diagramme**
- Il faut se placer du point de vue de l'acteur et non pas de celui du système
  ils sont symbolisés graphiquement par des ellipses contenant le nom du cas (un verbe à l'infinitif),
- On doit pouvoir dire : L'acteur **peut** <nom du cas>

7-3) Relation entre les cas

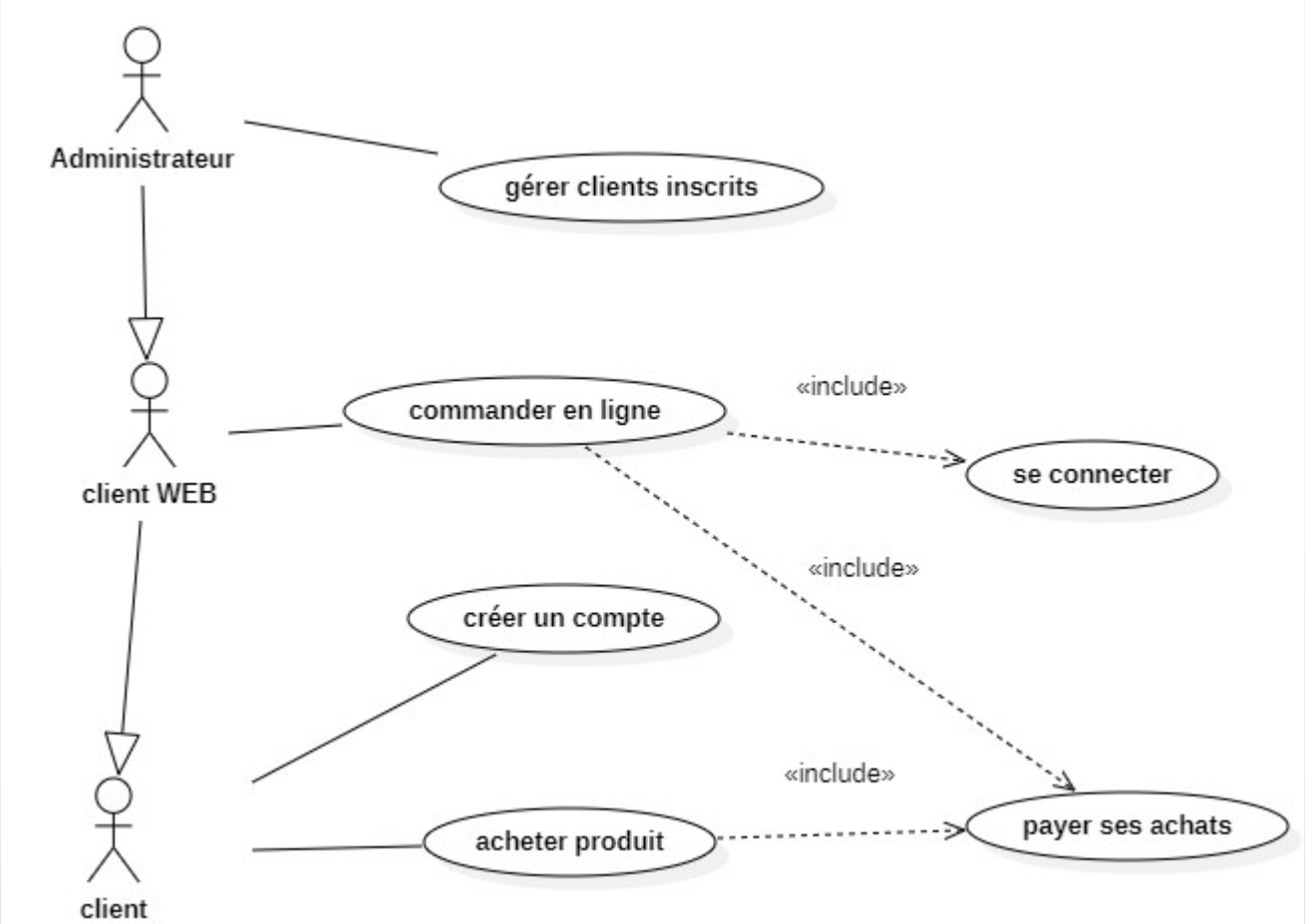
- UML définit trois types de relations entre acteurs et cas d'utilisation :
  - la relation de **communication**. La participation de l'acteur est signalée par un lien entre l'acteur et le cas d'utilisation.
  - la relation **d'inclusion**. Elle signifie qu'une instance du cas d'utilisation source **nécessite** également le comportement décrit par le cas d'utilisation destination. Lorsque A est sollicité, B l'est obligatoirement, comme une partie de A.
  - la relation **d'extension**. Elle signifie qu'une instance du cas d'utilisation source étend (enrichit) également le comportement décrit par le cas d'utilisation destination. Contrairement à l'inclusion, l'extension est **optionnelle**.
  - la relation de **généralisation**. Elle permet de regrouper des cas particuliers (**abstraits**). Attention, ces cas particuliers restent des cas et non des scénarios.





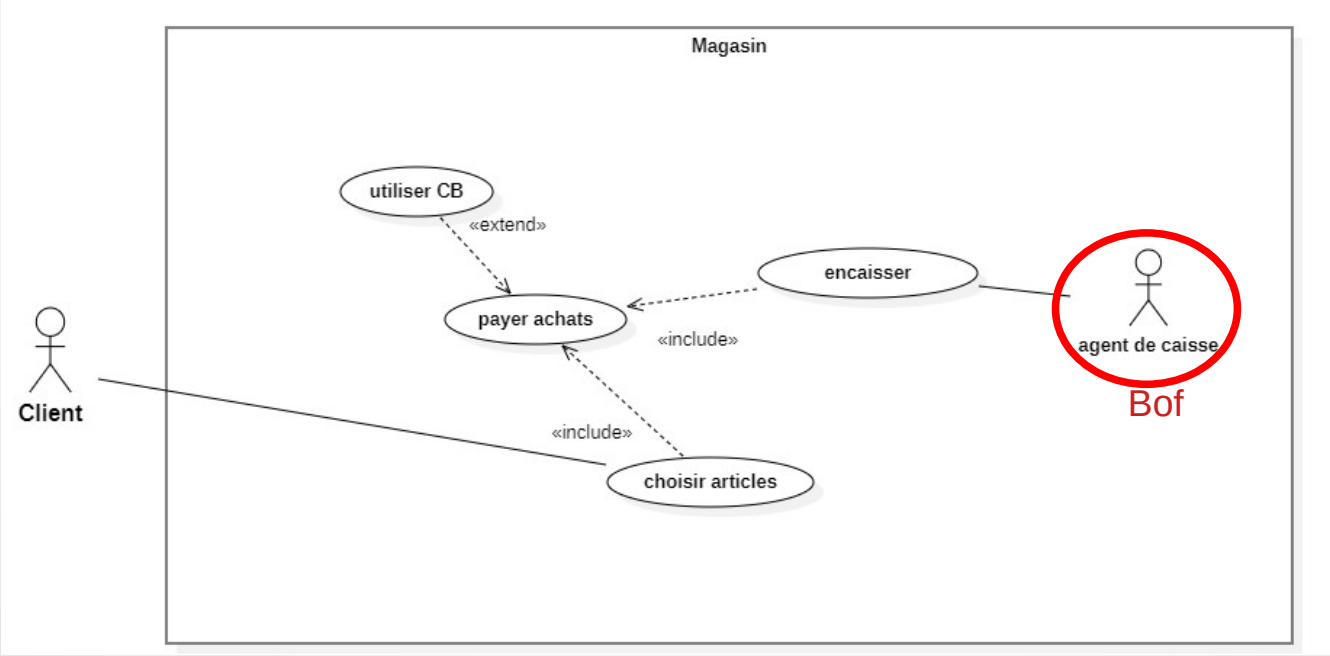
7-4) Relation entre les acteurs

- la relation de généralisation. Elle permet de regrouper des acteurs qui peuvent hériter du comportement d'un autre



7-5) Limites d'un système

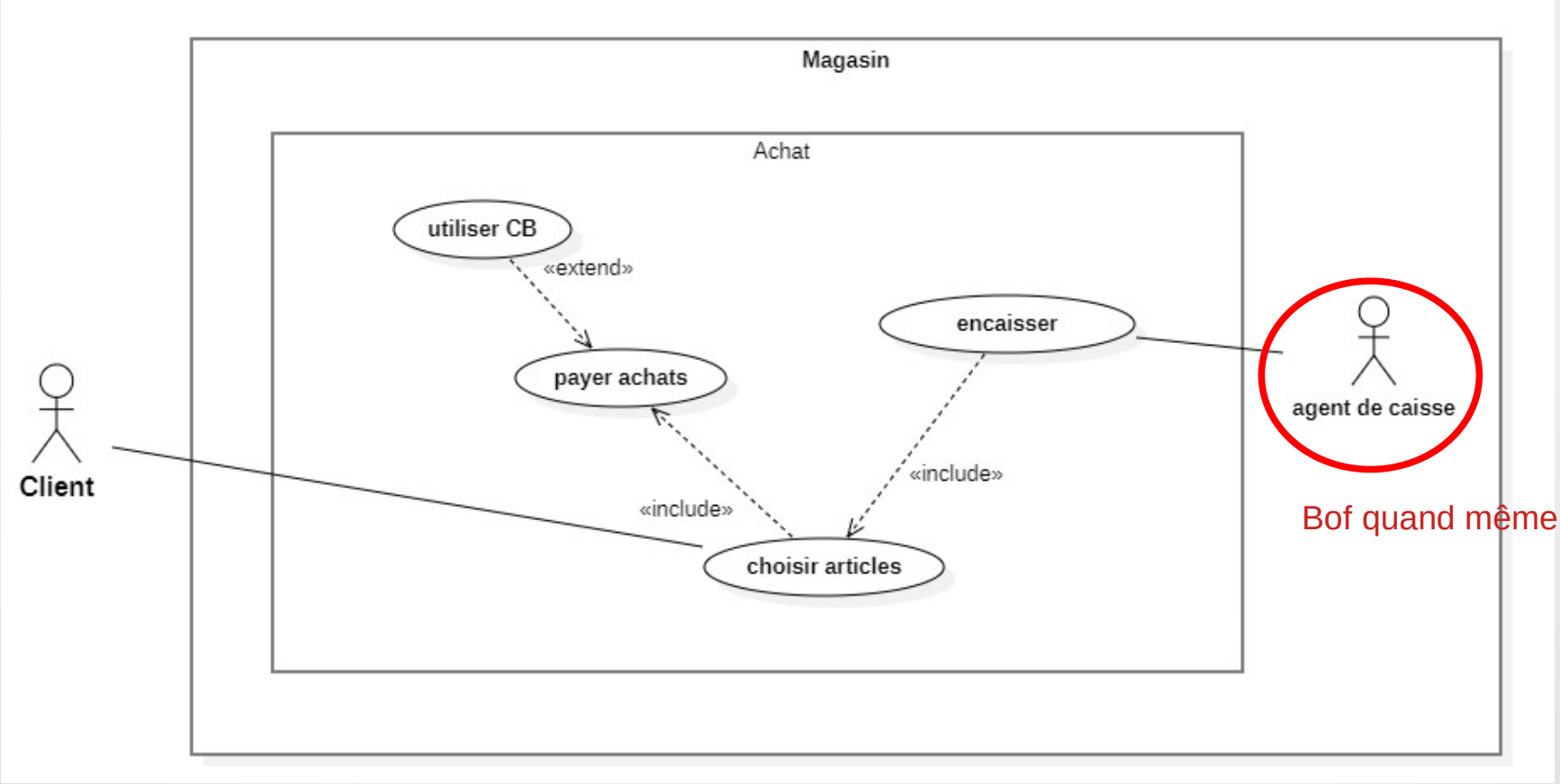
- La limite d'un système est un **rectangle** que vous pouvez dessiner dans un diagramme de cas d'utilisation pour séparer les cas d'utilisation appartenant au système et les acteurs extérieurs. Cette limite est une aide visuelle **facultative** pour le diagramme : elle n'apporte aucune valeur sémantique au modèle.



- Il est possible d'ajouter des acteurs dans le rectangle mais cela reste discutable car normalement **un acteur est en dehors du système**

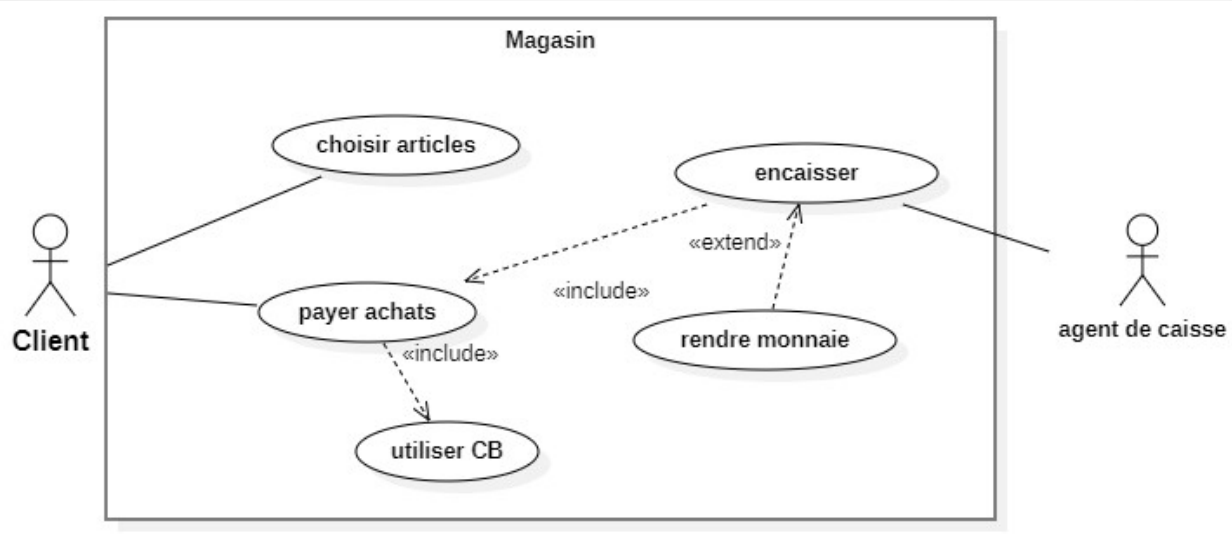
7-5) Limites d'un système

- On pourrait considérer qu'un acteur interne est en fait l'acteur d'un système plus petit



7-5) Limites d'un système

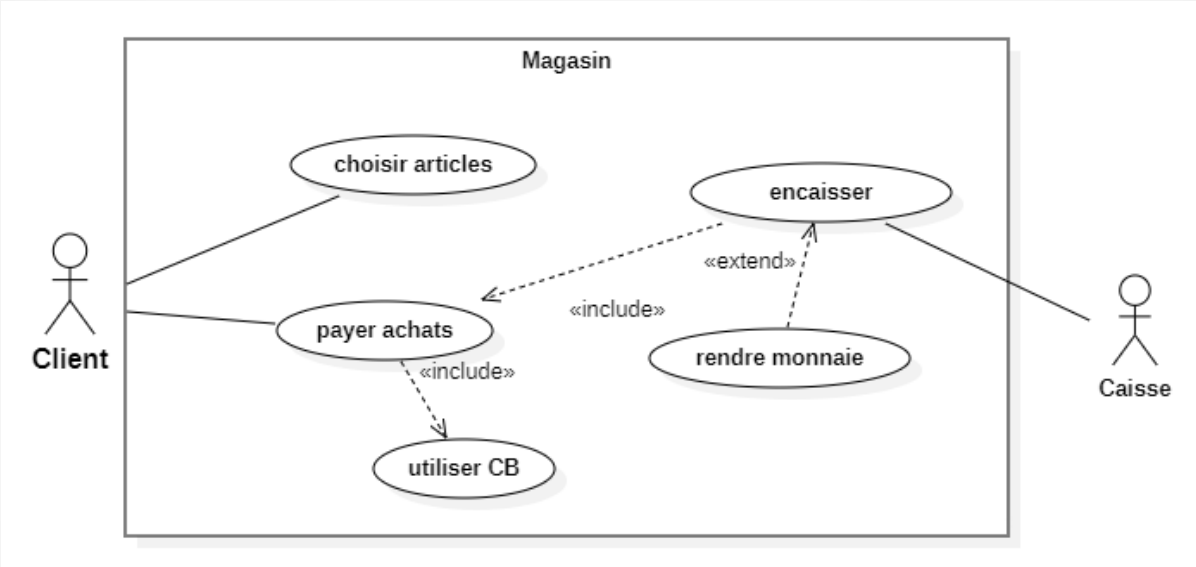
- L'agent de caisse peut-il est considéré comme un acteur ?



- Quand on retire un acteur, le système reste complet et fonctionnel. Mais une partie du système ne sert plus à rien.
- Sans agent de caisse, ce magasin reste complet mais ne permet plus d'encaisser les achats. L'agent est donc acteur du système.
- Sans Client, c'est toujours un magasin.
- La caisse, fait partie du magasin.

7-5) Limites d'un système

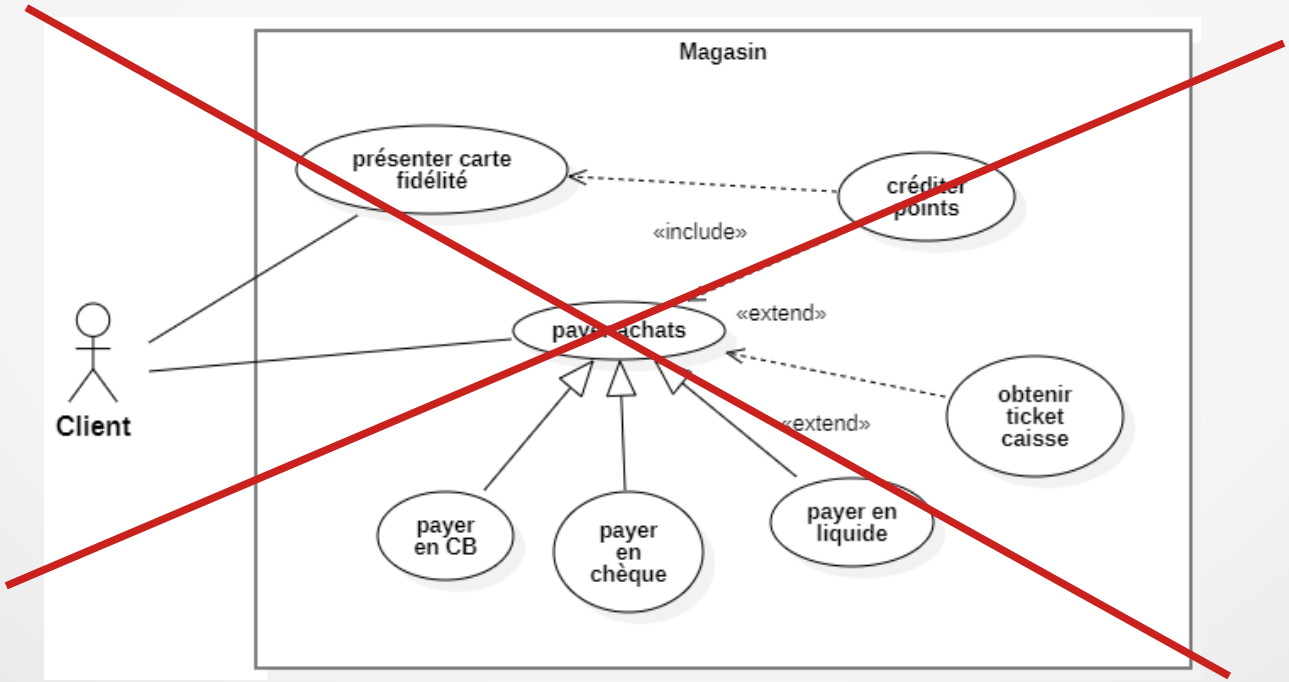
- L'agent de caisse peut-il est considéré comme un acteur ?



- Quand on retire un acteur, le système reste complet et **fonctionnel**. Mais une partie du système **ne sert plus** à rien.
- Sans caisse, ce magasin n'est plus complet. Il n'est plus fonctionnel.
- La caisse n'est donc pas un acteur du système.
- La caisse, fait partie du magasin. On aurait pu placer un acteur interne. Mais on peut s'en dispenser je l'ai déjà dit

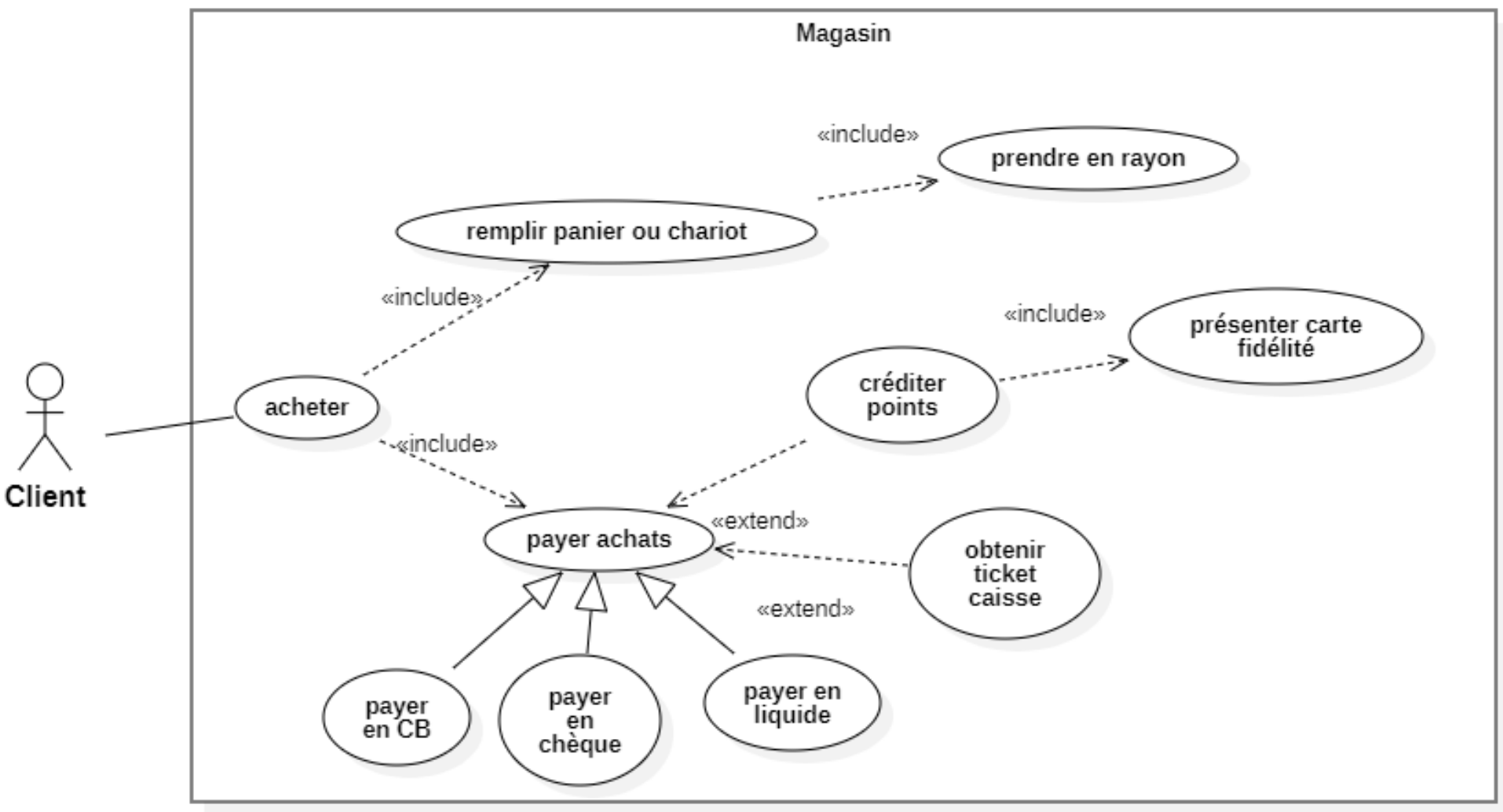
7-6) Cas principaux, cas secondaires

- Un cas principal est directement relié à un acteur. Il correspond à un objectif principal du système pour cet acteur.
- Un cas secondaire (ou interne) est relié à un autre cas.



- Dans cet exemple, on comprend que l'objectif principal d'un client de magasin est de payer ses achats et de présenter sa carte de fidélité ce qui est absurde

7-6) Cas principaux, cas secondaires



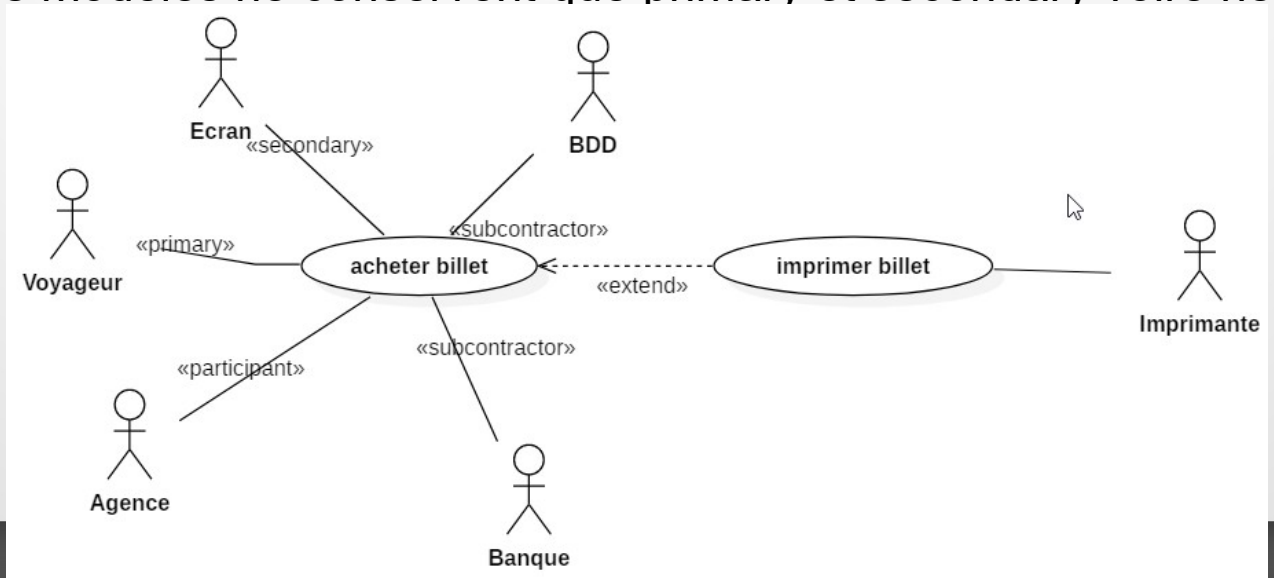
- Ici, seul « acheter » est un cas principal

7-7) Spécification d'une association Acteur-Cas

- Lorsqu'un acteur peut interagir plusieurs fois avec un cas d'utilisation, il est possible d'ajouter une multiplicité sur l'association du côté du cas d'utilisation
- Un acteur est qualifié de **principal** pour un cas d'utilisation lorsque ce cas rend service à cet acteur. Les autres acteurs sont alors qualifiés de **secondaires**. Un cas d'utilisation a au plus un acteur principal. Un acteur principal obtient un résultat observable du système tandis qu'un acteur secondaire est sollicité pour des informations complémentaires. En général, l'acteur principal initie le cas d'utilisation par ses sollicitations. Le stéréotype « primary » vient orner l'association reliant un cas d'utilisation à son acteur principal, le stéréotype « secondary » est utilisé pour les acteurs secondaires

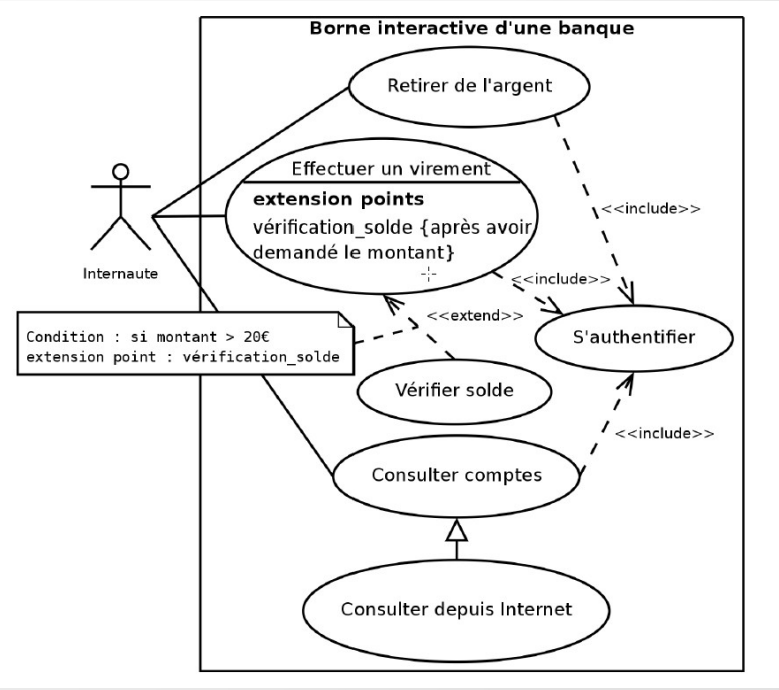
7-7) Spécification d'une association Acteur-Cas

- Les différents stéréotypes sont
  - <<primary>> : L'acteur lié au cas est initiateur de sa réalisation. Le cas lié lui rend service.
  - <<subcontractor>> : L'acteur lié est sollicité (comme un sous-traitant) par le cas pour en permettre la réalisation,pour des informations complémentaires… Il est nécessaire mais intervient "en second"
  - <<participant>> : L'acteur lié intervient dans la réalisation du cas, il participe directement à sa mise en œuvre.
  - <<secondary>> : L'acteur lié est sollicité par le cas ou intervient dans sa réalisation.
- La plupart des modèles ne conservent que primary et secondary voire ne spécifient rien.

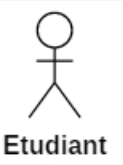


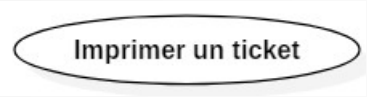
7-8) Point d'extension

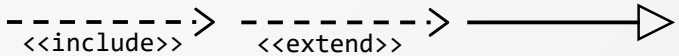
- L'extension peut intervenir à un point précis du cas étendu. Ce point s'appelle le point d'extension. Il porte un nom, qui figure dans un compartiment du cas étendu sous la rubrique point d'extension, et est éventuellement associé à une contrainte indiquant le moment où l'extension intervient. Une extension est souvent soumise à condition. Graphiquement, la condition est exprimée sous la forme d'une note.
- Il ne faut pas abuser de cette extension (**et même l'éviter si possible**) car on risq de compliquer le diagramme et de dépasser l'objectif des diagrammes de cas d'utilisation.
- Rappel : un dcu répond à la question
  " A quoi cela sert "
  et pas " Comment ça marche "

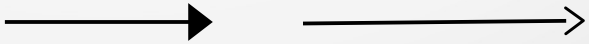


7-9) Norme

- Les éléments en UML ont une représentation précise qu'il convient de respecter rigoureusement :
  - Un acteur 

  - Un cas 

  - Une association acteur-cas
  - Une association cas-cas 

  - Une association acteur-acteur 
  - Rien du tout (faux)  **-1000**
  - Faux