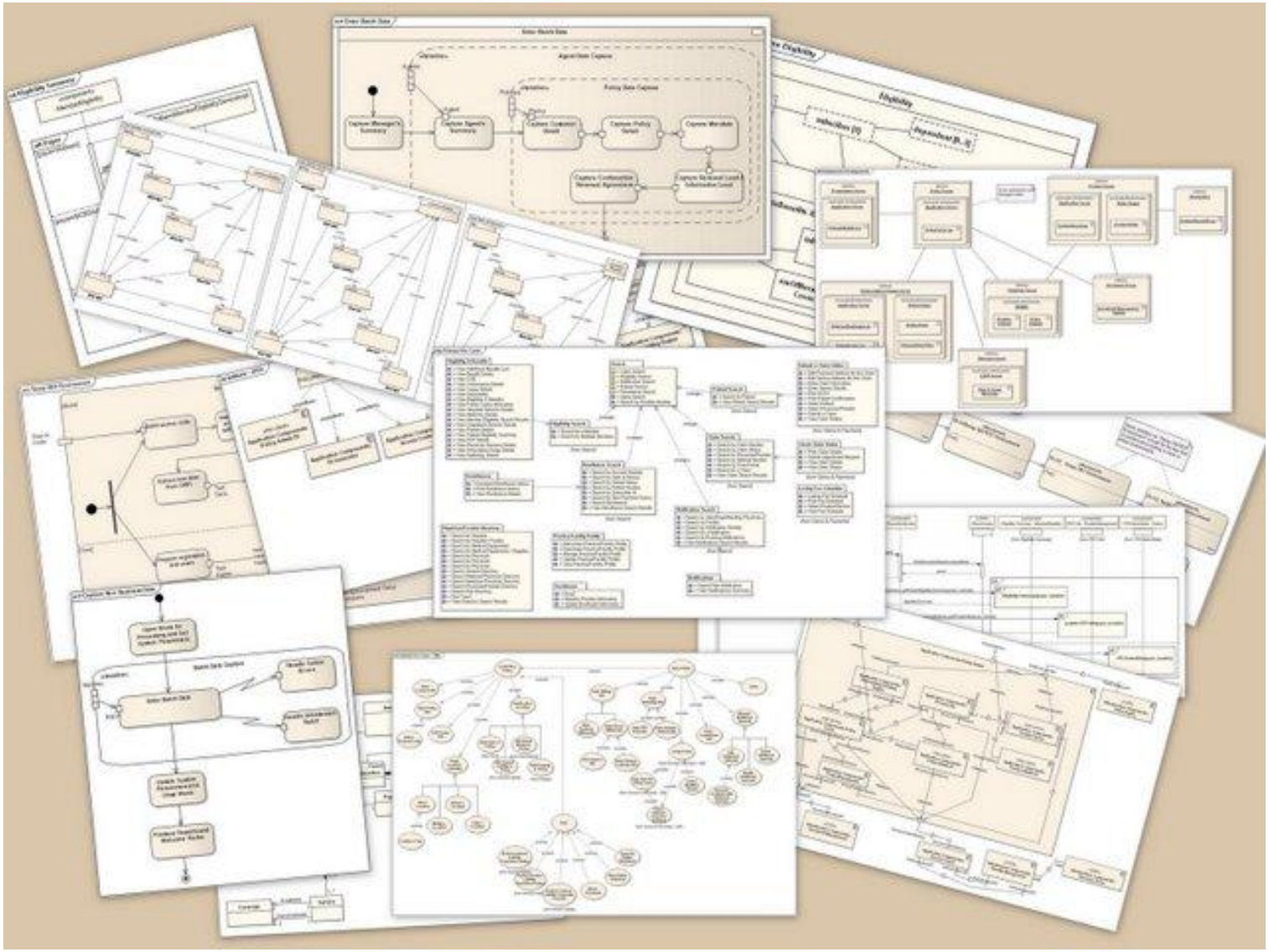
Olivier Augereau

Formation UML



basée sur le cours de Bertrand Legal, maître de conférences à l'ENSEIRB

www.enseirb.fr/~legal

Sommaire

Introduction

- Langage standard conçu pour permettre aux concepteurs d'élaborer les plans des logiciels qu'ils doivent développer.

- On se sert du langage UML pour:
  - *Visualiser* => Schématiser le travail à réaliser
  - *Spécifier* => Exprimer formellement les contraintes
  - *Construire* => Commencer à développer une solution
  - *Documenter* => Expliquer les choix réalisés (les problèmes)
  - *Communiquer* => Travailler en équipe

Introduction

•Langage => *vocabulaire* + *règles* => *communiquer.*

•Le vocabulaire et les règles d'écriture régissent la manière de construire et de lire les modèles correctement mis en forme.

Aucune décision d'implémentation n'est prise lors de la phase de spécification et d'analyse du cahier des charges.

•UML permet de résoudre les problématiques du partage et de mémorisation de l'information de manière **visuelle**.

Introduction

•Chaque symbole graphique possède sa sémantique propre (signification universelle).

•Un modèle peut être écrit par un concepteur et compris pas un autre sans ambiguïté à la lecture.

➥ Automatisation par outil => Génération de code source

➡ UML est un *métalangage* de modélisation permettant *d'unifier les* modèles utilisés dans les méthodes de développement.

I) Les bases

Dans le langage UML, il existe 3 types de briques de base pour modéliser tous systèmes :

1. Les *éléments*

✓ Ce sont les abstractions essentielles au modèle.

2. Les *relations*

✓ Les relations expriment les liens existants entre les différents éléments.

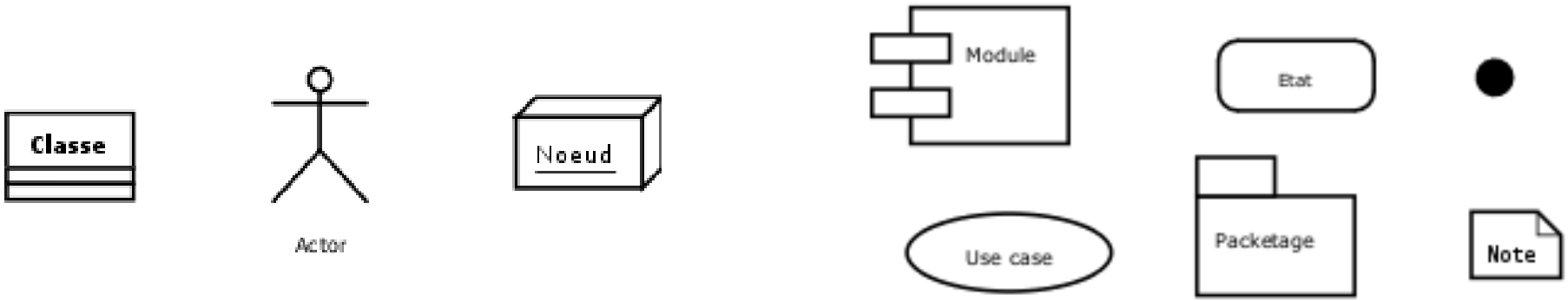
3. Les *diagrammes*

✓ Les diagrammes comprennent des ensembles d'éléments et de relations dignes d'intérêt.

I)1) Les éléments

➡ Définir des modèles

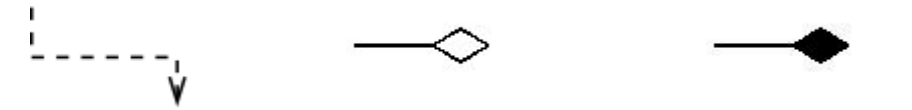
1. Les éléments *structurels* (classes, nœuds …)
2. Les éléments *comportementaux* (interactions)
3. Les éléments de *regroupement* (packages)
4. Les éléments *d'annotation* (commentaires)



I)2) Les relations

Afin d'exprimer les liens interconnectant les différents éléments des modèles, 4 relations de base ont été définies :

1. La *dépendance*
2. La *généralisation*
3. L'*association*
4. La *réalisation*



I)3) Les diagrammes

Un *diagramme est une représentation visuelle de l'ensemble des* éléments qui constituent le système.

Ils servent à visualiser un système sous différents angles (utilisateur, administrateur par ex.)

Dans les systèmes complexes, un diagramme ne fourni qu'une *vue partielle du système.*

- L'ensemble des diagrammes aboutés permet d'obtenir une vue globale du système à concevoir.
- Chaque diagramme va permettre de modéliser ou spécifier une vue (spécificité) du système à concevoir.

I)3) Les diagrammes

•Il existe 13 types de diagrammes différents.

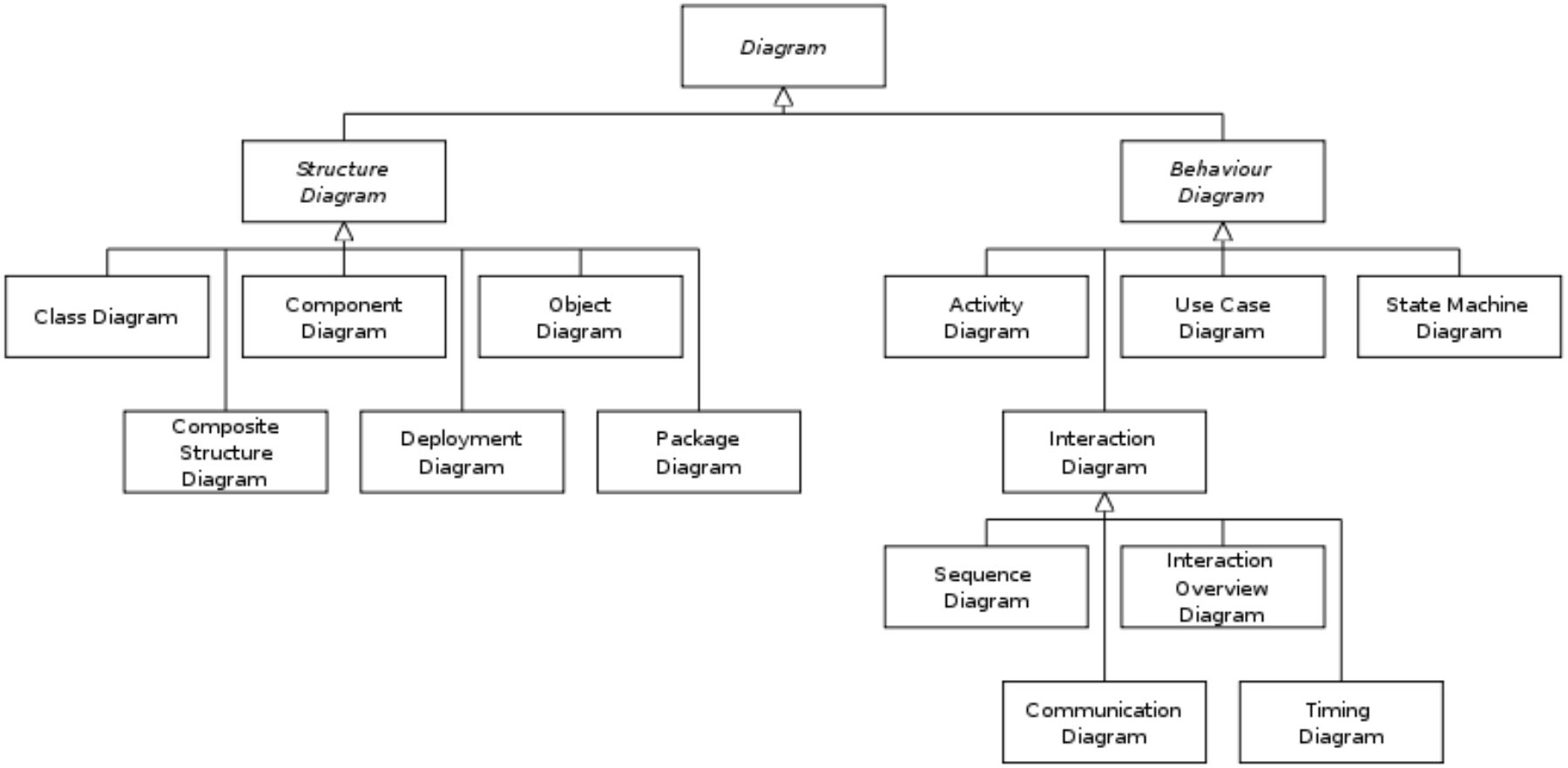
•Ces **13** types de diagrammes UML sont dépendants hiérarchiquement et se complètent, de façon à permettre la modélisation d'un projet tout au long de son cycle de vie.

•Chacun est dédié à la représentation d'un concept particulier.

Il existe deux grands groupes:
-Modélisation des *parties structurelles* (statiques) - 6 types
-Modélisation du *comportement* (dynamiques) - 7 types

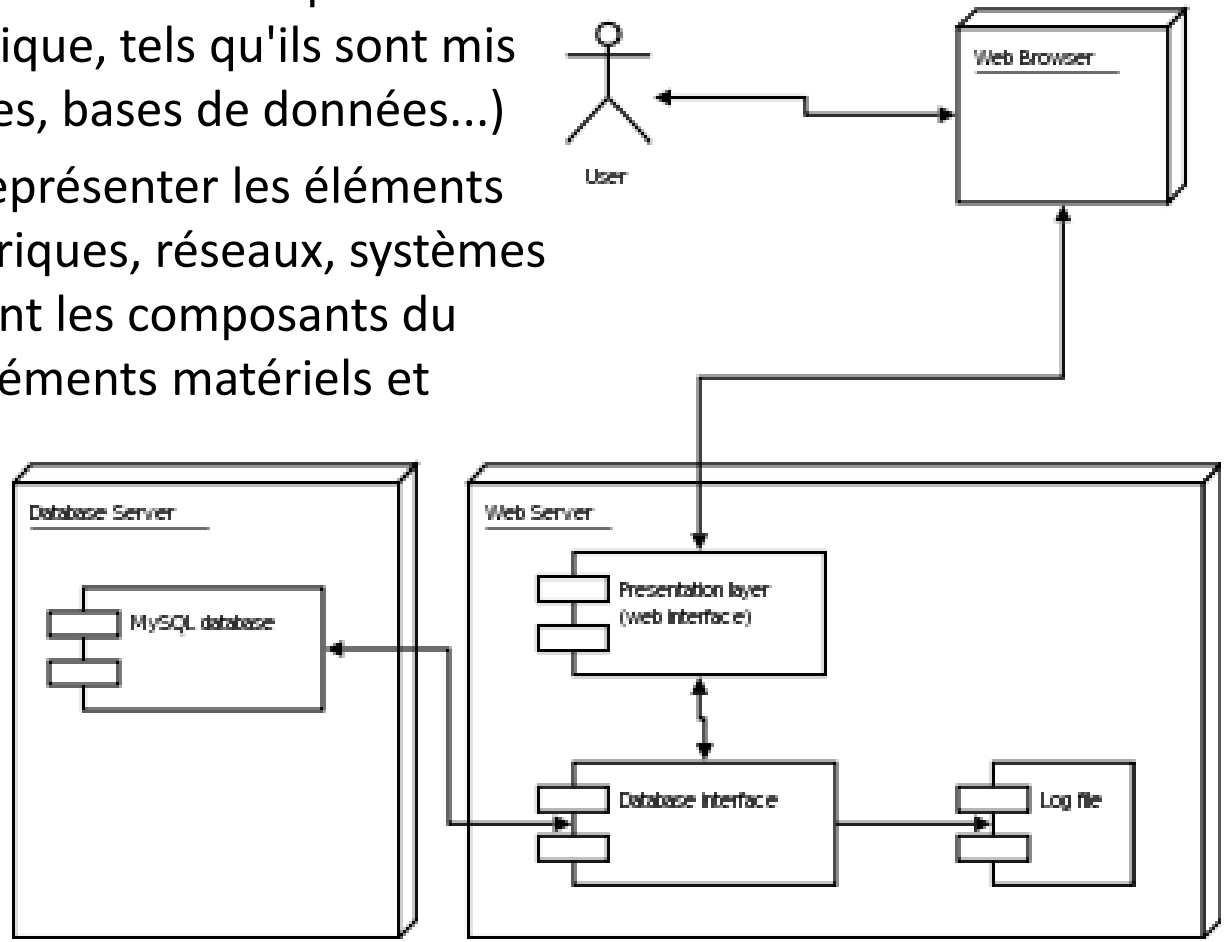
I)3) Les diagrammes

- **Diagrammes structurels ou statiques :**
  - Diagramme de classes : représenter les classes intervenant dans le système.
  - Diagramme d'objets : représenter les instances de classes utilisées dans le système.



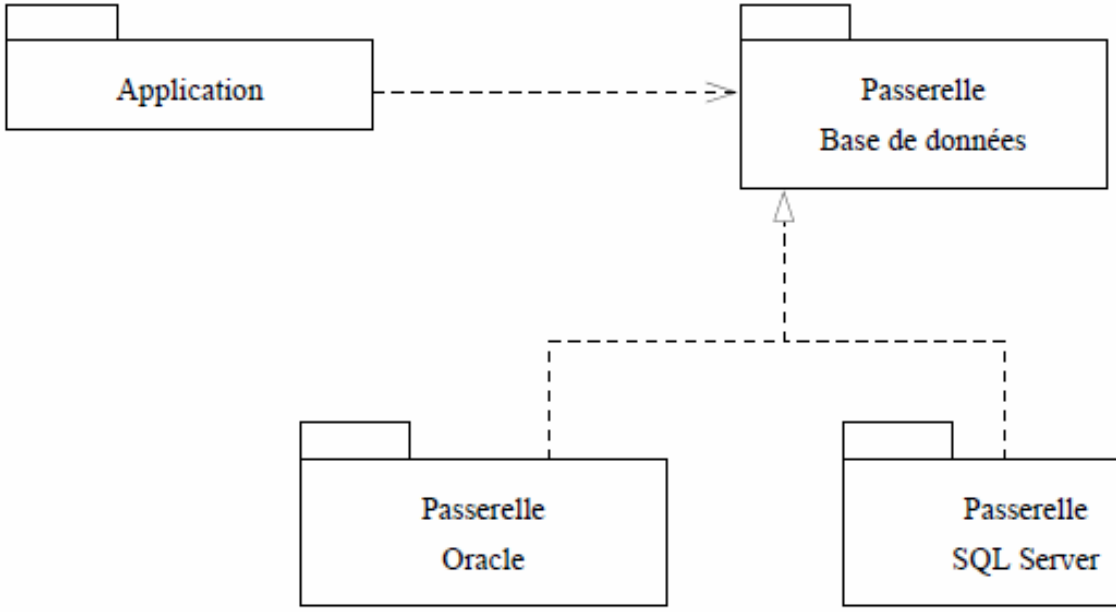
I)3) Les diagrammes

- **Diagrammes structurels ou statiques :**
  - Diagramme de composants : montrer les composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données...)
  - Diagramme de déploiement : représenter les éléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage...) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent entre eux.



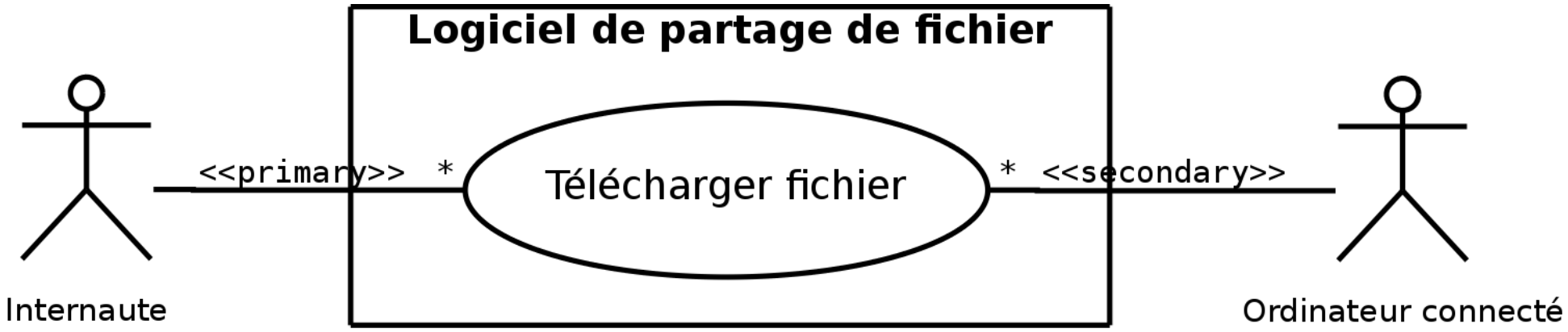
I)3) Les diagrammes

- **Diagrammes structurels ou statiques :**
  - Diagramme des paquetages : un paquetage est un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML. le Diagramme de paquetage sert à représenter les dépendances entre paquetages, c'est-à-dire les dépendances entre ensembles de définitions.
  - Diagramme de structure composite : décrire sous forme de boîte les relations entre composants d'une classe.



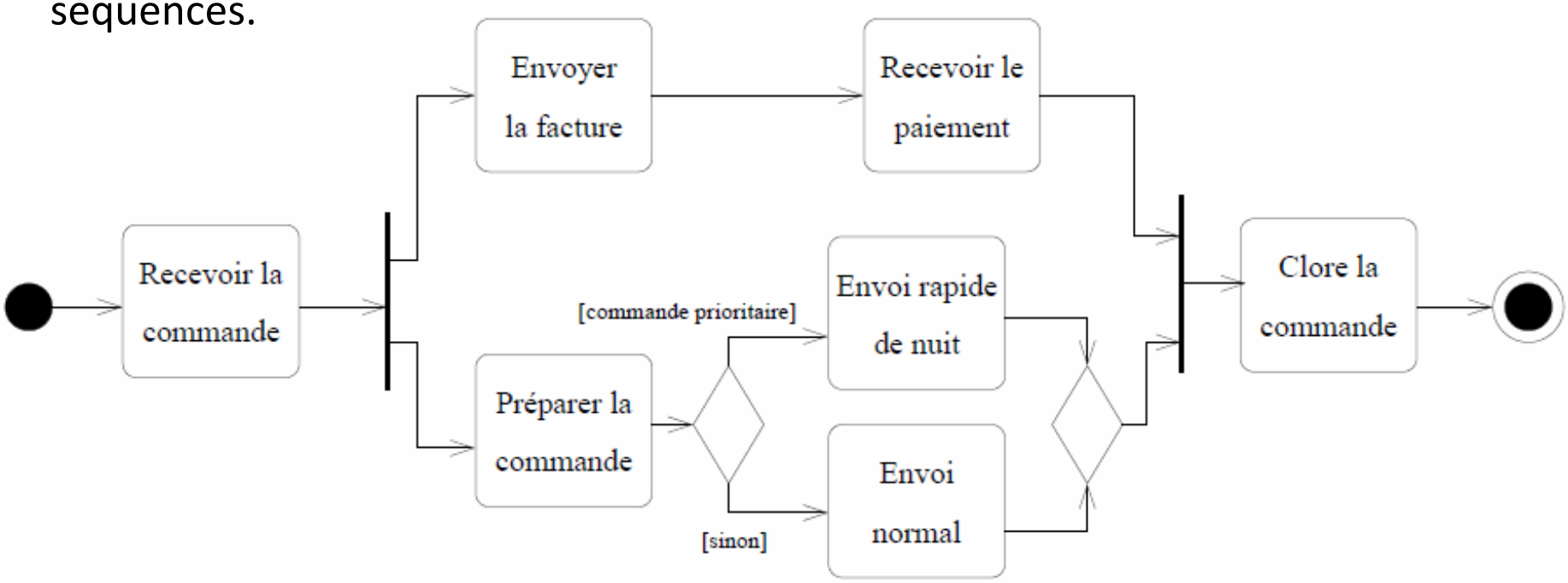
I)3) Les diagrammes

- **Diagrammes comportementaux ou dynamique :**
  - Diagramme des cas d'utilisation : identifier les possibilités d'interaction entre le système et les acteurs (=> toutes les fonctionnalités que doit fournir le système.)



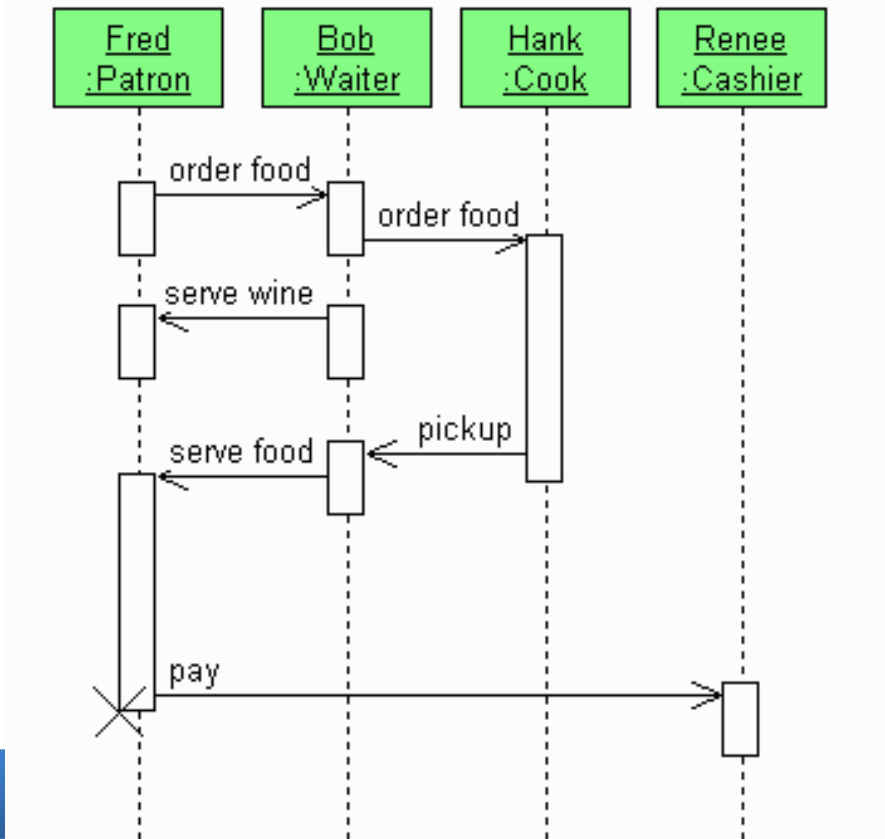
I)3) Les diagrammes

- **Diagrammes comportementaux ou dynamique :**
  - Diagramme états-transitions : décrire sous forme de machine à états le comportement du système ou de ses composants.
  - Diagramme d'activité : décrire sous forme de flux ou d'enchaînement d'activités le comportement du système ou de ses composants. (logigramme)
  - Diagramme global d'interaction : permet de décrire les enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences.

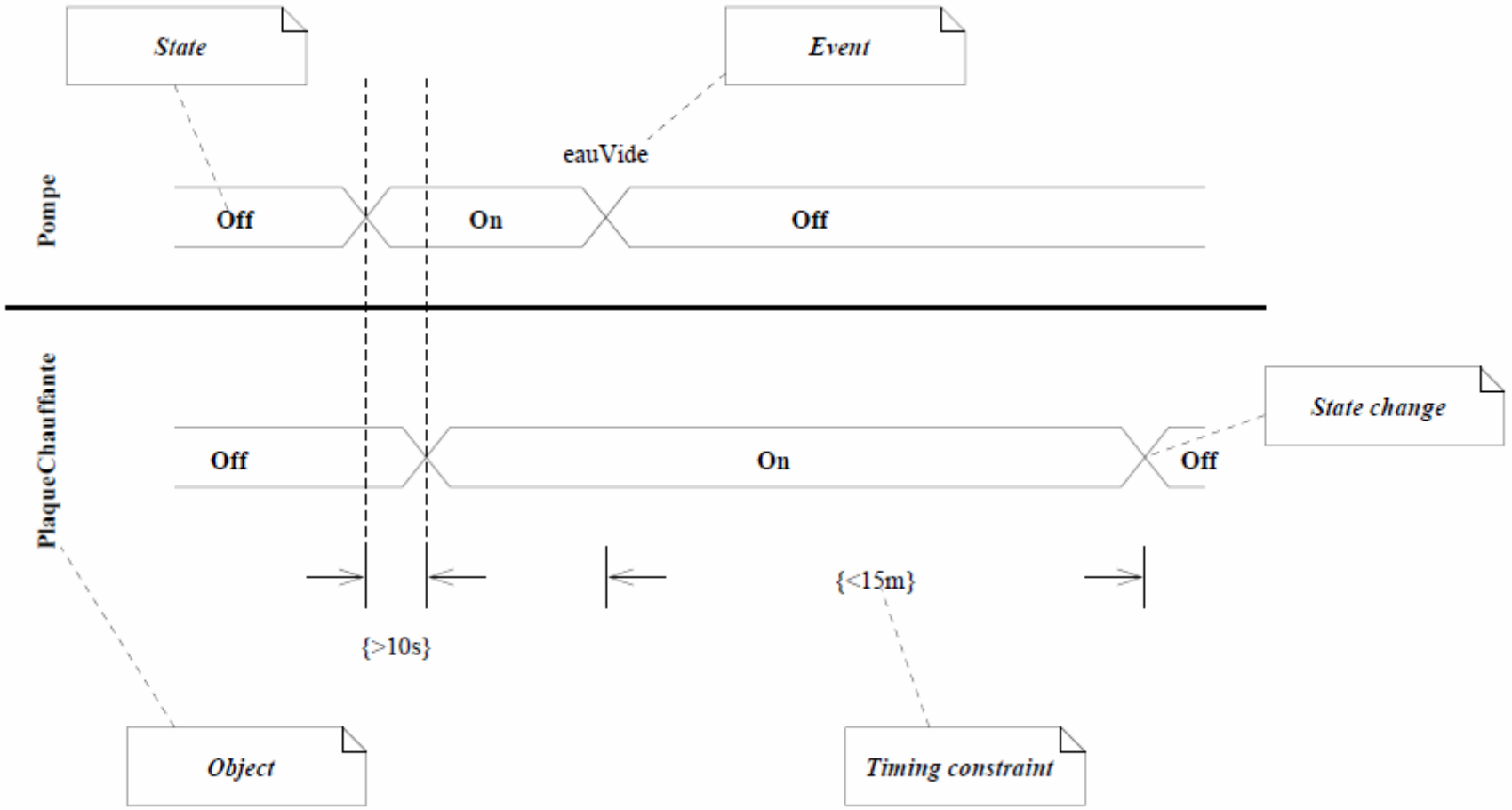


I)3) Les diagrammes

- **Diagrammes comportementaux ou dynamique :**
  - Diagramme de séquence : représentation du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs.
  - Diagramme de communication/collaboration : représentation d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets.



- **Diagrammes comportementaux ou dynamique :**
  - Diagramme de temps : décrire les variations d'une donnée au cours du temps.



II) Les diagrammes de cas d'utilisation

- Les diagrammes de cas d'utilisation :
  - Organisent les comportements du système.
  - Représentent les acteurs et leurs relations avec l'application.

- On ne se préoccupe pas de l'implémentation du système, juste des services qu'il doit rendre aux utilisateurs.

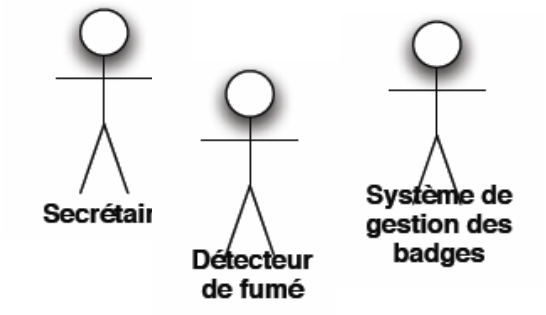
- Les données sont directement extraites du cahier des charges.

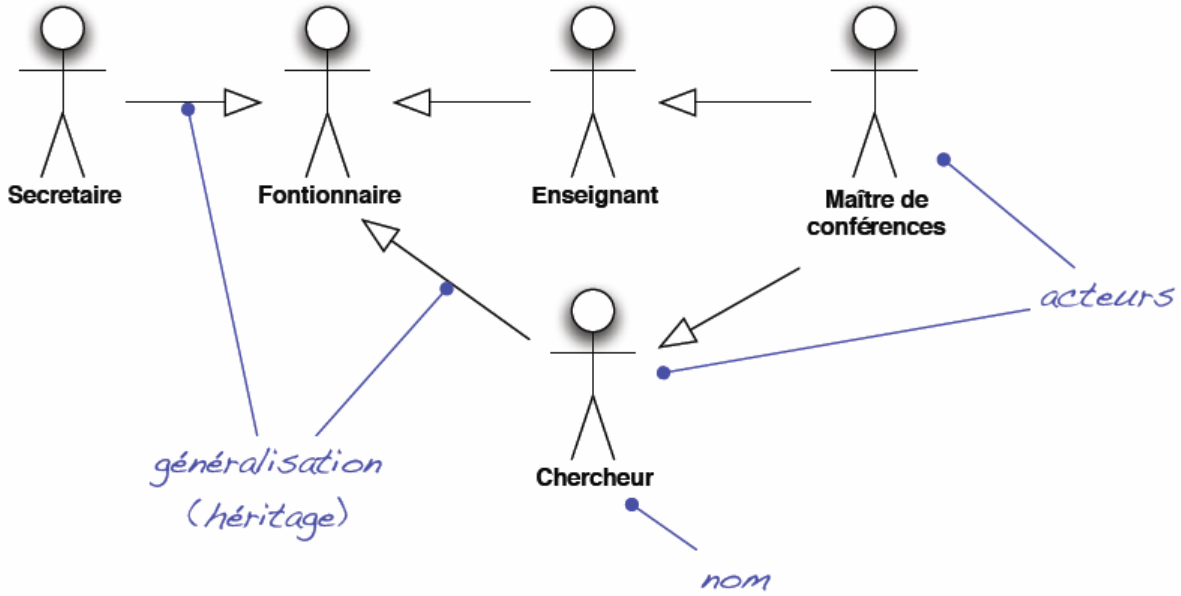
- L'expression des cas d'utilisation permet de déterminer les points suivants :
  - Quels sont les services que doit implémenter le système ?
  - A qui ces services doivent être rendus ?

II) Les diagrammes de cas d'utilisation

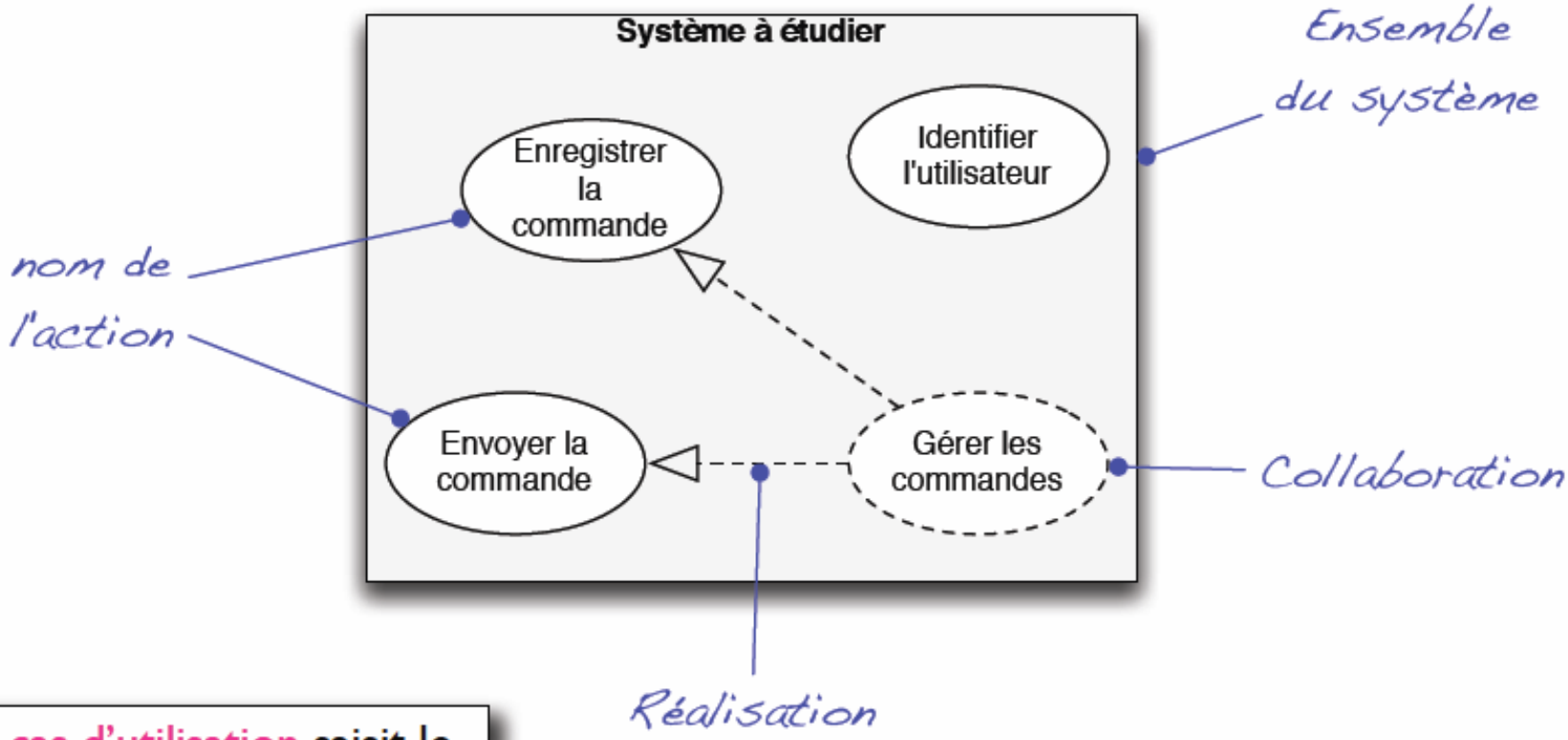
- Les cas d'utilisation se composent :
  - *D'acteurs externes au système (personnes, capteurs, autres* applications ...).
  - *Du système en lui même et des services qu'il doit rendre.*
  - *Des relations qui relient les acteurs aux services qui leurs sont rendus.*

- Un acteur est un utilisateur ou un autre système qui interagit avec le système.





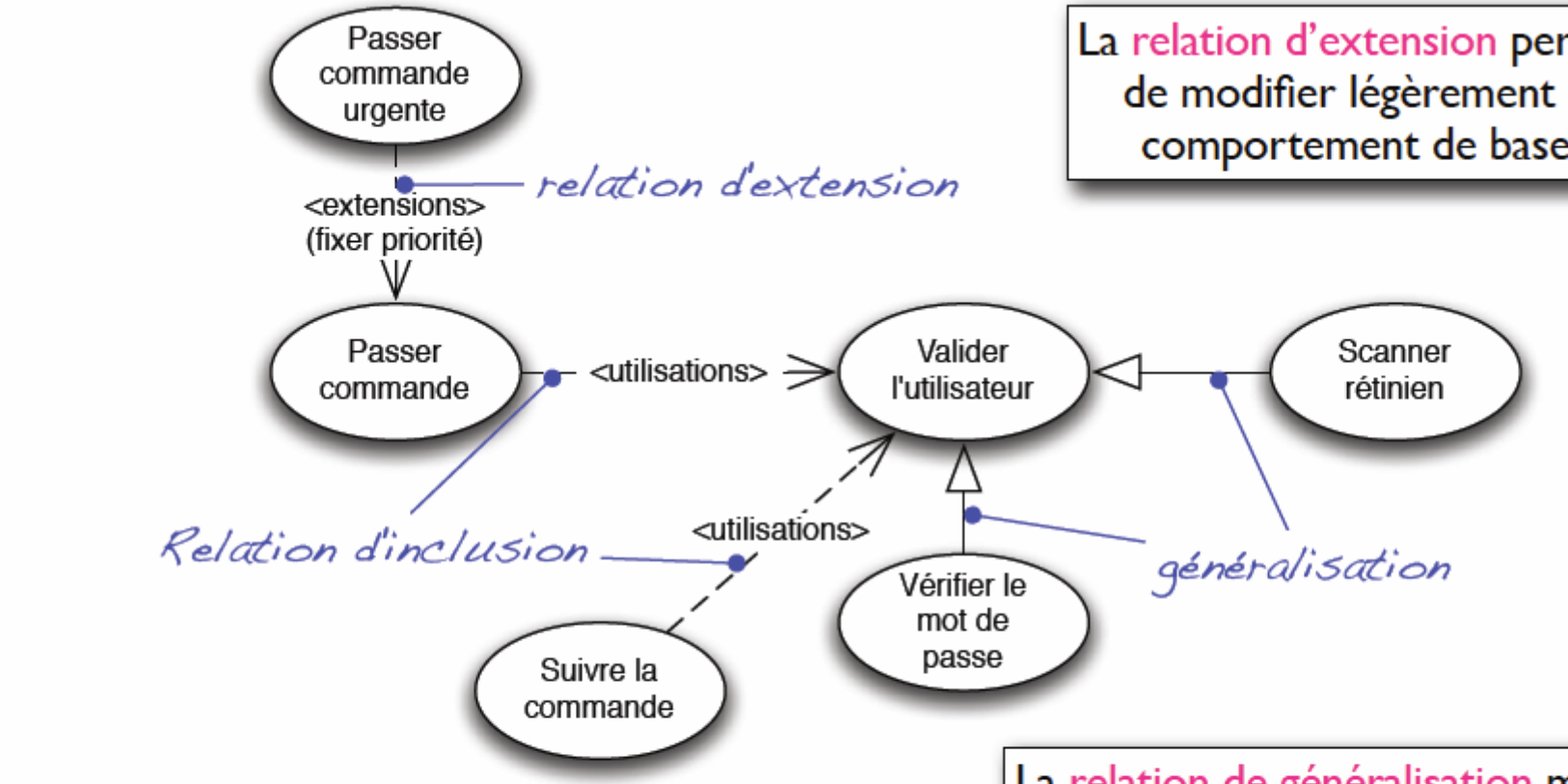
II) Les diagrammes de cas d'utilisation



Un cas d'utilisation saisit le comportement attendu du système sans que l'on précise comment cela est réalisé.

La collaboration permet de raffiner la vue implémentation du système en factorisant les points communs.

II) Les diagrammes de cas d'utilisation



La relation d'extension permet de modifier légèrement le comportement de base.

La relation d'inclusion est faite pour éviter la répétition des actions en factorisant les services rendus. Cela revient à déléguer une partie des actions.

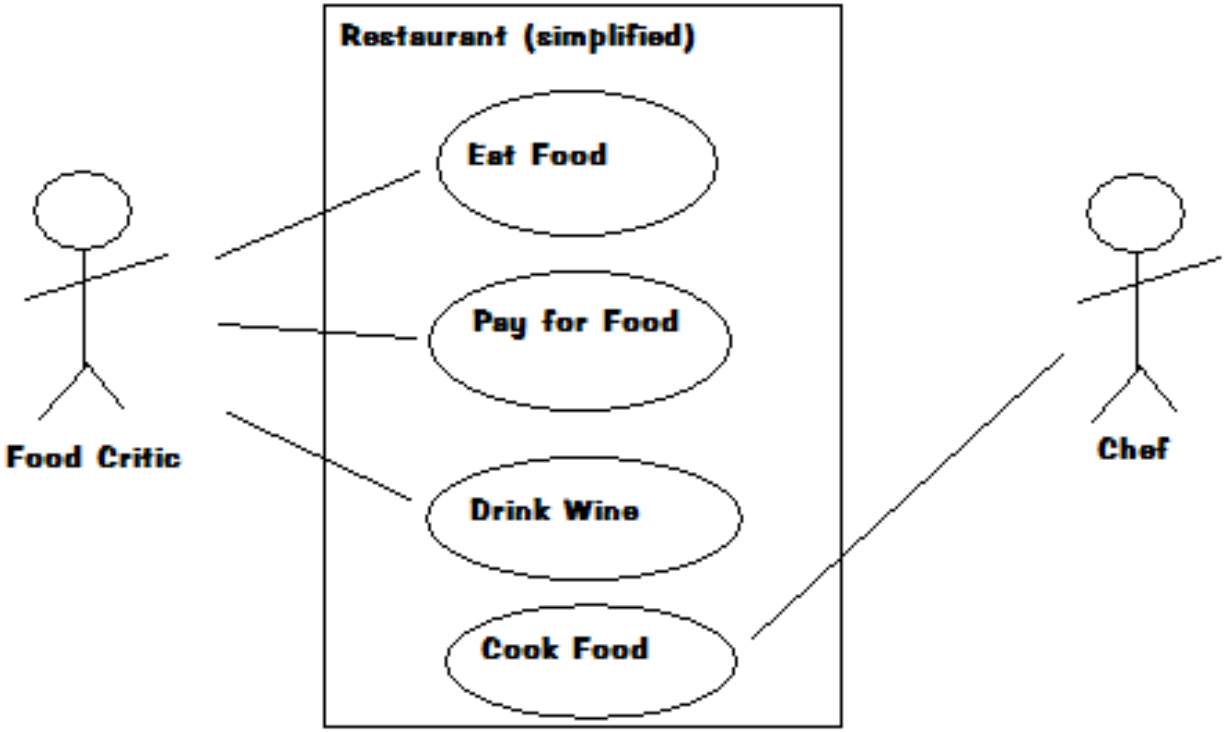
La relation de généralisation modélise un héritage du comportement de base afin de l'affiner.

II) Les diagrammes de cas d'utilisation

- Identifier l'ensemble des acteurs possibles qui vont interagir avec l'application :
  - Identifier tous les acteurs.
  - Les factoriser sous forme de groupes.
- Prendre en considération les besoins de tous les utilisateurs
  - Identifier tous les services que doit rendre le système.
  - Regrouper les besoins communs à l'aide des relations d'inclusion, de généralisation et d'extension.

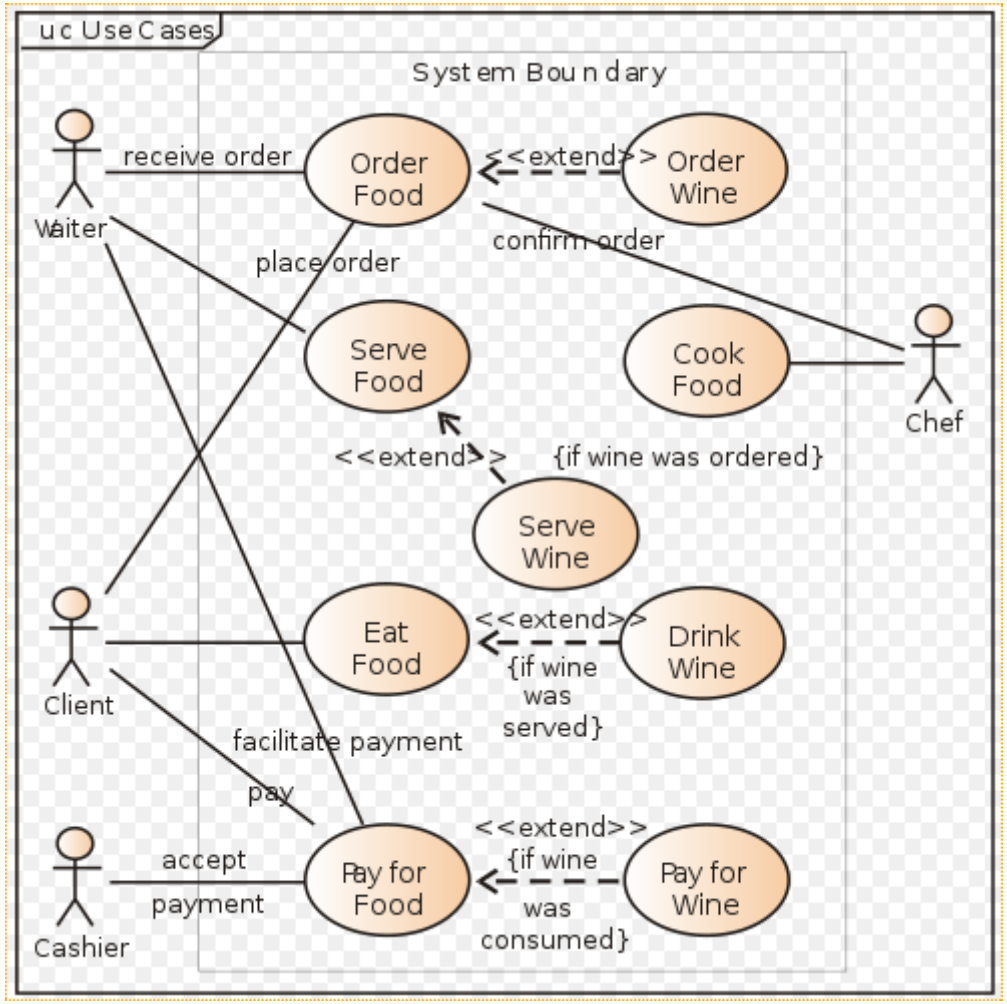
II) Les diagrammes de cas d'utilisation

- Exemple simple : restaurant



II) Les diagrammes de cas d'utilisation

- Exemple avancé : restaurant



II) Les diagrammes de cas d'utilisation

- Attention au niveau de précision: ne pas descendre trop bas.

- Pour chaque cas d'utilisation, il faut :
  - Exprimer ce que fourni l'utilisateur au système.
  - Ce que l'utilisateur reçoit en retour après exécution.
  - Il faut aussi penser à détailler les hypothèses réalisées lors de la description du cas d'utilisation (pré-requis et les post-conditions).

- L'expression des transactions entre le système et l'utilisateur sera détaillé par d'autres modèles plus adaptés :
  - Diagrammes de séquences.
  - Diagrammes d'activité, etc.

III) Les diagrammes de classes

- Le diagramme de classes peut être défini à différents niveaux d'abstraction :
  - Point de vue *conceptuel* (entités fournissant des services).
  - Point de vue *des spécifications* (raffinement fonctionnel des entités).
  - Point de vue *implémentation* (les classes que l'on doit mettre en œuvre).

- Un diagramme de classe se compose des éléments suivants :
  - Des classes.
  - Des interfaces.
  - Des relations (dépendances, associations, généralisation, …).

III) Les diagrammes de classes

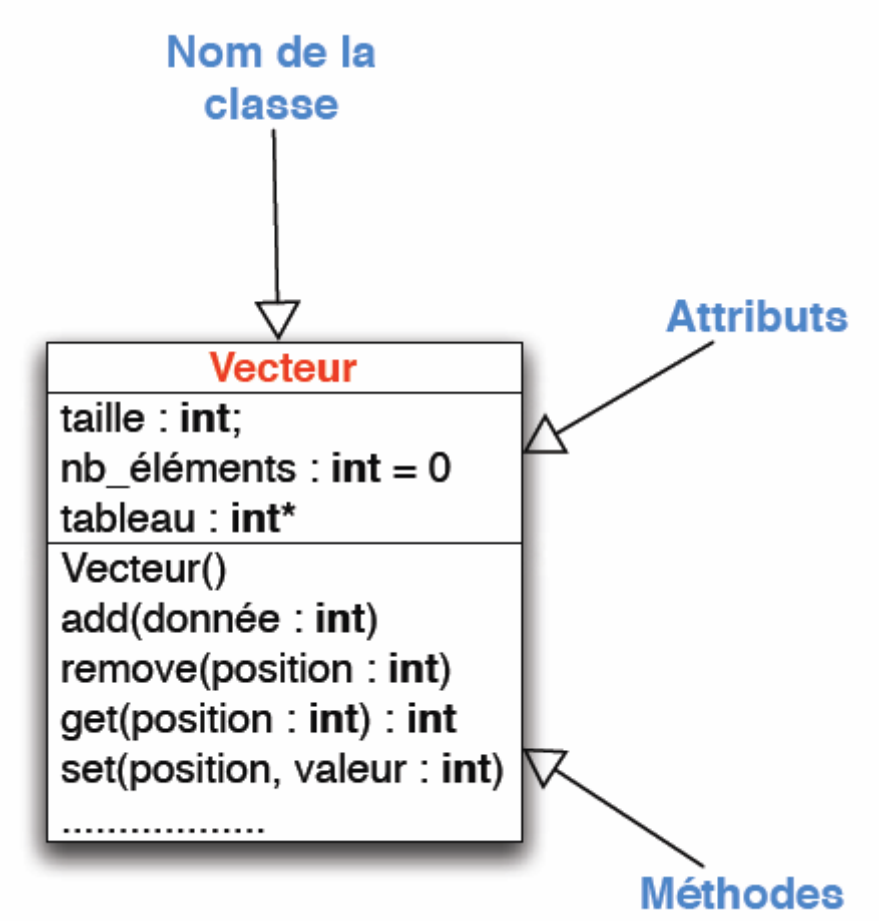
- **Les classes**

  – Une classe est la description élémentaire d'un ensemble d'objets qui partagent des attributs et des méthodes en communs.

  – La définition des attributs et des méthodes n'est *pas obligatoire* :
    - Les attributs et les méthodes sont trop nombreux pour être représentés dans la majorité des cas...
    - On indique alors les responsabilités de la classe sous forme de texte (ce qu'elle doit implémenter comme service).
    - On peut dès la définition d'une classe préciser les droits d'accès aux méthodes et aux attributs (+, - ou # correspondant respectivement à public, private ou protected)

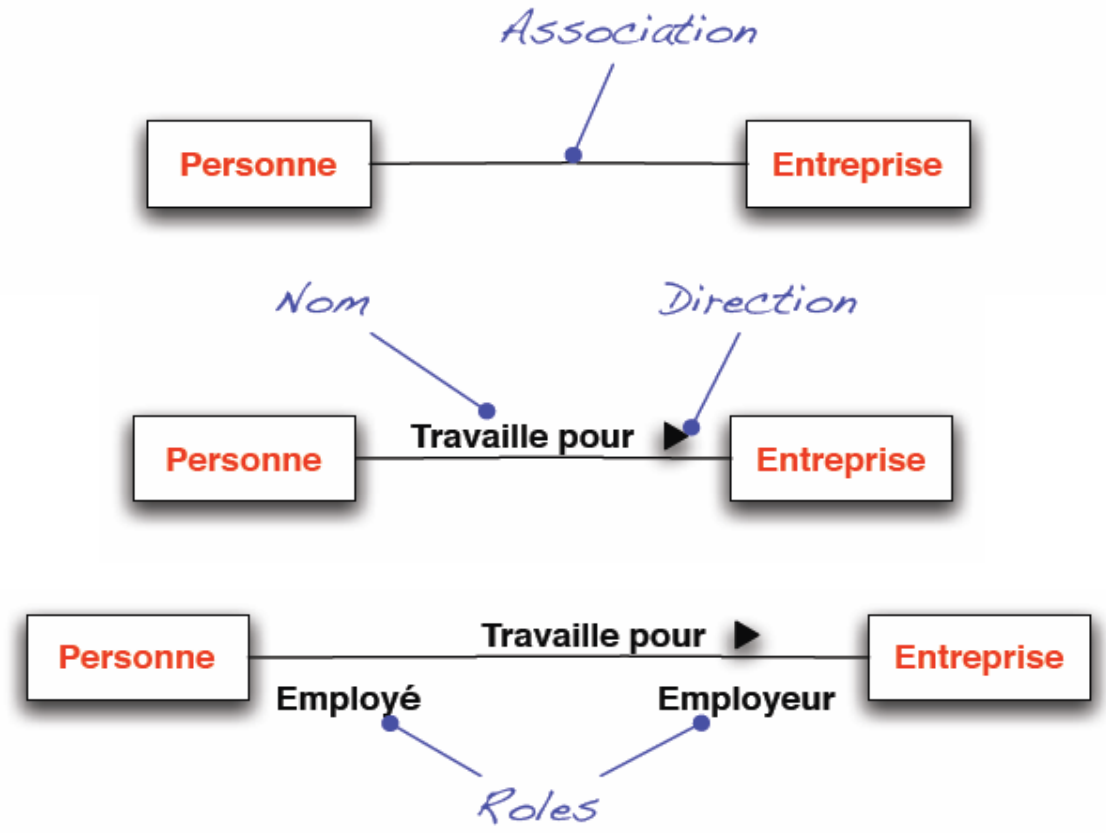
III) Les diagrammes de classes

- **Les classes**

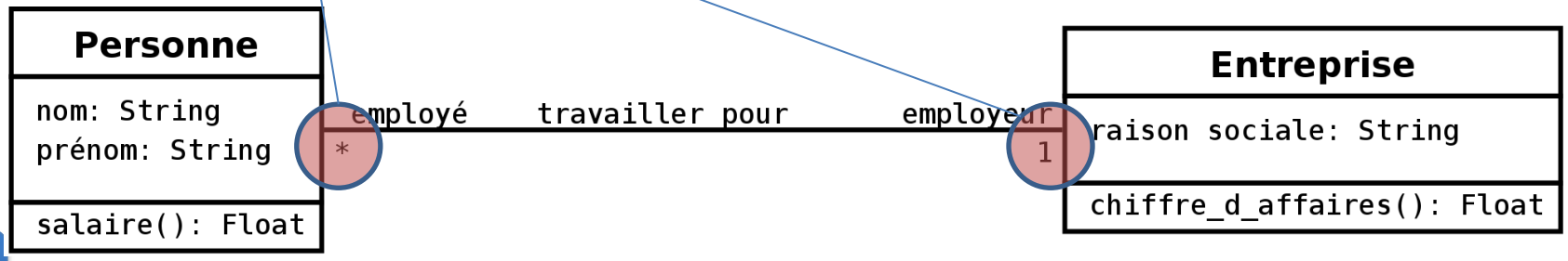


III) Les diagrammes de classes

- **Relations entre classes**
  - **Association**



Multiplicité :
-exactement un : 1 ou 1..1
-plusieurs : * ou 0..*
-au moins un : 1..*
-de un à six : 1..6



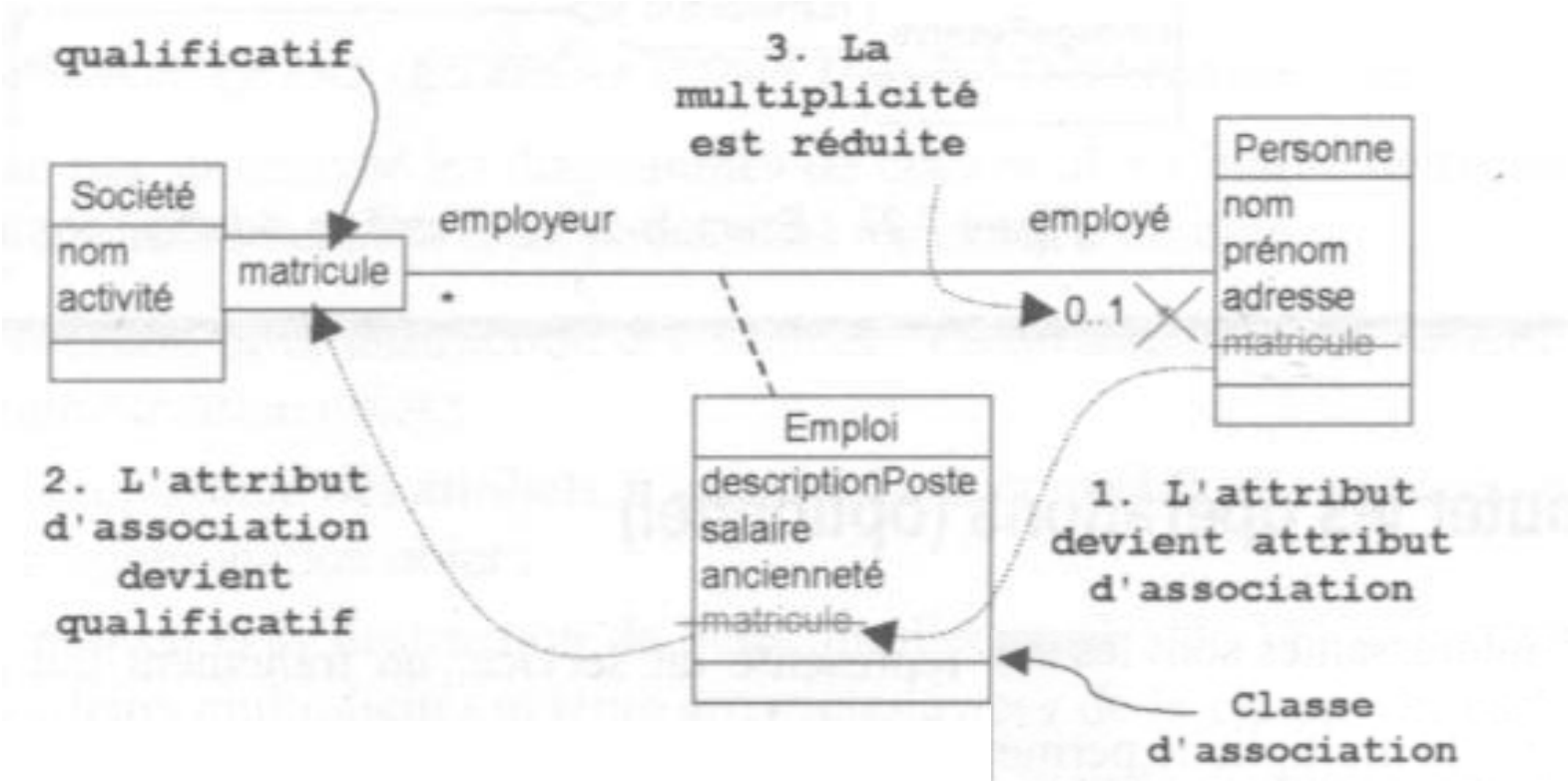
III) Les diagrammes de classes

- **Relations entre classes**
  - **Association**
    - Qualificatif

L'utilisation d'un qualificatif permet de sélectionner un sous-ensemble d'objets, parmi l'ensemble des objets d'une association

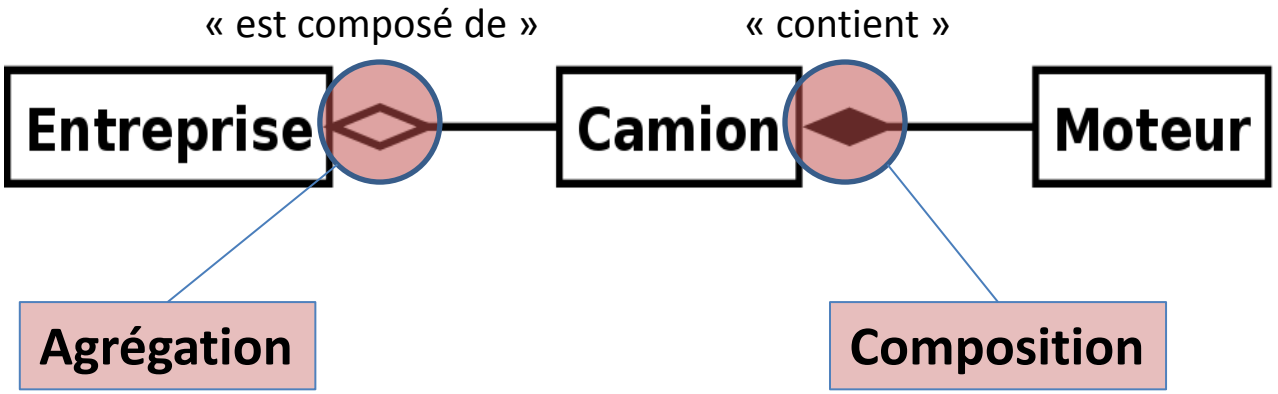
III) Les diagrammes de classes

- **Relations entre classes**
  - **Association**
    - Classe d'association



III) Les diagrammes de classes

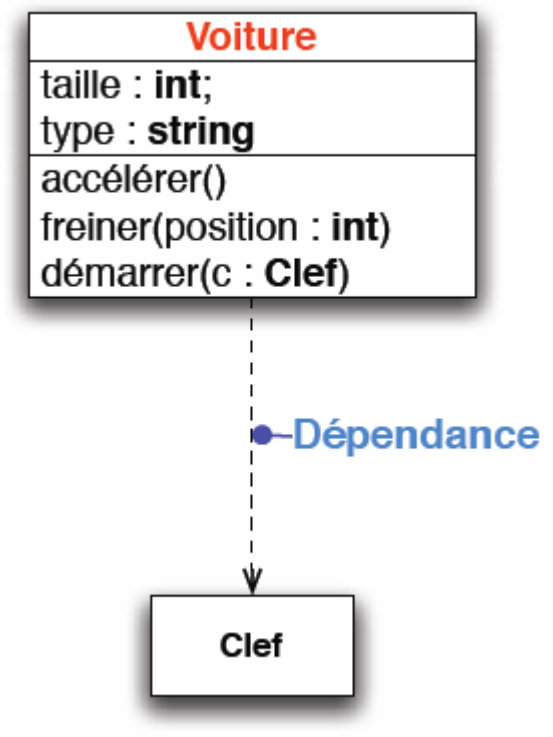
- **Relations entre classes**
  - **Agrégation et composition**



Une **agrégation** est une association qui représente une relation d'**inclusion structurelle** ou comportementale d'un élément dans un ensemble
La **composition**, décrit une **contenance structurelle** entre instances. Ainsi, la destruction de l'objet composite implique la destruction de ses composants.

III) Les diagrammes de classes

- **Relations entre classes**
  - **Dépendance**

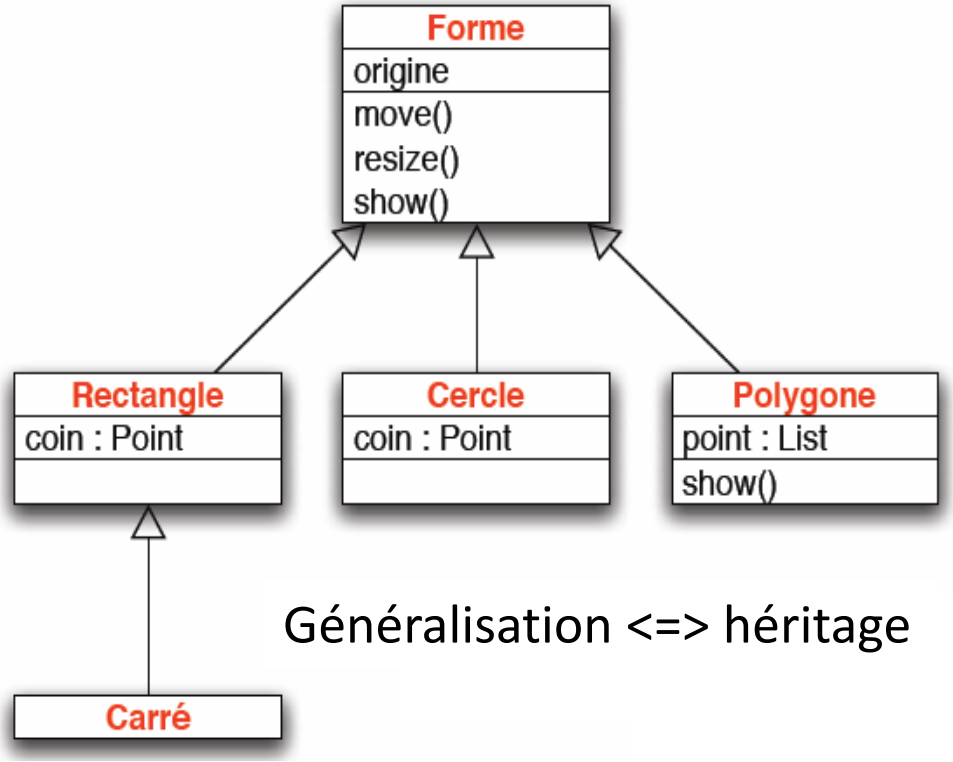


•Une dépendance est une relation unidirectionnelle exprimant une dépendance sémantique. Elle est représentée par un trait discontinu orienté. Elle indique que la modification de la cible peut impliquer une modification de la source. La dépendance est souvent stéréotypée.

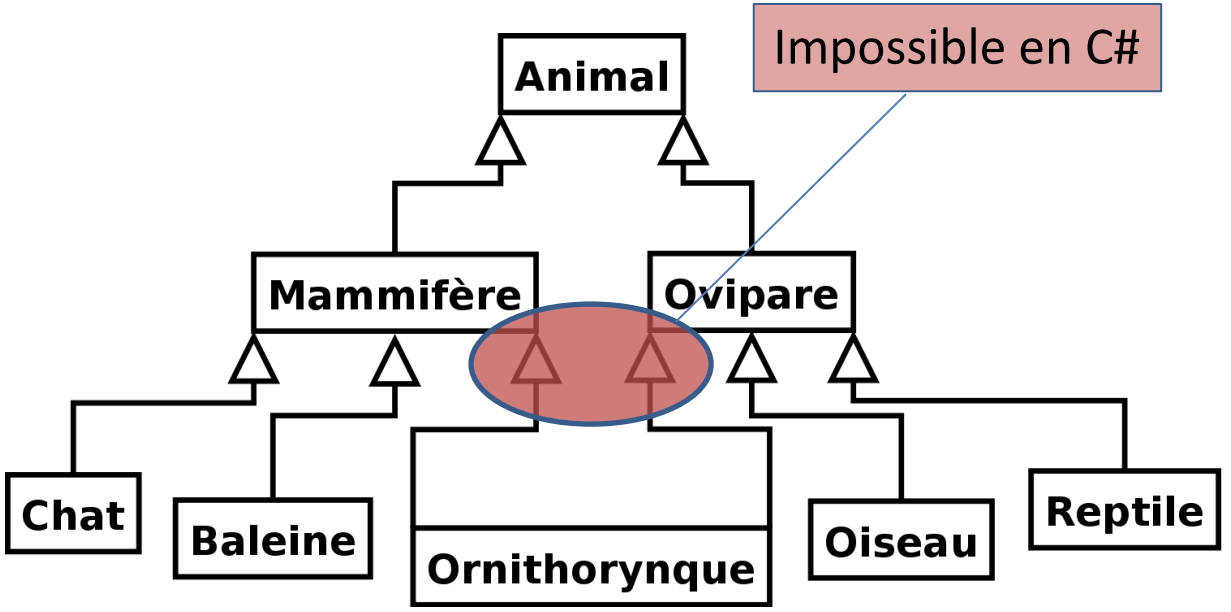
•On utilise souvent une dépendance quand une classe en utilise une autre comme argument dans la signature d'une opération.

III) Les diagrammes de classes

- **Relations entre classes**
  - **Héritage**



Généralisation <=> héritage



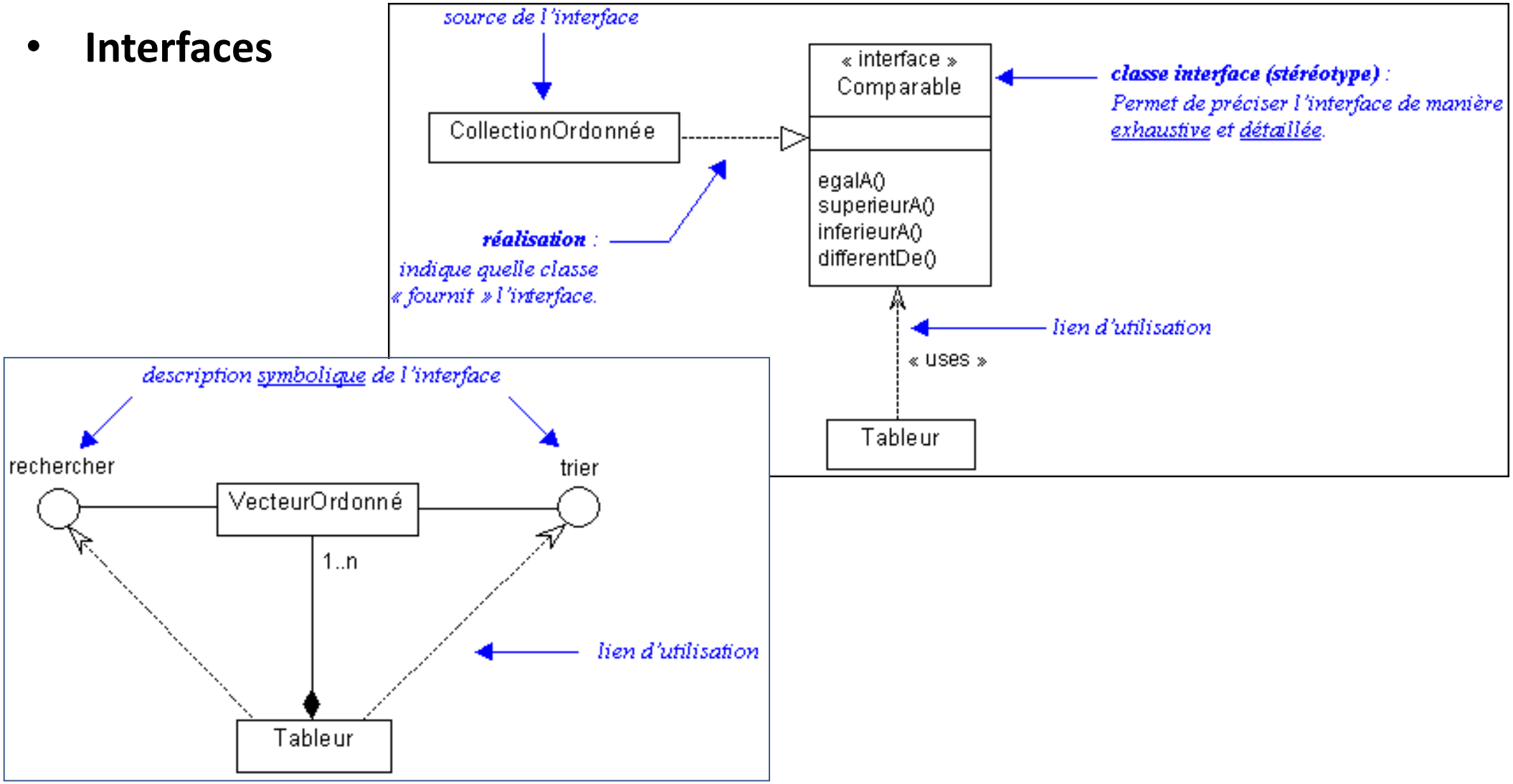
En UML, l'héritage peut être aussi utilisé pour les paquetages

III) Les diagrammes de classes

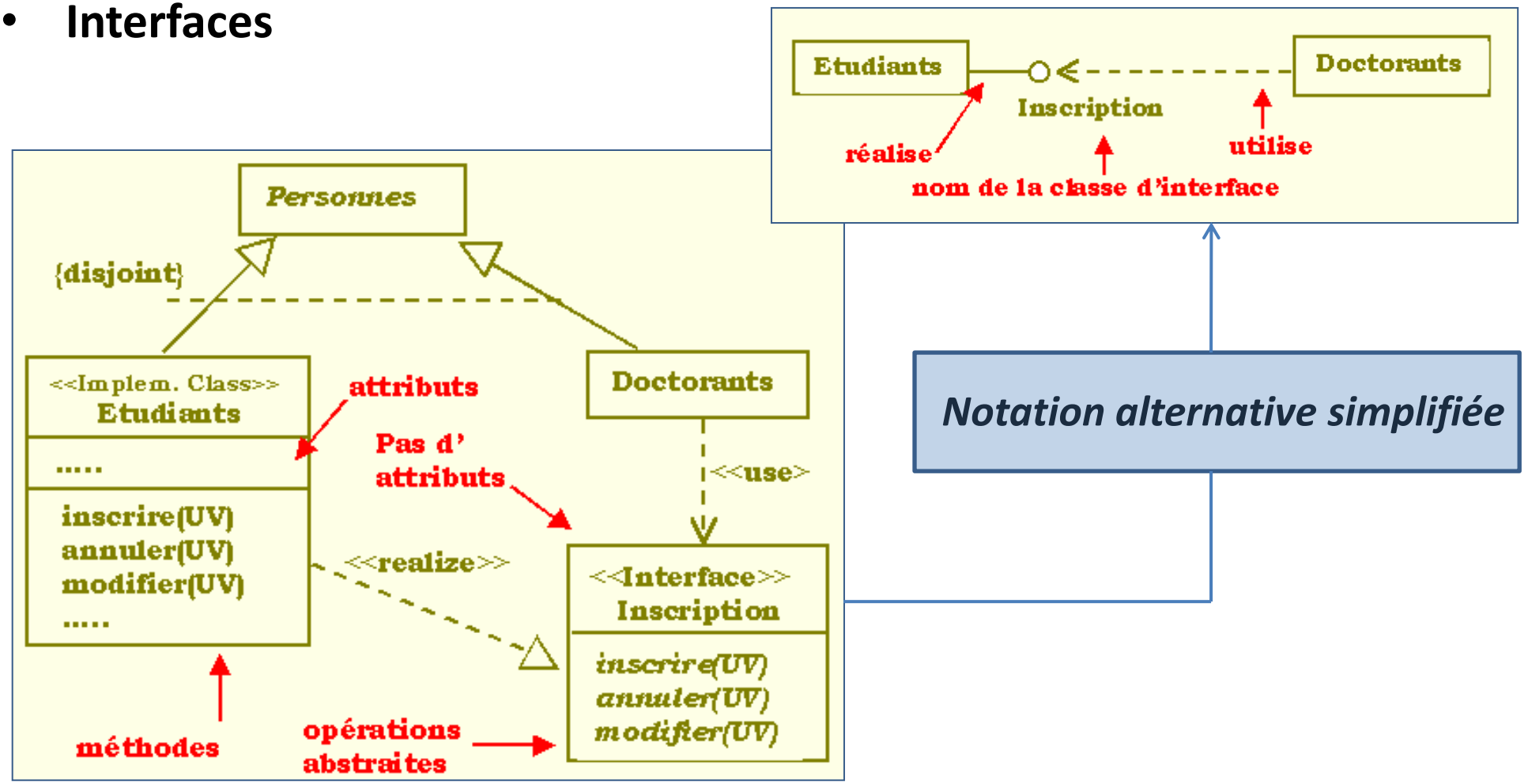
- **Interfaces**
  - Une interface est une spécification des opérations qui sont visibles par l'environnement d'une classe.
    - c'est une classe qui n'a pas d'implémentation (puisqu'il s'agit d'une spécification)
    - elle ne présente qu'une partie du comportement de la classe correspondante (puisque cette dernière peut implémenter des méthodes "internes")
    - elle ne possède que des opérations (pas d'attributs, ni d'associations, ni d'états puisqu'elle n'est pas implémentée). Une interface est donc équivalente à une classe abstraite qui serait réduite à des opérations abstraites (elle ne possède ni attributs ni méthodes) et qui n'aurait pas de descendants (et donc pas d'instances).
  - Une interface peut être vue comme un type particulier dont le domaine serait celui des objets d'une (ou plusieurs) classes d'implémentation.

III) Les diagrammes de classes

- **Interfaces**



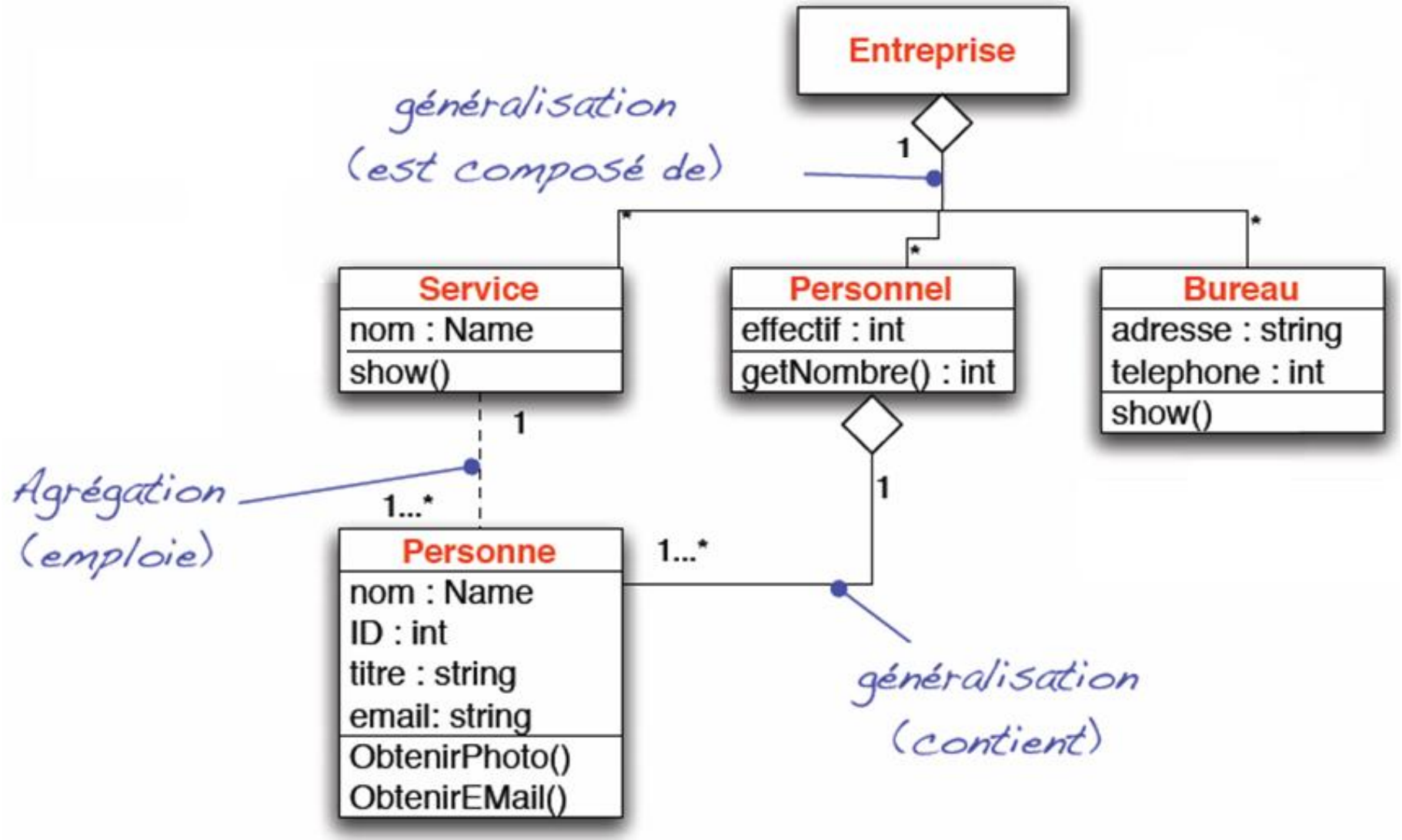
- **Interfaces**


Etudiants
Doctorants
Inscription
réalise
utilise
nom de la classe d'interface
Personnes
{disjoint}
<<Implem. Class>>
Etudiants
.....
inscrire(UV)
annuler(UV)
modifier(UV)
.....
attributs
Pas d'
attributs
Doctorants
<<use>>
<<realize>>
<<Interface>>
Inscription
inscrire(UV)
annuler(UV)
modifier(UV)
méthodes
opérations
abstraites
Notation alternative simplifiée

III) Les diagrammes de classes

- Nous allons maintenant essayer de modéliser une entreprise à l'aide d'un diagramme de classes (de manière simple):
  - Une entreprise est composée de plusieurs services, dans ces services officie un certain nombre de personnes.
  - Dans les locaux de l'entreprise se trouvent des bureaux
  - Chaque bureau porte une adresse et possède un numéro de téléphone.
  - Les employés sont reconnus à l'aide de leur nom et de leur identifiant personnel. Chaque employé possède un titre dans l'entreprise.

III) Les diagrammes de classes



III) Les diagrammes de classes

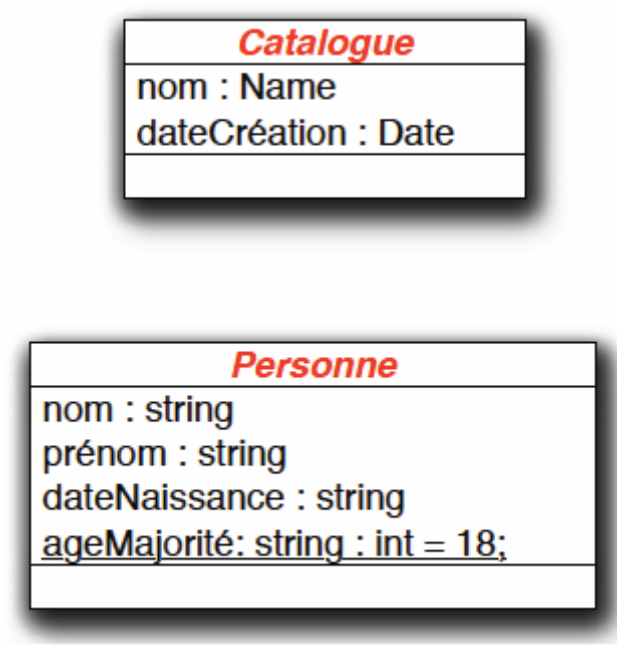
Implémentation

- Les diagrammes permettent de spécifier de manière claire, les classes et les relations existantes entre elles :
  - Génération de code source à partir du modèle.
  - Les modèles ne sont pas spécifiques à un langage (choix à la génération => 1 modèle plusieurs implémentations possibles suivant les contraintes).

- A partir de tout diagramme UML, il est possible de remonter vers le cahier des charges :
  - Les éléments du modèle ont une sémantique propre et unique.

III) Les diagrammes de classes

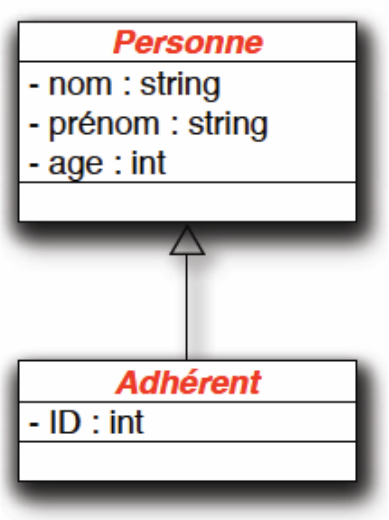
- Indépendance du modèle vis-à-vis du langage utilisé pour l'implémentation.
- Implémentation particulière => pas d'ambiguïté possible.




```
class Catalogue {
private:
     string nom;
      DateTime dateCreation;
      // ... ... ...
}

class Personne {
private:
   string nom;
   string prénom;
protected:
      DateTime dateNaissance;
private:
     static int ageMajorite = 18;
}
```

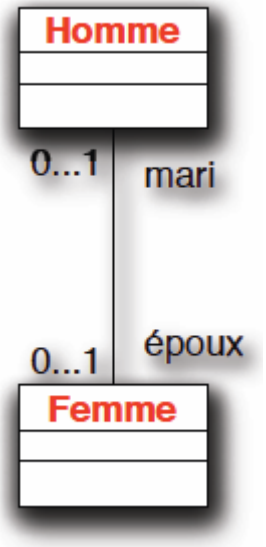
III) Les diagrammes de classes



généralisation → héritage

```
class Personne {
   // ... ... ...
}

class Adhèrent : Personne {
    private int iD;
}
```



composition → agrégation

```
class Homme {
private:
      Femme épouse;
      // ... ... ...
}

class Femme {
private:
      Homme mari;
      // ... ... ...
}
```

IV) Les diagrammes de séquences

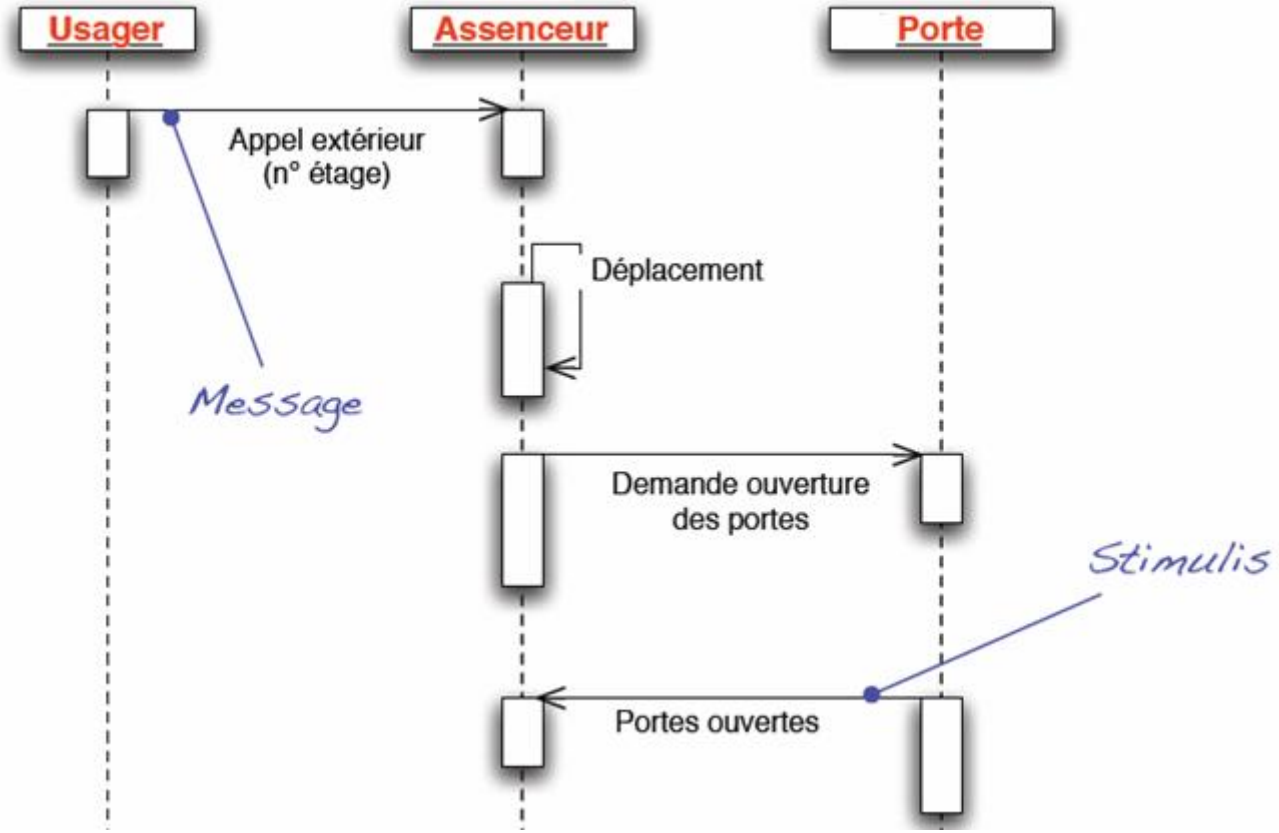
- Permet de mettre en évidence :
  - L'aspect temporel des traitements (ordre chronologique de réalisation).
  - Les objets et les messages échangés par les entités interagissant avec et/ou dans le système.

- La modélisation est basée sur l'affichage des objets participant à la séquence et à l'ordre des messages et des actions associées.

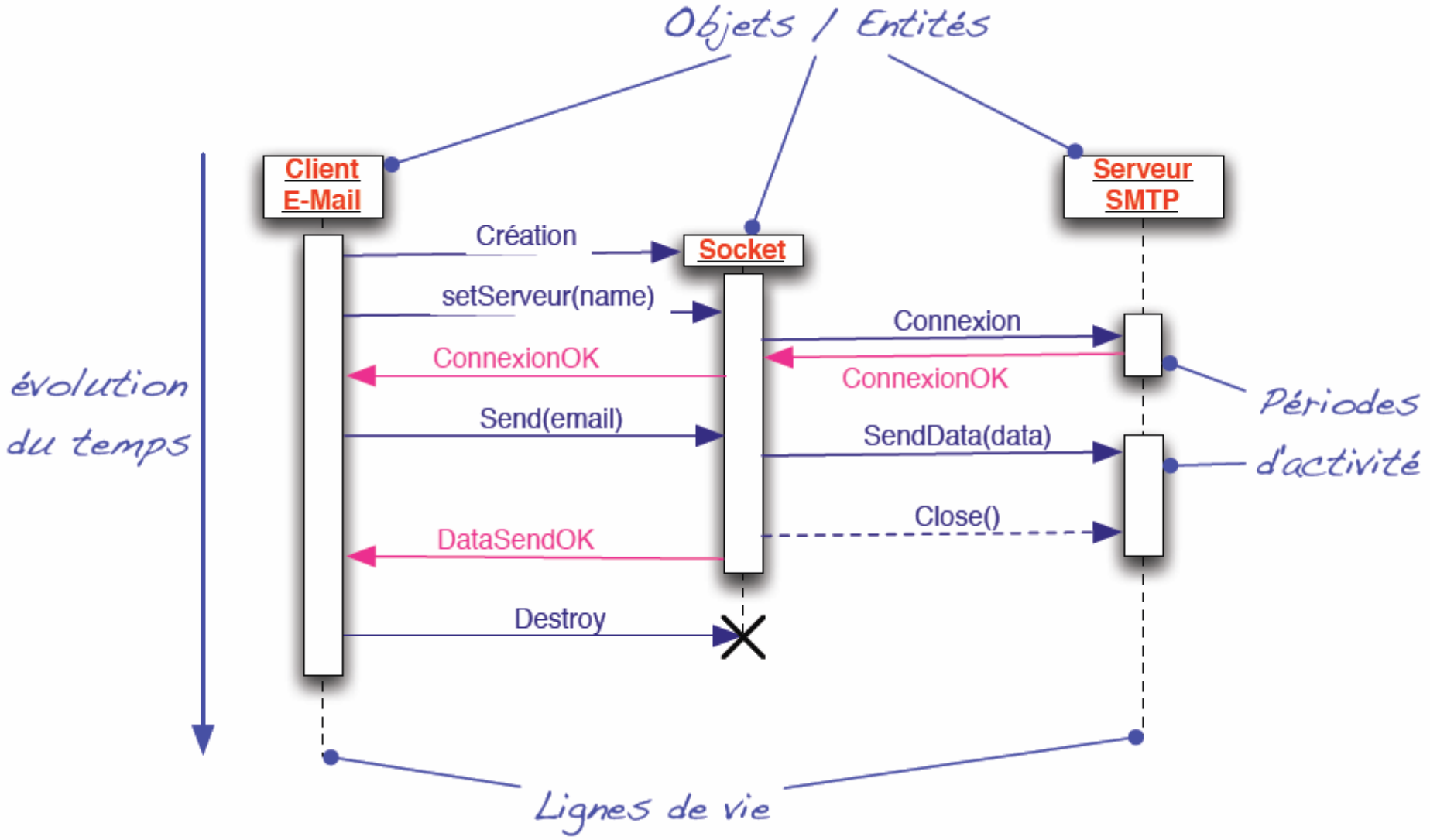
- Les *diagrammes de séquence permettent de modéliser* les interactions entre les objets :
  - Communications *synchrones (flèches pleines).*
  - Communications *asynchrones (flèches pointillés).*
  - Les données transmises entre les objets (annotation des flèches).

IV) Les diagrammes de séquences

- Ils permettent aussi de connaître pour chaque objet :
  - Ses dates de création et de destruction.
  - Sa durée de vie, si la création est externe au diagramme (ligne verticale en pointillés).



IV) Les diagrammes de séquences



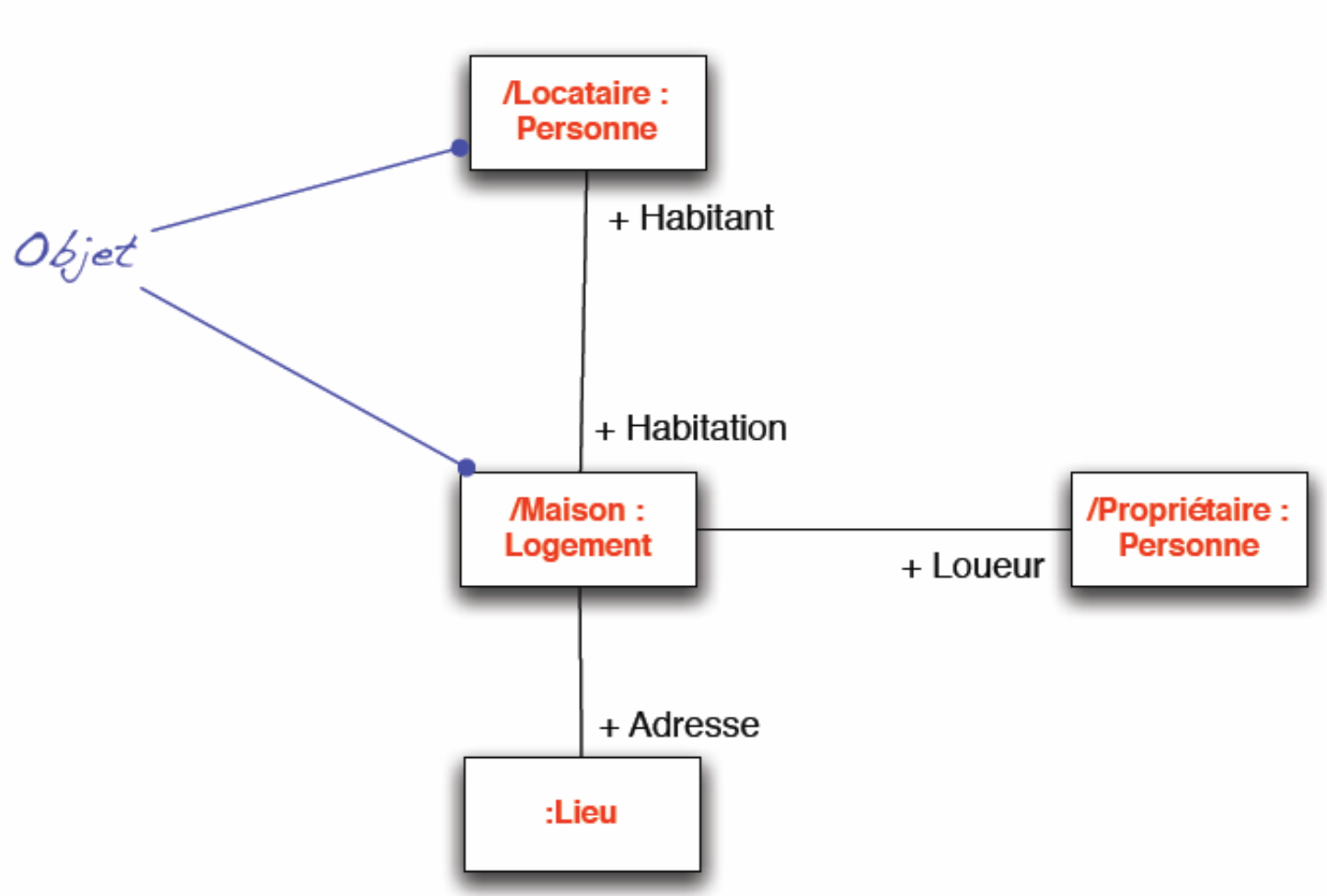
V) Les diagrammes de collaboration

- Mise en évidence de *l'organisation des objets qui vont collaborer pour effectuer une interaction.*
  - On représente les objets qui vont intervenir (les sommets).
  - On modélise les liens représentant les communications entre les objets (les arcs).
  - On annote les liens à l'aide des informations qui vont être échangées.

- Visualisation claire du flot de contrôle dans le contexte de l'organisation structurelle.

- 2 niveaux de représentation (d'abstraction)
  - Le niveau spécification ➡ définition des classes et de leurs rôles,
  - Le niveau instance ➡ définition des objets et des messages échangés,

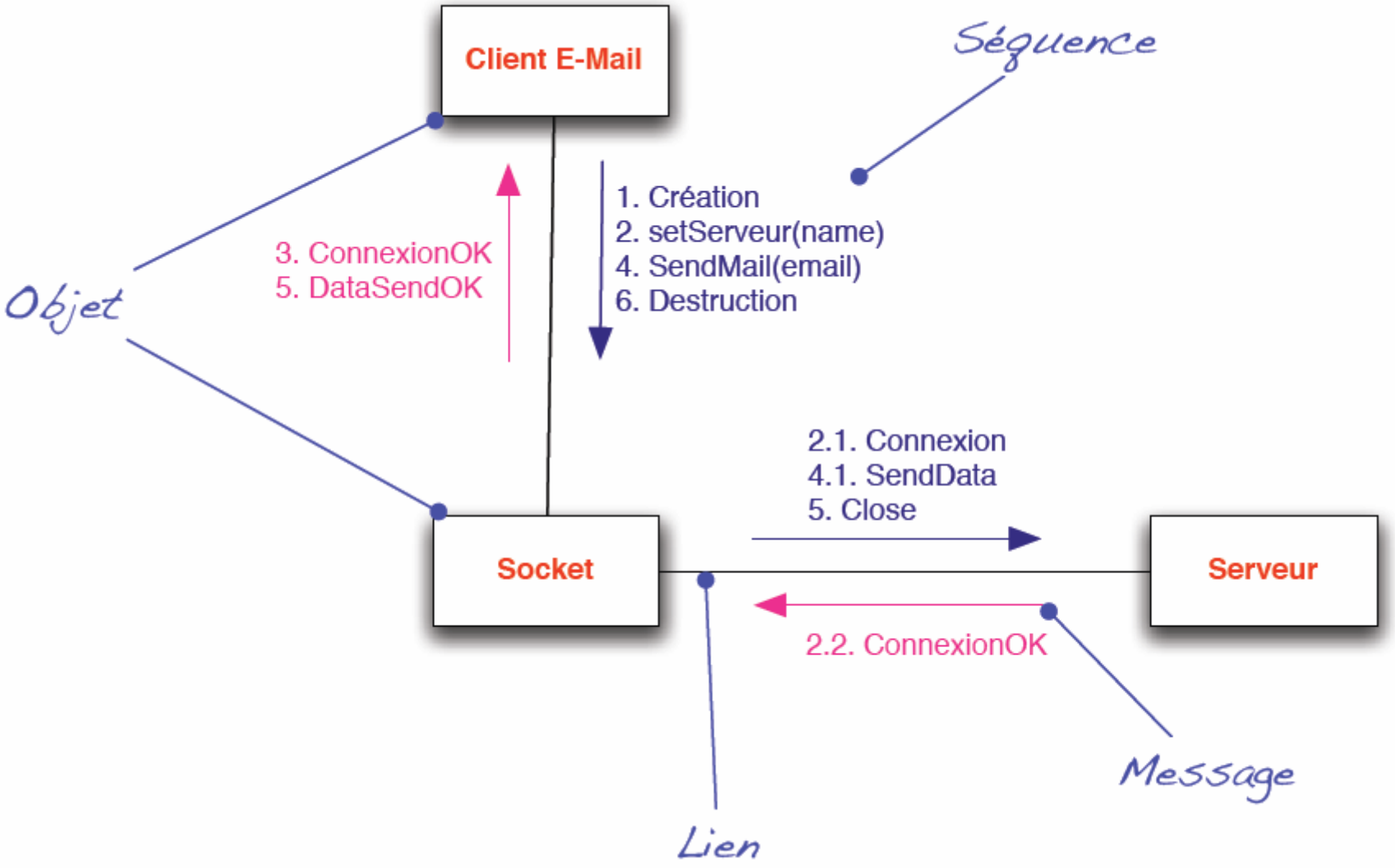
V) Les diagrammes de collaboration



V) Les diagrammes de collaboration

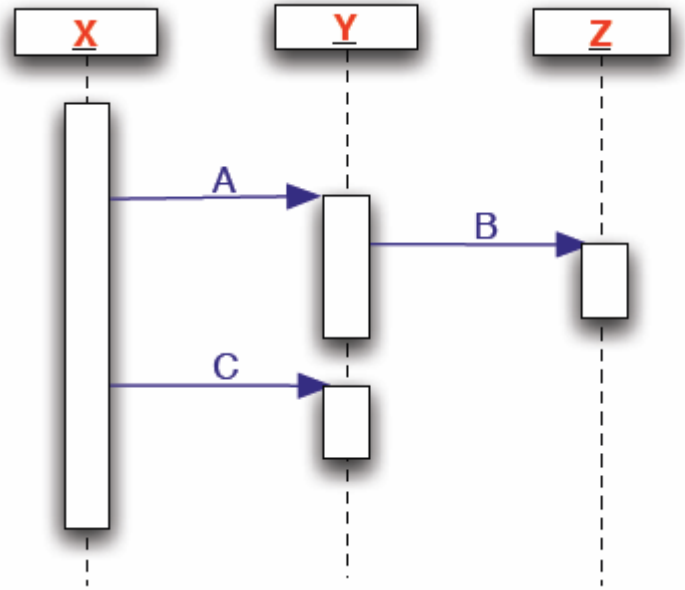
- La *modélisation temporelle est en partie masquée* => *u*tilisation de la numérotation des séquences pour ordonner les traitements réalisés.
- Contrairement au diagramme de séquence, certaines informations ne sont pas représentées :
  - On ne modélise pas le ligne de vie des objets.
  - On ne modélise pas les temps d'activité des objets.
- Par contre on modélise explicitement les liens qui existent entre les différents objets.

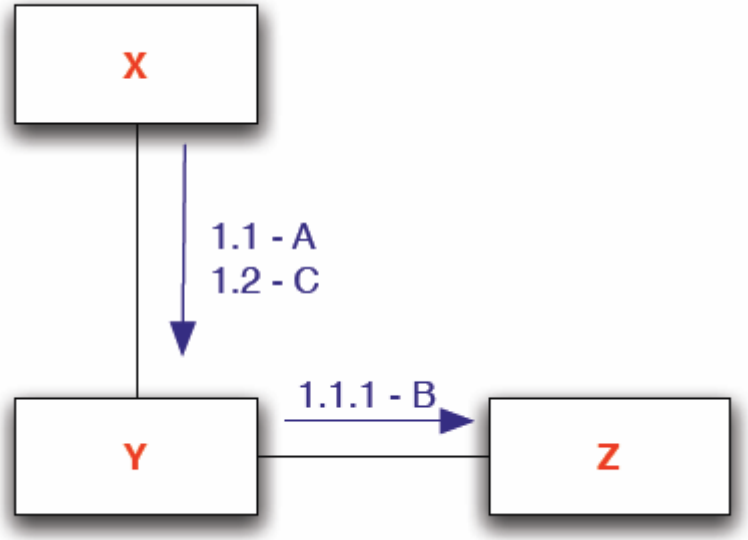
V) Les diagrammes de collaboration



V) Les diagrammes de collaboration

- Correspondance entre le diagramme de séquences et de collaboration





VI) Les diagrammes d'états

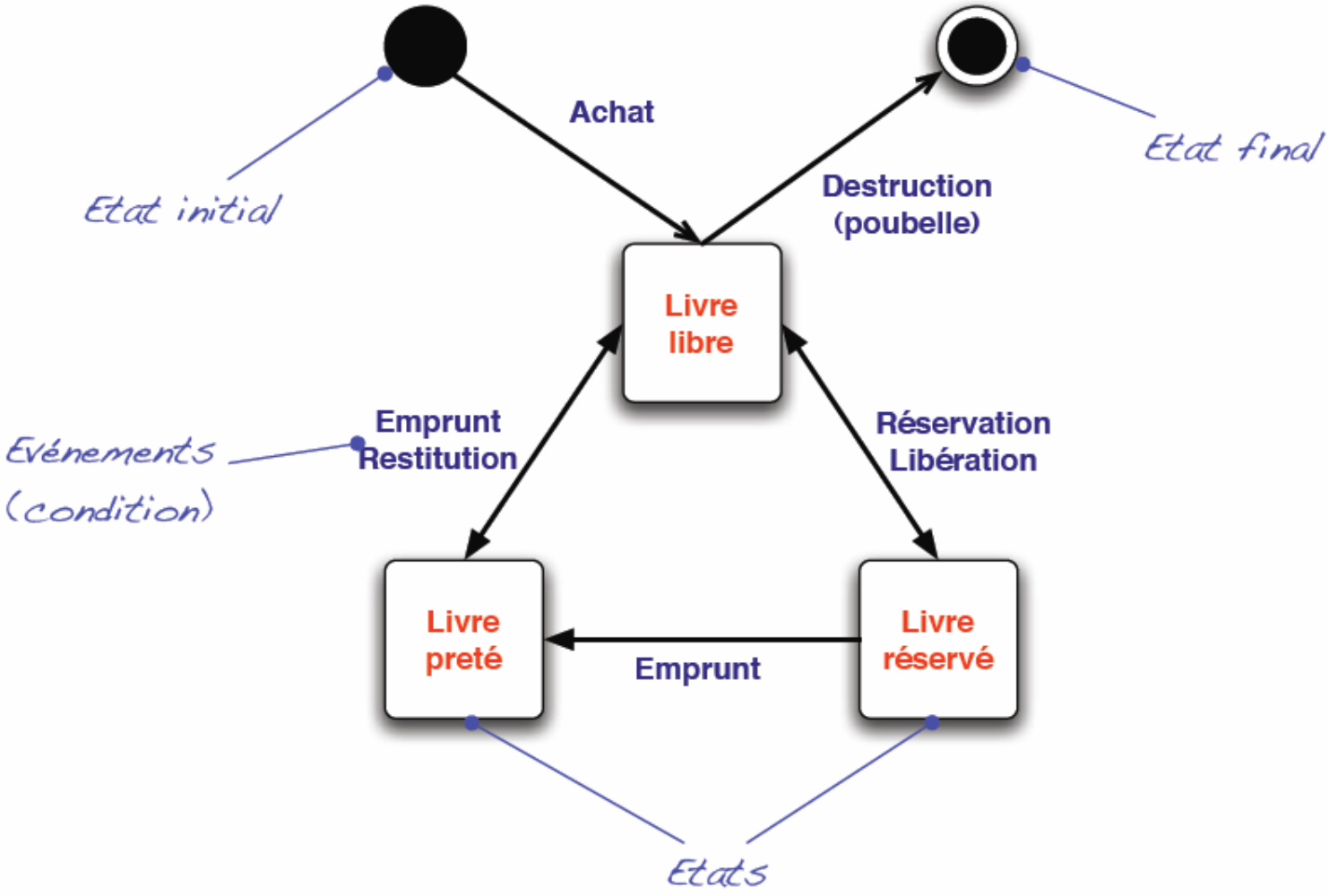
- Les diagrammes d'états-transitions d'UML décrivent le *comportement interne* d'un objet à l'aide d'un automate à états finis.

- Ils présentent 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 discrets (de type signaux, invocations de méthode).

VI) Les diagrammes d'états

- Diagramme d'états

La vision globale du système n'apparaît pas sur ce type de diagramme puisqu'ils ne s'intéressent qu'à un seul élément du système indépendamment de son environnement.



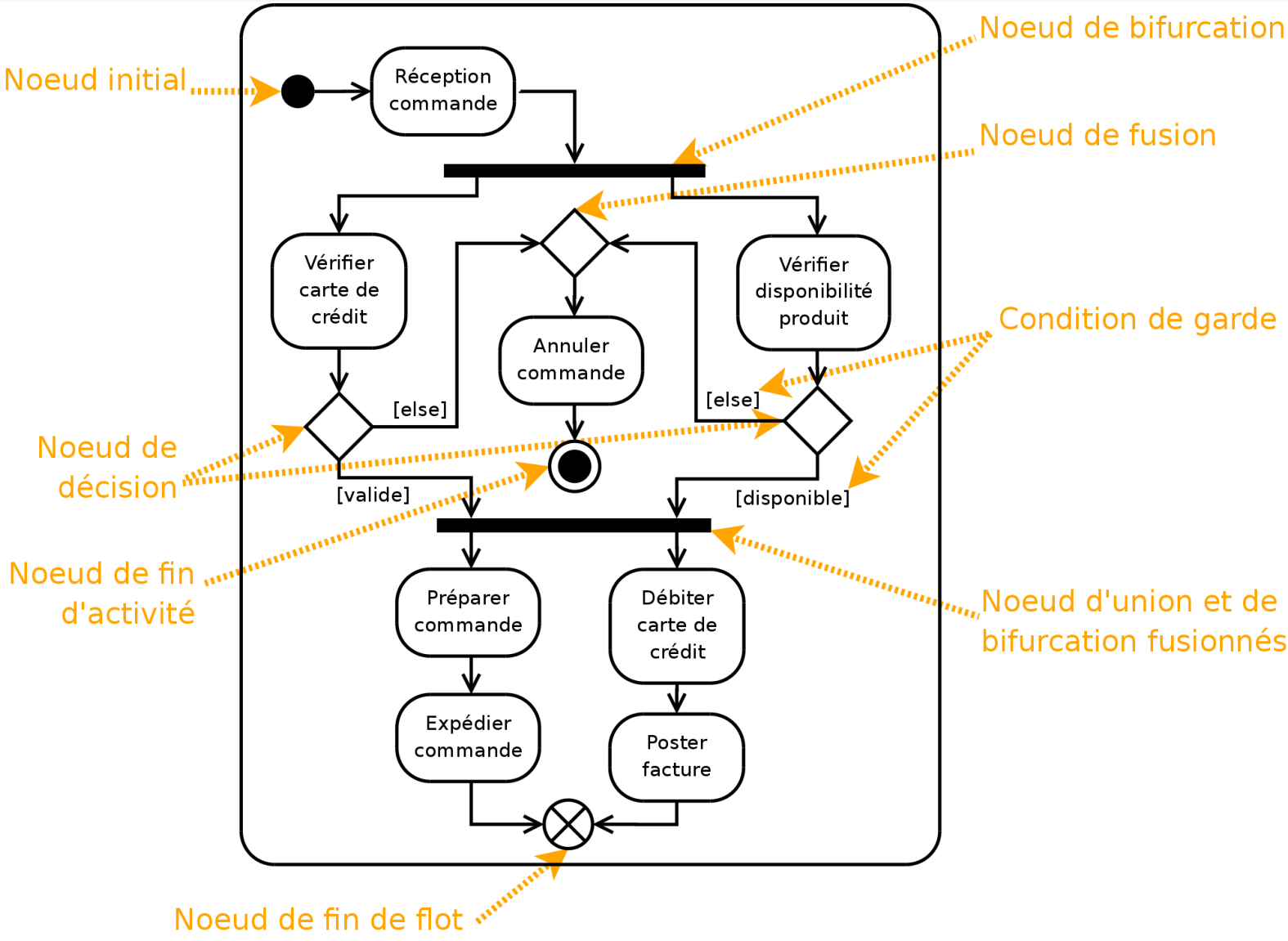
VII) Les diagrammes d'activité

- Les diagrammes d'activités sont adaptés à la modélisation du cheminement de flots de contrôle et de données. Ils permettent de représenter le comportement d'une méthode ou le déroulement d'un cas d'utilisation.

- Les diagrammes d'activités sont relativement proches des diagrammes d'états-transitions dans leur présentation, mais tandis que les premiers mettent l'accent sur le flot de contrôle d'un objet à l'autre, les seconds insistent sur le flot de contrôle d'une activité à l'autre.

VII) Les diagrammes d'activité

On peut attacher un diagramme d'activités à n'importe quel élément de modélisation afin de visualiser, spécifier, construire ou documenter le comportement de cet élément.



Les outils

- Logiciels disponibles
  - Visual studio 2008
  - Visio
  - ArgoUML (gratuit)
  - StarUML (gratuit)
- Logiciels Pro
  - Enterprise architect
  - UModeL
  - Visual Paradigm

Sondage developpez.net: Quel **outil de modélisation UML** utilisez vous ?		
[Télélogic] TAU UML / TAU GII	4	1,10%
[I-Logix] Rhapsody Modeler	6	1,65%
ArgoUML	42	11,54%
[Eclipse] Plugin Omondo	38	10,44%
Umbrello UML Modeller	17	4,67%
ClassBuilder	2	0,55%
[Sybase] PowerAMC/PowerDesigner	32	8,79%
[Microsoft] Visio	33	9,07%
[Objecteering Software] Objecteering/UML	23	6,32%
[Gentleware] Poseidon UML	43	11,81%
[IBM] Rational Rose / XDE	90	24,73%
[Borland] Together / ModelMaker	34	9,34%

Bibliographie

- Cours Bertrand Legal
- Wikipédia
- http://laurent-audibert.developpez.com/Cours-UML/html/
- http://uml.free.fr/index-cours.html
- http://lacl.univ-paris12.fr//polonowski/Cours/MOO/chapitre2.pdf

Ressources

- **Conception & realisation des bases de donnees : de uml à sql, Jacques Guyot (pdf)**
- http://uml.developpez.com/livres/ **Critiques des meilleurs livres sur UML**