

Modélisation Avancée Centrée UML

Manzoor AHMAD


Master TI et Big Data 1^{ère} année
Université de Pau et des Pays de l'Adour

manzoor.ahmad@univ-pau.fr

Remerciements

- Ce cours est basé initialement sur les cours de
 - Eric Cariou : <http://ecariou.perso.univ-pau.fr/>
 - Michael Mrissa : <http://mmrissa.perso.univ-pau.fr>
 - Laurence Duchien: <http://www.lifl.fr/~duchien/>
- Spec UML :
<http://www.omg.org/spec/UML/2.5/PDF/>


Plan

- 
- UML (Unified Modelling Language)
 - SysML (System Modelling Language)
 - Le Diagramme des Exigences
 - Evaluation
 - Contrôle Continu 50%
 - Contrôle Terminale 50%

Génie logiciel

- **Objectif du génie logiciel** : produire des logiciels de qualité
- **Qualité**
 - **conforme** (correct) : fait ce qu'il doit faire (comme définit dans sa spécification)
 - **robuste** : capable de réagir à des situations anormales
 - **extensible** : capable de s'adapter à des changements de ses spécifications
 - **réutilisable** : pouvoir être utilisé dans de nombreuses applications
 - **efficace** : économe en ressources
 - **portable, facile à utiliser, compatible, ...**

Génie logiciel

- 
- Spécification du problème
 - Objectif : formaliser les besoins du client
 - Résultat : cahier des charges
 - Modélisation du programme
 - Formelle (maths) ou informelle (langage naturel)
 - Plusieurs approches
 - Chaque approche met en avant des aspects différents

Modèles

► Modèle

► Outil pour la représentation d'un système

► Vue subjective et simplifiée

► **Système** = ensemble d'éléments interagissant entre eux et formant un tout

► Dans notre cas spécifique

- programme = système

► Représentation toujours partielle

- Met en avant certains éléments
- En cache d'autres

► Utilité des modèles

► Faciliter la compréhension d'un système

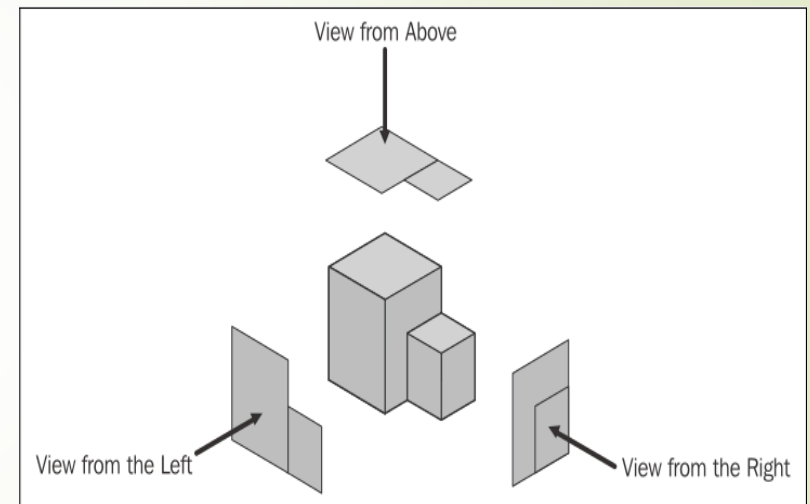
- Permettre également la communication avec le client

- Vision de communication, de documentation

► Définir voire simuler le fonctionnement d'un système

- Dans ce cas, on se doit d'être le plus précis possible dans le contenu des modèles pour s'approcher du code

- Vision de développement, de production



Source : UML 2.0 in action (Packt)

Introduction

- UML : *Unified Modeling Language*
- Normalisé par l'OMG (Object Management Group)
- <http://www.omg.org/spec/UML/>
- Dernière version : 2.5.1 (Déc. 2017)
- Notation standard pour la modélisation d'applications à base d'objets et de composants
- Langage utilisant une notation graphique


Introduction



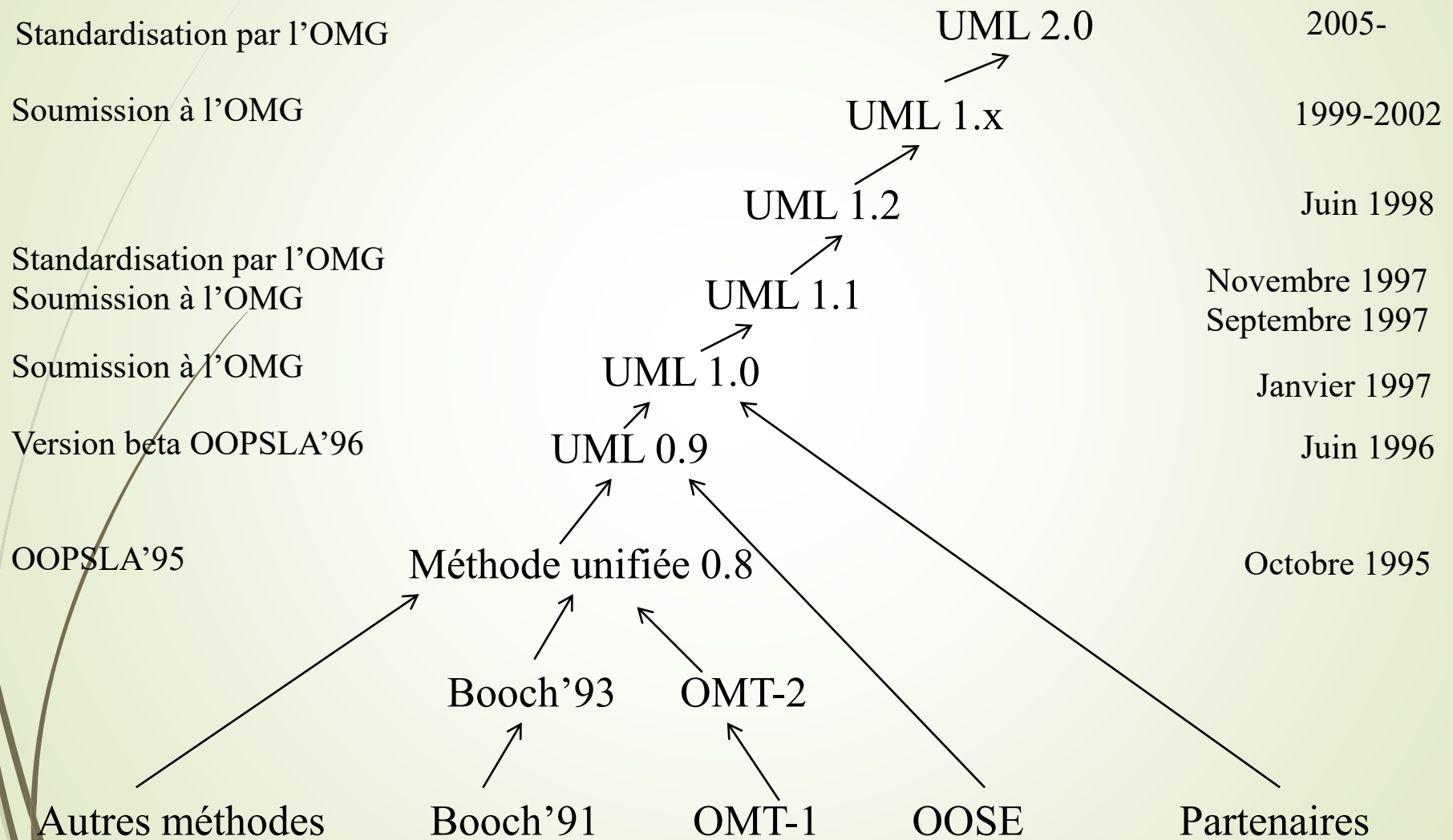
➤ UML : Objectifs

- Modélisation de systèmes pour:
 - Communication, visualisation
 - Vérification (Complet, cohérent, correct)
 - Génération du code (Model-driven engineering)
 - Non pas un modèle unique, mais un ensemble de diagrammes

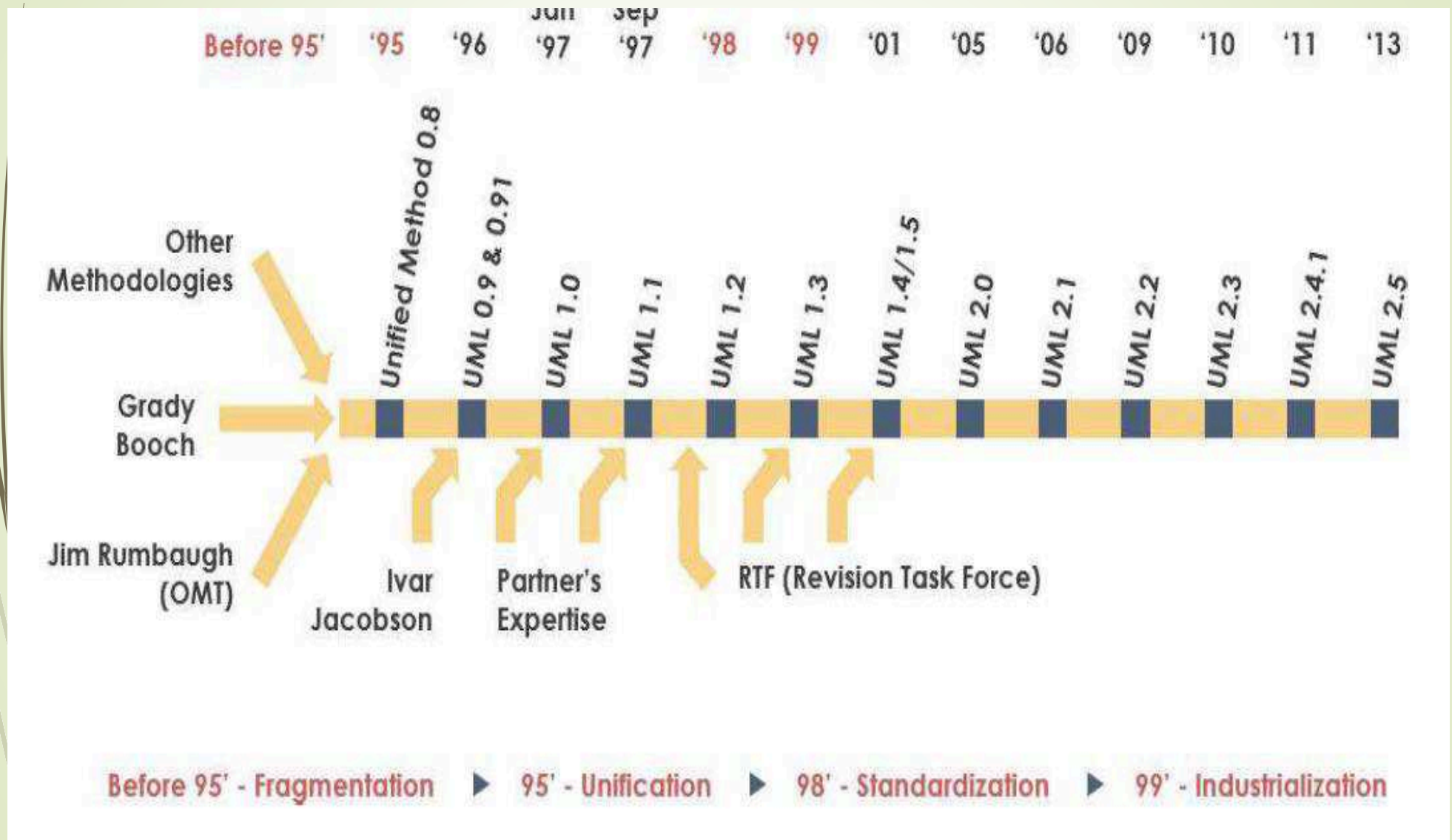
Historique

- 
- UML hérite principalement des méthodes objets de Booch (Booch), OMT (Object Modeling Techniques) de James Rumbaugh et OOSE (Object Oriented Software Engineering) de Ivar Jacobson
 - Mais intègre également d'autres approches, comme les machines à états de Harel
 - But initial
 - Définir un processus/méthode de développement complet (de l'analyse à l'implémentation) orienté objet
 - Problème
 - Pas de notation, langage pour écrire les modèles ou les artefacts définis par ce processus devenu le but final d'UML
 - UML n'est donc pas une méthode ou un processus
 - UML propose un ensemble de notations pour que chacun ait à sa disposition les éléments nécessaires à la conception d'une application


Historique




Historique




Pourquoi UML?

- 
- Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models
 - Provide extensibility and specialization mechanisms to extend the core concepts
 - Be independent of particular programming languages and development processes
 - Provide a formal basis for understanding the modeling language
 - Encourage the growth of the OO tools market
 - Support higher-level development concepts such as collaborations, frameworks, patterns and components
 - Integrate best practices

UML ≠ processus de développement

- 
- UML indépendant du processus de conception et de développement :
 - ne décrit pas comment il fonctionne
 - Exemple de processus de conception et de développement
 - Processus itératif et incrémental
 - Définition du cahier des charges
 - Elaboration du logiciel : cycle de vie à itérer
 - Analyse, Spécification, Implémentation, Test
 - Chaque itération permet l'ajout de fonctionnalités en les définissant, les réalisant, les testant et les intégrant
 - Arrêt du processus itératif lorsque le logiciel produit répond complètement au cahier des charges

UML ≠ processus de développement

- 
- UML fournit une notation/syntaxe pour les diagrammes et modèles définis pendant tout le cycle de développement
 - UML permet de définir des modèles de niveaux différents
 - Analyse
 - Conception
 - Spécification d'implémentation
 -
 - Il faut préciser à quel niveau correspond un modèle
 - On peut raffiner un modèle pour le spécifier à chaque niveau

► Diagrammes structurels

Les 13 diagrammes UML

► De classes (class diagram)

► D'objets (object diagram)

► De composants (component diagram)

► De structure composite (composite structure diagram)

► De déploiement (deployment diagram)

► De paquetages (package diagram)

► Diagrammes de comportement

► De cas d'utilisation (use case diagram)

► D'activité (activity diagram)

► D'états-transition (state diagram)

► Diagrammes d'interaction


► De séquence (sequence diagram)

► Vue générale d'interaction (interaction overview diagram)


► De communication (communication diagram)

► De temps (timing diagram)

Pourquoi 13 diagrammes?

- 
- Why there are a lot of different diagrams (models) in UML?
 - The reason for this is that it is possible to look at a system from many different viewpoints
 - A software development will have many stakeholders playing a part e.g.
 - Analysts
 - Designers
 - Coders
 - Testers
 - QA
 - The Customer
 - Technical Authors

Les diagrammes UML

- 
- Ces diagrammes permettent de définir une application selon plusieurs points de vue
 - **Fonctionnel** (cas d'utilisation)
 - **Statique** (classes, objets, composants, structure composite)
 - **Dynamique** (états, activité, interaction, séquence, communication, temps)
 - **Implémentation** (déploiement, packaging)
 - Les diagrammes seuls ne permettent pas de définir toutes les contraintes de spécification requises
 - Utilisation du langage textuel de contraintes OCL en complément
 - S'applique sur les éléments de la plupart des diagrammes

Plan

- 
- *Diagrammes fonctionnels*
 - *Cas d'utilisation*
 - Diagrammes statiques
 - Diagrammes dynamiques
 - Diagrammes d'implémentation

Diagramme de cas d'utilisation



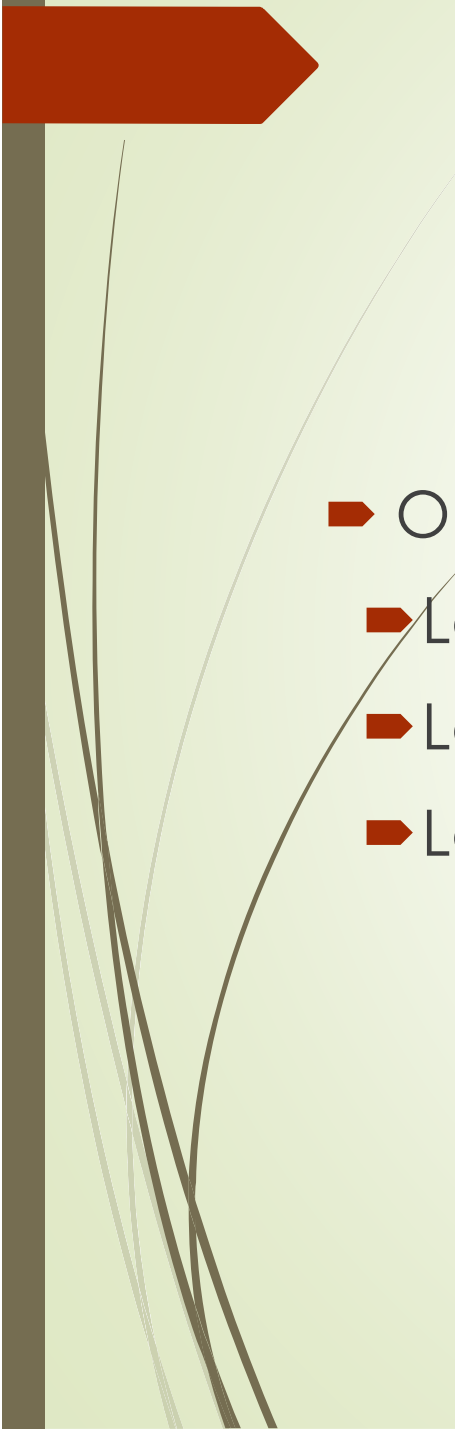
- 
- Description des interactions type entre un utilisateur et le système informatique
 - Définition des cas d'utilisation à partir de discussions avec l'utilisateur sur ses attendus du système
 - Énumération des principaux scénarios prévus
 - Exemple : écriture d'un texte avec un traitement de texte
 - 2 cas d'utilisation : mettre du texte en gras, créer un index
 - Propriétés des cas d'utilisation
 - Déterminer les fonctions visibles pour un utilisateur
 - Prendre en compte les objectifs des utilisateurs
 - De taille quelconque

Diagramme de cas d'utilisation

- 
- Deux grandes approches
 - Objectif de l'utilisateur
 - Interaction du système
 - Exemple : utilisation d'une feuille de style dans un traitement de texte
 - Objectif de l'utilisateur : assurer un formatage cohérent, faire un format de document identique à un autre
 - Interaction du système : définir un style, changer de style, déplacer un style d'un document vers un autre
 - Les interactions du système reflètent ce que l'utilisateur peut faire plus que le but réel de l'application
 - Description d'un cas d'utilisation : de manière informelle, généralement en langage naturel

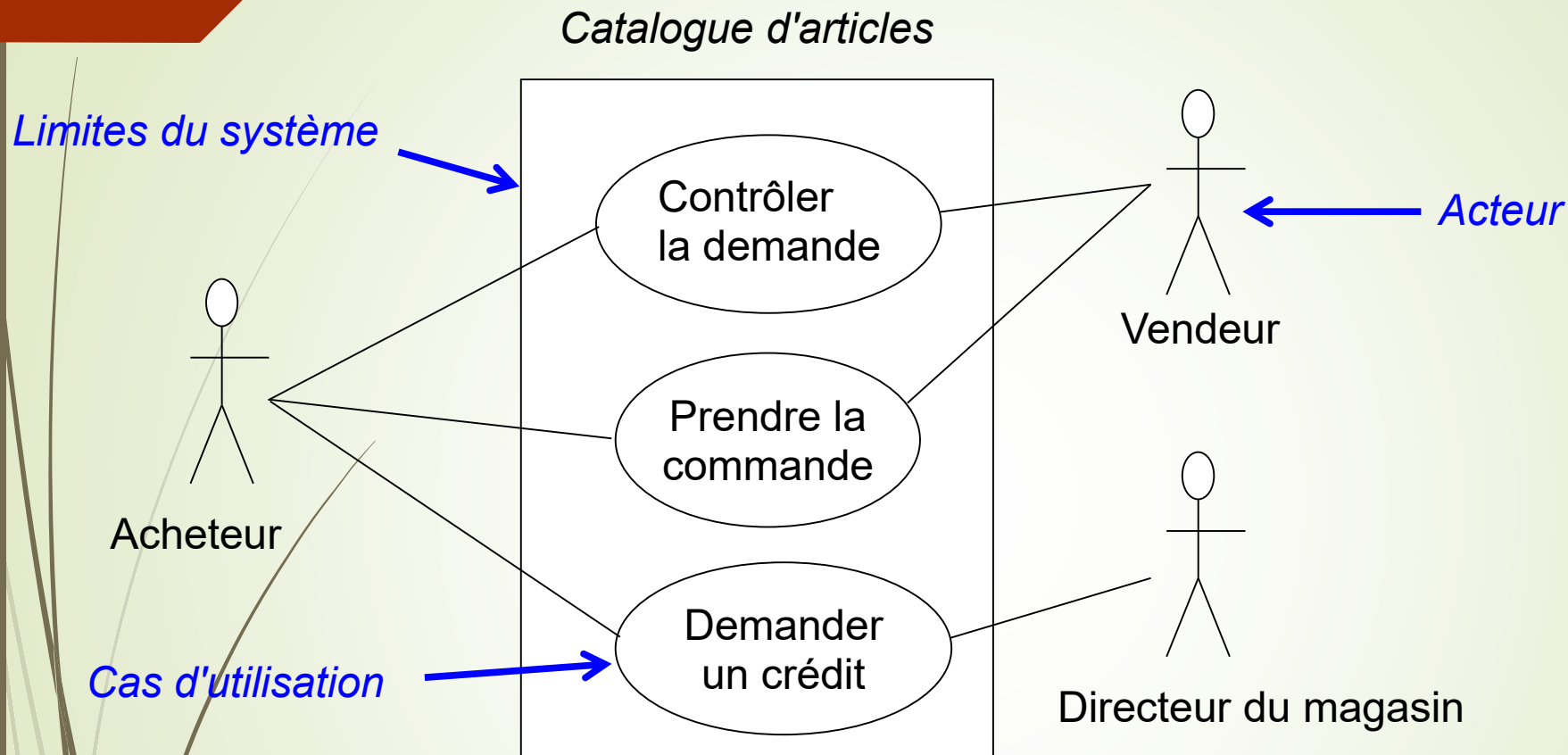
PHASE D'ANALYSE

- 
- On commence par identifier :
 - Le système
 - Les acteurs qui interagissent avec le système
 - Les actions des acteurs sur le système

Les acteurs

- Un acteur est une entité externe (personne, imprimante, serveur, SGBD, ...) qui interagit avec le système
- Un même humanoïde peut être plusieurs acteurs
- On définit donc un acteur par un ensemble de rôles qu'il a sur le système

Diagramme de cas d'utilisation



Différents liens entre les cas d'utilisation/acteurs :

—— association

—— << extend >> étend

- - - - - > généralise

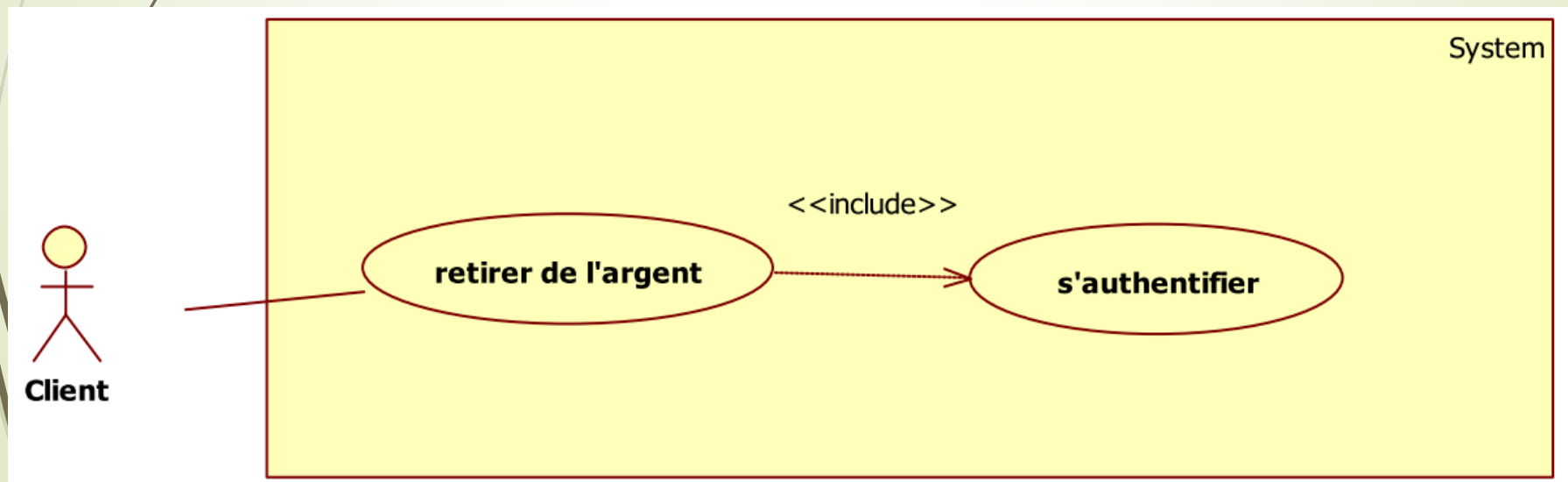
—— << include >> inclut

Relations entre cas d'utilisation

- Relation d'inclusion : <<include>>
- Relation d'extension : <<extend>>
- Relation de généralisation

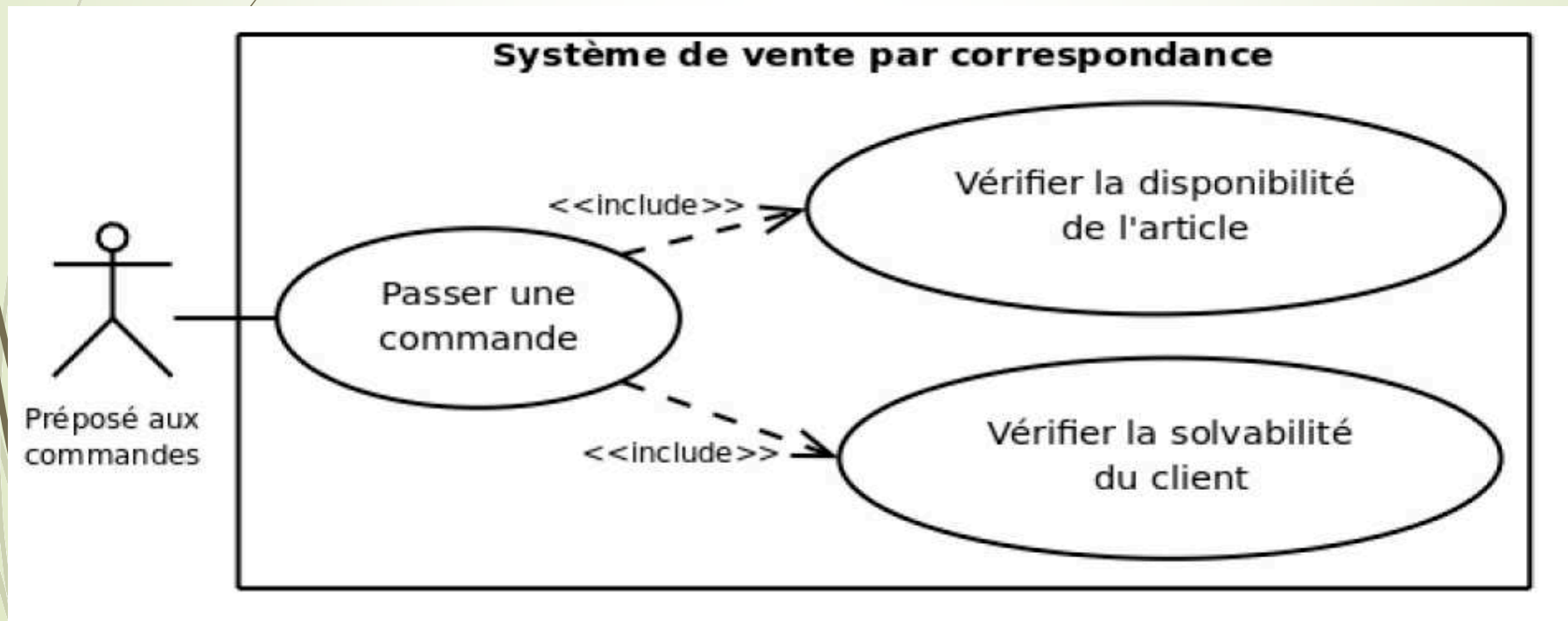
La relation <<include>>

- Un cas A inclut un cas B si le comportement décrit par le cas A inclut systématiquement le comportement du cas B, e.g:
- accès aux informations d'un compte bancaire inclut nécessairement une phase d'authentification avec un identifiant et un mot de passe



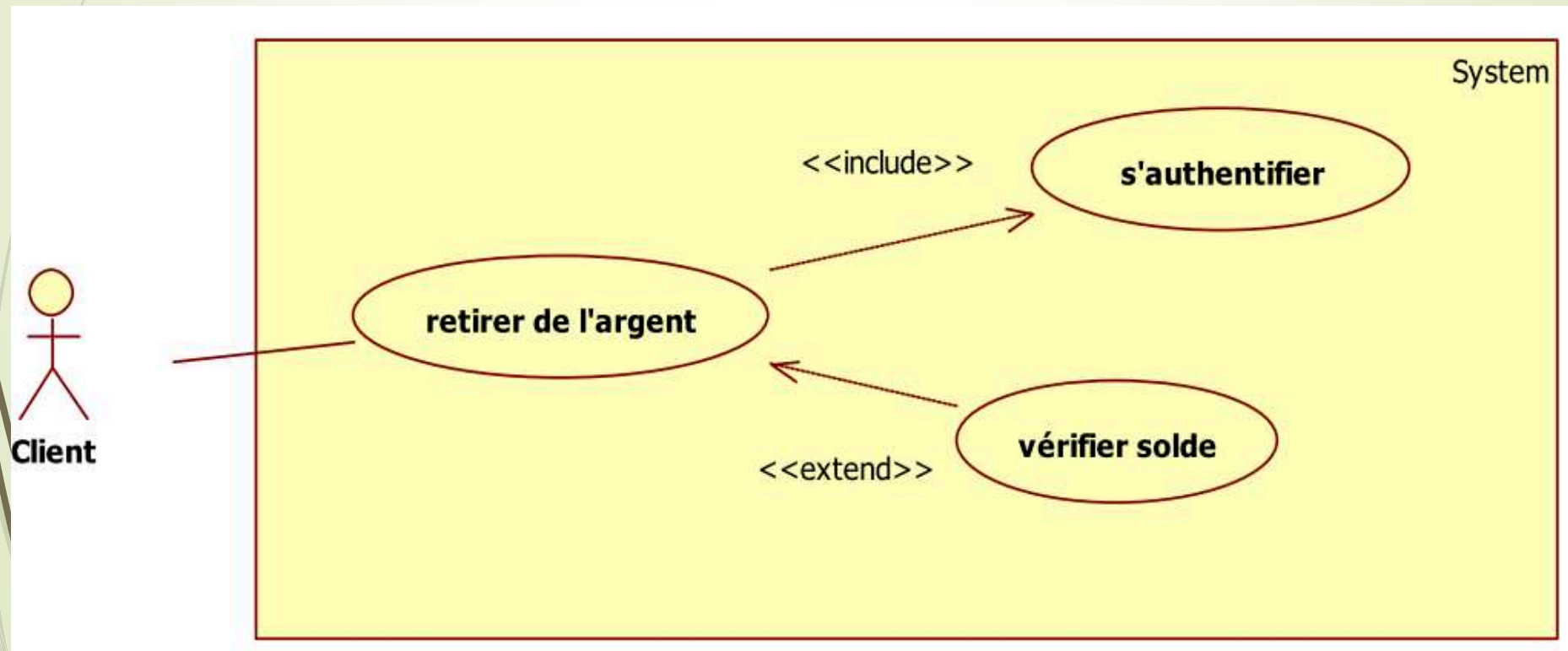
La relation <<include>>

- L'inclusion permet de :
 - factoriser la description d'un cas d'utilisation qui est commune à d'autres cas d'utilisation
 - décomposer un cas complexe en sous-cas plus simples



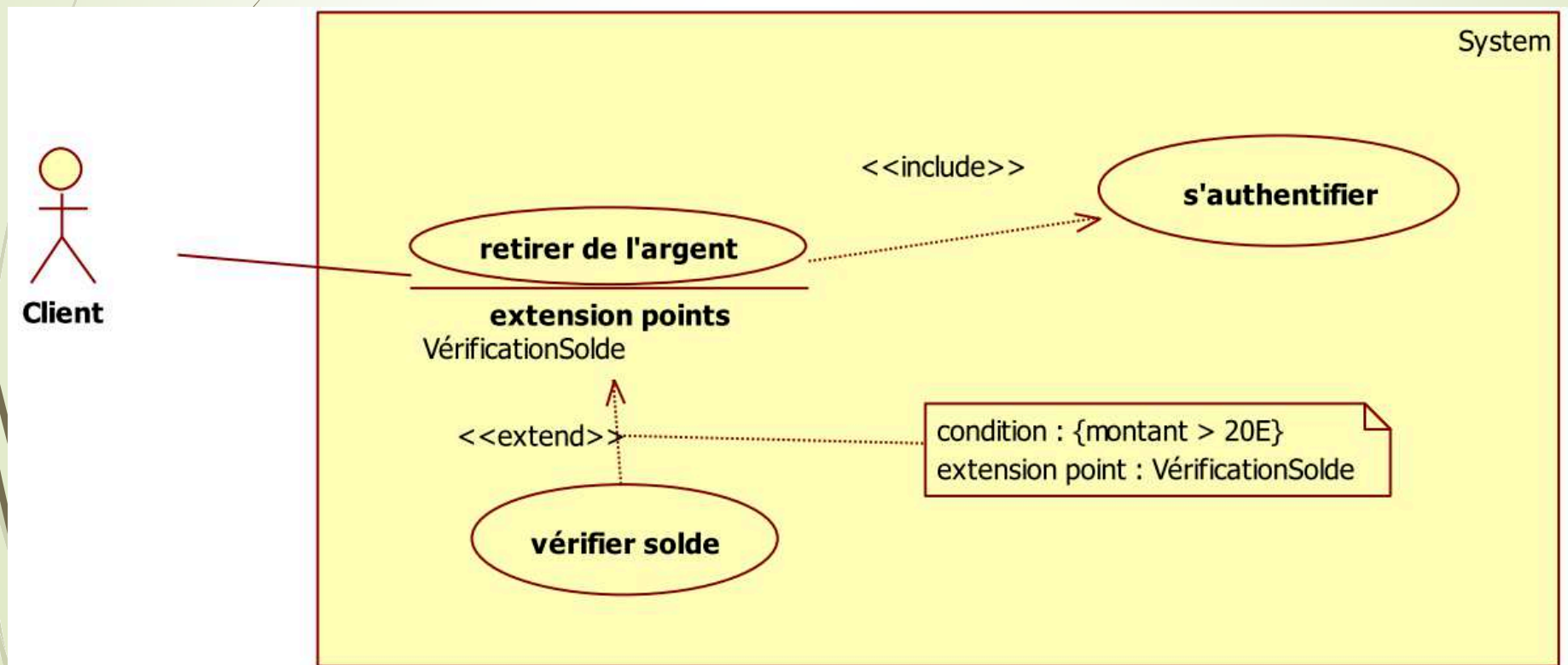
La relation <<extend>>

- Un cas A étend un cas B lorsque le cas A peut être appelé au cours de l'exécution du cas B e.g.
- quand la demande de retrait dépasse 20 euros, on fait une vérification du solde du compte



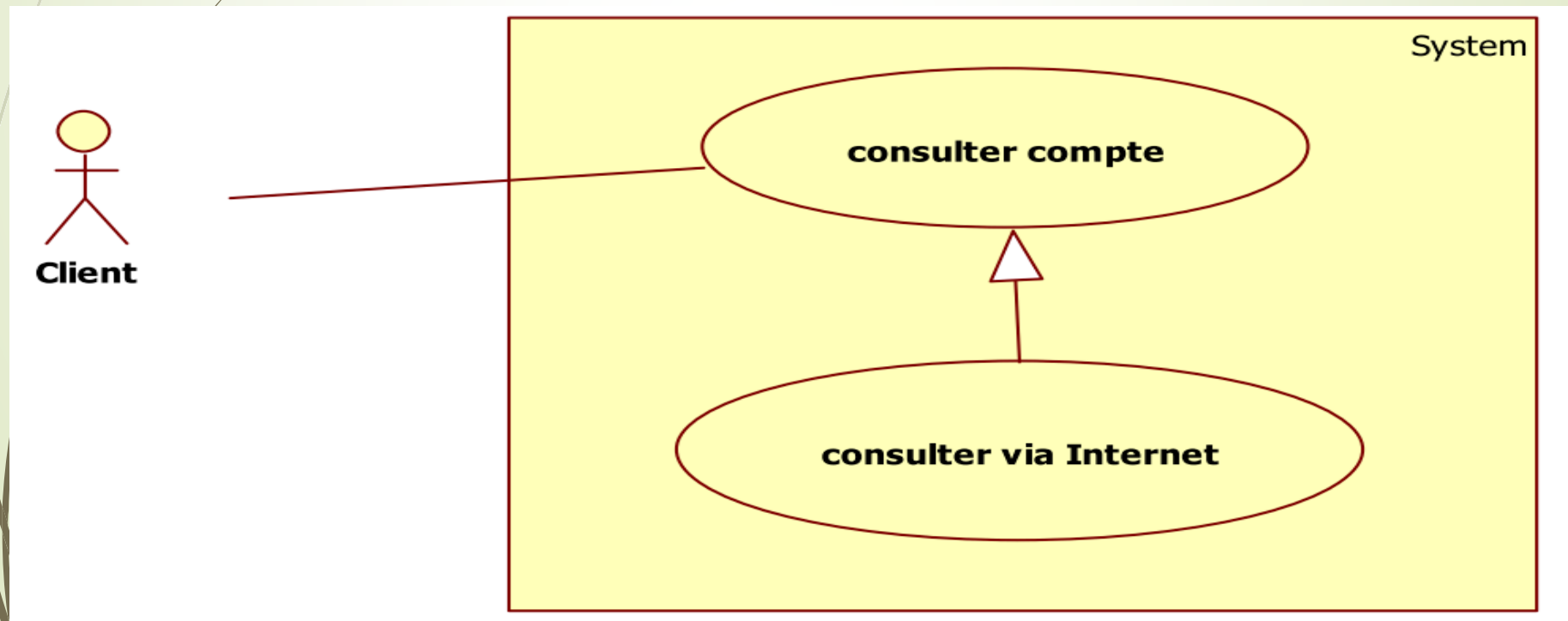
La relation <<extend>>

- L'extension peut intervenir à un point précis du cas étendu : point d'extension
- Condition d'extension : contrainte dans une note



La relation <<generalize>>

- Un cas A est une généralisation d'un cas B si B est un cas particulier de A, e.g.
 - La consultation d'un compte via Internet est un cas particulier de la consultation



Fiche descriptive d'un cas d'usage

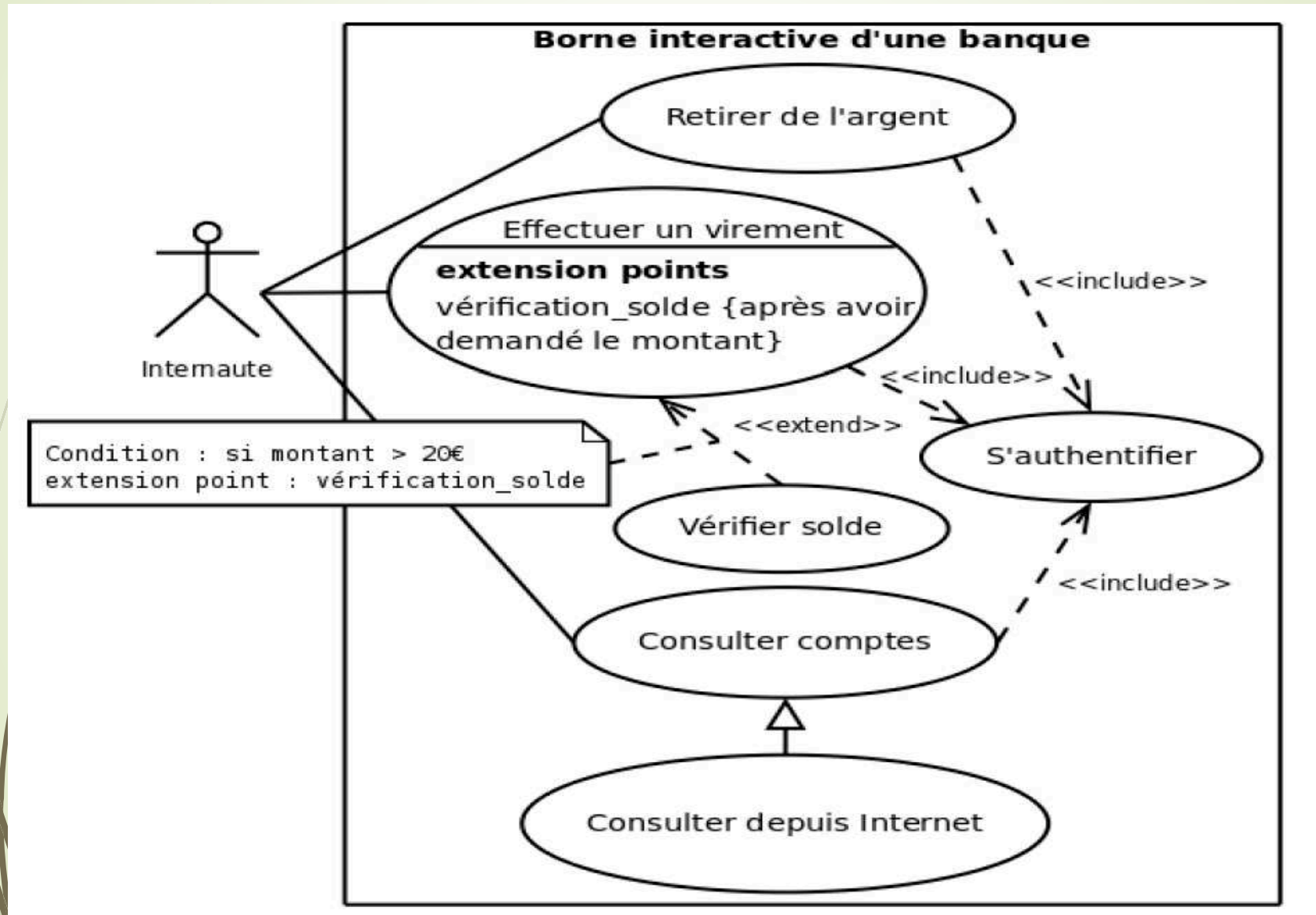
- A chaque cas d'usage, on associe une fiche
 - une **description** du cas d'usage
 - **règle d'initiation** : qu'est ce qui déclenche la transaction ?
 - **règle de terminaison** : qu'est ce qui termine la transaction ?
 - **règle d'exception** : qu'est ce qui déclenche le cas d'usage quand la règle d'initiation n'est pas vérifiée ?
 - les **relations** avec d'autres cas d'usage : extension, inclusion et généralisation

Fiche descriptive d'un cas d'usage


► Exemple : cas Retirer l'argent

- **Description** : il s'agit du cas d'usage qui permet à un client de retirer une somme d'argent à partir d'un distributeur
- **Règle d'initiation** : le client doit avoir sa carte bancaire valide et le code
- **Règle de terminaison** : le solde du compte bancaire est mis à jour
- **Règle d'exception** :
- **Relations** : ce cas inclut le cas s'authentifier

Exemple de diagramme de cas d'utilisation complet



Exemple drive

- 
- drive est un site commercial qui permet de faire des courses en ligne et de venir au magasin les retirer au drive
 - Sur l'écran de la borne automatique de magasin, il y a deux options :
 - commande sur place ou
 - retrait de marchandises
 - Pour retirer les marchandises, il faut obligatoirement saisir le code client et procéder au paiement si ce n'est pas encore fait sur le site
 - Modéliser les fonctionnalités proposées par la borne automatique via un diagramme de cas d'utilisation

Exemple drive

