# Analyse, Conception des Systèmes Informatiques

## Diagrammes de cas d'utilisation "Use Case"


O. Boissier, SMA/G2I/ENS Mines Saint-Etienne, Olivier.Boissier@emse.fr, **Septembre 2004**

## Sommaire

- Introduction
- Acteurs
- Cas d'utilisation
- Relations
- Diagramme de cas d 'utilisation





## Objectifs

- **Cas d'utilisation**
- Séquences
- Collaboration
- Classes
- Objets
- États/transitions
- Activités
- Composants
- Déploiement

- Développés par Ivar Jacobson.
- Description du système du point de vue de **l'utilisateur**
- Pour mettre en évidence les services rendus par le système
- Pour fixer le périmètre entre le système et son environnement



## Comportement du système

- Quelles fonctionnalités doivent être fournies par le système ?
- Les cas d'utilisation:
  - Les utilisations du système (**cas d'utilisation**)
  - Les limites (**les acteurs**)
  - Les relations entre les cas et les acteurs (**diagramme de cas d'utilisation**).
- Les cas d'utilisation servent à communiquer autant pour les usagers que pour les concepteurs, pour spécifier les besoins.

# Acteurs

- Les acteurs n 'appartiennent pas au système, MAIS ils interagissent avec celui-ci.
  - Ils fournissent de l'information en entrée.
  - et/ou reçoivent de l'information en sortie.
- Bien identifier les acteurs admissibles, et évaluer comment ils vont interagir avec le système.
- Les acteurs ne sont pas forcément des personnes.
- Possibilité de spécialiser des acteurs à partir d'autres acteurs.





# Identification (1)

- Qui est intéressé par un certain besoin ?
- Par qui le système est utilisé dans l'organisation?
- Qui bénéficiera de l'utilisation du système ?
- Qui fournira au système l'information, qui l'utilisera et qui la maintiendra ?
- Qui va supporter et maintenir le système ?
- Quelque chose est il produit automatiquement par le système?





# Identification (2)

- Acteurs principaux
  - personnes qui utilisent les fonctions principales du système
- Acteurs secondaires
  - personnes qui effectuent des tâches administratives ou de maintenance
- Matériel externe
  - dispositifs matériels faisant partie du domaine de l 'application
- Autres systèmes





# Représentation

- Dans UML, l 'acteur est représenté comme suit :


- Le nom de l 'acteur correspond au *rôle* joué par la personne
- Un acteur a un nom et si possible une description

# Cas d'utilisation

- Classe de scénarios :
  - Modélisant un dialogue entre un acteur et le système…
  - Représentant une fonctionnalité offerte par le système.
- Scénario :
  - instance d'un cas d'utilisation
  - séquence de transactions entre le système et l'acteur qui mène à un résultat tangible pour un acteur.
  - Un use case comprend au moins deux scénarios: un où tout se passe bien, et un autre où il y a problème.
- L'ensemble des cas d'utilisation forme toutes les façons dont le système pourra être utilisé.

→ Un Use Case sert a beaucoup de choses : à délimiter le système, à analyser les besoins, à concevoir des interfaces, à distribuer le travail, à préparer les tests.





# Identification

- Quelles sont les tâches de chaque acteur?
- Est-ce qu'un acteur crée, enregistre, modifie, enlève ou consulte une information dans le système?
- Quel cas d'utilisation sera responsable de ces tâches?
- Quels cas d'utilisation supportent ou maintiennent le système?





# Représentation

- Dans UML, le cas d'utilisation est représenté par un ovale:

cas d'utilisation  Inscription à des cours

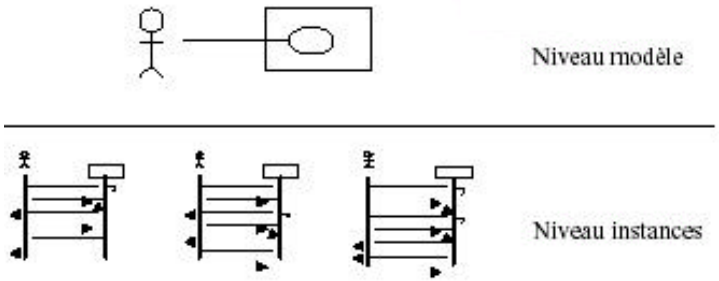
- Un cas d'utilisation peut être décrit par :
  - Un nom unique, des acteurs, des conditions d'entrée, des flots d'évènements (scénarios), des conditions de sortie, des besoins spécifiques,





# Cas d'utilisation vs. Scénario

- Un cas d'utilisation est une famille de scénarios (diag. séquence)



# Exemple

*Nom :* `Achat ticket`

*Acteur :* `Passager`

*Condition entrée :*

- `Passager` devant le distributeur de ticket.
- `Passager` a suffisamment d'argent pour acheter le ticket.

*Condition de sortie :*

- `Passager` a le ticket.

*Scenario :*

1. `Passager` choisit le nombre de zones traversées.
2. Le distributeur affiche la somme due.
3. `Passager` insère l'argent correspondant à au moins la somme due.
4. Distributeur rend la monnaie.
5. Distributeur donne le ticket.





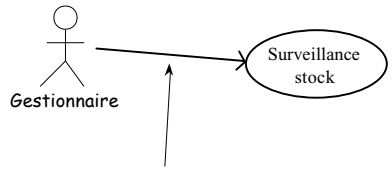
# Relations

- UML définit une relation de communication entre acteur et cas d'utilisation.
- UML définit des types de liens entre les cas d'utilisation pour les structurer :
  - La relation <<extend>>.
  - La relation <<include>>
  - La relation de généralisation





# Relation de communication

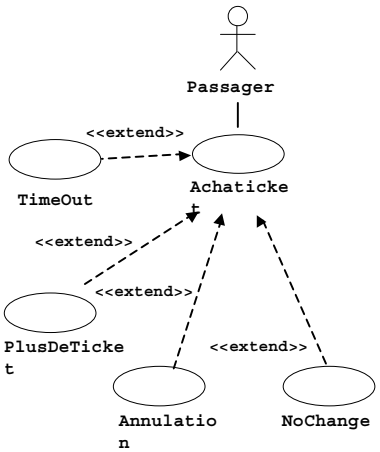



**La participation d'un acteur est représentée par une ligne solide entre l'acteur et le cas d'utilisation.**
**Le sens de la flèche indique l 'initiateur de l 'interaction.**
**C'est la <u>seule relation possible</u> entre un acteur et les cas d'utilisation.**



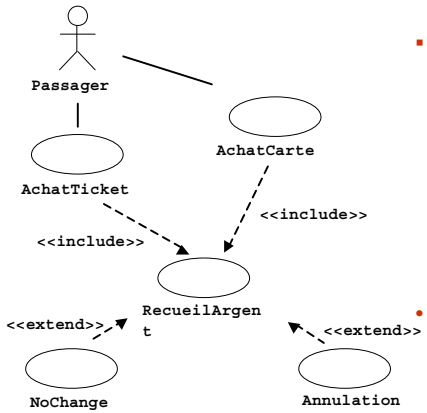


# Relation <<extend>>




- Représentation de cas exceptionnels ou rarement appelés, extensions de comportement.
- Les flots d'événements exceptionnels sont factorisés hors de la séquence principale d'événements pour gagner en clarté
- Les Use cases représentant des flots exceptionnels peuvent étendre plus d'un use case.
- Une condition est associée au cas d'utilisation étendue.
- La relation `<<extend>>` est orientée vers le use case étendu.

# Relation <<include>>




- <<include>> représente un comportement factorisé pour réutilisation

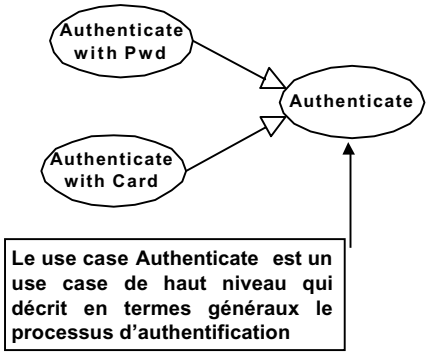
  (pas parce que c'est une exception)

- Orientation <<include>> vers le use case utilisé





# Relation <<generalize>>



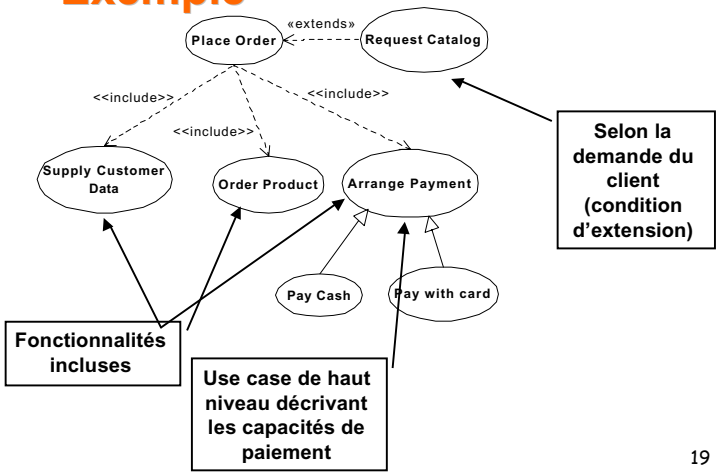

Un cas d'utilisation peut préciser un cas d'utilisation plus général.

La relation est orientée du cas d'utilisation spécialisé vers le cas d'utilisation général





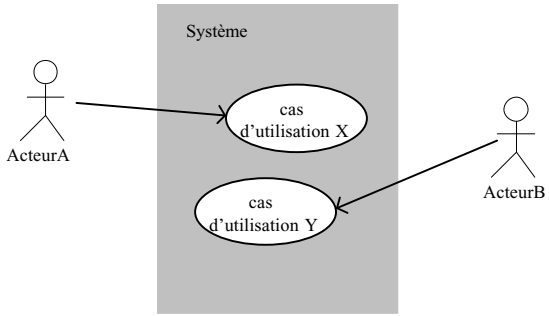
# Exemple



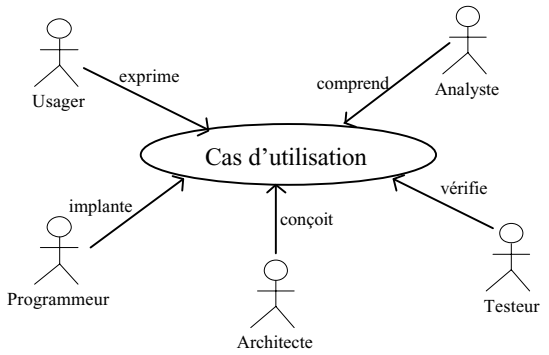





# Diagramme de cas d'utilisation



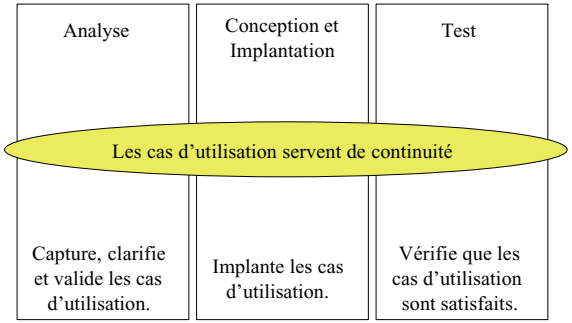
# Diagramme de cas d'utilisation








# Intérêts








# Construction

- Dans un premier temps identifier les cas d'utilisation et les acteurs.
- Décrire ensuite un cas d'utilisation en écrivant des scénarios sous la forme de texte ainsi que de diagrammes d'interactions.
- Faire un sommaire des scénarios principaux et des cas d'utilisation.
- S'assurer finalement de la cohérence.

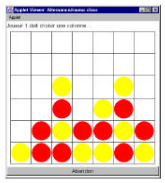




# ATTENTION !

- Les cas d 'utilisation ne sont qu 'une vue externe et non un contrôleur procédural
- Le logiciel est souvent structuré d 'une manière totalement différente des cas d 'utilisation
- Les cas d 'utilisation ne sont qu 'une certaine vision du logiciel !
- Ils servent à spécifier les besoins.

# Exo 1 :



- L'application doit permettre à deux joueurs de faire une partie de Puissance 4. On joue avec un tableau (cf. Fig).
- Règles du jeu : deux joueurs (jaune et rouge) jouent chacun à leur tour. A chaque coup, ils peuvent mettre une de leurs pièces dans une des colonnes du tableau qui n'est pas encore pleine. La pièce tombe dans la case la plus basse qui n'est pas encore occupée dans la colonne. Dès qu'un joueur a aligné soit horizontalement, verticalement ou en diagonale, 4 de ses pièces, il a gagné. Si aucun joueur n'y parvient et que le tableau est rempli, la partie est nulle.



# Exo 2 :

- Le système gère un ensemble (une petite base de données) de mots de passe et de noms d'utilisateur associés à une application.
- Le système permet d'effectuer quatre opérations à savoir la création, la suppression, la modification et la recherche de mots de passe dans la base.
- La base se présente comme un fichier crypté. Ainsi, celui-ci doit être décrypté et ré-encrypté à chaque traitement. L'utilisateur doit fournir la clef correspondante. Celle-ci est donnée une seule fois au lancement de l'utilitaire qui la mémorise jusqu'à la fin du traitement.



# Exo 3 :

- Système de gestion de bibliothèque qui doit permettre à des personnes d'adhérer, d'emprunter des livres. L'inscription est annuelle. Une personne non abonnée ne peut pas emprunter de livres. Un abonné peut à tout moment décider de résilier son abonnement.



# Exo 4 : Gestion des malades dans un cabinet médical

- Un médecin doit pouvoir
  - Créer des dossiers médicaux de patients, les modifier, les consulter et les supprimer, Créer des ordonnances, les modifier, les imprimer et les supprimer, Créer et modifier des instructions à l'attention du secrétariat, Créer et modifier les informations sur le patient (informations non médicales)
- Un secrétaire doit pouvoir
  - Créer et modifier les informations sur le patient (informations non médicales), Éditer une feuille de soin, Imprimer un récépissé lors de l'encaissement d'un paiement, etc.

# Exo 5 :

- Lorsque le navigateur crée une applet, la méthode init() est automatiquement appelée. Le développeur place dans cette méthode les instructions d'initialisation et, en particulier, la prise en compte d'éventuels paramêtres. Juste après l'appel de init(), la méthode start() est automatiquement appelée. La méthode stop() est également automatiquement appelée quand la page HTML contenant l'applet disparaît (changement de page par exemple) ou quand le navigateur est iconifié. Le fait de stopper une applet suspend son exécution, l'applet est dite alors en sommeil et pourra être réactivée par un appel à sa méthode start(). Les méthodes start() et stop() peuvent être appelées plusieurs fois. La méthode paint(), est sollicitée chaque fois que l'applet est découverte. Elle reçoit en argument un contexte graphique g, instance de classe Graphics, spécialisé dans des opérations de dessin. Lorsque le navigateur arrive en fin de vie (fin de la tâche d'exécution liée à la fermeture par l'utilisateur du navigateur par exemple), la méthode destroy() est automatiquement appelée. C'est ici que le développeur place le code nécessaire à la libération de ressources partageables comme, par exemple, la fermeture de fichiers ou la déconnexion à une base de données. Le garbage collector peut alors faire son office.