



## INTRODUCTION A UML

Christophe GNAHO

### PLAN

#### Présentation générale

- UML – Langage de Modélisation Objet
- Notion de *modèle*
- Vue d'ensemble des diagrammes UML
- Quelques outils

#### Modélisation fonctionnelle

- Vers une démarche de modélisation
- Diagramme de cas d'utilisation
- Description des cas d'utilisation
  - Diagramme de séquence (système)
  - Diagramme d'activités (système)

#### Modélisation statique

- Diagramme de classes

### Modélisation dynamique

- Diagrammes d'interaction
  - Diagramme de séquence (de conception)
  - Diagramme de communication
- Diagramme d'états
- Diagramme d'activités

### Passage de l'analyse au code

- Hypothèses
- Élaboration du diagramme de classes de conception
- Transformation du diagramme de classes de conception en code
- Exemple

### Implémentation

- Diagramme de composants
- Diagramme de déploiement

### Dossier d'analyse

- Rappel sur les phases de développement logiciel
- Plan de rédaction

## Présentation générale

- UML – Langage de modélisation objet
- Notion de *modèle*
- Vue d'ensemble des diagrammes UML
- Quelques outils

## PRESENTATION GENERALE (UML – Langage de Modélisation Objet)

UML (Unified Modeling Language)

Résultat de la fusion de trois méthodes Orientées Objet



James Rumbaugh  
(OMT)



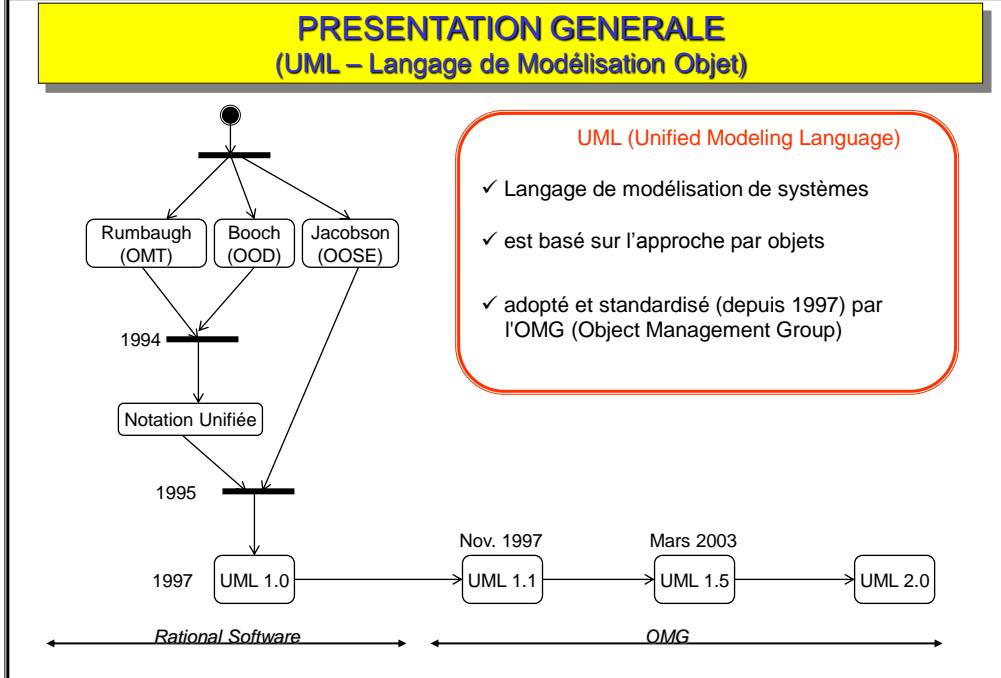
Grady Booch  
(OOD)



Ivar Jacobson  
(OOSE)

*Les trois amigos*

## PRESENTATION GENERALE (UML – Langage de Modélisation Objet)



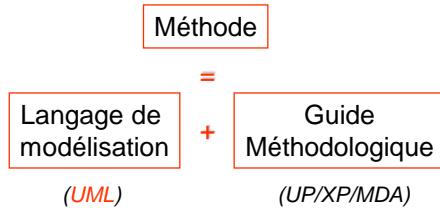
## PRESENTATION GENERALE (UML – Langage de Modélisation Objet)

UML fournit un formalisme graphique et textuel destiné à :

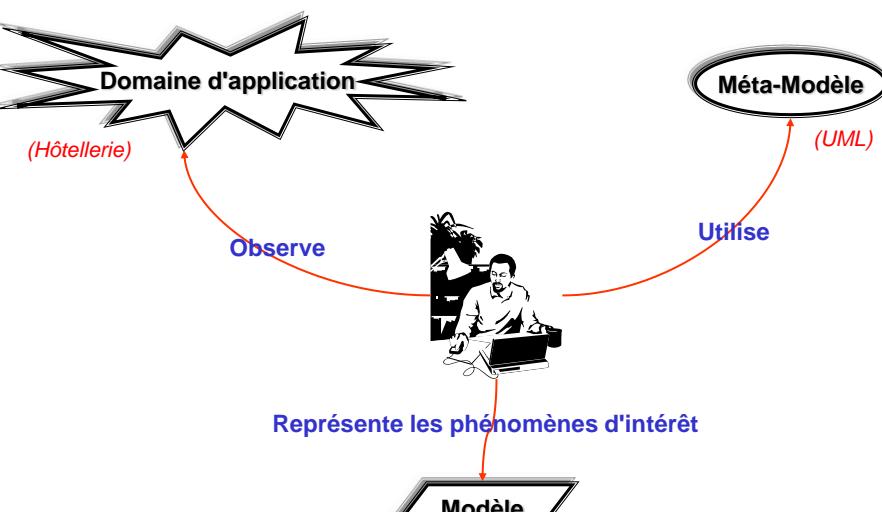
- comprendre et exprimer les besoins des utilisateurs,
- décrire les exigences fonctionnelles des systèmes
- analyser, spécifier, représenter et documenter les systèmes,
- élaborer des architectures logicielles,
- communiquer des points de vue.
- ...

## PRESENTATION GENERALE (UML – Langage de modélisation Objet)

UML n'est pas une méthode ...



## PRESENTATION GENERALE (Notion de modèle)

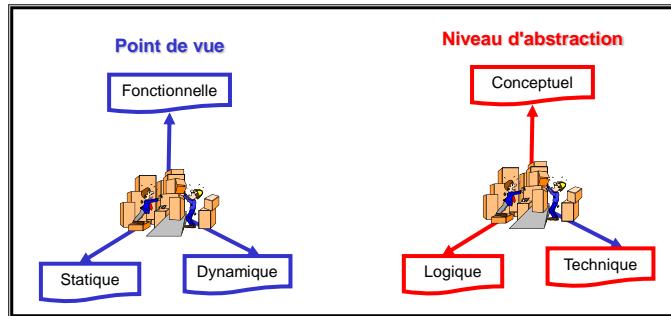


## PRESENTATION GENERALE (Notion de modèle)

On ne peut rendre compte de la complexité d'un système par une représentation unique; il est nécessaire d'avoir plusieurs points de vue complémentaires.

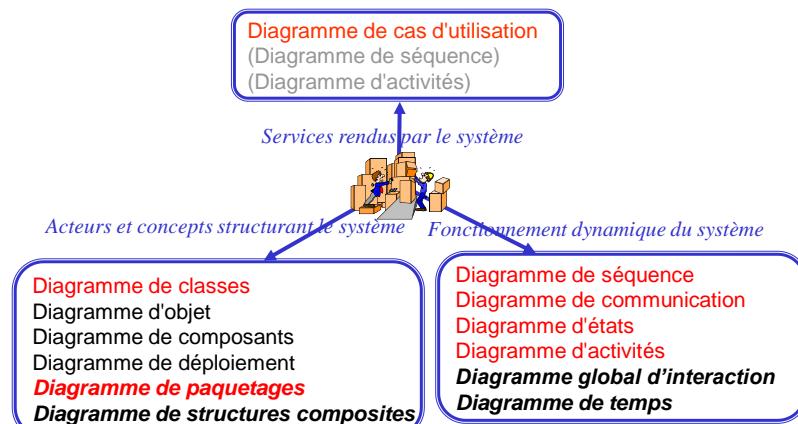
Un modèle est composé de plusieurs représentations sur un système. Chacune rend compte d'un **point de vue particulier**. Ces représentations peuvent cibler différents **niveaux d'abstraction**, et elles doivent être cohérentes

*En UML, chaque représentation correspond à un type de diagramme.*



## PRESENTATION GENERALE (Les diagrammes UML – Répartition suivant les trois points de vue)

Un diagramme est un schéma qui permet de donner une représentation graphique d'un système.



## PRESENTATION GENERALE

(Vue d'ensemble des diagrammes UML – Axe Fonctionnel)

### Diagramme de cas d'utilisation



L'axe fonctionnel s'appuie essentiellement sur le diagramme de cas d'utilisation.

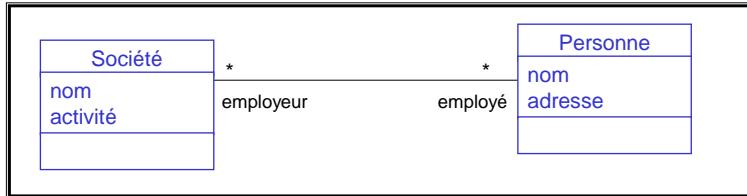
Ce diagramme permet de décrire les exigences fonctionnelles du système. Il détermine les frontières du système et ses relations avec son environnement.

Il est utilisé dans l'étape de capture et de spécification des besoins.

## PRESENTATION GENERALE

(Vue d'ensemble des diagrammes UML – Axe Statique)

### Diagramme de classes

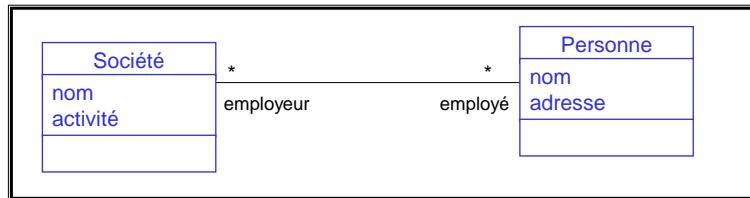


Alors que le diagramme de cas d'utilisation montre le système du point de vue des acteurs, le diagramme de classes en montre la **structure interne**. Il représente une description purement statique du système.

## PRESENTATION GENERALE

(Vue d'ensemble des diagrammes UML – Axe Statique)

### Diagramme de classes



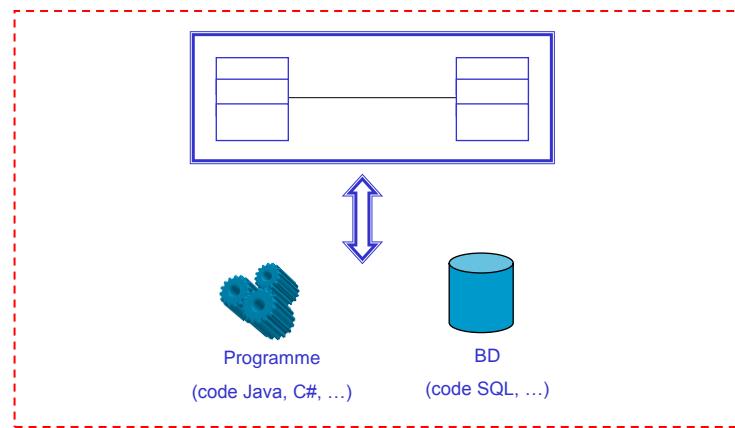
Le diagramme de classes peut être vu suivant **deux niveaux d'abstraction** :

- *Diagramme de classes d'analyse* : donne une représentation structurelle des entités manipulées par les utilisateurs.
- *Diagramme de classes de conception* : représente la structure d'un programme, ou les modules d'un programme orienté objet.

## PRESENTATION GENERALE

(Vue d'ensemble des diagrammes UML – Axe Statique)

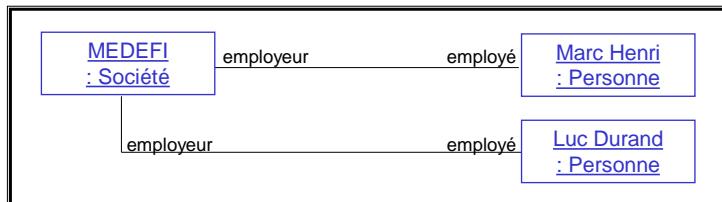
### Transformation d'un diagramme de classes en code



## PRESENTATION GENERALE

(Vue d'ensemble des diagrammes UML – Axe Statique)

### Diagramme d'objets



Le diagramme d'objets sert à illustrer des structures de classes compliquées. Il est utilisé en analyse pour vérifier l'adéquation d'un diagramme de classes à différents cas possibles.

## PRESENTATION GENERALE

(Vue d'ensemble des diagrammes UML – Axe Statique)

### Modélisation de l'architecture d'un système

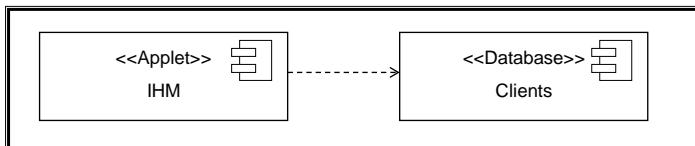
La modélisation UML de l'architecture d'un système comprend :

- la modélisation de l'architecture logicielle et sa structuration en composants ([Diagramme des composants](#))
- la modélisation de l'architecture matérielle et la répartition physique des logiciels ([Diagramme de déploiement](#)).

## PRESENTATION GENERALE

(Vue d'ensemble des diagrammes UML – Axe Statique)

### Architecture logicielle - Diagramme de composants



Un diagramme de composants montre le découpage du système en unités (logiciels) pouvant être distribuées.

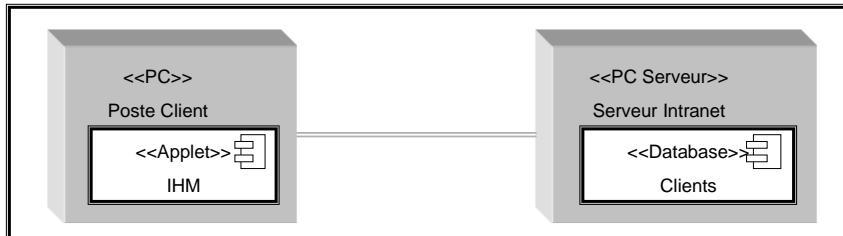
En UML 1, la notion de composant représente les *concepts logiciels*, comme par exemple : fichier, base de données, bibliothèque de fonctions, progiciel, applet, ...

En UML 2, elle désigne une *boîte noire* (Ex : une Interface au sens Java) qui offre des services logiciels décrits par une ou plusieurs interfaces.

## PRESENTATION GENERALE

(Vue d'ensemble des diagrammes UML – Axe Statique)

### Architecture matérielle (physique) - Diagramme de déploiement



Le diagramme de déploiement représente à la fois :

- (1) la structure du système informatique qui prend en charge le système logiciel,
- (2) et la façons dont les composants y sont installés.

## PRESENTATION GENERALE

(Vue d'ensemble des diagrammes UML – Axe Dynamique)

Diagrammes dynamiques : passerelle entre vision externe et interne

Diagramme de cas d'utilisation  
(Vision externe du système)

Diagrammes Dynamiques = **Passerelle**

Diagramme de classes  
(Vision interne du système)

Comment les objets interagissent  
pour réaliser une fonctionnalité !!!



## PRESENTATION GENERALE

(Vue d'ensemble des diagrammes UML – Axe Dynamique)

Diagrammes dynamiques : passerelle entre vision externe et interne

Diagramme d'interaction

- diagramme de séquence
- diagramme de communication

Diagramme états-transitions

Diagramme d'activités

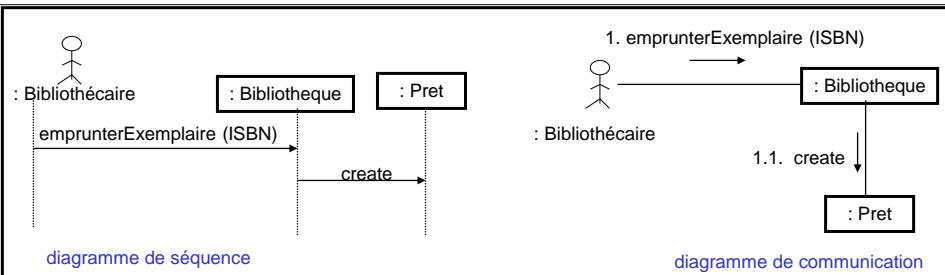
Diagramme global d'interaction

Diagramme de temps

## PRESENTATION GENERALE

(Vue d'ensemble des diagrammes UML – Axe Dynamique)

### Diagrammes d'interaction



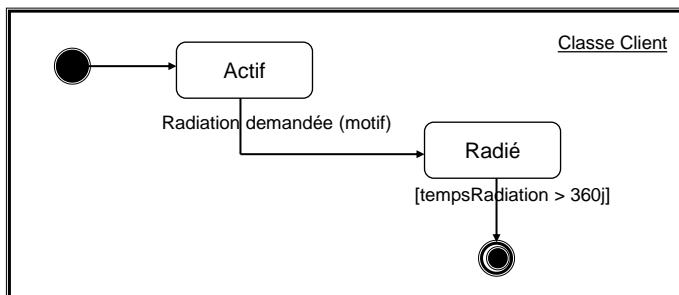
Les diagrammes de séquence et de communication sont appelés **diagrammes d'interaction**. Ils représentent les échanges de messages entre objets du système, **dans le cadre d'un fonctionnement particulier (scénario) du système**.

Ils peuvent être notamment utilisés en conception, pour définir et concevoir les méthodes des classes.

## PRESENTATION GENERALE

(Vue d'ensemble des diagrammes UML – Axe Dynamique)

### Diagrammes d'états

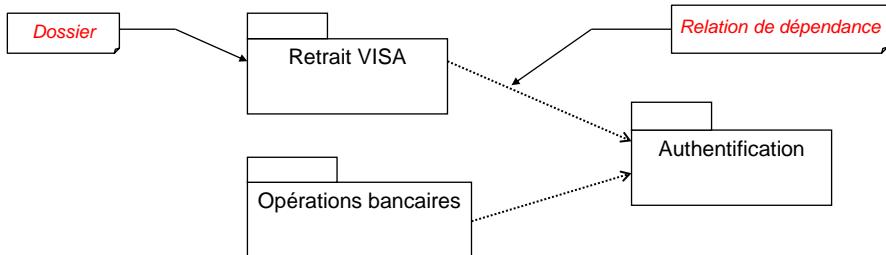


Un diagramme d'états est propre à une classe, il décrit le cycle de vie des objets de cette classe à l'aide d'un automate à état finis.

Il présente les séquences possibles d'états et d'actions qu'une instance de classe peut traiter au cours de son cycle de vie en réaction à des événements.

## PRESENTATION GENERALE

(La structuration des éléments de modélisation – **Diagramme de paquetages**)

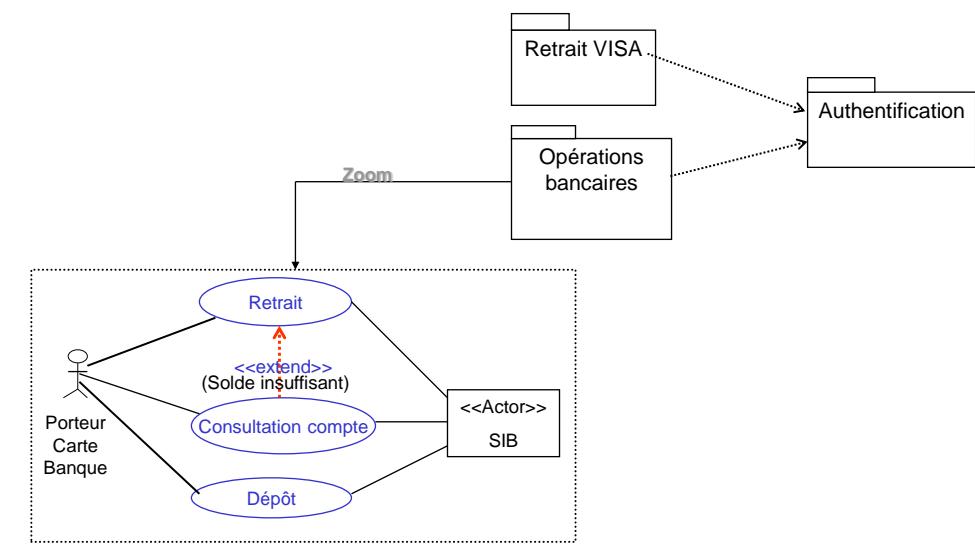


En UML 1, les paquetages faisaient partie du diagramme de classe et regroupaient exclusivement des ensembles de classes.

UML 2 propose un diagramme spécifique (*diagramme de paquetages*). Un paquetage permet d'organiser des éléments de modélisation en groupe. Il peut contenir des classes, des cas d'utilisation, des composants, autres paquetages, ...

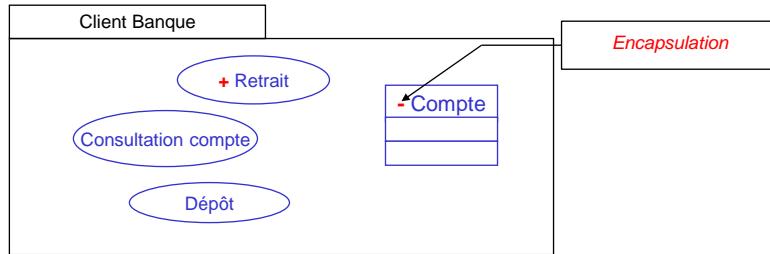
## PRESENTATION GENERALE

(La structuration des éléments de modélisation – **Diagramme de paquetages**)



## PRESENTATION GENERALE

(La structuration des éléments de modélisation – Diagramme de paquetages)



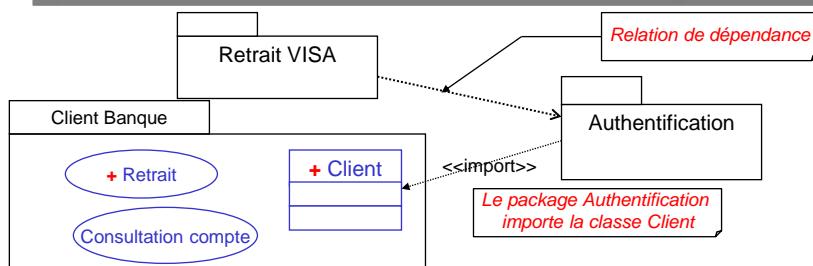
Il est possible d'inclure directement les éléments (avec leur propre représentation graphique) d'un paquetage à l'intérieur du dossier qui le représente.

*Un élément de modélisation ne peut être présent que dans un seul paquetage.*

Chaque élément d'un paquetage peut être accessible à l'extérieur (voir signe + précédant le nom de l'élément) ou encapsulé (non visible) à l'intérieur de celui-ci (voir signe -). Par défaut, un élément est accessible à l'extérieur.

## PRESENTATION GENERALE

(La structuration des éléments de modélisation – Diagramme de paquetages)



En UML1, la relation entre paquetage est définie par la notion de *dépendance*. Celle-ci a été spécialisée en UML2, en deux associations de dépendances (*association d'importation* et *association d'accès*) à l'aide du stéréotype *import* et *access*.

*L'association d'importation* consiste à amener dans le paquetage destination un élément (*visible*) du paquetage d'origine.

*L'association d'accès* consiste à accéder, depuis le paquetage de destination, à un élément du paquetage d'origine.

Ces deux associations peuvent également s'appliquer à un paquetage complet. Elle s'intéresse dans ce cas à tous les *éléments visibles* du paquetage

## PRESENTATION GENERALE

### (Quelques outils)

#### Poseidon for UML par Gentleware

La dernière version est disponible sous forme de plug-in pour Eclipse.

#### Rational Rose et XDE par IBM

XED est un plug-in pour Eclipse et Visual Studio.NET.

#### Visio par Microsoft

L'outil de dessin de Microsoft, propose un plug-in pour UML qui inclut le dessin, la génération de code et l'ingénierie inverse.

#### Eclipse UML par Omondo

Un plug-in gratuit pour Eclipse

#### StarUML

Open plate-forme open source UML/MDA

...

## Modélisation fonctionnelle

- Vers une démarche de modélisation
- Diagramme de cas d'utilisation
- Description des cas d'utilisation
  - Diagramme de séquence (système)
  - Diagramme d'activités (système)

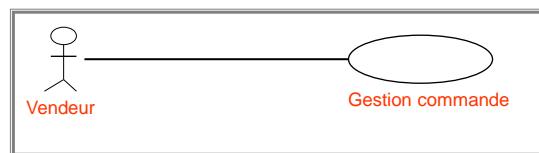
## Modélisation fonctionnelle

- Vers une démarche de modélisation

## MODELISATION FONCTIONNELLE

(Vers une démarche de modélisation)

Diagramme de cas d'utilisation



## MODELISATION FONCTIONNELLE

(Vers une démarche de modélisation)

### Description des cas d'utilisation

La description des cas d'utilisation repose sur la notion de *scénario*.

Un scénario représente un fonctionnement particulier du cas sous la forme d'une séquence de messages échangés entre les acteurs et le système.

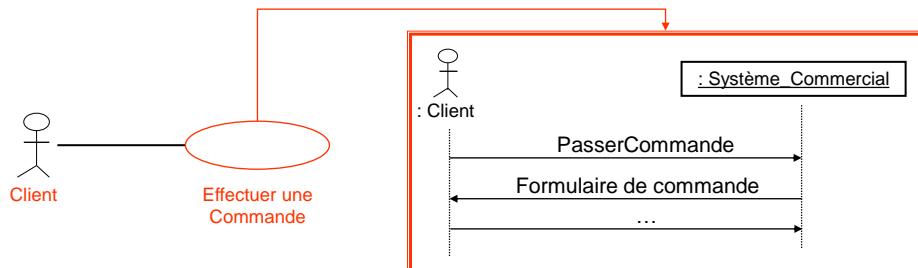
UML n'impose aucun diagramme pour la description d'un cas d'utilisation, le concepteur peut donc utiliser le type de diagramme qui lui paraît le plus adapté pour représenter des scénarios tout en restant dans la norme ...

Nous utilisons dans le cadre de ce cours le *diagramme de séquence* et le *diagramme d'activité*.

## MODELISATION FONCTIONNELLE

(Vers une démarche de modélisation)

### Description des cas d'utilisation - Diagramme de Séquence Système (DSS)

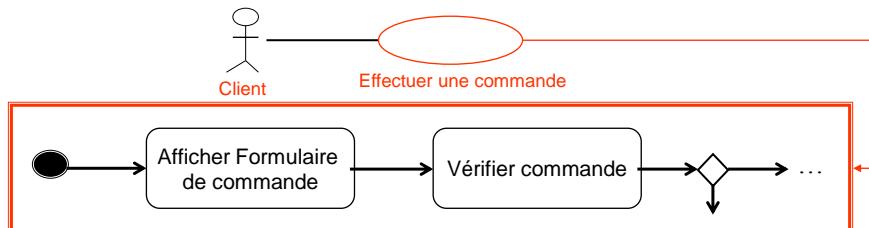


Le **Diagramme de Séquence Système** est une version «simplifiée» du diagramme de séquence (de conception). Il est utilisé pour compléter la description textuelle d'un cas d'utilisation par une description graphique. Il permet de décrire, sous forme d'une séquence de messages, les interactions entre les acteurs et le système.

## MODELISATION FONCTIONNELLE

(Vers une démarche de modélisation)

### Description des cas d'utilisation - Diagramme d'Activités Système (DAS)



Le Diagramme d'activités (dans sa version simplifiée) permet également de compléter la description textuelle d'un cas d'utilisation par une description graphique.

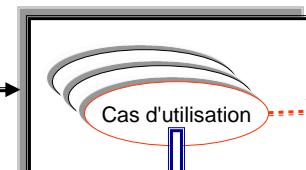
Contrairement au DSS qui ne décrit qu'un seul scénario, le diagramme d'activités **représente tous les scénarios**, sous la forme d'un ensemble d'activités (actions) **réalisées par le système** (pour accomplir l'objectif du cas), avec tous les branchements conditionnels et toutes les boucles possibles.

## MODELISATION FONCTIONNELLE

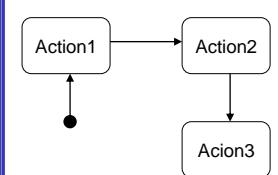
(Vers une démarche de modélisation)

### Diagramme de cas d'utilisation

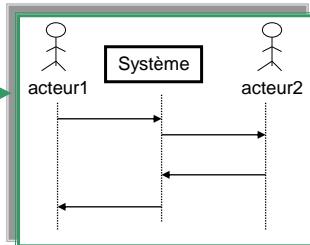
Expression textuelle du problème



### Diagramme d'activités



### Diagramme de séquence

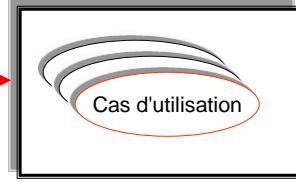


## MODELISATION FONCTIONNELLE

(Vers une démarche de modélisation)

Diagramme de cas d'utilisation

Expression  
textuelle  
du problème



### Une démarche en 4 étapes :

1. Identifier les acteurs
2. Identifier les Ucs (voir tableau)
3. Transformer le tableau en diagramme (=> première version)
4. Rechercher les relations inter-Uc (=>version finale du diagramme)

## Modélisation fonctionnelle

- Diagramme de cas d'utilisation

## MODELISATION FONCTIONNELLE

(Diagramme de cas d'utilisation)

### Concepts Fondamentaux

- Cas d'utilisation
- Acteur
- Relations
  - entre un cas d'utilisation et un acteur
  - entre acteur
  - entre les cas d'utilisation

## MODELISATION FONCTIONNELLE

(Diagramme de cas d'utilisation)

### Cas d'utilisation

Le concept de cas d'utilisation permet de donner une description du système étudié privilégiant le **point de vue de l'utilisateur**.

Un cas d'utilisation est une **manière spécifique** d'utiliser le système. Il spécifie **un service** (une fonctionnalité) rendu par le système à un utilisateur.

Un cas d'utilisation est composé d'un **ensemble d'actions** (déclenché par un acteur) réalisé par le système et qui **produisent un résultat significatif** pour un acteur particulier.

## MODELISATION FONCTIONNELLE

(Diagramme de cas d'utilisation)

### Cas d'utilisation

#### Exemples :

Gérer une commande, Consulter catalogue, ...

#### Formalisme graphique :



Verbe + compléments



Gérer une commande

## MODELISATION FONCTIONNELLE

(Diagramme de cas d'utilisation)

### Cas d'utilisation

#### Pour compléter la définition :

Chaque cas d'utilisation représente donc une **fonction métier** du système, **selon le point de vue d'un de ses acteurs**.

L'ensemble des cas d'utilisations doit décrire les exigences fonctionnelles et montrer **les frontières du système**, à savoir les fonctionnalités remplies et celles qui lui sont externes.

## MODELISATION FONCTIONNELLE

(Diagramme de cas d'utilisation)

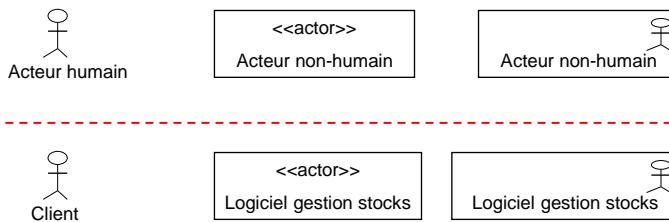
### Acteur

Un acteur représente un rôle joué par une entité externe (humain, dispositif matériel ou autre système) qui interagit directement avec le système étudié.

Exemple : Client, Service commercial, Logiciel de gestion de stocks, ...

Un acteur est donc une entité appartenant à l'environnement du système

### Formalisme graphique



## MODELISATION FONCTIONNELLE

(Diagramme de cas d'utilisation)

### Acteur principal – Acteur secondaire

Un cas d'utilisation (qui n'est pas interne) a toujours au moins un acteur principal pour qui le système produit un résultat observable, et éventuellement d'autres acteurs ayant un rôle secondaire.

Nous appelons *acteur principal* celui pour qui le cas d'utilisation produit un résultat observable. Par contre nous appelons *acteur secondaire* celui qui peut uniquement consulter ou informer le système (qui est sollicité) lors de l'exécution du cas d'utilisation.

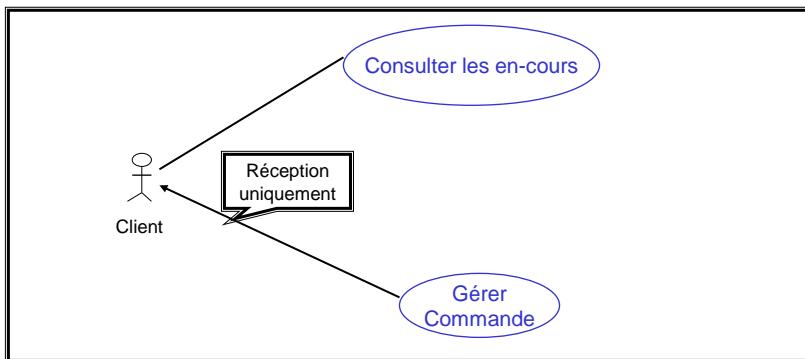
Par exemple, l'acteur *Porteur carte VISA* est un acteur principal du cas d'utilisation *Retirer de l'argent*, tandis que le *SI Banque* est un acteur secondaire. L'objectif du cas d'utilisation n'est pas essentiel pour ce dernier.

## MODELISATION FONCTIONNELLE

(Diagramme de cas d'utilisation)

### Relation entre un cas et un acteur - Association

Une association (ou **relation de communication**) permet de relier un acteur et un cas d'utilisation par une relation qui signifie "*participe à*"

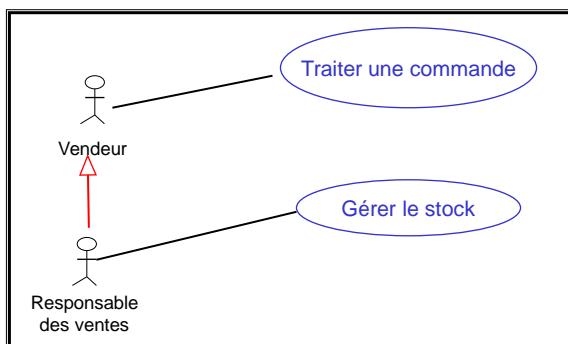


## MODELISATION FONCTIONNELLE

(Diagramme de cas d'utilisation)

### Relations entre acteur

La seule relation possible entre deux acteurs est la **généralisation** : un acteur A (Exemple : Vendeur) est une généralisation d'un acteur B (Exemple : Resp. des ventes) si l'acteur A peut être remplacé par l'acteur B. **Tous les cas d'utilisation accessibles à A le sont aussi à B, mais l'inverse n'est pas vrai.**



## MODELISATION FONCTIONNELLE (Diagramme de cas d'utilisation)

### Relations entre les cas d'utilisation

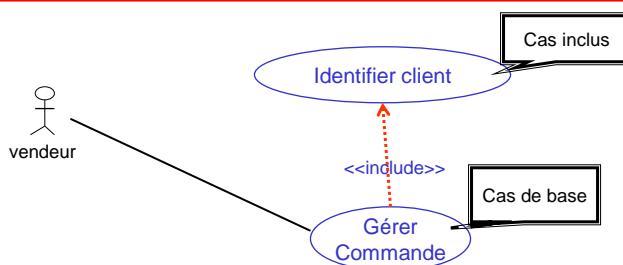
- Relation d'inclusion
- Relation d'extension
- Relation de spécialisation / généralisation

## MODELISATION FONCTIONNELLE (Diagramme de cas d'utilisation)

### Relation d'inclusion

Une relation d'inclusion représentée par le stéréotype `<<include>>` permet d'enrichir un cas d'utilisation (cas de base) par un autre cas d'utilisation (cas inclus). Cet enrichissement est réalisé par une inclusion **obligatoire**.

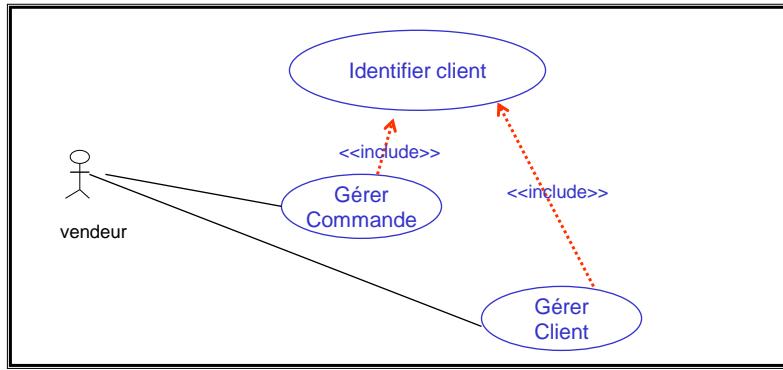
En général, le cas inclus existe uniquement dans ce but, il ne répond pas à un objectif d'un acteur principal. Un tel cas n'est pas directement accessible par un acteur, il est appelé **cas interne**, il représente une sous-fonctionnalité



## MODELISATION FONCTIONNELLE (Diagramme de cas d'utilisation)

### Relation d'inclusion

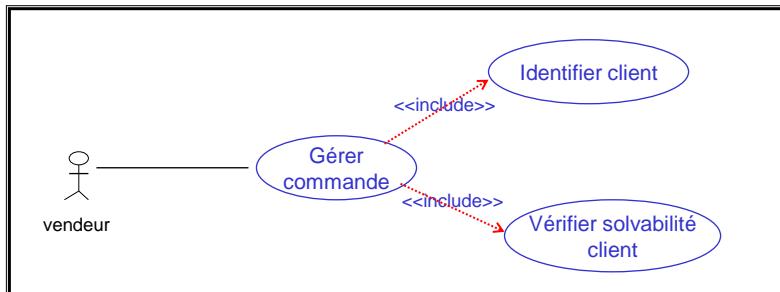
L'inclusion permet généralement d'identifier une partie commune aux différents cas d'utilisation et de la factoriser dans un nouveau cas inclus dans ces derniers.



## MODELISATION FONCTIONNELLE (Diagramme de cas d'utilisation)

### Relation d'inclusion

L'inclusion peut également être utilisée pour décomposer l'intérieur d'un cas d'utilisation sans que le cas inclus soit partagé. Le cas inclus représente alors une sous-fonction du cas de base.

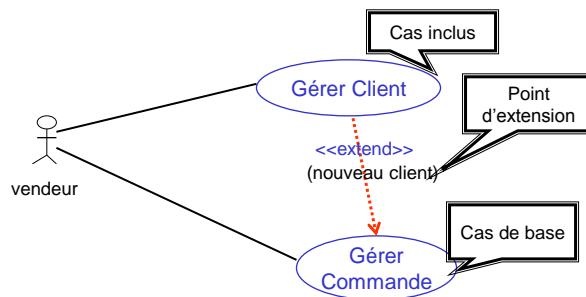


## MODELISATION FONCTIONNELLE (Diagramme de cas d'utilisation)

### Relation d'extension

Une relation d'extension (représentée par le stéréotype `<<extend>>`) permet comme la relation d'inclusion d'enrichir un cas d'utilisation par un autre, cependant, cet enrichissement est **optionnel**.

L'extension se fait dans le cas d'utilisation de base, en des points précis appelés **points d'extension**.

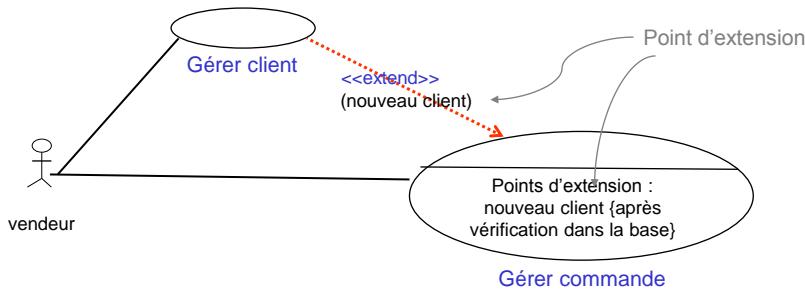


## MODELISATION FONCTIONNELLE (Diagramme de cas d'utilisation)

### Relation d'extension

#### Notion de 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 sur le lien ou dans un compartiment du cas étendu. Le nom est éventuellement associé à une contrainte indiquant le moment où l'extension intervient.



## MODELISATION FONCTIONNELLE (Diagramme de cas d'utilisation)

### Relation d'extension

L'inclusion étant optionnelle, les deux cas d'utilisation (cas de base et cas inclus) peuvent s'exécuter indépendamment.

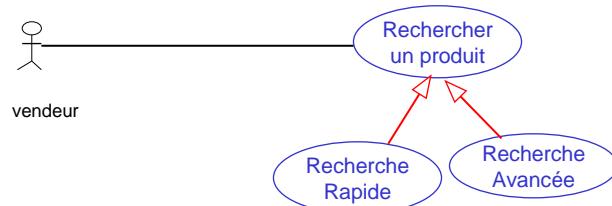
Dans l'exemple précédent, on exprime que "Gérer Client" peut également venir s'intercaler à l'intérieur de "Gérer Commande", au point d'extension "Nouveau client".

## MODELISATION FONCTIONNELLE (Diagramme de cas d'utilisation)

### Relation de généralisation/spécialisation

Une relation de généralisation/spécialisation permet d'exprimer que les cas d'utilisation descendants héritent de la description de leur parent commun. Ils peuvent cependant comprendre chacune des interactions spécifiques supplémentaires, ou modifier les interactions dont ils ont hérités.

Cette relation permet principalement de formaliser les **variations importantes** sur le même cas d'utilisation.



## MODELISATION FONCTIONNELLE

(Diagramme de cas d'utilisation)

### Relation de généralisation/spécialisation vs Relation d'extension



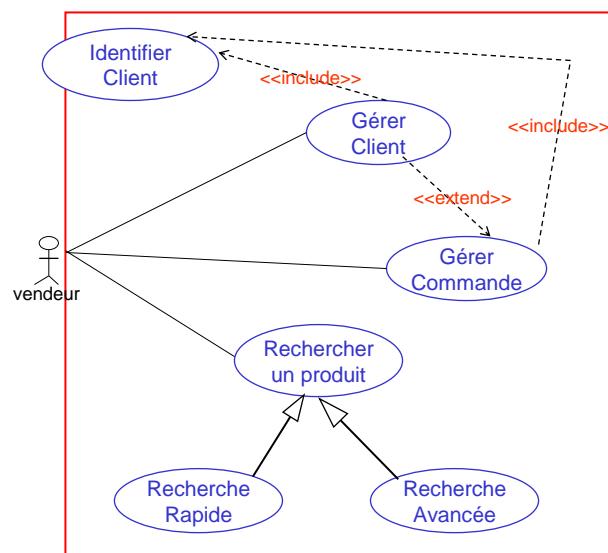
Le cas spécifique (Recherche Avancée) représente un traitement particulier qui va remplacer le traitement standard (Rechercher un produit).



Lors du traitement standard (Rechercher un produit), il peut y avoir quelque chose de plus à faire (Recherche Avancée).

## MODELISATION FONCTIONNELLE

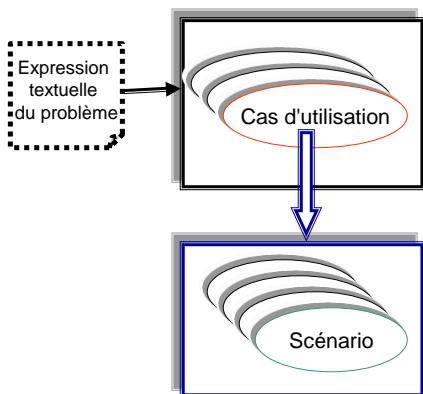
(Diagramme de cas d'utilisation "Gestion clients")



## Modélisation fonctionnelle

- Description des cas d'utilisation
  - Diagramme de séquence (système)
  - Diagramme d'activités (système)

### MODELISATION FONCTIONNELLE (Description des Cas d'Utilisation)



Une fois les cas d'utilisation identifiés, il faut les décrire. Cette description repose sur la notion de **scénario**.

Un **scénario** représente une succession particulière d'enchaînements, s'exécutant du début à la fin du cas d'utilisation.

Un cas d'utilisation contient en général un **scénario nominal** et plusieurs **scénarios alternatifs** (qui se termine de façon normale) ou **d'erreur** (qui se termine en échec, i.e. l'objectif n'est jamais atteint).

Scénarios : un nominal, 0 à n alternatifs, et cas d'échecs

## MODELISATION FONCTIONNELLE

(Description textuelle des cas d'utilisation)

### Plan Type

Titre  
Objectif  
Acteurs  
Pré-conditions  
Post-conditions  
Descriptif du scénario nominal  
Descriptif des scénarios alternatifs  
Descriptif des scénarios d'erreur



## MODELISATION FONCTIONNELLE

(Description textuelle des cas d'utilisation)



**Titre :** Réserver un véhicule

**Objectif :** ce cas d'utilisation permet à un client internaute de saisir une demande de réservation.

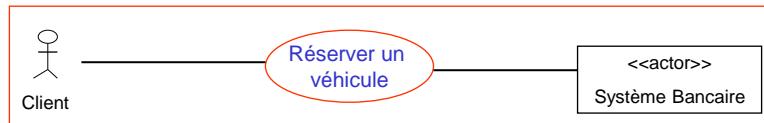
**Acteurs :** Client (principal), Système Bancaire (secondaire)

**Pré-conditions :** Le parc de véhicules n'est pas vide

**Post-conditions :** Une demande de réservation a été enregistrée par le système avec toutes les informations nécessaires.

## MODELISATION FONCTIONNELLE

(Description textuelle des cas d'utilisation)



### Descriptif du scénario nominal

1. Le client saisit son code d'identification
2. Le système vérifie le code d'identification
3. Le système demande au client de saisir les informations sur la réservation
4. Le client saisit les informations sur la réservation
5. Le système interroge l'acteur *système bancaire* pour vérifier l'acompte
6. Le *système bancaire* donne une réponse favorable
7. Le système envoie au client, un message de confirmation de la demande

## MODELISATION FONCTIONNELLE

(Description textuelle des cas d'utilisation)



### Descriptif des scénarios alternatifs

SA1 : code d'identification erroné pour la première ou la deuxième fois

SA1 démarre au point 2 du scénario nominal

3. Le système indique au client que le code est erroné, pour la première ou la deuxième fois.

Le scénario nominal reprend au point 1.

## MODELISATION FONCTIONNELLE

(Description textuelle des cas d'utilisation)



### Descriptif des scénarios d'erreur

SE1 : code d'identification erroné pour la troisième fois

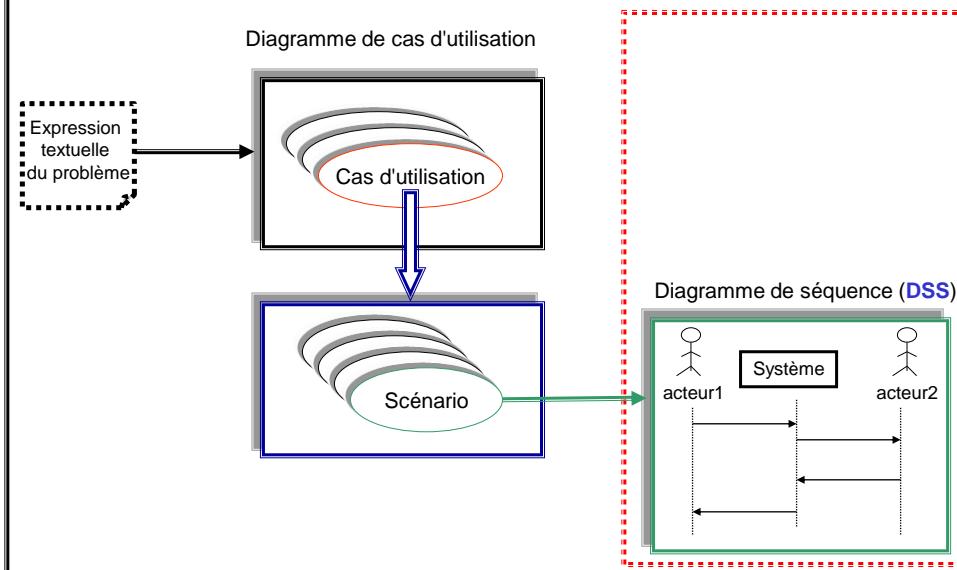
SE1 démarre au point 2 du scénario nominal

3. Le système indique au client que le code est erroné pour la troisième fois.

Le cas d'utilisation se termine en échec (l'objectif n'est pas atteint).

## MODELISATION FONCTIONNELLE

(Description graphique des Cas d'Utilisation)



## MODELISATION FONCTIONNELLE

(Description graphique des Cas d'Utilisation - Diagramme de Séquence Système)

### Rappel :

Le DSS est une simplification (quelques concepts) du Diagramme de séquence pour représenter graphiquement un scénario d'un cas d'utilisation.

Un DSS permet de donner une description graphique d'un scénario représentatif d'un cas d'utilisation. Il représente les messages échangés entre les acteurs et le système (vu comme une boîte noire).

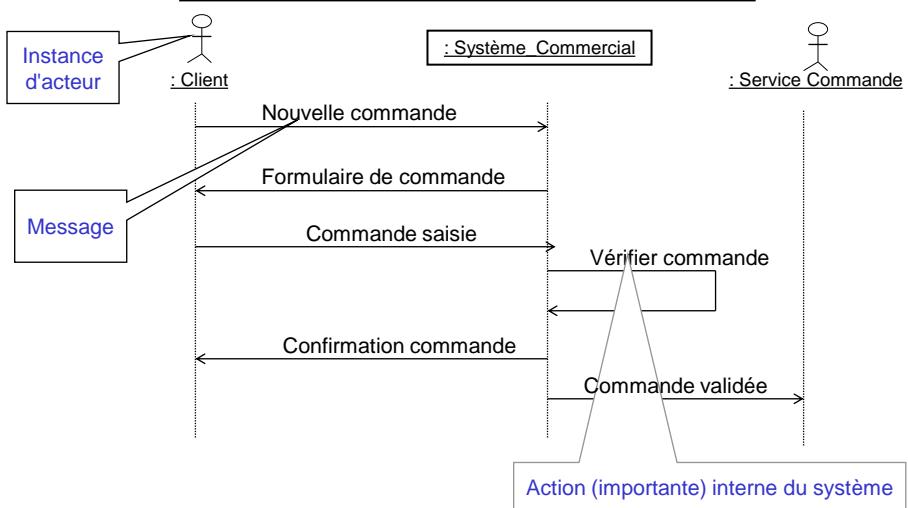
Concepts :

- Acteur (instance)
- Système (instance)
- Message

## MODELISATION FONCTIONNELLE

(Description graphique des Cas d'Utilisation - Diagramme de Séquence Système)

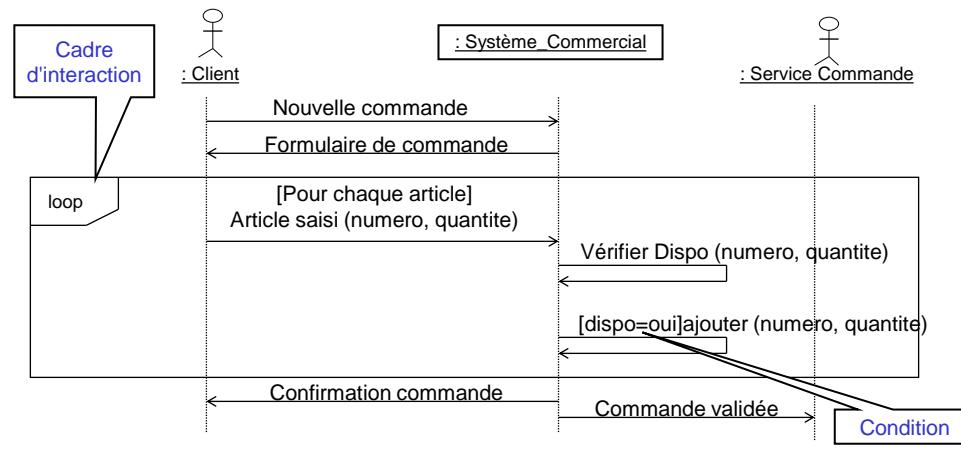
### Scénario nominal du cas *Effectuer une commande*



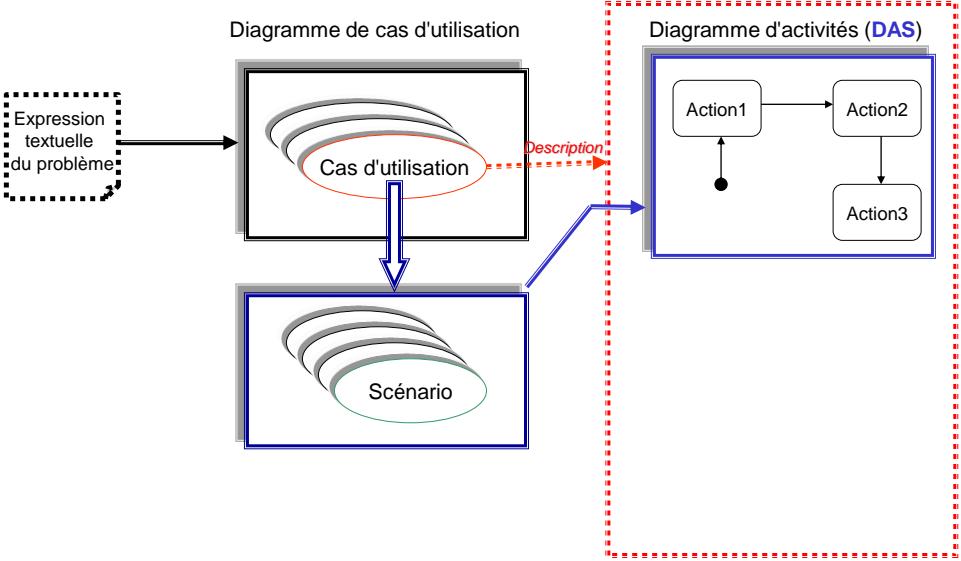
## MODELISATION FONCTIONNELLE

(Description graphique des Cas d'Utilisation - Diagramme de Séquence Système)

## **Scénario nominal du cas *Effectuer une commande***



## MODELISATION FONCTIONNELLE (Description graphique des Cas d'Utilisation)



## MODELISATION FONCTIONNELLE

(Description graphique des cas d'utilisation - Diagramme d'Activités Système)

Nous utilisons également une version simplifiée (quelques concepts) du diagramme d'activités, pour donner une description graphique des *cas d'utilisation*. Afin d'éviter toute confusion, nous appellerons ce diagramme : *Diagramme d'Activité Système (DAS)*.

Le *DAS* permet de donner *une vision globale* de l'ensemble des *actions réalisées par le système* (pour accomplir l'objectif du cas), avec tous les branchements conditionnels et toutes les boucles possibles.

Il se présente sous la forme d'un *graphe orienté* d'activités (d'actions) et de transitions. Les transitions sont franchies lors de la fin des activités; des étapes peuvent être réalisées en parallèle ou en séquence.

## MODELISATION FONCTIONNELLE

(Description graphique des cas d'utilisation - Diagramme d'Activités Système)

Concepts :

- Activité
- Transition
- Condition de garde
- Décision
- Parallélisme
- Point de départ / Point d'arrivée

## MODELISATION FONCTIONNELLE

(Description graphique des cas d'utilisation - Diagramme d'Activités Système)

### Activité

Une *activité* est une série d'actions (Ex: créer ou modifier un objet, réaliser une opération, ...) qui vise à effectuer un certain travail. Il peut s'agir d'un calcul, de la recherche de données, de la manipulation d'informations, ...

Dans le contexte de la représentation graphique de cas d'utilisation, le concept d'activité représente une action importante du système dans le cadre de l'objectif visé. Les actions des acteurs ne seront donc pas représentées.

#### Formalisme graphique

Nom de l'action du système

Vérifier commande

## MODELISATION FONCTIONNELLE

(Description graphique des cas d'utilisation - Diagramme d'Activités Système)

### Transition

Les activités sont reliées par des *transitions*, représentées par des flèches. La transition a lieu lorsque l'activité d'origine est terminée.

#### Formalisme graphique

Afficher Formulaire de commande

Vérifier commande



## MODELISATION FONCTIONNELLE

(Description graphique des cas d'utilisation - Diagramme d'Activités Système)

### Condition de garde

Une condition peut être affectée à une transition pour restreindre son utilisation. La condition est notée entre crochets sur la ligne de la transition. Elle doit être vraie pour que l'exécution se poursuive avec la prochaine activité

### Formalisme graphique



## MODELISATION FONCTIONNELLE

(Description graphique des cas d'utilisation - Diagramme d'Activités Système)

### Décision

Comme dans un organigramme, les losanges, sur le diagramme d'activités, représentent des décisions. Ils peuvent s'agir d'un simple test vrai/faux, ou d'un choix parmi plusieurs options. Chaque option est identifiée à l'aide d'une condition de garde. Les conditions doivent être mutuellement exclusives de façon qu'un seul choix soit possible.

### Formalisme graphique



## MODELISATION FONCTIONNELLE

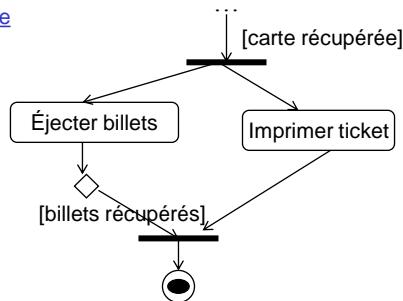
(Description graphique des cas d'utilisation - Diagramme d'Activités Système)

### Parallélisme

Le diagramme d'activités fournit une notation pour modéliser le parallélisme. Il s'agit d'une simple barre qui permet de représenter une *fourche* ou une *synchronisation*.

La fourche représente le passage d'une transition à plusieurs. La synchronisation représente le passage de plusieurs transitions à une seule.

#### Formalisme graphique



## MODELISATION FONCTIONNELLE

(Description graphique des cas d'utilisation - Diagramme d'Activités Système)

### Point de départ / Point d'arrivée

Deux icônes permettent de représenter respectivement le début et la fin de l'exécution de l'ensemble des activités d'un diagramme.

L'icône de fin n'est pas forcément unique et n'est pas obligatoire.

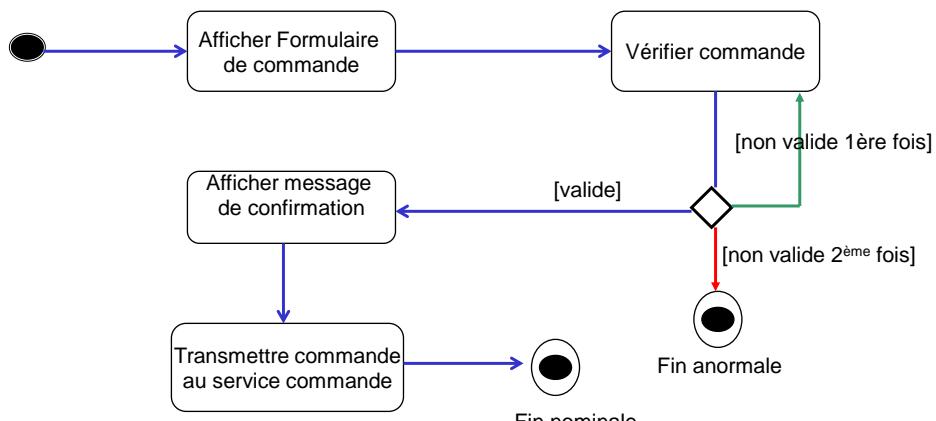
#### Formalisme graphique



## MODELISATION FONCTIONNELLE

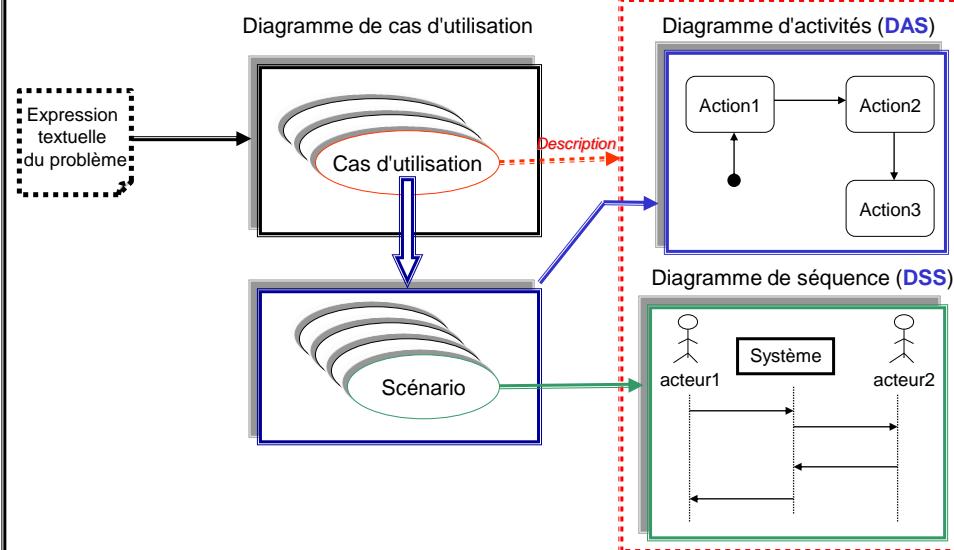
(Description graphique des cas d'utilisation - Diagramme d'Activités Système)

### Description du cas *Effectuer une commande*



## MODELISATION FONCTIONNELLE

(Description graphique des Cas d'Utilisation)



## Modélisation statique

### - Diagramme de classes

## MODELISATION STATIQUE (Diagramme de classes)

Le diagramme de classes permet de décrire les classes d'objets (description structurelle et comportementale) et les relations qui interviennent, [dans le cadre du problème posé](#).

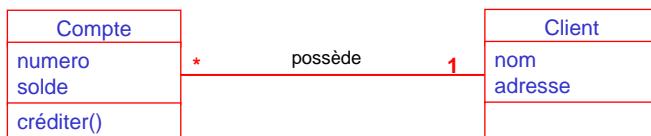
Cette description est indépendante de la solution technique choisie, elle représente seulement le problème.

Le diagramme de classes est au centre de la modélisation UML. C'est celui qui permet la génération automatique du code et c'est celui que produit l'ingénierie inverse. [Les autres diagrammes servent donc de supports pour la construction du diagramme des classes](#).

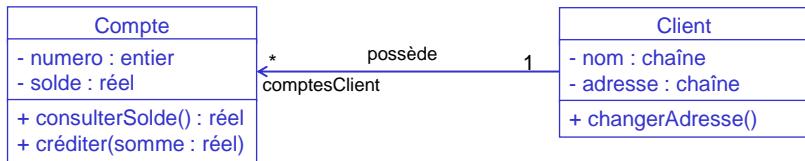
## MODELISATION STATIQUE (Diagramme de classes)

On distingue principalement deux niveaux d'abstraction

### Diagramme de classes d'analyse



### Diagramme de classes de conception



## MODELISATION STATIQUE (Diagramme de classes)

### Les principaux concepts

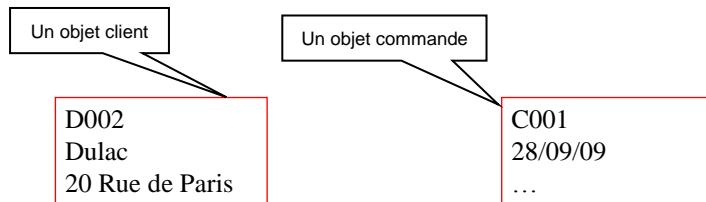
- Objet
- Classe
  - attribut ou propriété
  - opération
- Association
  - généralisation/specialisation
  - association simple
  - agrégation
  - composition

## MODELISATION STATIQUE (Diagramme de classes)

### Le concept d'objet

Le concept d'objet permet de représenter une **entité pertinente** ayant une signification **dans le domaine du problème posé**.

Un objet possède une identité et encapsule un état et un comportement.



## MODELISATION STATIQUE (Diagramme de classes)

### Le concept de classe

Une classe représente la description abstraite d'un **ensemble d'objets** possédant les mêmes caractéristiques et les mêmes comportements.

Classe



Objets



## MODELISATION STATIQUE (Diagramme de classes)

### Le concept de classe - suite

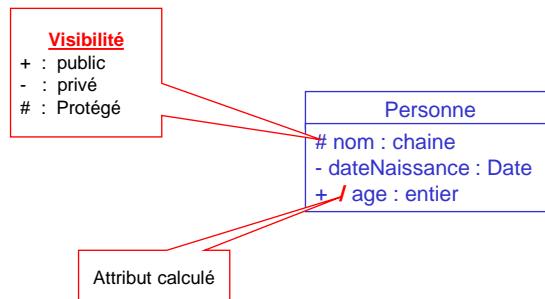
On dit qu'un objet est une **instance** d'une classe.

Une classe est définie par un ensemble *d'attributs* (**caractéristiques**) et *d'opérations* (**comportements**)

## MODELISATION STATIQUE (Diagramme de classes)

### Attribut

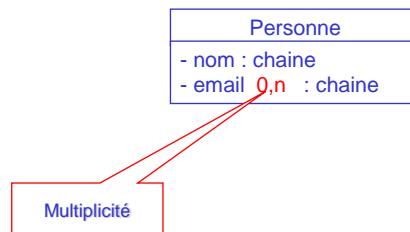
Un attribut représente une donnée intéressante (**dans le cadre du problème**) contenue dans une classe.



## MODELISATION STATIQUE (Diagramme de classes)

### Attribut

Un attribut peut avoir une **multiplicité** pour indiquer le nombre minimum et le nombre maximum de **valeurs possibles** de l'attribut pour une instance de la classe. La multiplicité par défaut est 1,1.

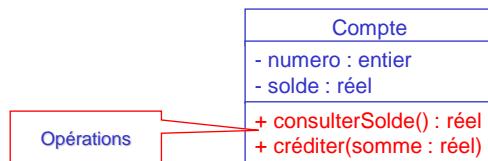


## MODELISATION STATIQUE (Diagramme de classes)

### Opération

Une opération représente un élément de comportement (un service) contenu dans une classe.

N.B. Dans une classe d'analyse, il convient de ne faire figurer que les opérations qui définissent de façon pertinente le comportement des classes.



## MODELISATION STATIQUE (Diagramme de classes)

### Représentation graphique d'une classe

Nom\_Classe  
Attribut1  
Attribut2 ...  
Opération1()  
Opération2() ...

Nom\_Classe  
Attribut1  
Attribut2 ...

Nom\_Classe

On peut se limiter à un ou deux compartiments.

## MODELISATION STATIQUE (Diagramme de classes)

### Le concept d'association

Une association représente une *relation sémantique et durable* entre deux classes. Cette relation est considérée comme une *relation structurelle*.

Exemple : la phrase "une personne possède une voiture" se traduit par une association entre la classe *Personne* et la classe *Voiture*.

On distingue principalement les types d'association suivants :

- généralisation/spécialisation,
- simple
- agrégation,
- composition

## MODELISATION STATIQUE

(Diagramme de classes)

### Généralisation / spécialisation

La généralisation/spécialisation permet de définir une relation entre une classe plus générale (appelée super-classe) et une ou plusieurs autres classes plus spécialisées (sous-classes).

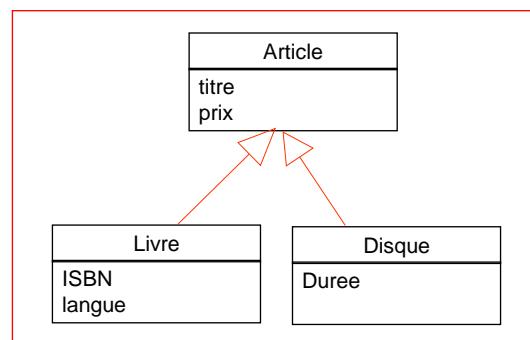
Elle permet soit de factoriser certains éléments soit de préciser de nouvelles contraintes sur le diagramme de classe.

Les sous classes héritent des propriétés, des opérations, des relations et des contraintes définies dans leur super-classe. Elles peuvent comporter des éléments spécifiques supplémentaires.

## MODELISATION STATIQUE

(Diagramme de classes)

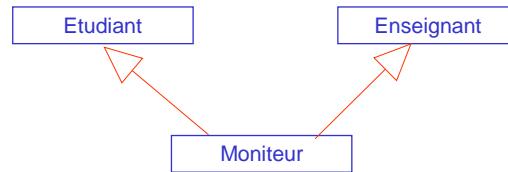
### Généralisation / spécialisation



## MODELISATION STATIQUE (Diagramme de classes)

### L'héritage multiple

L'héritage multiple met en évidence des classes spécialisées correspondant à plusieurs super-classes.

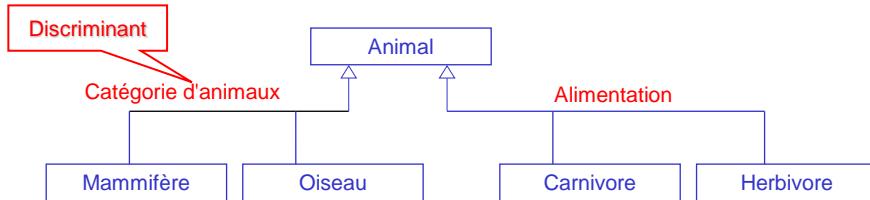


Une instance de *Moniteur* hérite à la fois des éléments (attributs, méthodes,) d'une instance de *Etudiant* et ceux d'une instance de *Enseignant*.

## MODELISATION STATIQUE (Diagramme de classes)

### Ensemble de spécialisation

Une classe peut être spécialisée selon plusieurs critères simultanément. Chaque critère est indiqué dans le diagramme en associant un **discriminant** à la relation de généralisation.



## MODELISATION STATIQUE (Diagramme de classes)

### Contraintes sur la relation de généralisation/specialisation

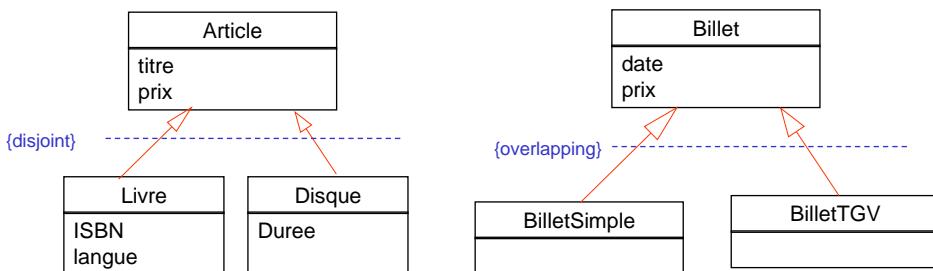
UML propose quatre contraintes sur la relation de généralisation/specialisation. Elles permettent de préciser si la décomposition de la classe générique en classes spécialisées est disjointe ou non, complète ou non :

- *incomplete*
- *complete*
- *disjoint*
- *overlapping*

## MODELISATION STATIQUE (Diagramme de classes)

### Contraintes sur la relation de généralisation/specialisation

La contrainte *{disjoint}* : un objet est au plus instance d'une seule des sous-classes.

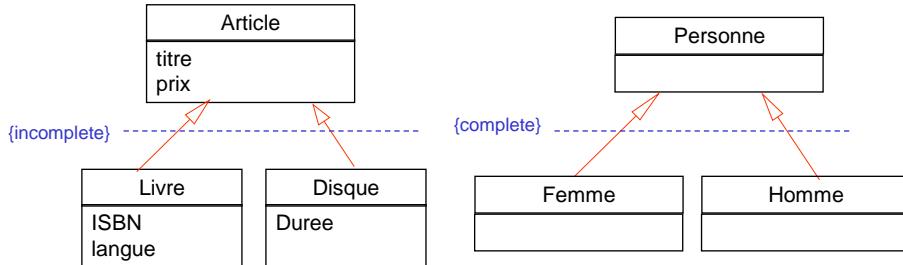


La contrainte *{overlapping}* : une instance de la super-classe est l'instance de deux ou plusieurs sous-classes.

## MODELISATION STATIQUE (Diagramme de classes)

### Contraintes sur la relation de généralisation/specialisation

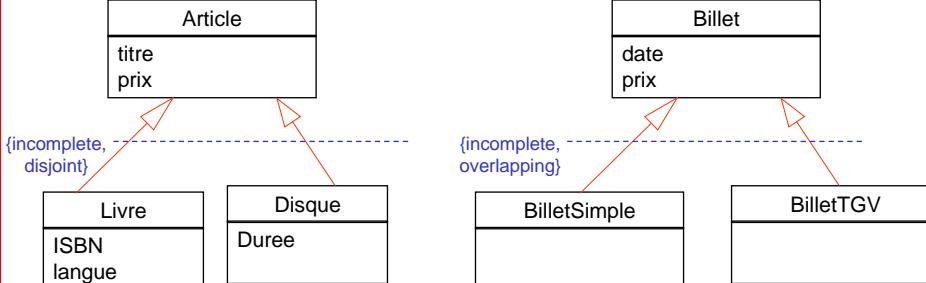
La contrainte **{complete}** : l'ensemble des sous-classes est complet et couvre la super-classe. On ne peut y rajouter de nouvelles sous-classes



La contrainte **{incomplete}** : l'ensemble des sous-classes est incomplet et ne couvre pas la super-classe. Il existe d'autres sous-classes qui peuvent être introduites.

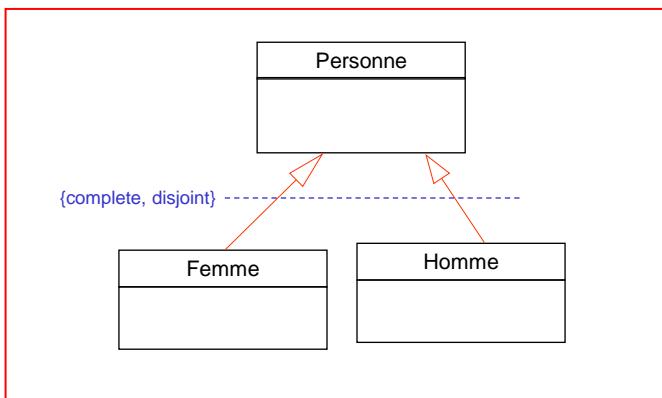
## MODELISATION STATIQUE (Diagramme de classes)

### Contraintes sur la relation de généralisation/specialisation



## MODELISATION STATIQUE (Diagramme de classes)

### Contraintes sur la relation de généralisation/specialisation

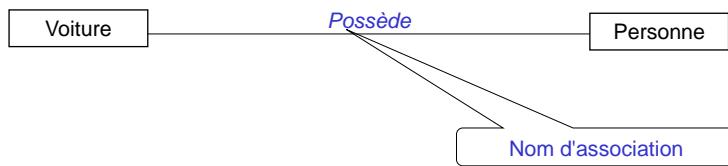


## MODELISATION STATIQUE (Diagramme de classes)

### Le concept d'association simple

Le concept d'association simple représente des relations structurelles et sémantiques entre classes d'objets.

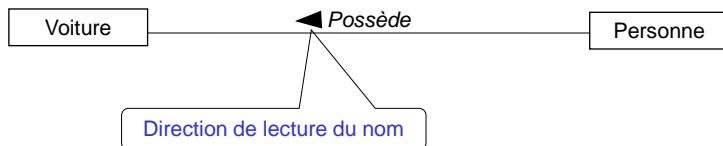
Chaque association porte un nom qui traduit sa signification. Il est recommandé d'utiliser des verbes, à l'actif ou au passif selon le sens de lecture que l'on souhaite privilégier. Le nom est en italique et placé au milieu de la ligne qui symbolise l'association.



## MODELISATION STATIQUE (Diagramme de classes)

### Le concept d'association simple

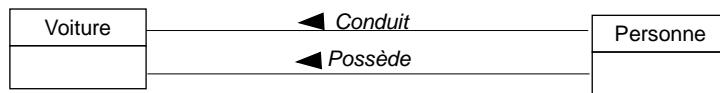
Au niveau analyse, les associations ne sont pas orientées. Pour améliorer la lisibilité, on peut indiquer le sens de lecture du nom par un petit triangle dirigé vers la classe désignée par la forme verbale.



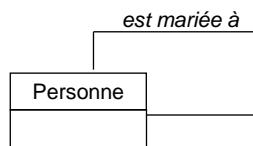
## MODELISATION STATIQUE (Diagramme de classes)

### Le concept d'association simple

Il peut y avoir deux ou plusieurs associations de nature différente entre les deux mêmes classes.



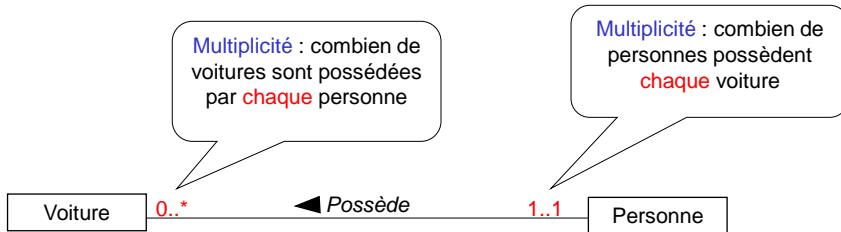
Une association peut affecter une seule classe



## MODELISATION STATIQUE (Diagramme de classes)

### Multiplicité d'une association

La multiplicité d'une association indique le nombre **minimum** et **maximum** d'objets de la classe qui peuvent être liés à **un** objet de l'autre classe. C'est la traduction d'une règle de gestion.



## MODELISATION STATIQUE (Diagramme de classes)

### Multiplicité d'une association

Pour **une** instance de la classe Voiture, il y a au minimum 1 instance de la classe Personne et au maximum 1.



Pour **une** instance de la classe Personne, il y a au minimum 0 instance de la classe Voiture et au maximum n (avec n>1).

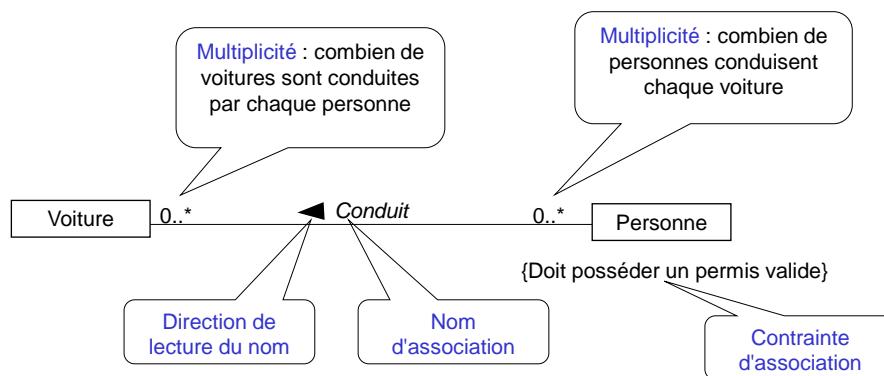
## MODELISATION STATIQUE (Diagramme de classes)

### Multiplicité d'une association

1..1 (ou 1)	Un et un seul
0..1	Zéro ou un
M..N	De M à N (entiers naturels)
*	De zéro à plusieurs
0...*	De zéro à plusieurs
1...*	De un à plusieurs

## MODELISATION STATIQUE (Diagramme de classes)

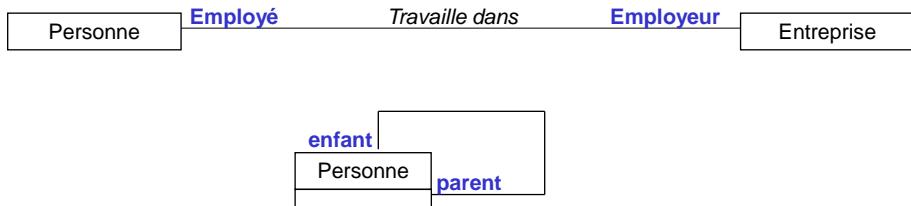
### Le concept d'association – synthèse



## MODELISATION STATIQUE (Diagramme de classes)

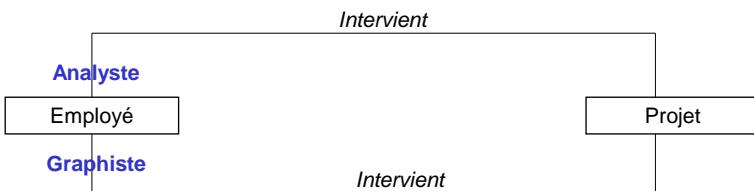
### Les rôles dans les associations

Le rôle permet d'exprimer la manière dont l'objet participe à la relation. Chaque rôle est placé à l'extrémité de l'association, du côté de l'objet qui joue le rôle.



## MODELISATION STATIQUE (Diagramme de classes)

### Les rôles dans les associations

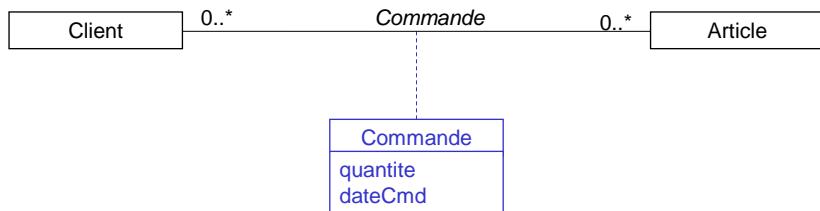


## MODELISATION STATIQUE (Diagramme de classes)

### Notion de classe d'association

Une classe d'association encapsule des informations (attributs ou opérations) concernant une association. Elles sont souvent utilisées dans les associations de type plusieurs à plusieurs.

Une classe de ce type peut participer à d'autres relations dans le modèle.

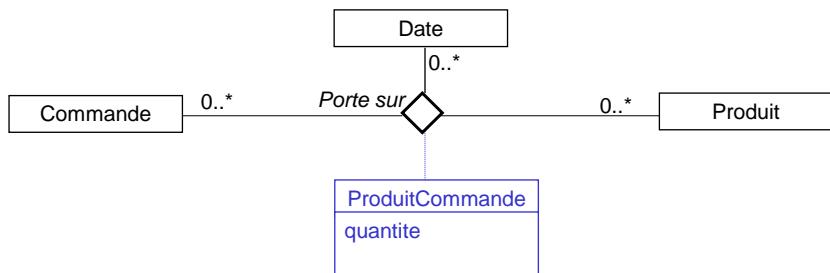


## MODELISATION STATIQUE (Diagramme de classes)

### Association ternaire

Une association ternaire fait intervenir trois classes. Elle est graphiquement représentée par un losange.

Une association ternaire est généralement porteuse d'attributs

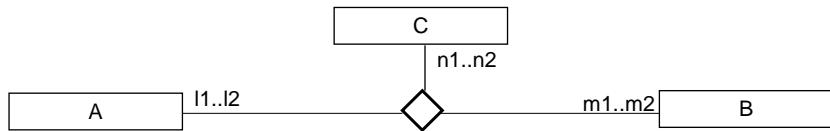


## MODELISATION STATIQUE (Diagramme de classes)

### Association ternaire

Pour une association ternaire, les multiplicités se lisent de la façon suivante :

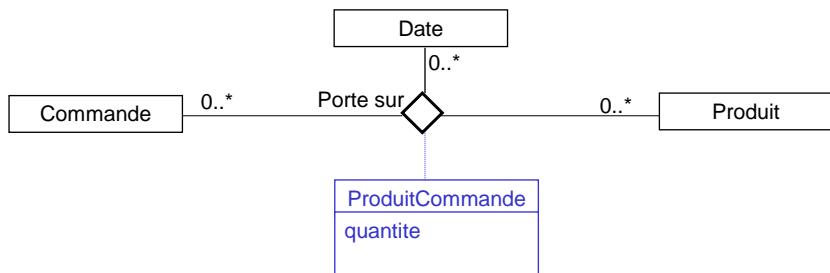
*Pour un couple d'instances de la classe A et de la classe B, il y a au minimum n1 instances de la classe C et au maximum n2.*



## MODELISATION STATIQUE (Diagramme de classes)

### Association ternaire

Pour une commande et un produit donnée, on peut trouver une date pour laquelle le produit n'a pas été commandé, le produit peut également avoir été commandé pour plusieurs dates.



## MODELISATION STATIQUE (Diagramme de classes)

### Contraintes sur les associations

De manière générale, une contrainte UML est une règle de gestion attachée à un élément du modèle. Elle est exprimée entre accolades, en langage naturel ou formel (ex : en OCL).

Une contrainte peut porter sur une association ou sur un groupe d'associations.

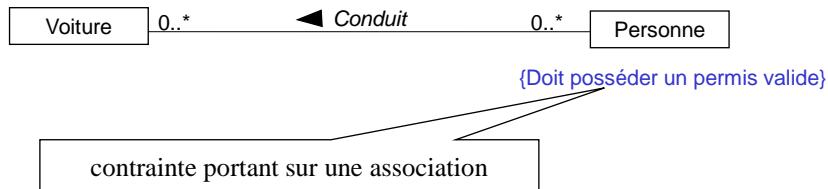
On distingue principalement deux types de contraintes sur les groupes d'associations :

contrainte d'inclusion

contrainte d'exclusion

## MODELISATION STATIQUE (Diagramme de classes)

### Contrainte sur une association



## MODELISATION STATIQUE (Diagramme de classes)

### Contrainte d'inclusion

La contrainte d'inclusion permet de préciser qu'une collection est incluse dans une autre. Par exemple, le délégué d'une classe fait obligatoirement partie des étudiants de cette classe.

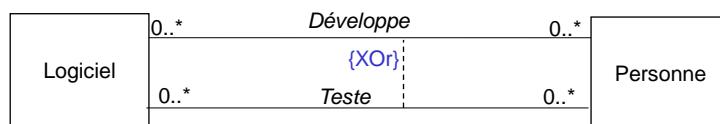


S'il existe un lien "Gère" entre un objet Personne et un objet Classe alors il existe également un lien "*Est dans*" entre ces mêmes objets.

## MODELISATION STATIQUE (Diagramme de classes)

### Contrainte d'exclusion

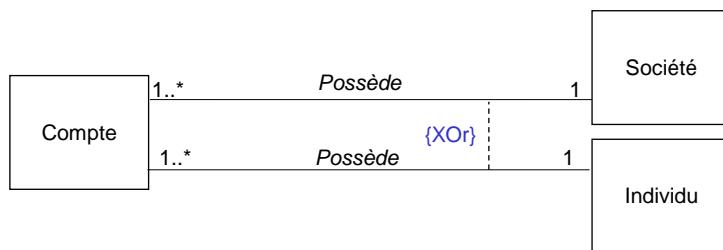
La contrainte d'exclusion permet de préciser qu'une instance d'association exclut une autre instance



La personne qui développe un logiciel ne doit pas tester lui-même ce logiciel.

## MODELISATION STATIQUE (Diagramme de classes)

### Contrainte d'exclusion



Un compte appartient à une société ou à un individu mais pas les deux.

## MODELISATION STATIQUE (Diagramme de classes)

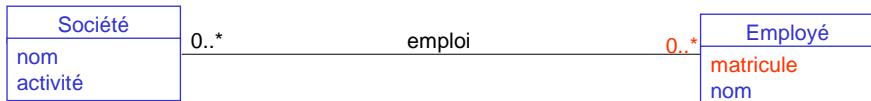
### Association qualifiée

La *qualification d'une association* permet principalement :

- (1) de transformer une multiplicité indéterminée ou infinie en une multiplicité finie
- (2) de limiter l'impact de l'association sur les classes associées.

## MODELISATION STATIQUE (Diagramme de classes)

### Association qualifiée – Changement de multiplicité avec *un qualificateur*



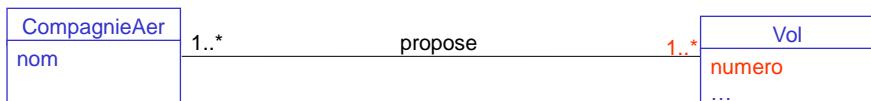
Un employé possède un matricule pour chacun de ses employeurs. Cet attribut permet donc de référencer un employé au sein de sa société. Il s'agit d'un identifiant relatif, dont la valeur permet d'accéder à une instance particulière de Employé, pour une Société donnée.



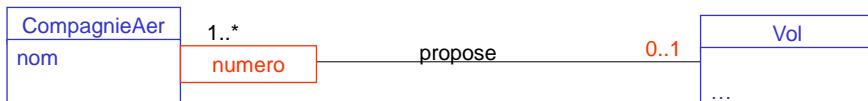
On appelle *qualificateur* d'une instance, une valeur ou un ensemble de valeurs qui permettent de retrouver cette instance.

## MODELISATION STATIQUE (Diagramme de classes)

### Association qualifiée – Changement de multiplicité avec *un qualificateur*

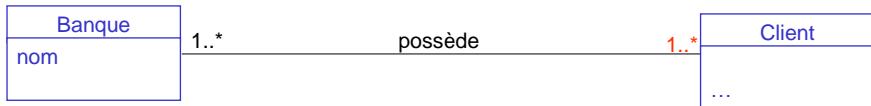


Chaque vol est identifié d'une façon unique par un numéro propre à chaque compagnie

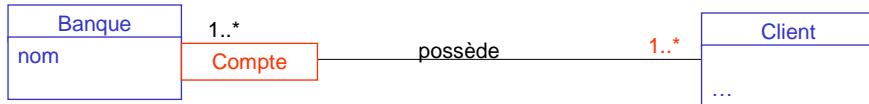


## MODELISATION STATIQUE (Diagramme de classes)

### Association qualifiée – Limiter l'impact d'une association sur les classes associées



L'association entre le client et la banque se limite à la possession de plusieurs comptes, les autres activités de la banque ne concernent pas le client. Pour préciser cette association, on ajoute une *classe qualifiante* (Compte) à la classe Banque.

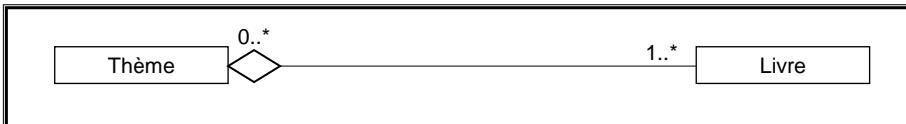


## MODELISATION STATIQUE (Diagramme de classes)

### Agrégation et Composition

L'**agrégation** est une forme particulière d'association non symétrique exprimant une relation de contenance. Elle indique que les objets participants peuvent être assemblés pour créer de nouveaux objets plus complexes.

Les agrégations n'ont pas besoin d'être nommées : implicitement elles signifient « contient », « est composé de ».



**Remarque** : si un thème disparaît, les livres continuent d'avoir une existence propre.

## MODELISATION STATIQUE (Diagramme de classes)

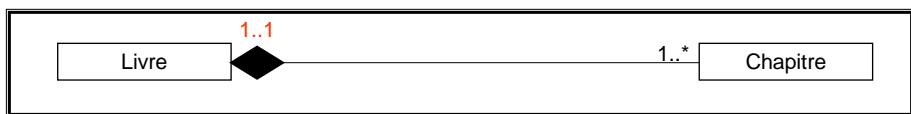
### Agrégation et Composition

La **composition** est une forme particulière d'agrégation (une agrégation plus forte).

Elle implique :

- un élément ne peut appartenir qu'à un seul agrégat composite (agrégation non partagée)
- la destruction de l'agrégat composite entraîne la destruction de tous ses éléments (la durée de vie des parties dépend de celle de l'agrégation).

Cette forme plus forte d'agrégation est indiquée avec un **losange noir**.



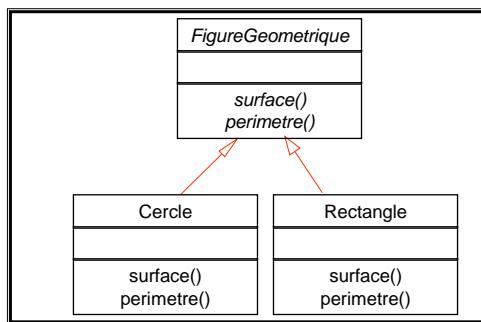
**Remarque** : si un livre est détruit, ses chapitres sont détruits également.

## MODELISATION STATIQUE (Diagramme de classes)

### Classe abstraite

Une **classe abstraite** ne fournit pas une description complète, elle contient au moins une **méthode abstraite** (méthode avec sa seule signature et sans code).

En UML, une classe ou une méthode abstraite sont représentées par le stéréotype «*abstract*». Graphiquement, celui-ci est représenté soit explicitement, soit implicitement avec une mise en italiques du nom de la classe ou de la méthode.

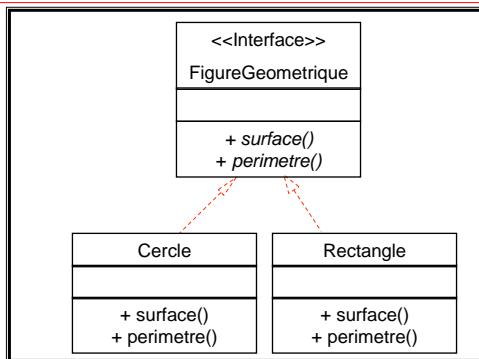


## MODELISATION STATIQUE (Diagramme de classes)

### Interface

Une interface est une classe totalement abstraite, c'est à dire sans attribut et toutes les méthodes sont abstraites et publiques. Elle est graphiquement représentée comme une classe avec le stéréotype « interface ».

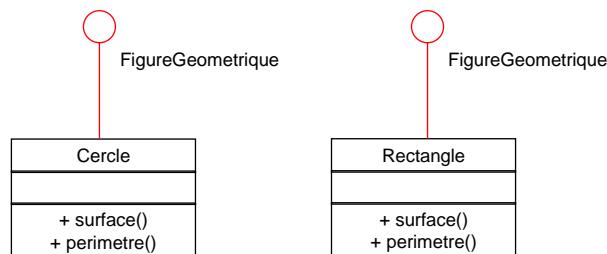
L'implémentation des méthodes est réalisée par une ou plusieurs classes concrètes. La relation entre ces classes et l'interface est appelée *relation de réalisation*.



## MODELISATION STATIQUE (Diagramme de classes)

### Interface

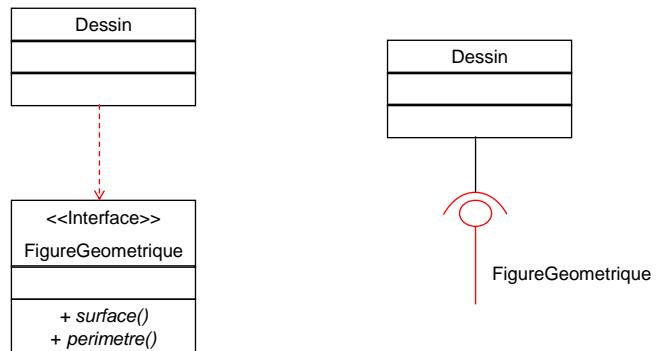
La relation de réalisation peut également être représentée par une lollipop



## MODELISATION STATIQUE (Diagramme de classes)

### Interface

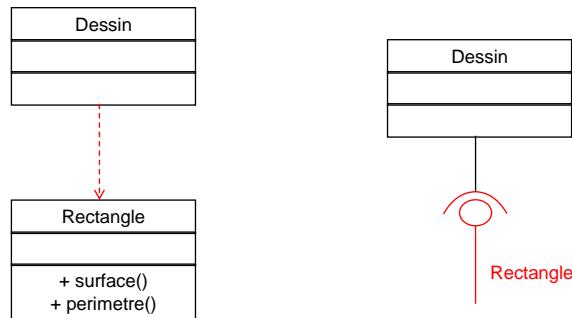
Une classe peut également dépendre d'une interface pour réaliser ses opérations. Cette dernière est alors employée comme type au sein de la classe (attribut, paramètre de l'une des méthodes ou variable locale de l'une des méthodes).



## MODELISATION STATIQUE (Diagramme de classes)

### Remarque

La relation de dépendance entre deux classes utilise la même notation.



## Modélisation dynamique

- Diagrammes d'interaction
  - Diagramme de séquence
  - Diagramme de communication
- Diagramme d'états
- Diagramme d'activités

## Modélisation dynamique

- Diagrammes d'interaction
  - Diagramme de séquence
  - Diagramme de communication

## MODELISATION DYNAMIQUE

(Diagramme d'interaction)

Diagrammes dynamiques : passerelle entre vision externe et interne



Diagramme de cas d'utilisation  
(Vision externe du système)

Diagrammes Dynamiques = **Passerelle**

Diagramme de classes  
(Vision interne du système)

Comment les objets interagissent  
pour réaliser une fonctionnalité !!!

## MODELISATION DYNAMIQUE

(Diagramme d'interaction)

Le diagramme de cas d'utilisation donne une vision fonctionnelle et **externe** du système. Le diagramme de classes, quant à lui en donne une vision statique et structurelle (structure **interne**). Les diagrammes d'interaction permettent de faire la **liaison entre ces deux approches** : ils montrent comment les objets du système interagissent pour réaliser une certaine exigence fonctionnelle.

Une *interaction* décrit le comportement d'une partie du système (sous-système) en se focalisant sur l'échange d'informations entre les éléments du sous-système.

Le terme *diagramme d'interaction* regroupe trois types de diagrammes : *diagramme de séquence*, *diagramme de communication* et *diagramme des temps* (pas abordé ici).

## MODELISATION DYNAMIQUE

(Diagramme d'interaction)

Une *interaction* décrit le comportement d'une partie du système (sous-système) en se focalisant sur l'échange d'informations entre les éléments du sous-système.

Le terme *diagramme d'interaction* regroupe trois types de diagrammes : *diagramme de séquence*, *diagramme de communication* et *diagramme des temps* (pas abordé ici).

## MODELISATION DYNAMIQUE

(Diagramme d'interaction)

Pour interagir entre eux, les objets (instances de classes) s'envoient des messages. Lors de la réception d'un message, un objet devient actif et exécute l'une de ses méthodes (en général la méthode de même nom).

Par conséquent, si nous pouvons modéliser les interactions entre les objets, nous verrons apparaître les opérations que ces objets doivent comporter.

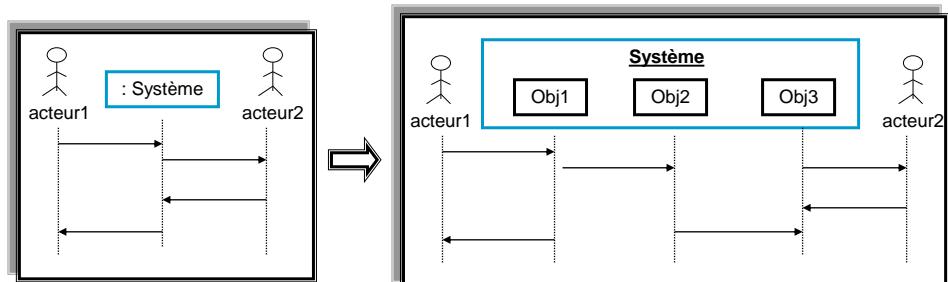
Les diagrammes d'interaction peuvent donc être utilisés comme outil pour identifier les opérations à attribuer à chaque classe.

## MODELISATION DYNAMIQUE

(Diagramme de séquence vs Diagramme de Séquence Système)

Dans l'étape de modélisation fonctionnelle, nous avons vu qu'un cas d'utilisation décrit un ensemble de scénarios ; et que le Diagramme de Séquence Système (DSS) donne une représentation graphique d'un scénario, sous la forme d'une séquence d'interaction entre le système (représenté par une boîte noire) et ses acteurs.

La construction d'un diagramme de séquence revient à remplacer dans le DSS le système (la boîte noire) par une collaboration d'objets.

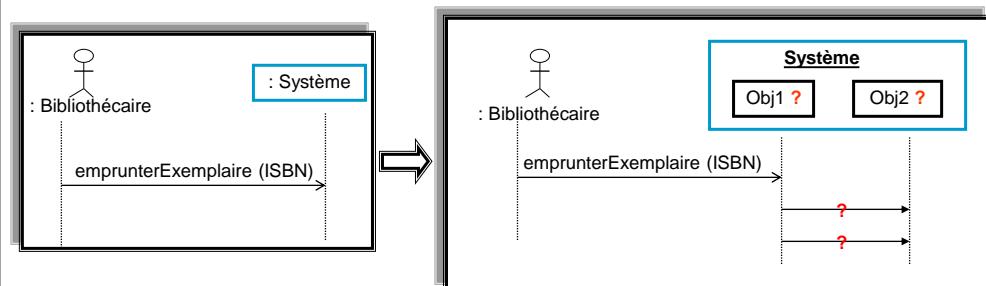


## MODELISATION DYNAMIQUE

(Diagramme de séquence vs Diagramme de Séquence Système)

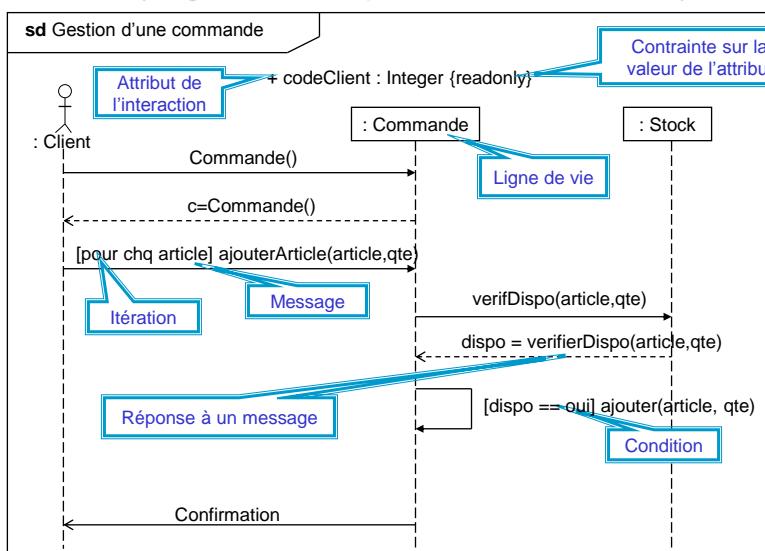
Le problème revient à étudier le *diagramme de classes d'analyse* pour trouver les objets (instances de classes) susceptibles d'intervenir dans le cadre du scénario (représenté par le DSS).

**Attention** : on peut également introduire des objets artificiels (objets de conception)



## MODELISATION DYNAMIQUE

(Diagramme de séquence – notation standard)



## MODELISATION DYNAMIQUE

(Diagramme de séquence – notation standard)

On distingue principalement trois types de messages :

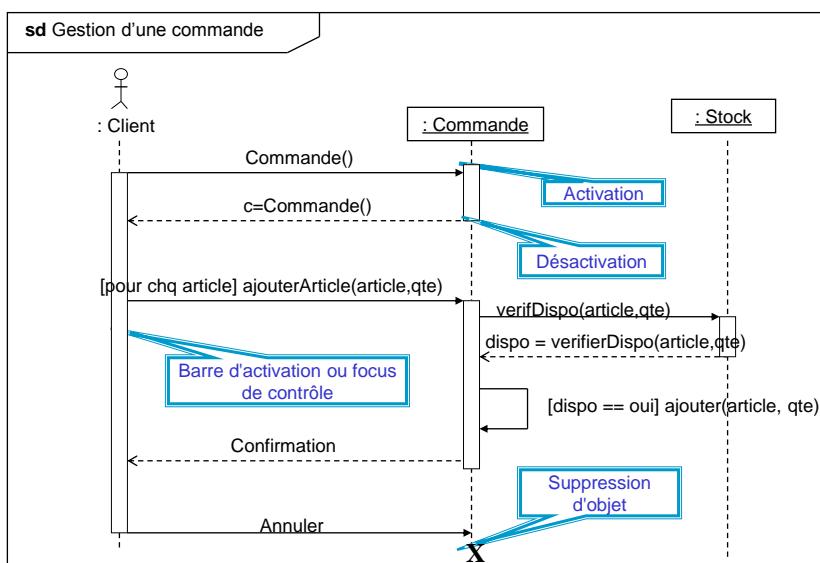


Un message *synchrone* signifie que l'objet émetteur se bloque en attendant la réponse du récepteur du message. En d'autres termes, l'objet émetteur attend que l'activation de la méthode invoquée chez le récepteur soit terminée avant de continuer son activité.

Un message *asynchrone* signifie que l'objet émetteur n'attend pas la réponse du récepteur et poursuit sa tâche sans se soucier de la réception de son message.

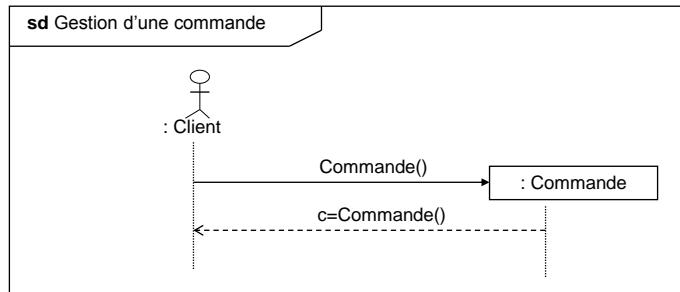
## MODELISATION DYNAMIQUE

(Diagramme de séquence – notation étendue)



## MODELISATION DYNAMIQUE

(Diagramme de séquence – modélisation de la création des objets)

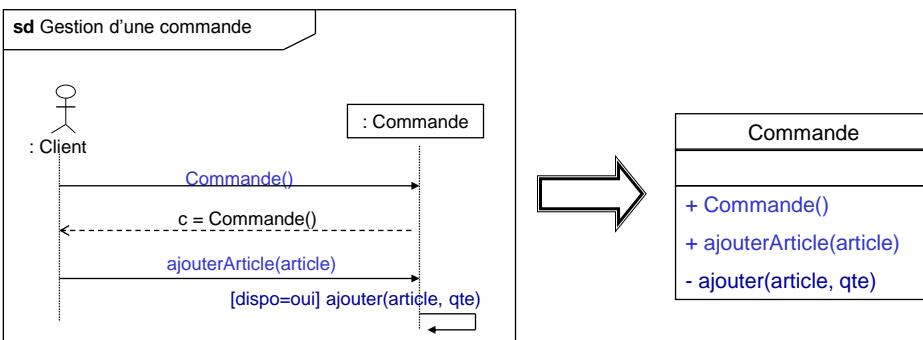


Le constructeur pointe directement sur l'objet, qui est donc placé plus bas, là où sa création est effectuée. Cette notation implique que les objets situés en haut du diagramme existent avant le démarrage du scénario.

## MODELISATION DYNAMIQUE

(Diagramme de séquence vs Opération)

De manière générale, la réception d'un message est suivie de l'exécution d'une méthode d'une classe. Un message entre deux objets se traduit par la création d'une opération publique sur la classe de l'objet récepteur. Un message envoyé par un objet à lui même correspond à un opération privée.

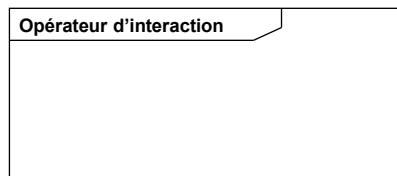


## MODELISATION DYNAMIQUE

(Diagramme de séquence - Les cadres d'interaction)

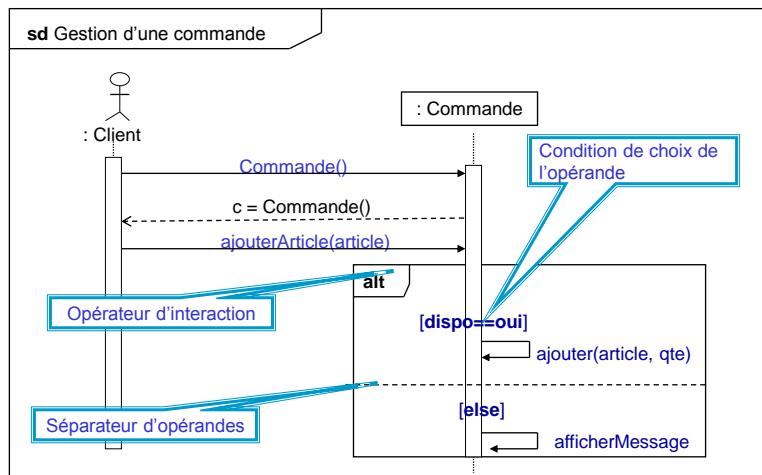
Un cadre d'interaction (fragment combiné) est une partie du diagramme de séquence associé à une étiquette. Elle contient un opérateur qui en détermine la modalité d'exécution.

Un cadre d'interaction se représente de la même façon qu'une interaction



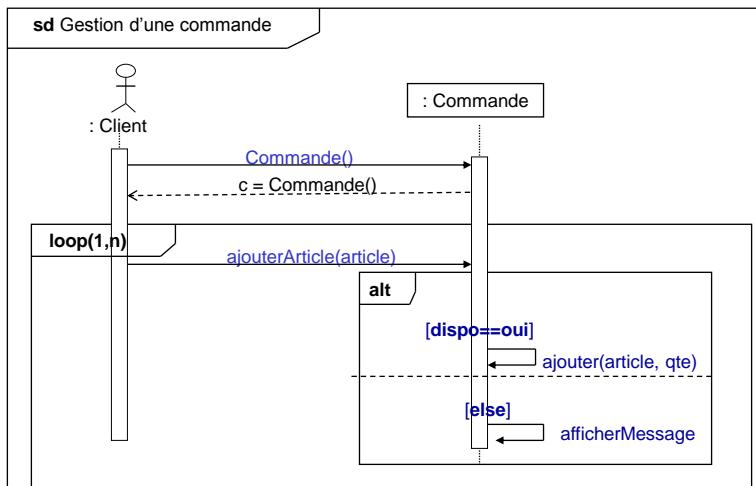
## MODELISATION DYNAMIQUE

(Diagramme de séquence - Utilisation des cadres d'interaction)



## MODELISATION DYNAMIQUE

(Diagramme de séquence - Utilisation des cadres d'interaction)



## MODELISATION DYNAMIQUE

(Diagramme de séquence - Les cadres d'interaction)

### Les opérateurs d'interaction

On distingue principalement les opérateurs suivants :

- ✓ Les opérateurs de choix et de boucle : alternative (alt), option (opt), loop
- ✓ Les opérateurs contrôlant l'envoi en parallèle de messages : parallel (par), critical region (région d'une interaction où les traitements sont atomiques)
- ✓ Les opérateurs contrôlant l'envoi de messages : ignore, consider, assertion (assert) et negative
- ✓ Les opérateurs fixant l'ordre d'envoi des messages : weak sequencing (weak), strict sequencing (strict)

## MODELISATION DYNAMIQUE

(Diagramme de séquence - Les cadres d'interaction)

### L'opérateur *loop*

loop [(min [,max])]

La boucle est répétée au moins *min* fois avant qu'une éventuelle condition booléenne (placée entre crochets sur la ligne de vie) ne soit testée. La boucle continue au plus *max* fois tant que la condition est vraie.

### L'opérateur *ignore*, *consider* et *assertion*

ignore {liste des noms de messages à ignorer}

consider {liste des noms de messages à considérer}

Les messages sont séparés par des virgules

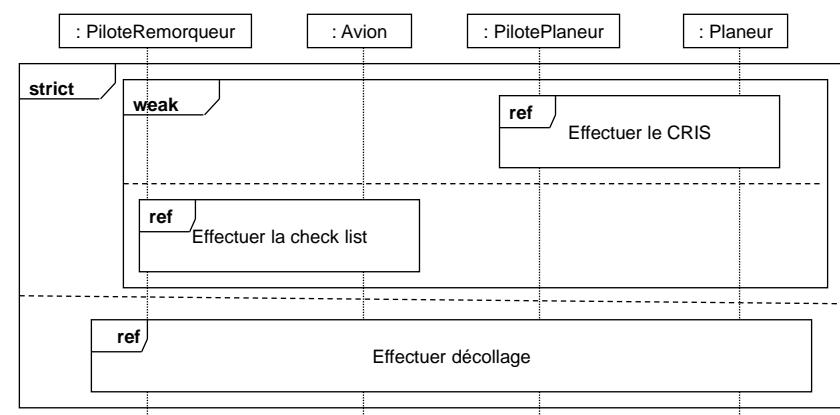
Ces deux opérateurs permettent de ne pas tenir compte de tous les messages.

L'opérateur assert permet de rendre obligatoire certains messages

## MODELISATION DYNAMIQUE

(Diagramme de séquence - Utilisation des cadres d'interaction)

sd Décoller un planeur



L'opérateur *strict sequencing* impose que ses opérandes se déroulent dans l'ordre strict de leur présentation. L'ordre imposé ne s'applique qu'à ses opérandes. Les opérandes sous forme de fragments peuvent avoir en interne, un séquencement quelconque.

## MODELISATION DYNAMIQUE

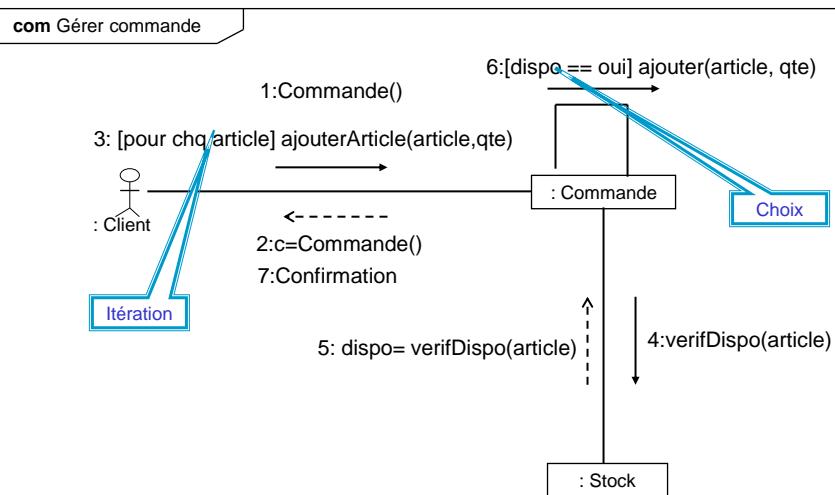
(Diagramme de communication)

Le diagramme de communication répond aux mêmes objectifs que le diagramme de séquence, mais avec une vision différente. Au lieu de représenter les messages dans le temps, il les place par dessus un diagramme d'objets. Certains outils (comme Rational Rose) proposent une commande pour passer de l'un à l'autre.

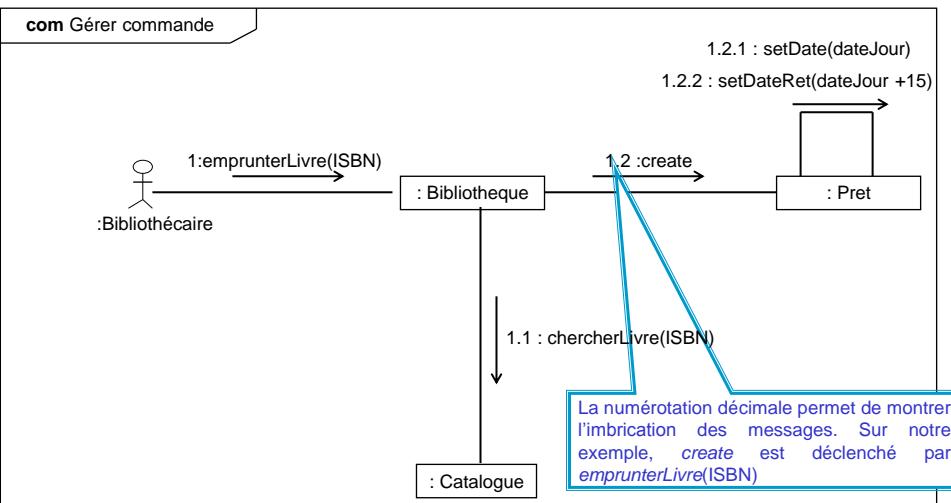
Le diagramme de communication permet donc de représenter les relations structurelles entre les objets qui communiquent.

## MODELISATION DYNAMIQUE

(Diagramme de communication)



## MODELISATION DYNAMIQUE (Diagramme de communication)



## MODELISATION DYNAMIQUE (Diagramme d'interaction vs Diagramme de classes)

Lors de la conception du *diagramme de classes d'analyse*, toutes les opérations des classes ne sont pas identifiées. L'identification complète se fait pendant la construction du *diagramme de classes de conception*. Construire un *diagramme de classes de conception* revient à compléter (affiner) le *diagramme de classes d'analyse*.

Les *diagrammes d'interaction* peuvent être utilisés comme outils pour faciliter le passage du *diagramme de classes d'analyse* au *diagramme de classes de conception*.

## Modélisation dynamique

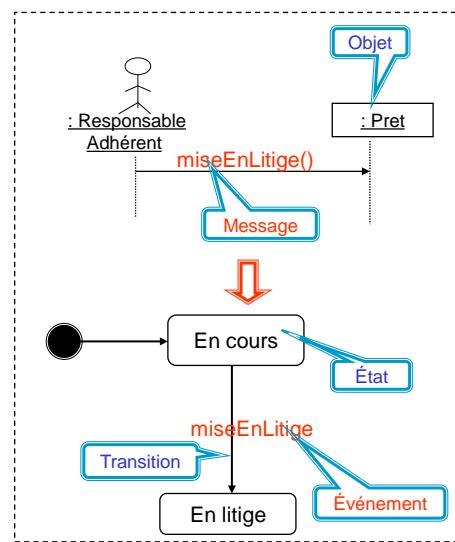
### Diagramme d'états

## MODELISATION DYNAMIQUE

(Diagramme d'interaction vs Diagramme d'états)

Les **diagrammes d'interaction** (séquence et communication) modélisent les interactions entre les objets du système; le **diagramme d'états** modélise les effets de ces interactions sur la configuration interne des objets.

Un diagramme d'interaction ne porte que sur un scénario particulier d'utilisation du système, alors que le diagramme d'état concerne la vie entière (tous les scénarios) d'un objet.



## MODELISATION DYNAMIQUE

(Diagramme d'états)

Les concepts fondamentaux du diagramme d'états sont :

- État
- Événement/Message
- Transition.

## MODELISATION DYNAMIQUE

(Diagramme d'états – concept d'état)

### Le concept d'état

L'état d'un objet représente une situation (de gestion) durable de sa vie et à laquelle on associe des règles de gestion et des activités particulières.

Pendant qu'il se trouve dans un état, un objet peut :

- exécuter une certaine activité (exécution d'une série de méthodes et d'interactions avec d'autres objets)
- ou bien attendre un certain événement

## MODELISATION DYNAMIQUE

(Diagramme d'états – concept d'état)

### Le concept d'état

L'état d'un objet est défini à la fois par les valeurs de ses attributs et par les valeurs de ses associations avec d'autres objets.

L'état d'un objet a une **durée finie**, quantifiable à l'échelle du système, un espace de temps séparé par deux événements.

## MODELISATION DYNAMIQUE

(Diagramme d'états – concept d'état)



Représentation graphique d'un état

Profession = Informaticien  
Participation à l'association "Travaille dans"



En poste

Profession = aucune  
Pas de Participation à l'association "Travaille dans"



A la retraite

## MODELISATION DYNAMIQUE

(Diagramme d'états – concept d'événement)

### Le concept d'événement

Un événement est un **fait significatif** qui peut déclencher un changement d'état. Il est lié à la réception d'un signal (réception d'un message) par l'objet.

Exemple : l'arrivée d'une commande, l'arrivée d'une demande de réservation ...

Un événement peut transporter des informations (paramètres). Il **n'a pas de durée** contrairement à un état.

Exemple de paramètres : un bon de commande, un formulaire de réservation, ...

## MODELISATION DYNAMIQUE

(Diagramme d'états – concept de transition)

### Le concept de transition

Une *transition* est le changement (possible) d'état d'un objet, causé par un événement. Elle est représentée par un lien orienté entre deux états, la flèche porte le nom (et les paramètres) de l'événement associé.



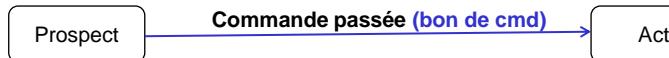
## MODELISATION DYNAMIQUE

(Diagramme d'états – concept de transition)

### Le concept de transition

#### Attributs

Les *attributs* correspondent à des informations ou à des paramètres portés par les événements ; ils sont représentés entre parenthèses après le nom de l'événement.



## MODELISATION DYNAMIQUE

(Diagramme d'états – concept de transition)

### Le concept de transition

#### Gardiens

Les *gardiens* ou *conditions de garde* sont des expressions booléennes ; ils sont représentés entre crochets après la liste des attributs. Une transition portant un gardien ne peut être effectuée que si le gardien est vérifié.

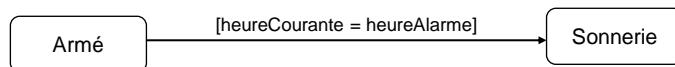


## MODELISATION DYNAMIQUE

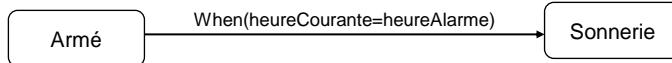
(Diagramme d'états – transitions particulières)

### Transition automatique

Une transition automatique ne correspond pas à la réception d'un message. Elle est franchie dès que l'objet a terminé l'activité liée à l'état d'origine.



UML propose le mot-clé *When (expression booléenne)* pour exprimer les transitions déclenchées par les événements internes.



## MODELISATION DYNAMIQUE

(Diagramme d'états – transitions particulières)

### Transition réflexive

Une transition réflexive possède le même état d'origine et de destination. Un événement associé à une telle transition ne fait pas changer l'état de l'objet, mais provoque une réaction (appel d'une méthode, envoi d'un signal à d'autres objets).



## MODELISATION DYNAMIQUE

(Diagramme d'états – concept de transition)

### Les traitements

Les opérations utilisées pour décrire les classes sont représentées dans le diagramme d'états par des activités (ou des actions).

Ces activités sont associées à des transitions ou à des états.

Une activité (ou une action) peut donc être déclenchée :

- au moment où une transition se produit
- ou bien par un événement interne à un état

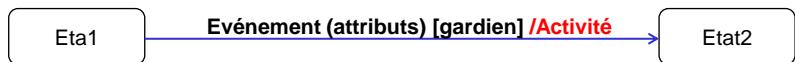
## MODELISATION DYNAMIQUE

(Diagramme d'états – concept de transition)

### Les traitements

#### Activité déclenchée par une transition

Les activités spécifiées lors du franchissement d'une transition sont représentées directement sur la transition, précédée d'un /



## MODELISATION DYNAMIQUE

(Diagramme d'états – concept de transition)

### Les traitements

#### **Activités spécifiées dans un état**

Les activités spécifiées dans un état sont déclenchées par *trois types d'événements* :

- l'entrée dans l'état*
- la sortie de l'état*
- un événement, appelé transition interne.*

*Une transition interne laisse l'objet dans le même état.*

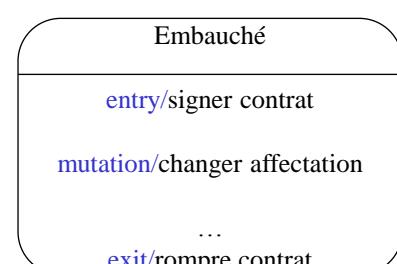
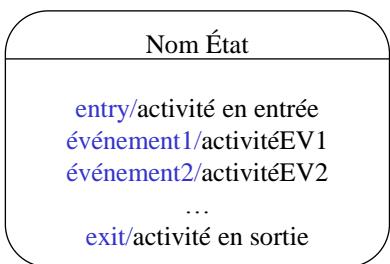
## MODELISATION DYNAMIQUE

(Diagramme d'états – concept de transition)

### Les traitements

#### **Activités spécifiées dans un état**

La représentation graphique



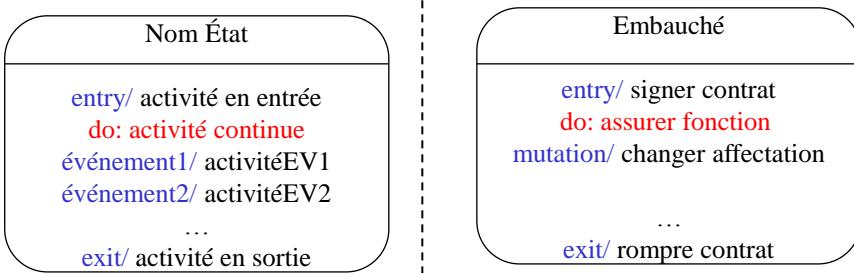
## MODELISATION DYNAMIQUE

(Diagramme d'états – concept de transition)

### Les traitements

#### Activités continues

On peut également associer à un état des activités continues qui durent tant que l'objet est dans l'état. Une activité continue n'est pas interrompue par les transitions internes. Elle ne s'arrête qu'à la sortie de l'état. Elle est représentée par le mot clé *do*.



## MODELISATION DYNAMIQUE

(Diagramme d'états – concept de transition)

### Les traitements

#### Le branchement et la synchronisation

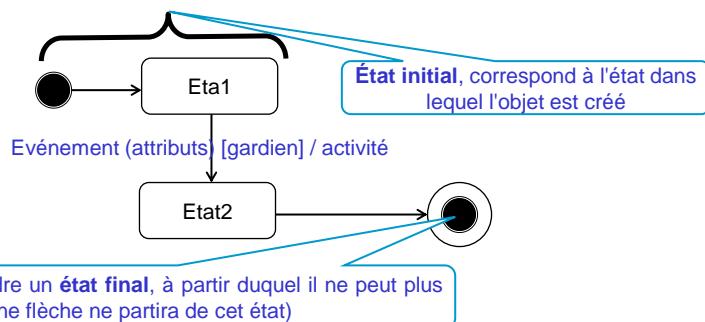
Le diagramme d'état permet également de représenter des chemins alternatifs (branchement conditionnel) ou parallèles (synchronisation et fourche). Pour cela, on utilise le même formalisme que dans le diagramme d'activités.

## MODELISATION DYNAMIQUE

(Diagramme d'états – concept de transition)

### État initial – état final

L'état initial d'un objet est obligatoire et unique. L'action de création est généralement associée à cet état. Un objet peut avoir plusieurs états finaux ; les événements conduisant à ce type d'état sont généralement associés à une action de destruction ou d'archivage.

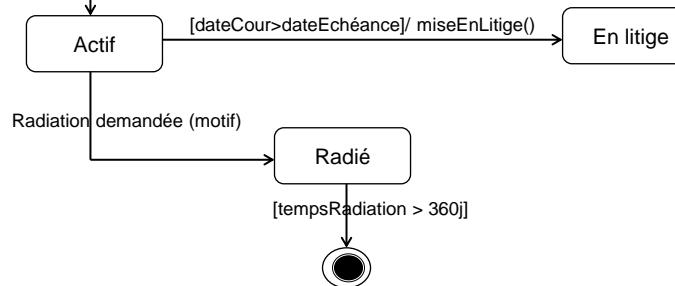


## MODELISATION DYNAMIQUE

(Diagramme d'états – Exemple)

### Classe Client

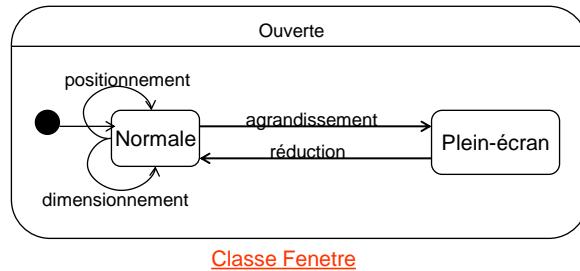
Commande passée (bon de cmd) / ajouterCmd(commande)



## MODELISATION DYNAMIQUE (Diagramme d'états – État composé)

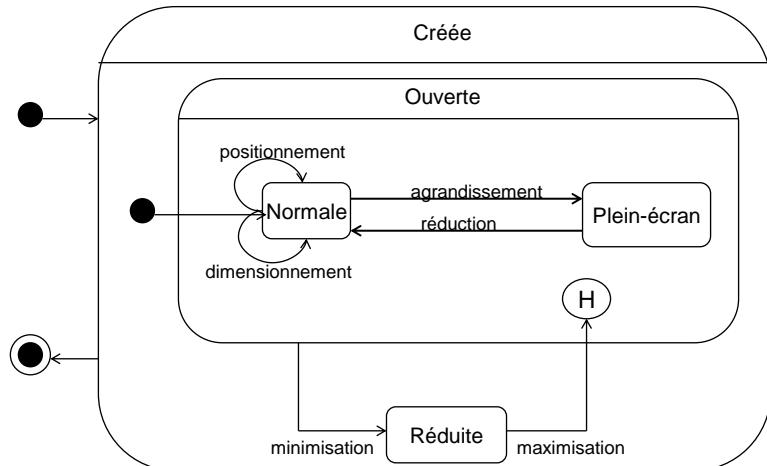
Un état peut être décrit lui-même par un diagramme d'états. On parle alors *d'état composé*. Les états qui le composent sont appelés *sous-états*.

Dès que l'objet passe dans l'état composé, il passe automatiquement dans le sous-état initial du diagramme interne. Si l'objet franchit une transition qui fait sortir de l'état composé, il quitte également les sous-états.



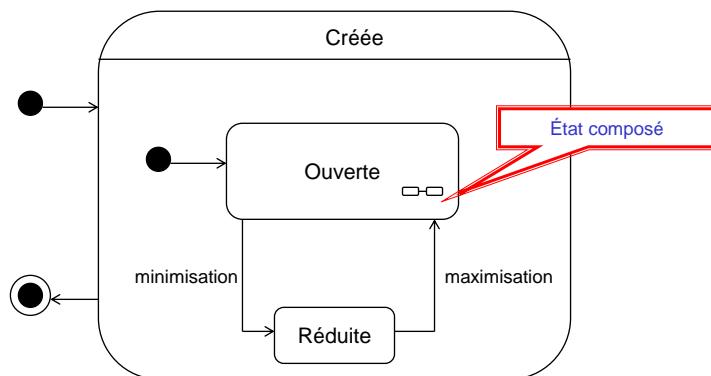
## MODELISATION DYNAMIQUE (Diagramme d'états – État composé)

Classe Fenetre



## MODELISATION DYNAMIQUE (Diagramme d'états – État composé)

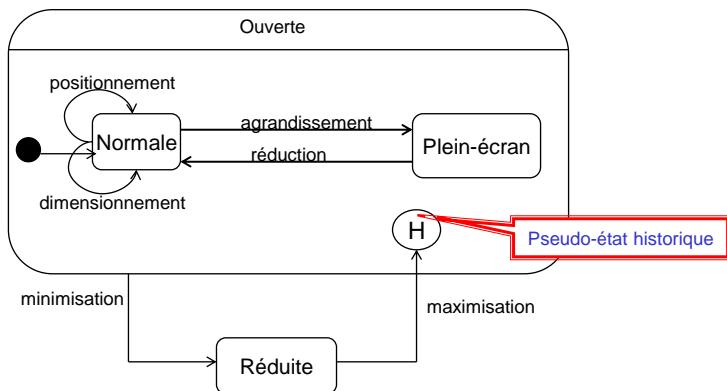
### Notation abrégée des états composés



## MODELISATION DYNAMIQUE (Diagramme d'états – État composé)

### Pseudo-état historique

Il est possible, lorsqu'un objet quitte un état composé, de mémoriser le sous-état actif pour y revenir. On utilise pour cela le **pseudo-état historique** qui représente dans l'état composé le dernier sous-état actif mémorisé.

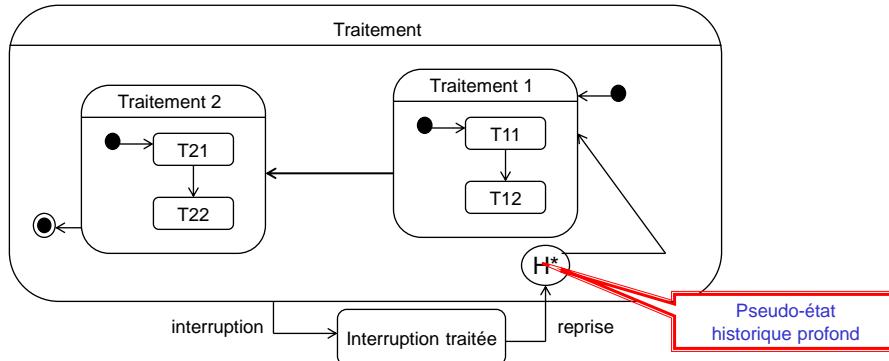


## MODELISATION DYNAMIQUE (Diagramme d'états – État composé)

### Pseudo-état historique profond

Un sous-état peut être lui-même composé de sous-états. Dans ce cas, il est possible de définir un *pseudo-état historique profond* qui permet d'atteindre le dernier sous-état visité quel que soit son niveau d'imbrication.

Il est également possible de définir un dernier *sous-état visité par défaut* à l'aide d'une transition ayant pour source le pseudo-état (H ou H\*).



## MODELISATION DYNAMIQUE (Diagramme d'états – État composé)

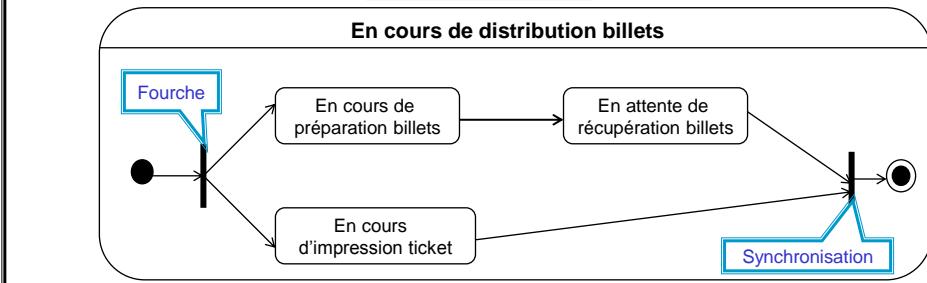
### Transition concurrente

Il est possible d'avoir au sein d'un état composé, des sous-états qui évoluent en parallèle. Pour représenter cela, on utilise les *transitions concurrentes*.

*Fourche* : une fois franchie, l'objet se trouve dans tous les sous-états de destination

*Synchronisation* : pour pouvoir continuer, tous les sous-états d'origine doivent être prêts à franchir la transition de rendez-vous.

### Un sous-état du GAB



## Modélisation dynamique

### Diagramme d'activités

## MODELISATION DYNAMIQUE

(Diagramme d'activités)

### Objectif

Le diagramme d'activités permet de représenter la dynamique du système. C'est un graphe orienté qui décrit un enchaînement de traitements qui peut être soumis à des branchements ou à des synchronisations.

Un traitement peut être une action (une opération élémentaire) ou une activité, pouvant elle-même être décrite à un niveau plus fin.

La visualisation de couloirs d'activités permet de représenter la répartition de la responsabilité des activités entre différents acteurs.

## MODELISATION DYNAMIQUE

(Diagramme d'activités)

### Utilisation

Le diagramme d'activités peut être utilisé à deux niveaux :

#### Niveau microscopique

- Il permet pour une classe donnée, d'exprimer graphiquement une activité (donc une méthode) du diagramme d'état en un algorithme
- traduction directe possible dans un langage de programmation O.O

#### Niveau macroscopique

Il permet de donner une vue macroscopique de l'enchaînement des fonctionnalités.

Exemple : enchaînement temporelle des cas d'utilisation d'un diagramme de cas d'utilisation, représentation d'un processus métier,

## MODELISATION DYNAMIQUE

(Diagramme d'activités)

Concepts de base (Rappel des concepts présentés dans la chapitre *description des cas d'utilisation*) :

- Activité
- Transition
- Condition de garde
- Décision
- Parallélisme
- Point de départ / Point d'arrivée

## MODELISATION DYNAMIQUE (Diagramme d'activités)

### Activité et Transition

Une *activité* est une série d'actions (Ex: créer ou modifier un objet, réaliser une opération, ...) qui vise à effectuer un certain travail. Il peut s'agir d'un calcul, de la recherche de données, de la manipulation d'informations, ...

Les activités sont reliées par des *transitions*, représentées par des flèches. La transition a lieu lorsque l'activité d'origine est terminée.

### Condition de garde

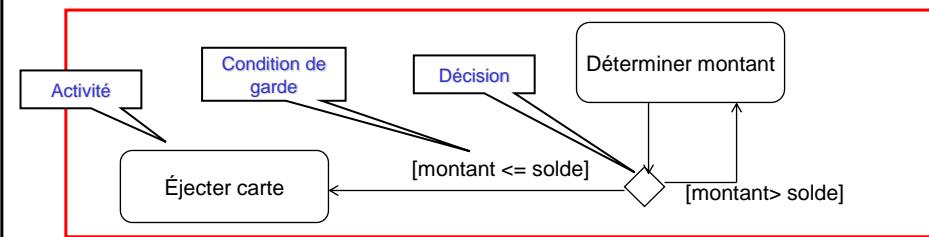
Une condition peut être affectée à une transition pour restreindre son utilisation. La condition est notée entre crochets sur la ligne de la transition. Elle doit être vraie pour que l'exécution se poursuive avec la prochaine activité

## MODELISATION DYNAMIQUE (Diagramme d'activités)

### Décision

Les losanges, sur le diagramme d'activités, représentent des décisions. Ils peuvent s'agir d'un simple test vrai/faux, ou d'un choix parmi plusieurs options. Chaque option est identifiée à l'aide d'une condition de garde. Les conditions doivent être mutuellement exclusives de façon qu'un seul choix soit possible.

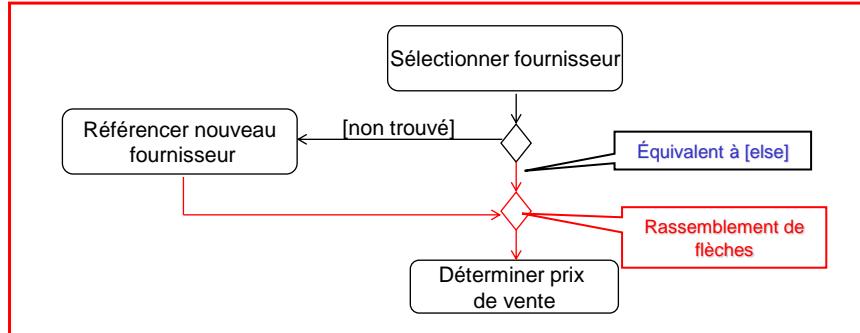
Il peut y avoir une transition sans condition de garde, elle a par défaut l'étiquette "else".



## MODELISATION DYNAMIQUE (Diagramme d'activités)

### Rassemblement de flèches

Indépendamment, le losange sert aussi à rassembler des flèches provenant de plusieurs activités

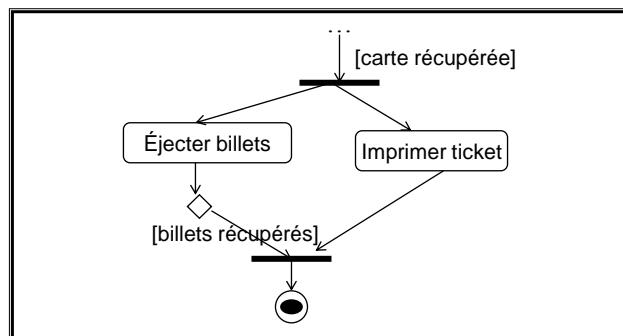


## MODELISATION DYNAMIQUE (Diagramme d'activités)

### Parallélisme

Le parallélisme est modélisé par le concept de *fourche* et/ou *le concept de synchronisation*.

La fourche représente le passage d'une transition à plusieurs. La synchronisation représente le passage de plusieurs transitions à une seule.



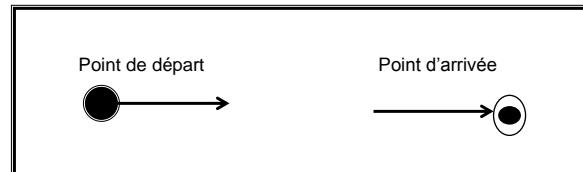
## MODELISATION DYNAMIQUE

(Diagramme d'activités)

### Point de départ / Point d'arrivée

Deux icônes permettent de représenter respectivement le début et la fin de l'exécution de l'ensemble des activités d'un diagramme.

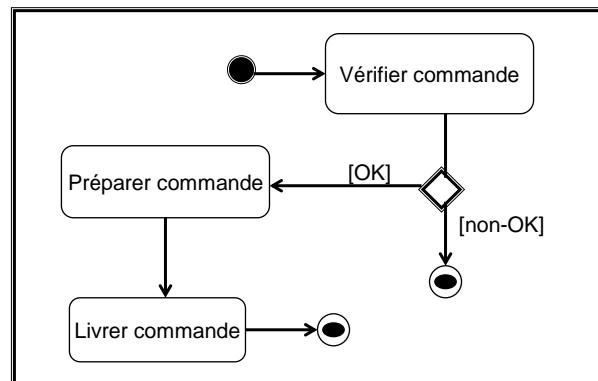
L'icône de fin n'est pas forcément unique et n'est pas obligatoire.



## MODELISATION DYNAMIQUE

(Diagramme d'activités)

### Point de départ / Point d'arrivée



## MODELISATION DYNAMIQUE

(Diagramme d'activités)

Concepts avancés :

- Signal
- Activités composées
- Travées ou partitions
- Entrées et sorties
  - Pin
  - Flot d'objet

## MODELISATION DYNAMIQUE

(Diagramme d'activités)

### Signal

Un signal est une information provenant d'une action externe à l'activité en cours de description. Par exemple l'arrivée d'une commande.

Un signal temporel exprime le passage du temps. Par exemple "attendre 10 seconde", "fin du mois", ...



Fin du mois

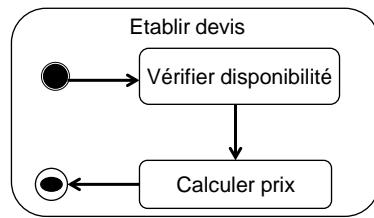


## MODELISATION DYNAMIQUE

(Diagramme d'activités)

### Les activités composées

Une activité peut être composée d'autres activités. Dans ce cas, un diagramme d'activités spécifique en décrit la composition en sous-activités.

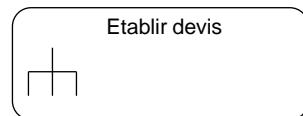


## MODELISATION DYNAMIQUE

(Diagramme d'activités)

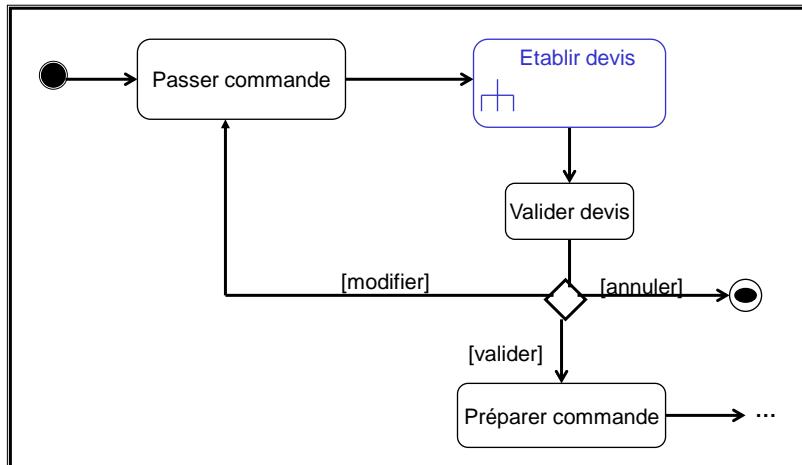
### Les activités composées

Dans les diagrammes où une activité composée est incluse, elle est représentée avec le formalisme ci-dessous :



## MODELISATION DYNAMIQUE (Diagramme d'activités)

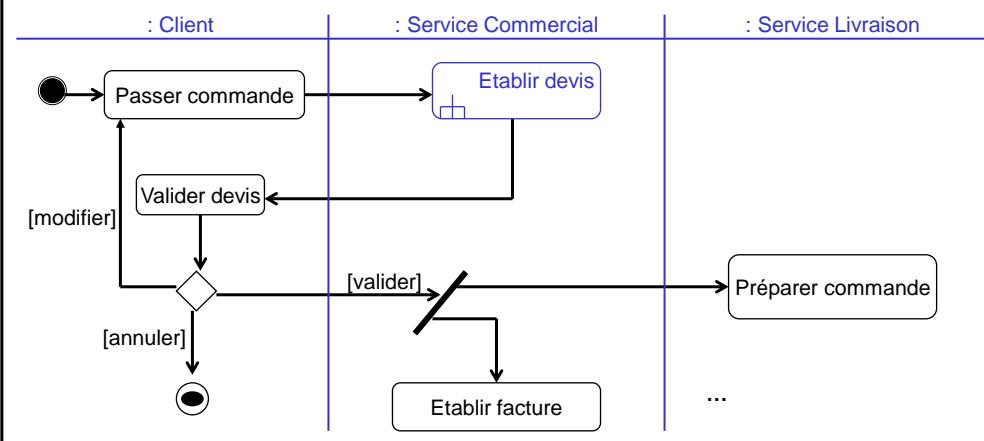
### Les activités composées



## MODELISATION DYNAMIQUE (Diagramme d'activités)

### Les travées ou les partitions

Le diagramme d'activités peut être divisé en travées ou en partitions, afin de mettre en évidence les acteurs et/ou les objets responsables de la réalisation des activités.



## MODELISATION DYNAMIQUE

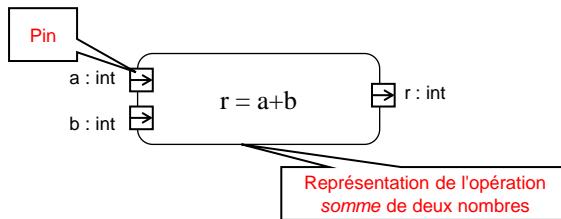
(Diagramme d'activités)

### Entrées et sorties

#### 1- Argument et valeur rentrée

Le concept de **pin** permet de spécifier les valeurs passées en argument à une activité et les valeurs de retour.

Un **pin** représente un point de connexion pour une activité : l'activité ne peut débuter que si l'on affecte une valeur à chacun de ses **pins d'entrée** ; quand elle se termine, une valeur doit être affectée à chacun de ses **pins de sortie**.



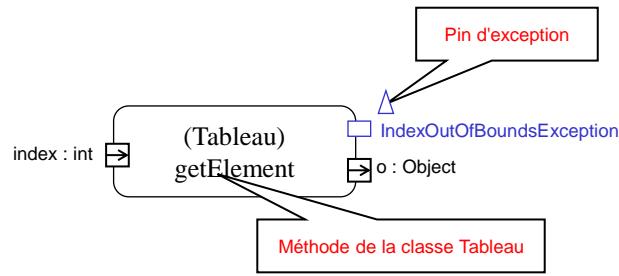
## MODELISATION DYNAMIQUE

(Diagramme d'activités)

### Entrées et sorties

#### 2- Représentation des exceptions

Les pins servent également à représenter les exceptions potentiellement levées par une activité (une opération d'une classe), on parle alors de **pin d'exception**.



## MODELISATION DYNAMIQUE (Diagramme d'activités)

### Entrées et sorties

#### 3 – Flot d'objet

Un flot d'objet permet de faire figurer les objets impliqués dans les activités. Chaque objet est représenté par un rectangle.

Les objets sont attachés aux activités qu'ils concernent par des flèches en pointillées.

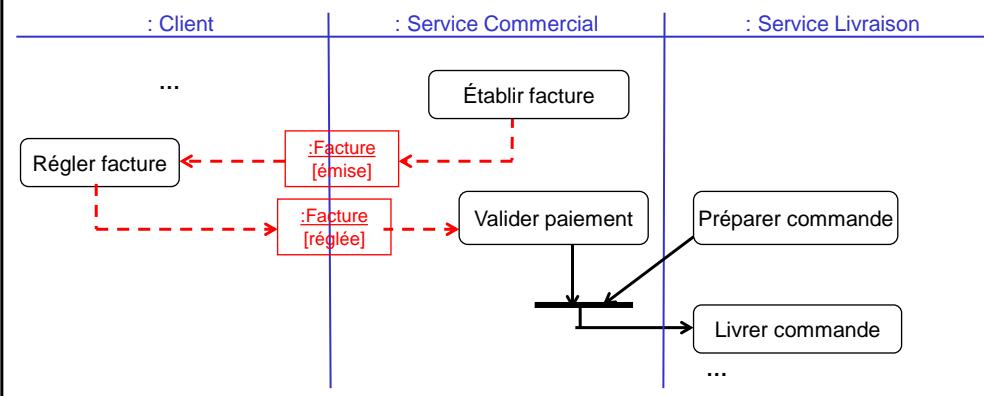
Si un objet sort de l'activité1 et est mis en entrée de l'activité2, il n'est pas nécessaire de faire une flèche directe de l'activité1 vers l'activité2.

On peut faire apparaître plusieurs fois le même objet, avec différents états, les états sont alors mis entre crochets.

## MODELISATION DYNAMIQUE (Diagramme d'activités)

### Entrées et sorties

#### 3 – Flot d'objet - Exemple



## Passage de l'analyse au code

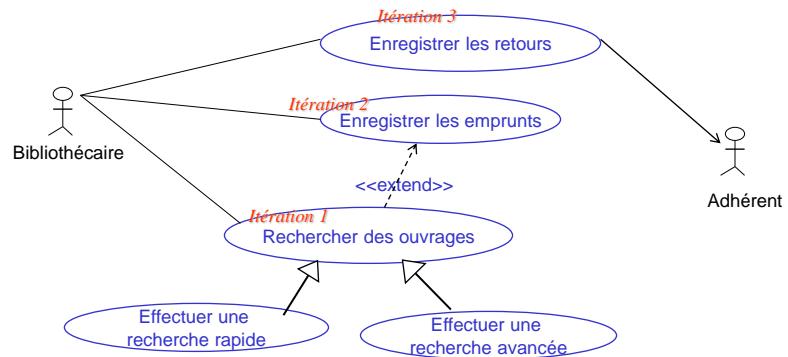
- Hypothèses
- Élaboration du diagramme de classes de conception
- Transformation du diagramme de classes de conception en code
- Exemple

## Passage de l'analyse au code

- Hypothèses

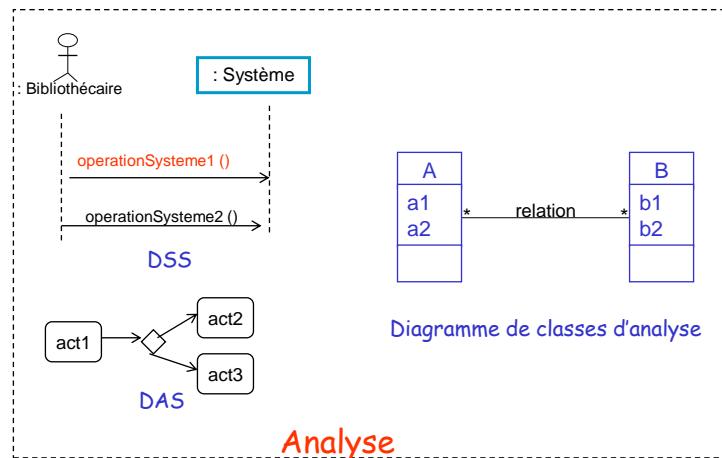
## PASSAGE DE L'ANAYSE AU CODE (Hypothèses)

- (1) La modélisation fonctionnelle est terminée et il existe une version stable du diagramme de cas d'utilisation
- (2) Le diagramme de cas d'utilisation est découpé en plusieurs itérations
- (3) Les itérations du diagramme de cas d'utilisation sont hiérarchisées.



## PASSAGE DE L'ANAYSE AU CODE (Hypothèses)

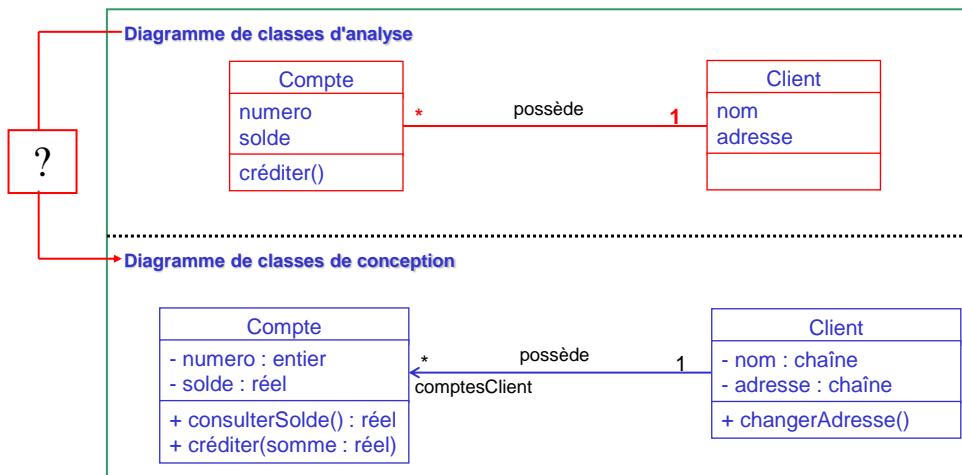
Pour chaque itération (cas d'utilisation)



## Passage de l'analyse au code

- Élaboration du diagramme de classes de conception

### PASSAGE DE L'ANAYSE AU CODE (Élaboration du diagramme de classes de conception)



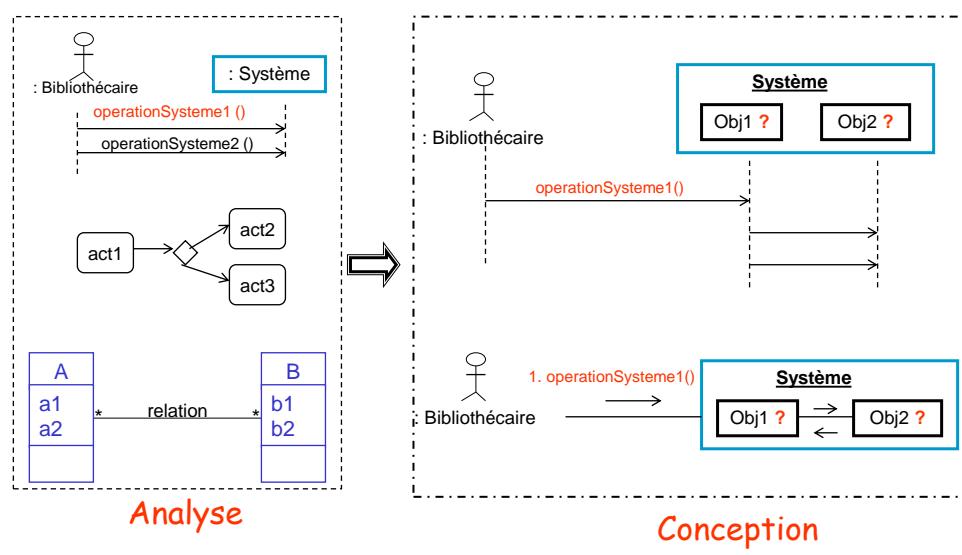
## PASSAGE DE L'ANAYSE AU CODE

(Élaboration du diagramme de classes de conception)



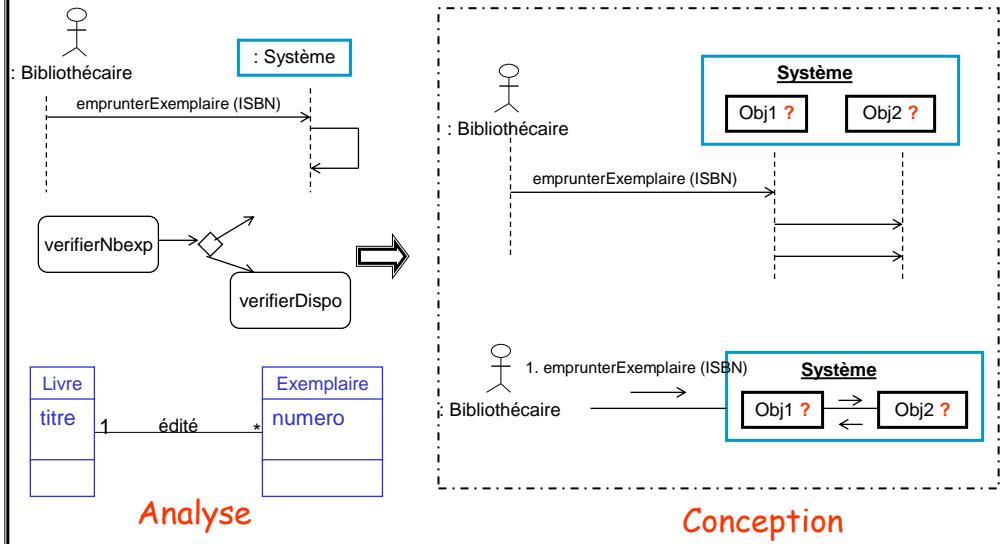
## PASSAGE DE L'ANAYSE AU CODE

(Élaboration du diagramme de classes de conception)



## PASSAGE DE L'ANAYSE AU CODE

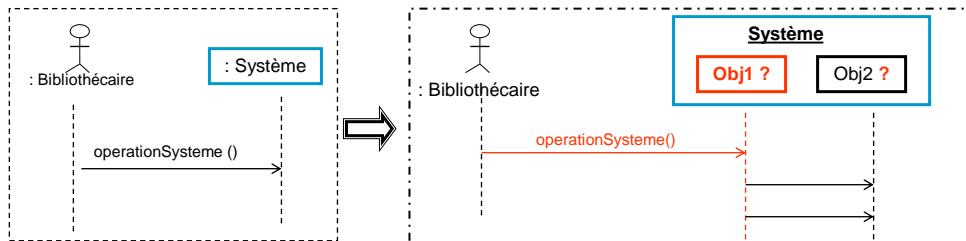
(Élaboration du diagramme de classes de conception)



## PASSAGE DE L'ANAYSE AU CODE

(Élaboration du diagramme de classes de conception)

Choix de l'objet récepteur du message système (opérateur système)



### Première solution

introduire un objet artificiel de conception de type « contrôleur »

### Deuxième solution

(cas d'un système contenant un nombre restreint d'opérations)

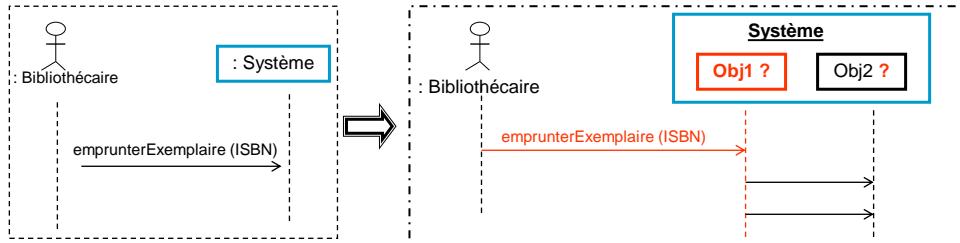
utiliser comme contrôleur une instance d'une classe d'analyse existante

- un objet représentant le système entier ou l'organisme elle-même,
- un objet représentant un rôle qui aurait réalisé l'opération système

## PASSAGE DE L'ANAYSE AU CODE

(Élaboration du diagramme de classes de conception)

Choix de l'objet récepteur du message système (opérateur système)



### Première solution

un objet de la classe *ControleurEmprunts*

### Deuxième solution

(cas d'un système contenant un nombre restreint d'opérations)

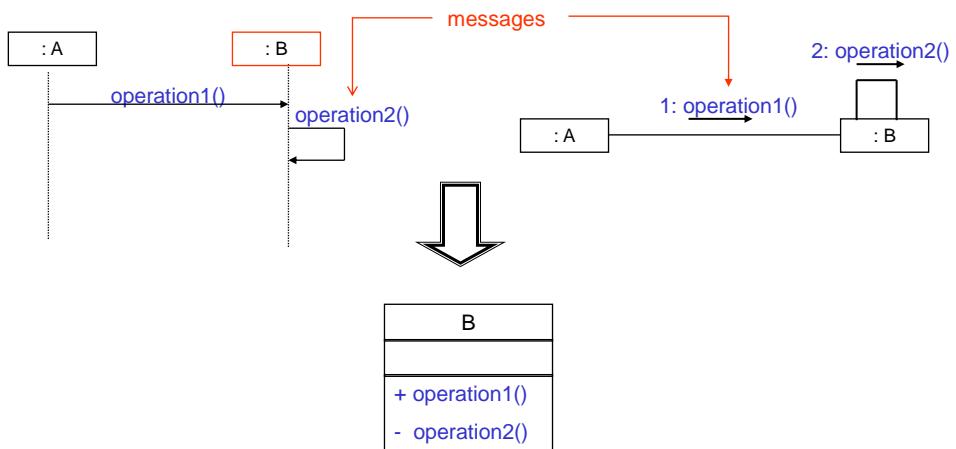
utiliser comme contrôleur une instance d'une classe d'analyse existante

- un objet de la classe *Bibliotheque*,
- un objet de la classe *Bibliothetaire*

## PASSAGE DE L'ANAYSE AU CODE

(Élaboration du diagramme de classes de conception)

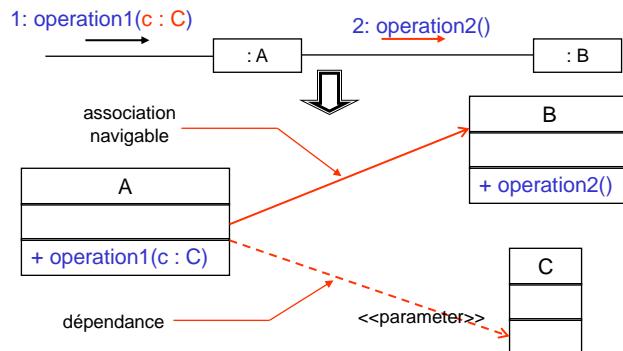
Rapport entre message et opération des classes



## PASSAGE DE L'ANAYSE AU CODE

(Élaboration du diagramme de classes de conception)

### Navigabilité des associations – Dépendances entre classes



Lien durable entre objet



Association navigable entre les classes correspondantes

Lien temporaire (par paramètre ou variable locale)



Dépendance entre les classes correspondantes

## PASSAGE DE L'ANAYSE AU CODE

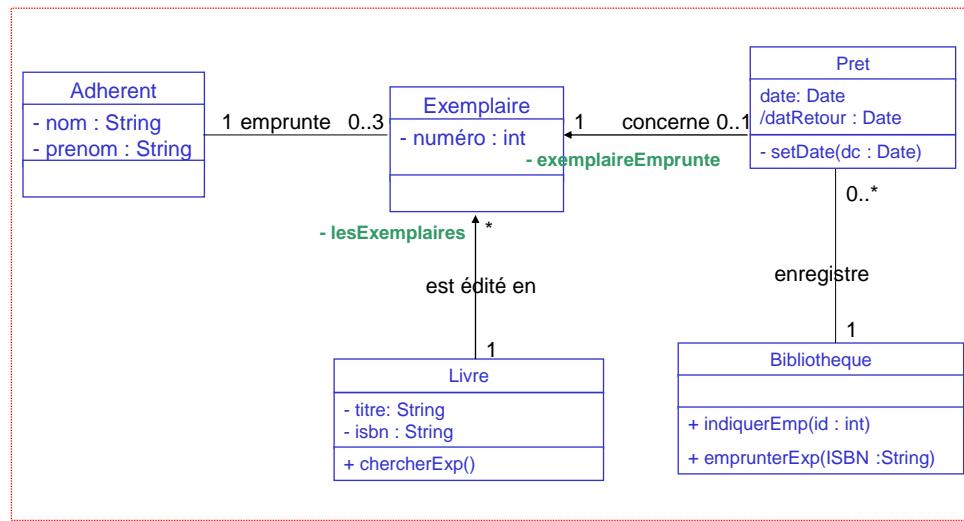
(Élaboration du diagramme de classes de conception)

### Quelques règles

- Ajouter les opérations (voir [diagramme de séquence](#))
- Typer les attributs ainsi que les paramètres et le retour des opérations
- Indiquer la visibilité des attributs en les faisant précédé par + pour *public*, # pour *protected*, - pour *private*
- Restreindre la navigabilité des associations (voir [diagramme de communication](#))
- Préciser les noms des rôles du côté navigable des associations
- Supprimer les classes et associations inutiles.
- Ajouter les dépendances entre classes suite aux liens temporaires entre objets

## PASSAGE DE L'ANAYSE AU CODE

(Élaboration du diagramme de classes de conception)



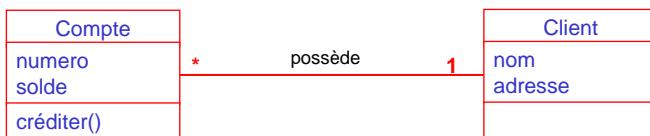
### Passage de l'analyse au code

- Transformation du diagramme de classes de conception en code

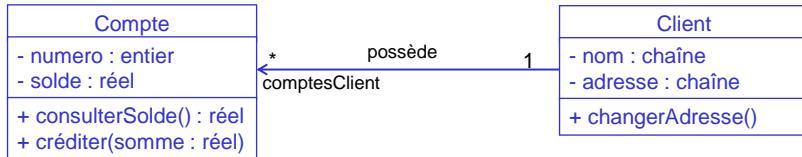
## MODELISATION STATIQUE (Diagramme de classes)

On distingue principalement deux niveaux d'abstraction

### Diagramme de classes d'analyse



### Diagramme de classes de conception



## PASSAGE DE L'ANAYSE AU CODE

(Transformation du diagramme de classes de conception en code)

Les règles présentées ci-dessous visent à produire le squelette de code de l'application à partir du [diagramme de classes de conception](#). Elles sont basées sur le langage java, mais applicables à d'autres langages objets.

### Classe UML

Une [classe UML](#) se transforme en une [classe Java](#). Par défaut, chaque classe UML devient un fichier `.java`.

### Attribut

Les [attributs](#) deviennent des [variables](#) en Java. Leur type est soit un type primitif, soit une classe fournie par la plate-forme (`String`, `Date`, ...). Ne pas oublier dans ce cas la directive d'importation du package correspondant.

Les attributs de classe (soulignés en UML) deviennent des membres statiques(`static`)

## PASSAGE DE L'ANAYSE AU CODE

(Transformation du diagramme de classes de conception en code)

### Opérations

Les **opérations** deviennent des **méthodes**, leur visibilité est définie avec les mêmes conventions que les attributs.

Les opérations de classes (soulignées en UML) deviennent des méthodes statiques, les opérations abstraites (en italique) se traduisent par le mot-clé *abstract*.

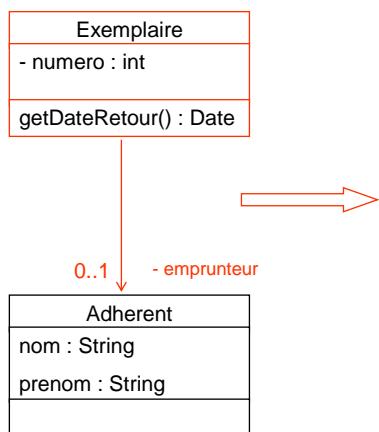
### Généralisation

Le concept de généralisation se traduit directement par le mécanisme de l'héritage (mot-clé *extends* en java).

## PASSAGE DE L'ANAYSE AU CODE

(Transformation du diagramme de classes de conception en code)

### Association



```
import java.util.Date;

public class Exemplaire
{
    private int numero;
    private Adherent emprunteur;

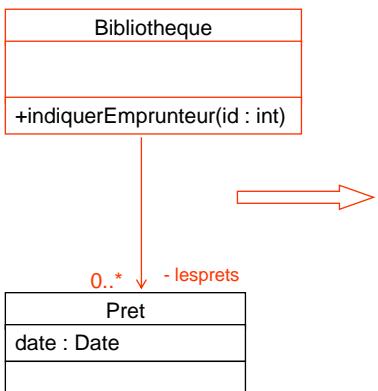
    public Exemplaire()
    {
    }

    public Date getDateRetour()
    {
    }
}
```

## PASSAGE DE L'ANAYSE AU CODE

(Transformation du diagramme de classes de conception en code)

### Association



```

import java.util.Date;

public class Bibliotheque
{
    private Prêt[] lesprets ;

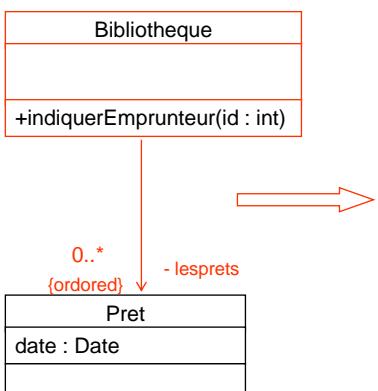
    public Bibliotheque()
    {
    }

    public void indiquerEmprunteur(id : int)
    {
    }
}
  
```

## PASSAGE DE L'ANAYSE AU CODE

(Transformation du diagramme de classes de conception en code)

### Association



```

import java.util.Date;

public class Bibliotheque
{
    private List lesprets = new ArrayList() ;

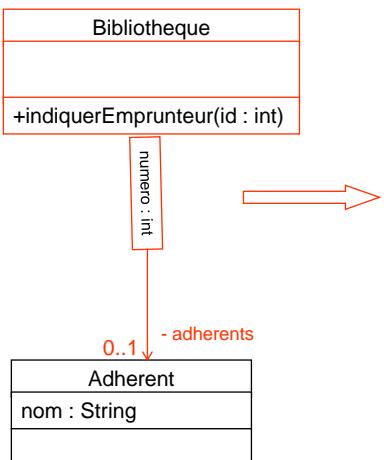
    public Bibliotheque()
    {
    }

    public void indiquerEmprunteur(id : int)
    {
    }
}
  
```

## PASSAGE DE L'ANAYSE AU CODE

(Transformation du diagramme de classes de conception en code)

### Association



```

import java.util.Date;

public class Bibliotheque
{
    private Map adherents = new Hashmap();

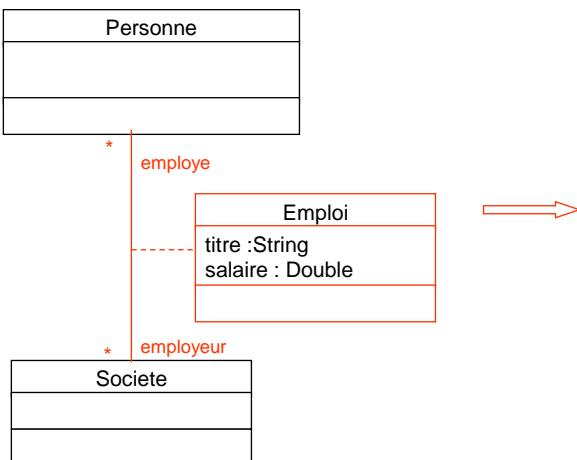
    public Bibliotheque()
    {
    }

    public void indiquerEmprunteur(id : int)
    {
    }
}
  
```

## PASSAGE DE L'ANAYSE AU CODE

(Transformation du diagramme de classes de conception en code)

### Classe d'association



```

public class Emploi
{
    private Personne employe;
    private Societe employeur;
    private String titre;
    private double salaire;

    ...
}
  
```

## PASSAGE DE L'ANAYSE AU CODE

(Transformation du diagramme de classes de conception en code)

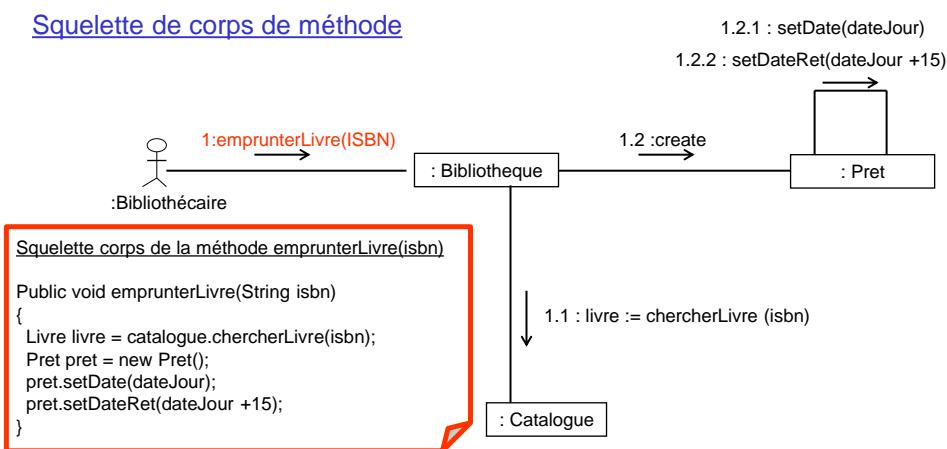
### Association bidirectionnelle

Une association bidirectionnelle se traduit simplement par une paire de références, une dans chaque classe impliquée dans l'association. Les noms des rôles aux extrémités d'une association servent à nommer les variables de type référence.

## PASSAGE DE L'ANAYSE AU CODE

(Transformation du diagramme de classes de conception en code)

### Squelette de corps de méthode



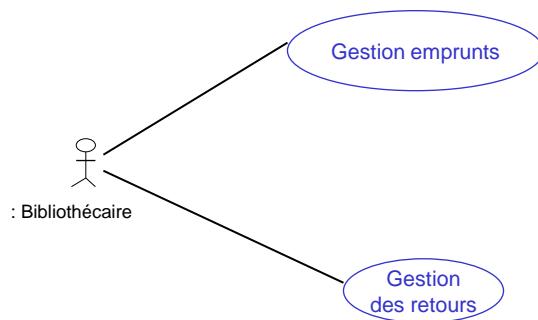
**N.B.** On peut également utiliser un diagramme de séquence pour représenter le corps d'une méthode (voir cadres d'interaction ).

## Passage de l'analyse au code

- Exemple

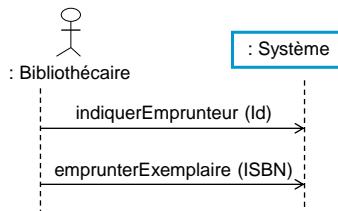
## PASSAGE DE L'ANAYSE AU CODE (Exemple)

Diagramme de cas d'utilisation



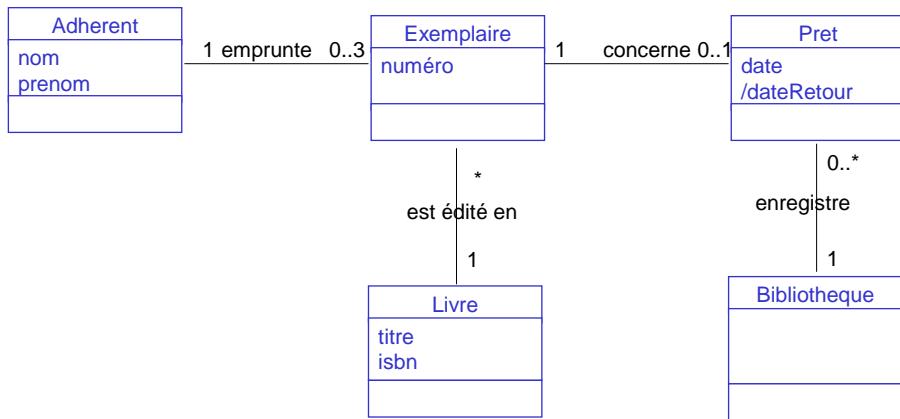
## PASSAGE DE L'ANAYSE AU CODE (Exemple)

DSS du scénario nominal du cas *Gestion emprunts*



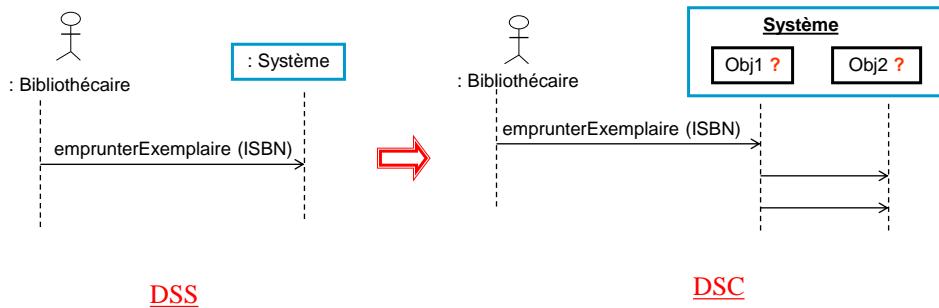
## PASSAGE DE L'ANAYSE AU CODE (Exemple)

Diagramme de classes d'analyse



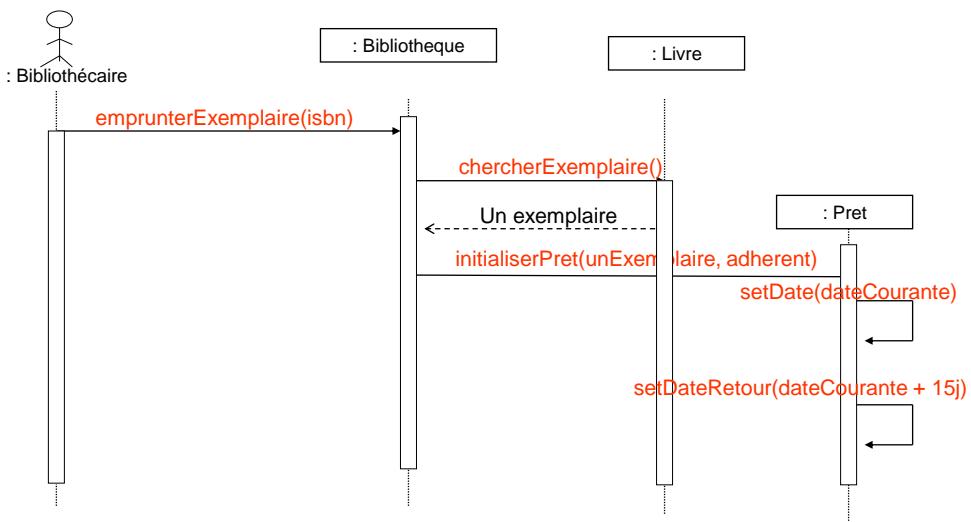
## PASSAGE DE L'ANAYSE AU CODE (Exemple)

Passage de l'analyse à la conception



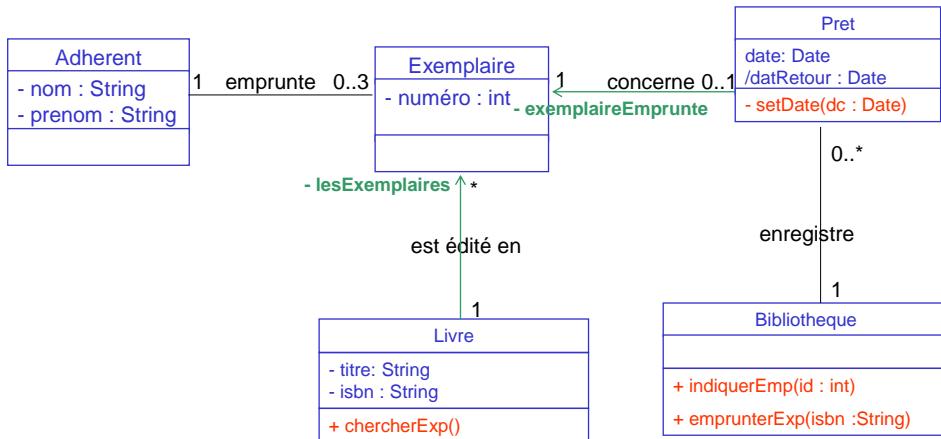
## PASSAGE DE L'ANAYSE AU CODE (Exemple)

Passage de l'analyse à la conception



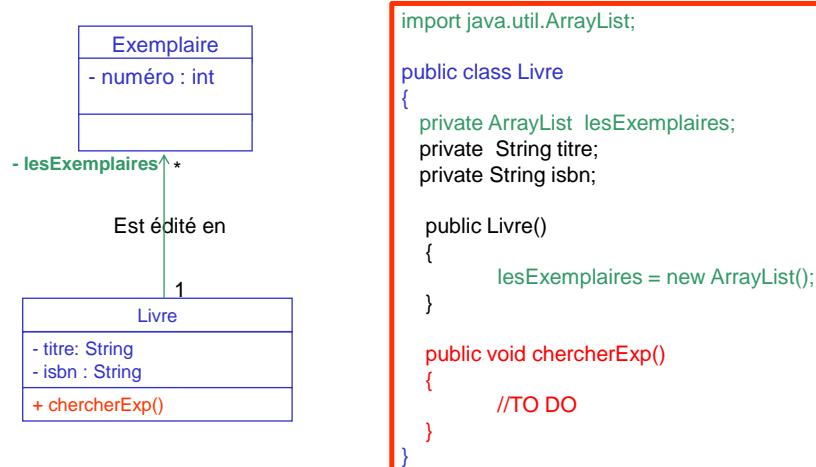
## PASSAGE DE L'ANAYSE AU CODE (Exemple)

### Passage de l'analyse à la conception



## PASSAGE DE L'ANAYSE AU CODE (Exemple)

### Transformation du diagramme de classes de conception en code



## Implémentation

- Diagramme de composants
- Diagramme de déploiement

### IMPLEMENTATION (Diagramme de composants)

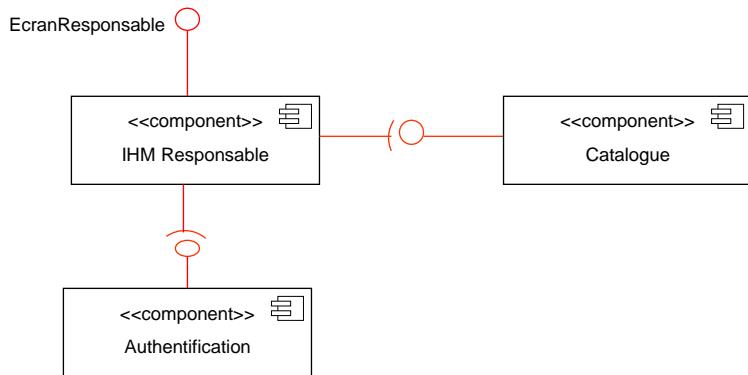
Un composant est une unité autonome offrant des services au travers d'une ou de plusieurs interfaces. Il est considéré comme une boîte noir car son contenu et son comportement interne sont totalement masqués de l'extérieur.

Un composant peut être déployé dans diverses configurations en fonction des connexions avec les autres éléments du système.



## IMPLEMENTATION (Diagramme de composants)

Le diagramme de composants décrit l'architecture logicielle d'un système comme un ensemble de composants liés par des interfaces offertes et requises.



## IMPLEMENTATION (Diagramme de déploiement)

Le diagramme de déploiement décrit l'architecture physique du système. Il est constitué de *nœuds* connectés par des *liens physiques*.

Un *nœud* représente une unité matérielle (par exemple un ordinateur) capable de recevoir et d'exécuter du logiciel. Les nœuds contiennent des parties du système (sous leur forme physique) appelées *artefacts*. Un fichier exécutable, un fichier source, une table de BD, ... sont des exemples d'artefact.

Un *lien* représente un chemin de communication comme par exemple des bus, des câbles réseaux, ...



## Dossier d'analyse

- Rappel des phases de développement
- Plan de rédaction

## DOSSIER D'ANALYSE (Rappel des phases de développement)

Trois grandes phases :

- 1 . Cahier des charges
- 2 . Organisation interne
- 3 . Développement

## **DOSSIER D'ANALYSE**

**(Phase 1 – Cahier des Charges)**

- 1 . Rencontre du client
- 2 . Analyse de l'existant
- 3 . Etude des spécifications

## **DOSSIER D'ANALYSE**

**(Phase 1 – Cahier des Charges)**

Il sera constitué des éléments ci-dessous :

- Une présentation générale du sujet : situation actuelle, contexte, objectifs généraux
- Une description précise du système à réaliser avec la liste exhaustive des résultats attendus
- Dans le cas où le logiciel réalisé ne représenterait pas la totalité du produit, indiquer les spécifications des tranches suivantes
- Les moyens matériels et logiciels nécessaires
- Organisation, planning général dont date de la recette et conditions financières

## DOSSIER D'ANALYSE (Phase 2 – Organisation Interne)

Il s'agit de définir :

- L'organisation de l'équipement,
- Le planning,
- La technique de développement (UML, Merise, ...)
- Les procédures de tests externes (fichiers d'essai),
- Les moyens informatiques.

Toutes ces informations sont à regrouper dans un document appelé "dossier Industriel", contenant le cahier des charges en annexe.

## DOSSIER D'ANALYSE (Phase 3 – Développement)

Analyse fonctionnelle (document de spécifications fonctionnelles)

Conception (document de spécifications détaillées ou logiques)

- Conception objet préliminaire
- Conception objet détaillée

Codage (classes dans un langage comme java, C#, ...)

Intégration et tests

Tests externes

Documentation

## DOSSIER D'ANALYSE (Phase 3 – Analyse fonctionnelle)

### Analyse des besoins

- Les attentes réelles des utilisateurs
- Élaboration et description de scénarios

### Description des fonctionnalités du système

- Description détaillée des services proposés
- Description des interfaces

### Modélisation conceptuelle du domaine

- Description des concepts du domaine

### Description de l'architecture logicielle

## DOSSIER D'ANALYSE (Phase 3 – Conception préliminaire)

Cette étape a pour but d'affiner et de compléter les diagrammes de classes d'analyse obtenus lors de l'analyse fonctionnelle, indépendamment des choix technologiques.

Pour cela, il faut :

- ajouter ou préciser les opérations dans les classes,
- ajouter des types aux attributs et aux paramètres et retours des opérations,
- affiner les relations entre les classes.

L'idée générale consiste à décider pour chaque service, quelle est la classe qui va le contenir (Décision d'allocation de responsabilités)

## DOSSIER D'ANALYSE

(Phase 3 – Conception détaillée)

Cette étape consiste à affiner et enrichir les diagrammes de l'étape de conception préliminaire, en fonction des choix technologiques effectués.

Par exemple, une classe peut représenter une servlet java, ou une classe CodeBehind d'ASP.net

## DOSSIER D'ANALYSE

(Plan)

### Initialisation ou étude d'opportunité

1. Client du projet
2. Objectifs du projet
3. Domaine du projet
4. Contraintes du projet

### Analyse des besoins

1. Liste et description des acteurs
2. Diagramme de cas d'utilisation
3. Description des cas d'utilisation (DSS et DAS)

## DOSSIER D'ANALYSE (Plan)

### Analyse-conception

1. Diagramme de classes métier avec glossaire des mots métier
2. Diagrammes de séquence et diagrammes de communication
3. Diagrammes d'états (pour les classes complexes)
4. Contraintes du projet
5. Diagramme de classes de conception

### Implémentation

1. Diagramme de composants

### Déploiement

1. Diagramme de déploiement