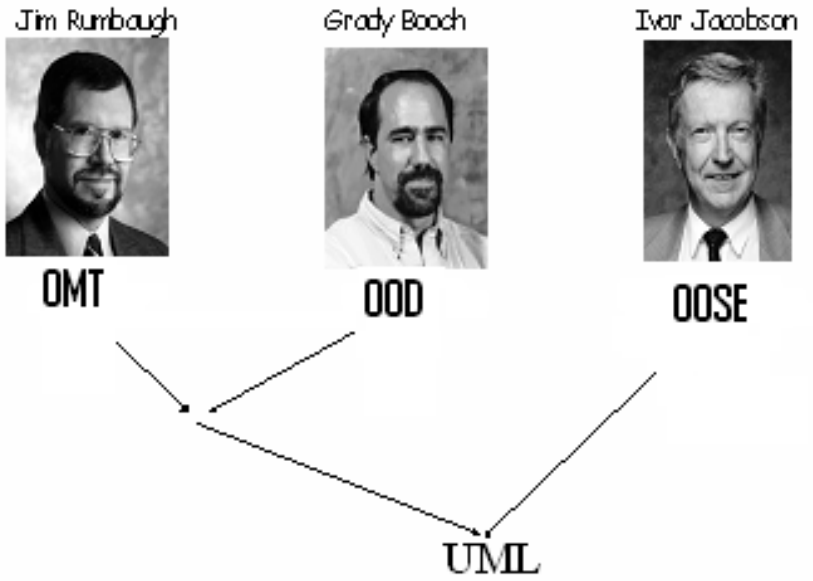
Génie logiciel



UML : Unified Modeling Language

A. Madani (madaniabdellah@gmail.com)

Génie Logiciel
UML – Partie I

- Introduction
  - Historique d'UML
- Diagrammes UML
  - Diagrammes des cas d'utilisation
  - Diagrammes de classes et d'objets
  - Diagramme de séquences
  - Autres diagrammes
- Etudes de cas

Historique d'UML

Début des années 1990

- les premiers processus de développement **OO** apparaissent
- prolifération des méthodes et notations étaient la cause de grande confusion :
  - méthode **OOD** de Grady Booch *(1991)*
  - méthode **OMT** de James Rumbaugh *(1991)*
  - méthode **OOSE** de Ivar Jacobson *(1991)*
  - méthode **OOA/OOD** de **Coad and Yourdon** *(1992)*
  - méthode de **Schlaer and Mellor** *(1992)*
  - Etc.

Historique d'UML

Fin 1994

- J. Rumbaugh rejoint G. Booch chez Rational Software
- OMT + OOD ➔ **Unified Method** *(oct 1995)*

Fin 1995

- I. Jacobson les rejoint chez Rational Software
- Unified Method + OOSE ➔ **UML 0.9** *(juin 1996)*

Début 1997

- Partenaires divers : Microsoft, Oracle, IBM, HP et autres leaders collaborent
- ➔ **UML 1.0** *(jan 1997)*

Fin 1997

- l'OMG (Object Management Group) retient **UML 1.1** comme **norme de modélisation**

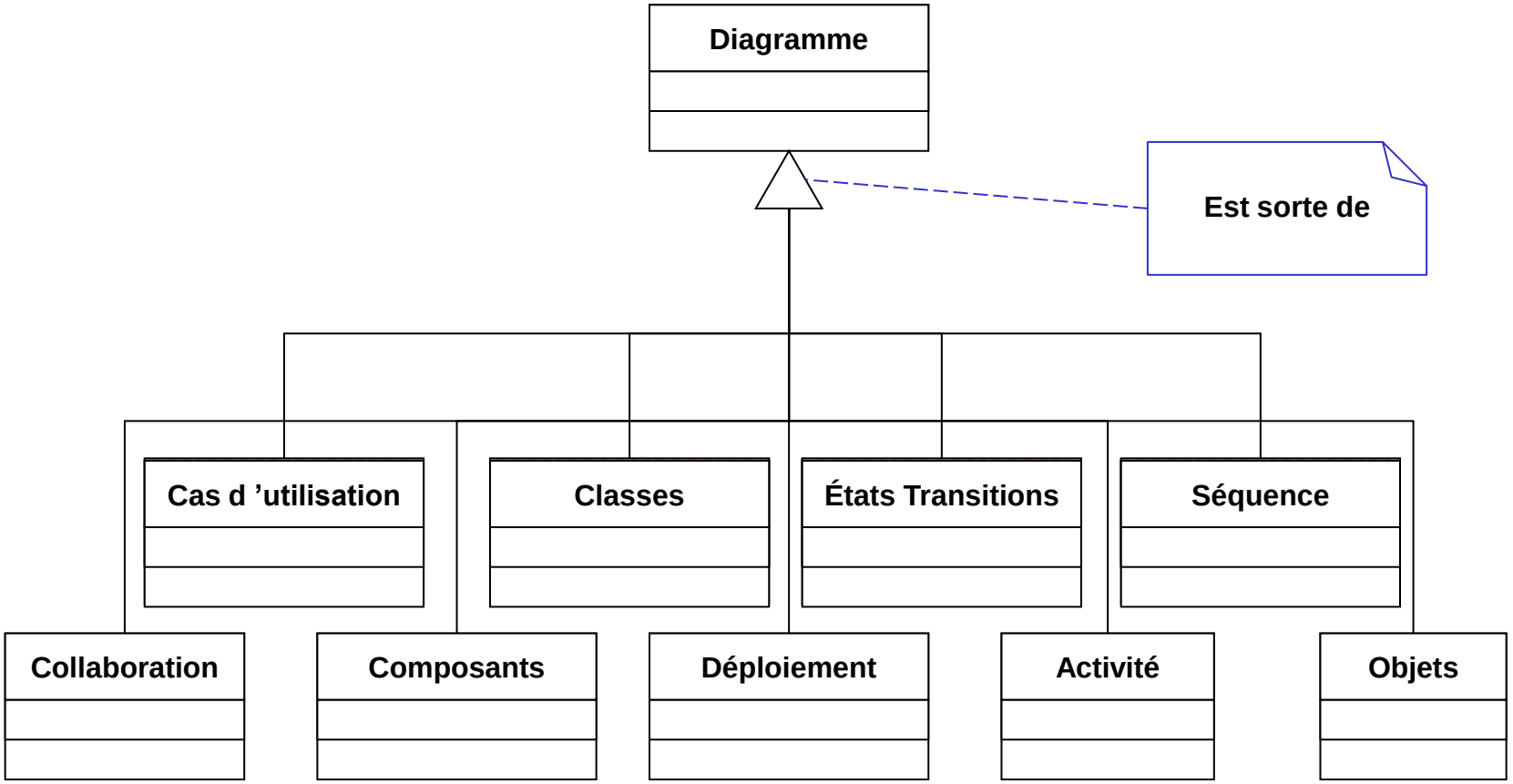
Historique d'UML

Les versions se succèdent :

- Début 1998
  - UML 1.2
- En 1998
  - UML 1.3
- En 2001
  - UML1.4
- En 2003
  - UML 1.5
- En 2005
  - UML 2.0

Diagrammes d'UML

UML1.1 comprend 9 de diagrammes :



UML

Diagrammes de cas d'utilisation

Diagramme des cas d'utilisation

- Avant de développer un système, il faut savoir précisément à **QUOI** il devra servir, càd à quels besoins il devra répondre.
- Modéliser les besoins permet de :
  - Faire l'inventaire des fonctionnalités attendues ;
  - Organiser les besoins entre eux, de manière à faire apparaître des relations (comportements partagés, exceptions, ...).

Exemple d'un diagramme de cas d'utilisations

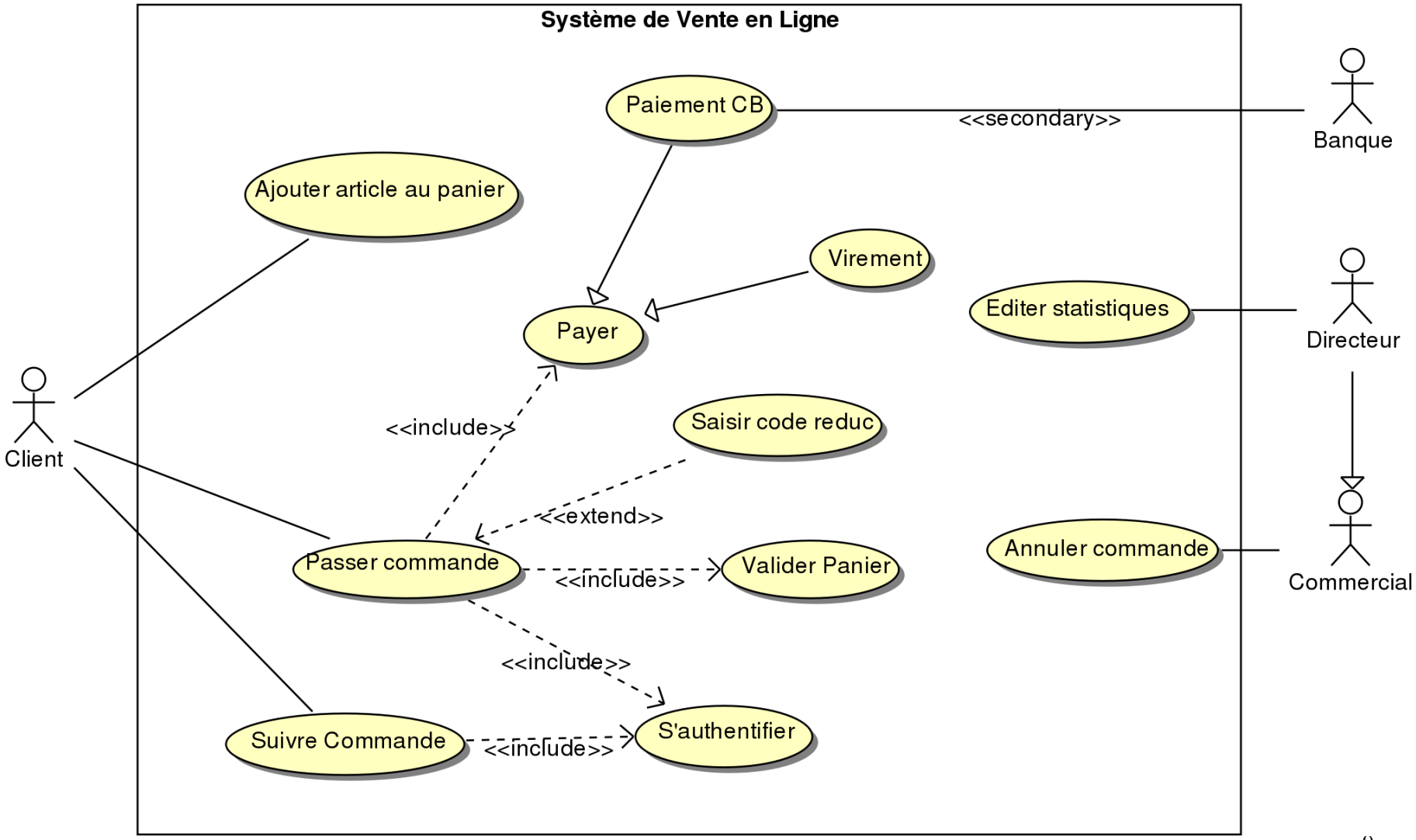


Diagramme de cas d'utilisation

- Sert à modéliser les aspects dynamiques d'un système.
- Décrit, sous forme d'actions et de réactions, le comportement d'un système du point de vue d'un utilisateur.
- Fait ressortir les acteurs et les fonctions offertes par le système.
- Utilisé pour modéliser les exigences (besoins) du client

Diagrammes des cas d'utilisation

Comportent plusieurs éléments :

- Acteurs
- Cas d'utilisation
- Relations de dépendances, de généralisations et d'associations

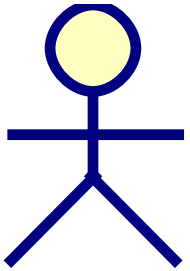
Acteurs

- UML n'emploi pas le terme d'utilisateur mais d'acteur.
- Le terme acteur ne désigne pas seulement des utilisateurs humains mais également les autres systèmes (machines, programmes, …)
- Un acteur est un rôle joué par une entité externe qui agit sur le système (Comptabilité, service commercial, …), en échangeant de l'information (en entrée et en sortie)

Acteurs

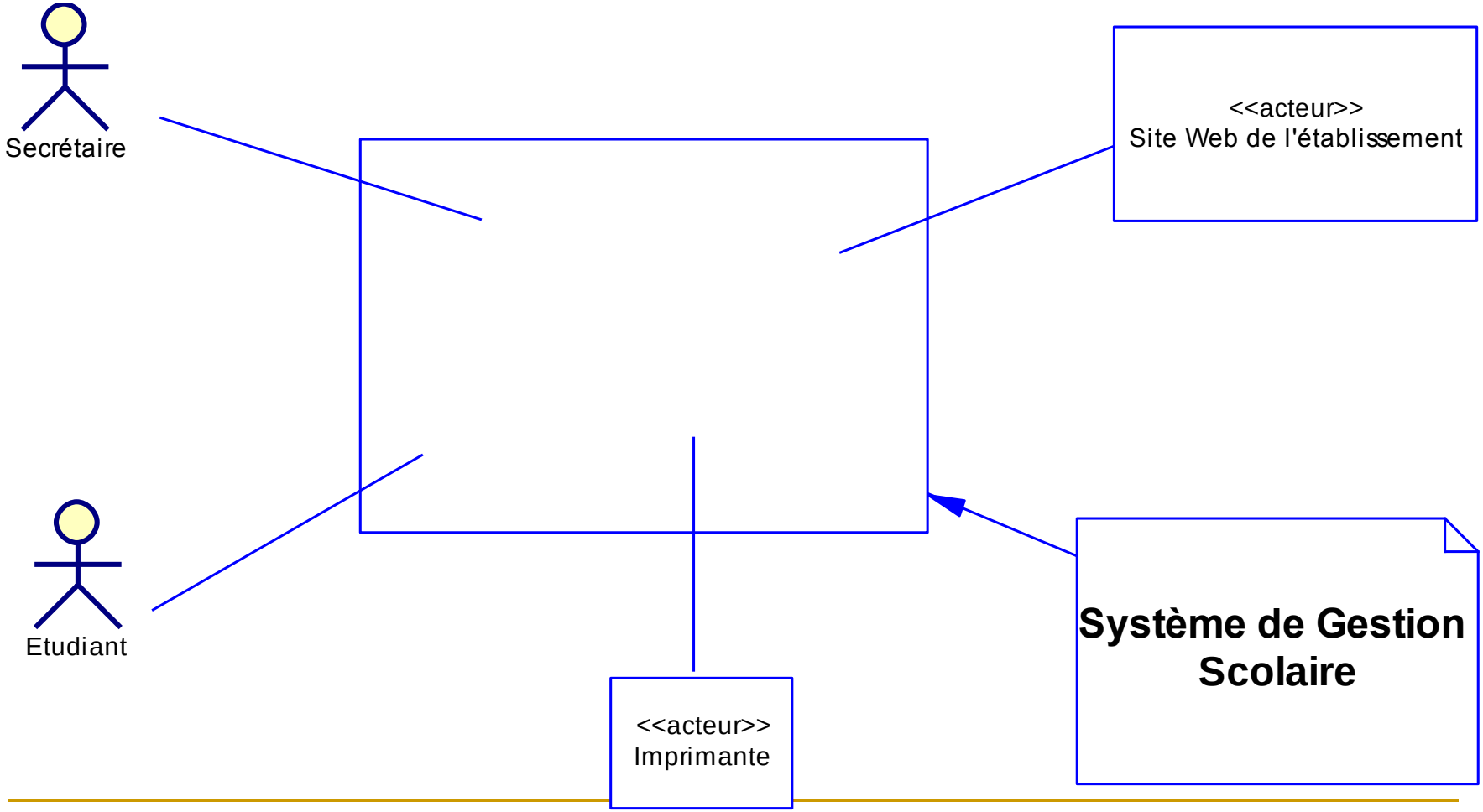
Peut être représenté de deux manières différentes :

- Petit personnage (stick man)



- Classe stéréotypée

Acteurs

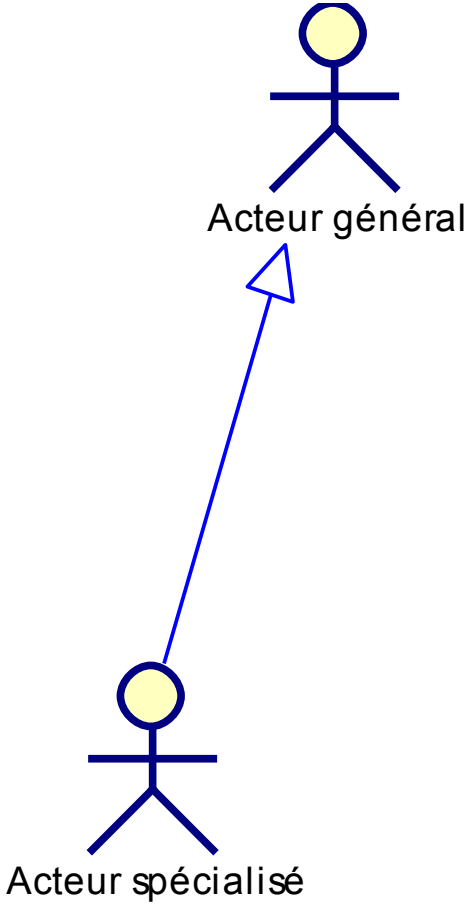

Secrétaire
Etudiant
<<acteur>>
Site Web de l'établissement
<<acteur>>
Imprimante
Système de Gestion Scolaire

Acteurs

Un acteur peut être une spécialisation d'un autre acteur déjà défini.

→

Dans ce cas, on utilise la relation de généralisation/spécialisation.



Cas d'utilisation

Les cas d’utilisations

- Permettent de modéliser les attentes (besoins) des utilisateurs
- Représentent les fonctionnalités du système
- Suite d’événements, initiée par des acteurs, qui correspond à une utilisation particulière du système

Cas d'utilisation

Un cas d'utilisation est représenté par une ellipse en trait plein, contenant son nom.

Structuration des cas d'utilisation

Après avoir identifié les acteurs et les cas d'utilisation, il est utile de restructurer l'ensemble des cas d'utilisation que l'on a fait apparaître afin de rechercher les :

- Comportements partagés
- Cas particuliers, exceptions, variantes
- Généralisations/spécialisations.

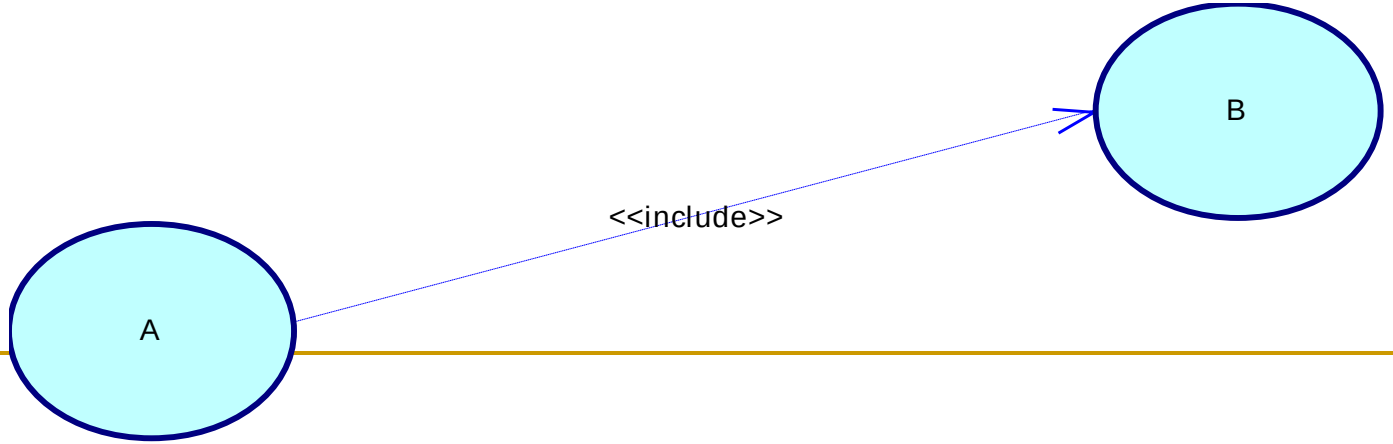
Structuration des cas d'utilisation

UML définit trois types de relations standardisées entre cas d'utilisation :

- Une relation d'inclusion, formalisée par la dépendance «include»
- Une relation d'extension, formalisée par la dépendance «extend»
- Une relation de généralisation/spécialisation

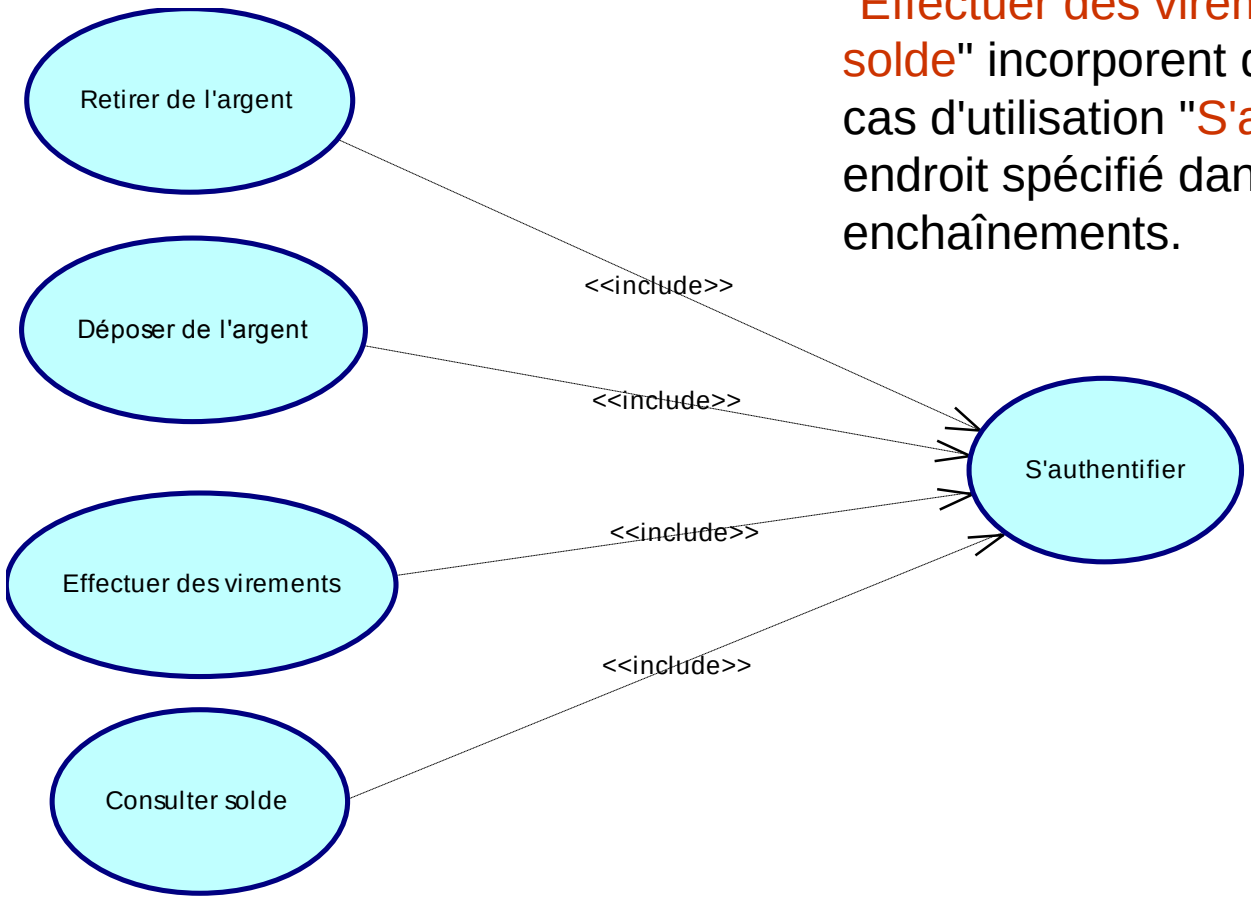
Relation d'inclusion

- A inclut B : le cas A inclut obligatoirement le comportement définit par le cas B; permet de factoriser des fonctionnalités partagées
- Le cas d'utilisation pointé par la flèche (dans notre cas B) est une sous partie de l'autre cas d'utilisation (A, dans notre exemple).



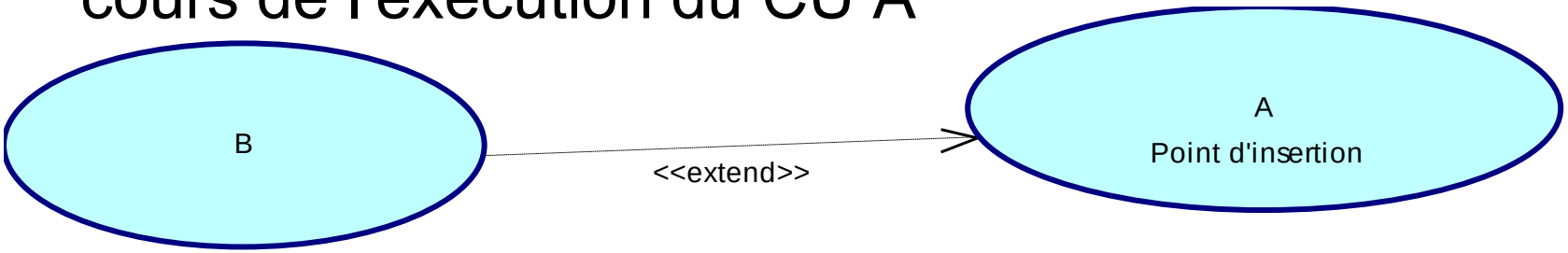
Relation d'inclusion

Les cas d'utilisation "Déposer de l'argent", "Retirer de l'argent", "Effectuer des virements" et "Consulter solde" incorporent de façon explicite le cas d'utilisation "S'authentifier", à un endroit spécifié dans leurs enchaînements.



Relation d'extension

- Le CU source (B) ajoute, sous certaines conditions, son comportement au CU destination (A)
- En d'autres termes, le CU B peut être appelé au cours de l'exécution du CU A

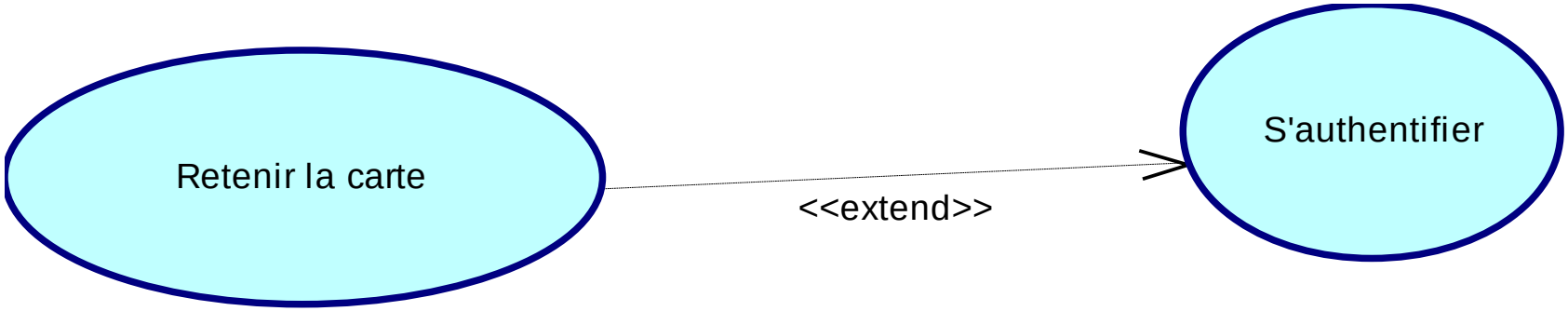


- Le comportement ajouté s'insère au niveau d'un point d'extension définit dans le CU destination

Relation d'extension

Exemple :

Au moment de l'authentification, il se peut que le guichet retient la carte.



Relation d'héritage

- Il peut également exister une relation d'héritage entre cas d'utilisation.
- Cette relation exprime une relation de spécialisation/généralisation au sens classique.

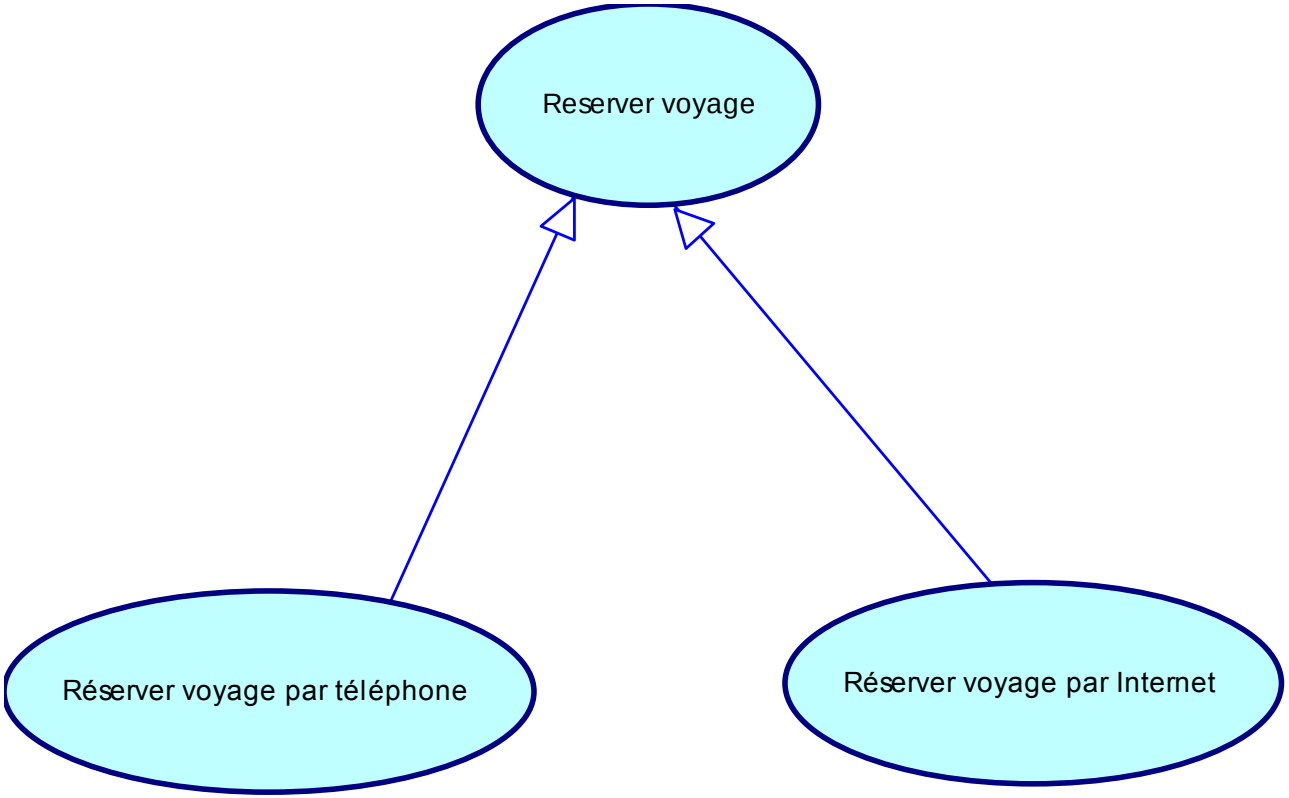
Relation d'héritage : Exemple

Dans un système d'agence de voyage, un acteur "Touriste" peut participer à un cas d'utilisation de base qui est "Réserver voyage", qui suppose par exemple, des interactions basiques au comptoir de l'agence. Une réservation peut être réalisée par téléphone ou par Internet.

Relation d'héritage : Exemple

- On voit qu'il ne s'agit pas d'une relation "extend", car la réservation par Internet n'étend pas les interactions ni les fonctionnalités du cas d'utilisation "Réserver voyage".
- Les deux cas d'utilisation "Réservation voyage" et "Réserver voyage par Internet" sont liés : la réservation par Internet est un cas particulier de réservation.
- De façon générale en objet, une situation de cas particulier se traduit par une relation de généralisation/spécialisation.

Relation d'héritage : Exemple



Structuration entre cas d'utilisation

Résumé

Les cas peuvent être structurées par des relations :

- A inclut B : le cas A inclut obligatoirement le comportement définit par le cas B; permet de factoriser des fonctionnalités partagées
- A étend B : le cas A est une extension optionnelle du cas B à un certain point de son exécution.
- A généralise B : le cas B est un cas particulier du cas A.

Relations entre cas d'utilisation : Exemple

Un client peut effectuer un **retrait bancaire**. Le retrait peut être **effectué sur place** ou par **Internet**. Le client **doit être identifié** (en fournissant son code d'accès) pour effectuer un retrait, mais si le **montant dépasse 500DH**, la **vérification du solde de son compte** est réalisée.

Relations entre cas d'utilisation

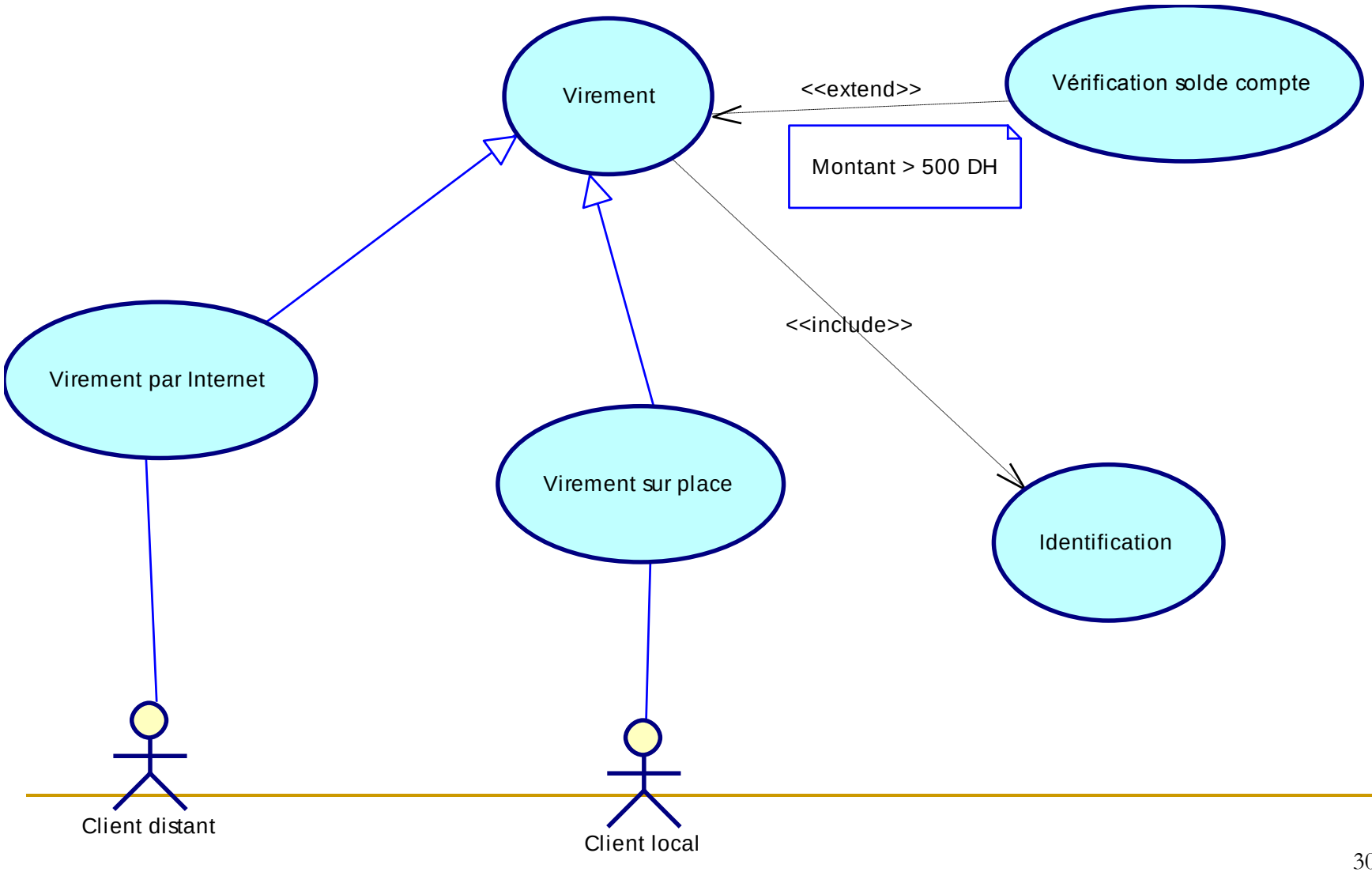
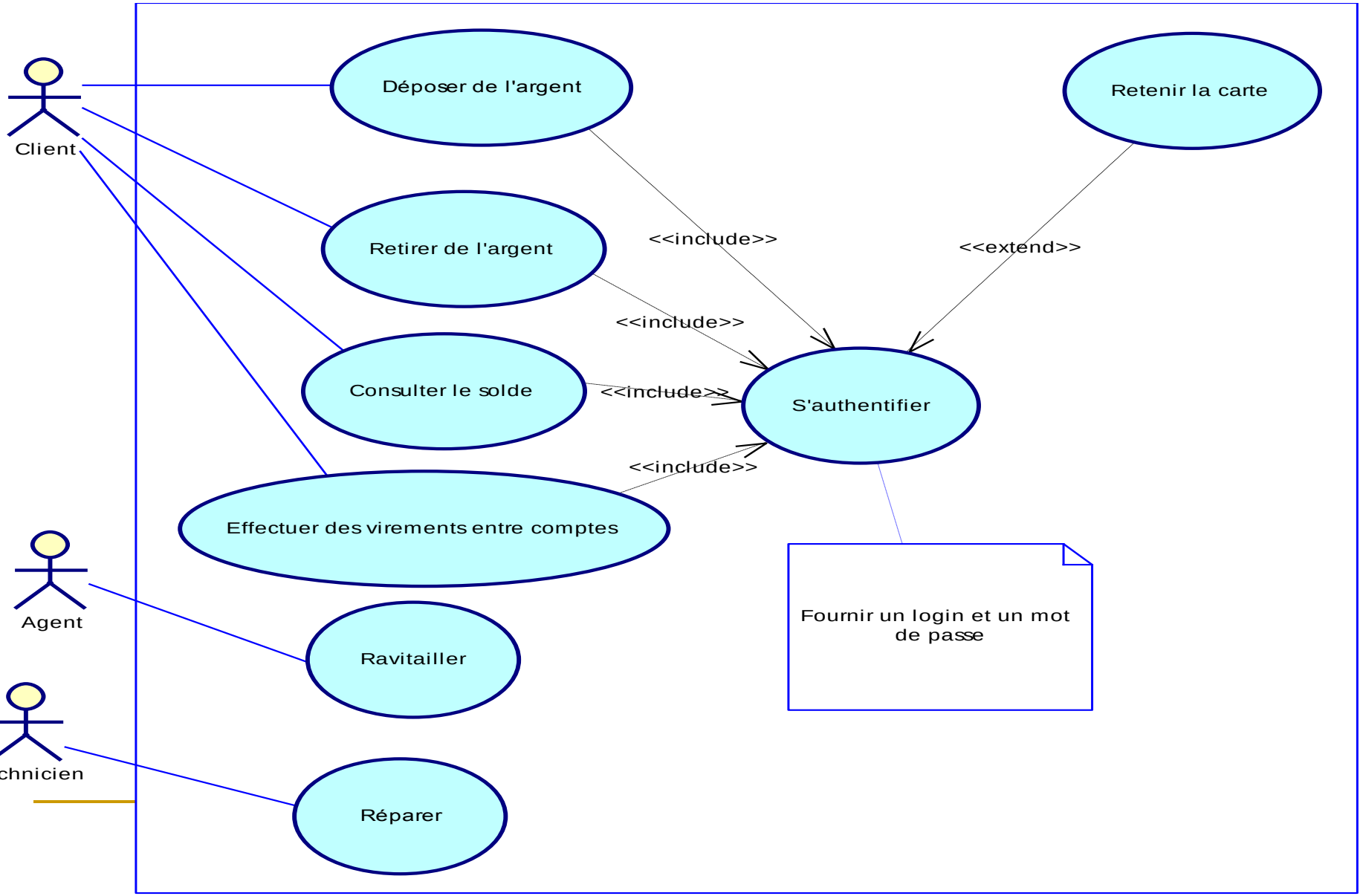


Diagramme des cas d'utilisation



UML

Diagrammes de classes et d'objets

Diagramme de classes

- Permet de donner une vue statique du système en terme de :
  - Classes d'objets
  - Relations entre classes
    - Associations
    - agrégation/composition
    - héritage
- La description du diagramme de classes est centrée sur trois concepts :
  - Le concept d’objets
  - Le concept de classes d’objets comprenant des attributs et des opérations
  - Les différents types de relations entre classes.

Concept d'objet

Objet = un concept, abstraction ou une chose autonome qui a un sens dans le contexte du système à modéliser

- une **personne** : le client « El Alami M. »
- un **objet concret** : le livre intitulé « Initiation à… »
- un **objet abstrait** : le compte bancaire n° 1915233C
- …

Concept d'objet

Remarque

- Un objet doit :
  - Être autonome
  - Avoir une signification dans le système
  - En relation avec d'autres objets
- Ne pas confondre "autonomie" avec "indépendance"!!
- Exemples
  - Gestion de stock : Clients, Commandes, Articles, …
  - Gestion scolaire : Étudiants, Modules, Filières, …

Concept d'attribut

- Un attribut est une propriété, caractéristique d'un objet. Par exemple :
  - un client a un **nom**, un **prénom**, une **adresse**, un **code client**, …
  - un compte bancaire a un **numéro**, un **solde**, …
- Un attribut doit (généralement) avoir une **valeur atomique**

Concept d'attribut

La description d'un attribut comporte :

Visibilité attribut:type[= valeur initiale]

Où :

- Visibilité :
  - + (publique, public) : visible par tous
  - - (privée, private) : visible seulement dans la classe
  - # (protégée, protected) : visible seulement dans la classe et dans les sous-classes de la classe.
- Nom d'attribut
- Type de l'attribut
- Valeur initiale (facultative)

Concept d'attribut

Le type d’un attribut peut être :

- Un type de base : entier, réel, …
- Une expression complexe : tableaux, enregistrements, …
- Une classe

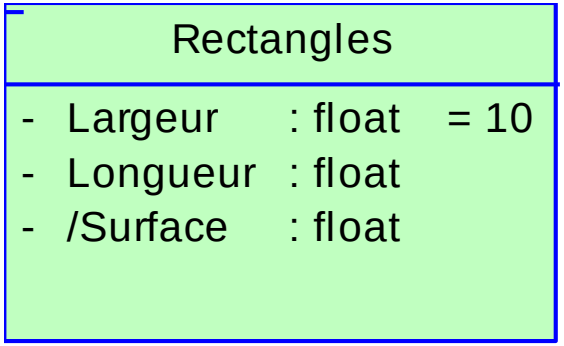
Exemples d’attributs :

- *- couleur : enum{Rouge, Vert, Bleu}*
- *# b : boolean = vrai*
- *- Client : Personne*

Concept d'attribut

Lorsqu'un attribut peut être dérivé ou calculé à partir d'autres attributs, il est précédé d'un **/**. Par exemple, une classe « Rectangle » peut contenir les attributs suivants :

- longueur : réel,
- largeur : réel,
- /surface : réel.

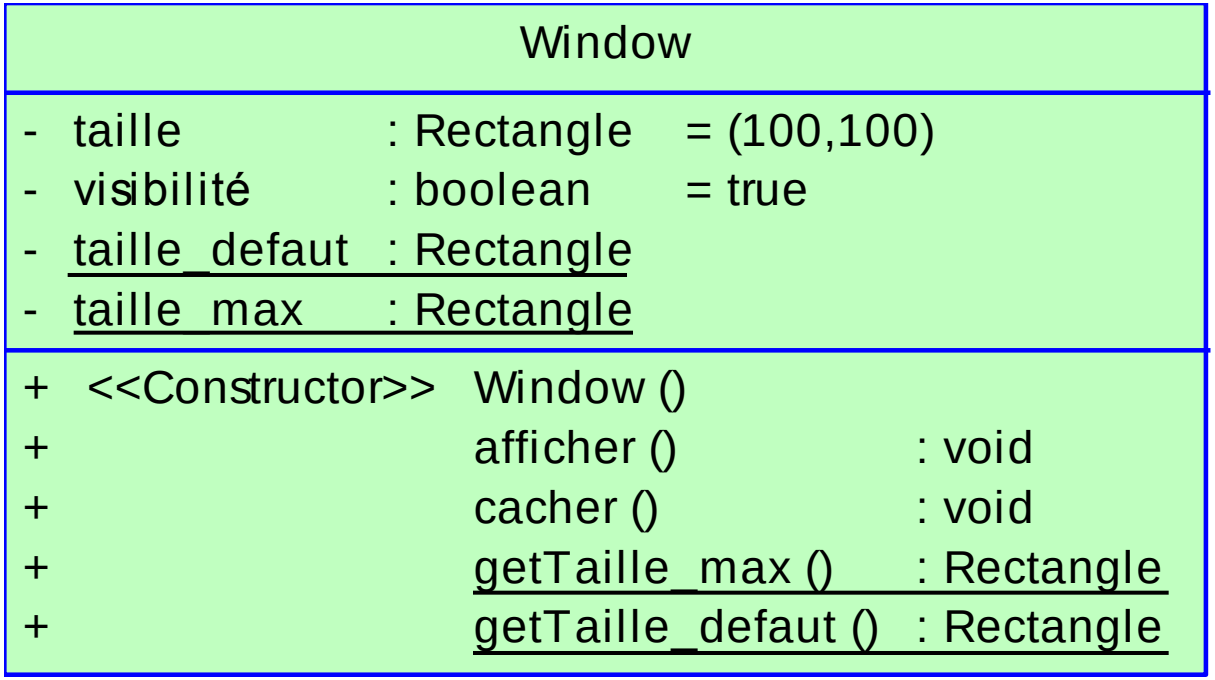


Concept d'attribut

On distingue deux types d'attributs :

- Attribut d'instance :
  - Chaque instance de la classe possède une valeur particulière pour cet attribut
  - Notation : **Visibilité attribut:type[= valeur initiale]**
- Attribut de classe
  - Toutes les instances de la classe possède la même valeur pour cet attribut
  - Notation : **Visibilité attribut:type[= valeur initiale]**
  - Équivalent en C++, Java : static

Concept d'attribut



Attributs d'instances

Attributs de classes

Opérations d'instances

Opérations de classes

Concept d'opération et méthode

Une opération est :

- un service offert par la classe
- une fonction ou une transformation qui peut être appliquée aux objets d'une classe.
- permet de décrire le comportement d'un objet. Par exemple, « Embaucher », « Licencier » et « Payer » sont des opérations de la classe « Société ».

Concept d'opération et méthode

Une méthode est

- l'implémentation d'un service offert par la classe (opération).
- de différents types :
  - accesseurs (get...): renvoie une information sur l'état d'un objet (fonction)
  - modifieurs (set...): modifie l'état de l'objet (procédure)
  - constructeurs: initialise une nouvelle instance

Concept d'opération et méthode

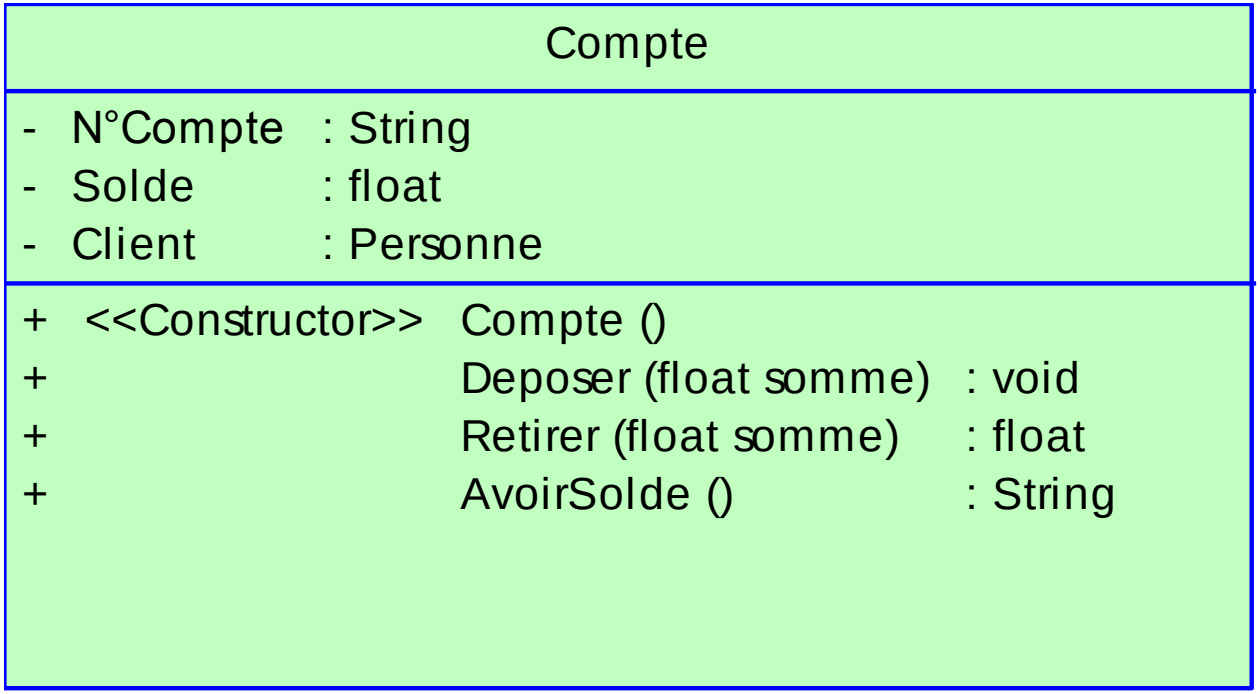
La description d'une opération comporte :

Visibilité opération([arguments:type[=valeur initiale]]):type de résultat

- Visibilité de l'opération (-, +, #)
- Nom de l'opération
- Liste des arguments avec leurs types et éventuellement leurs valeurs par défaut
- Le type du résultat retourné

Concept d'opération et méthode

Exemples d'opérations :



Concept de classes d'objets

- **Classe** = ensemble d'objets ayant les mêmes propriétés (attributs) et le même comportement (opérations)
  - tous les clients sont décrits par un **nom**, un **prénom**, … et peuvent **marcher**, **parler**,**courir**, …
  - tous les comptes bancaires ont un **numéro**, un **solde**, … et sur lesquels on peut **déposer** ou **retirer** l'argent, ou les **consulter**
  - …
- Un objet est **instance** d'une classe, et le fait de créer un objet d'une classe est dite instanciation.

Concept de classes d'objets

Classe représentée par un rectangle à trois parties :

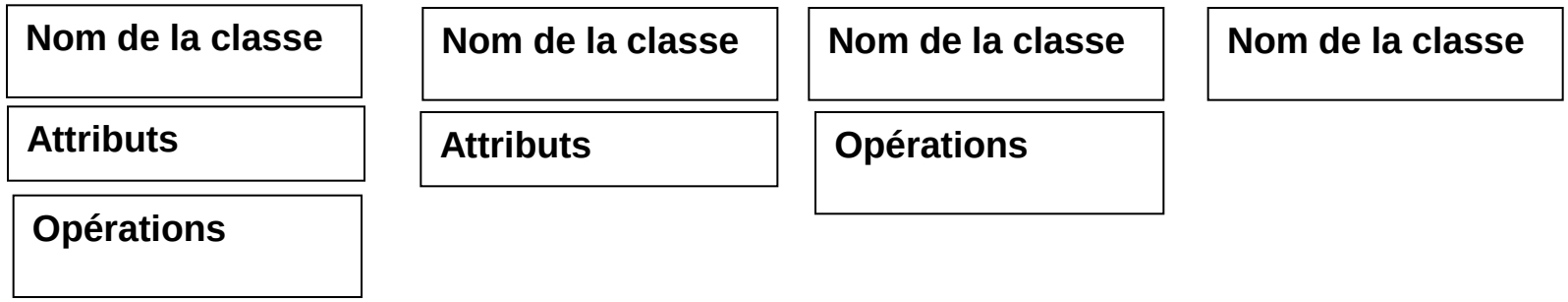
- Partie 1 : Nom de la classe
- Partie 2 : Attributs (propriétés, champs)
- Partie 3 : Méthodes (fonctions, opérations)

Concept de classes d'objets

Compte
- N°Compte : String - Solde : float = 100 # Client : Personne
+ <<Constructor>> Compte () + Deposer (float somme) : void + Retirer (float somme) : float + AvoirSolde () : String

Concept de classe d'objets

On peut ne pas visualiser les attributs et/ou les opérations, afin de ne pas alourdir inutilement le schéma.



Associations

- Relation existant entre une, deux ou plusieurs classes.
- Une association porte un nom (signification)
- Représentée par une ligne rectiligne

Associations

Remarques

- une association fonctionne (généralement) dans les 2 sens (bidirectionnelle)
- termes associés : Nom, Sens de lecture, degré (arité), Multiplicité, Rôle, navigabilité , …

Associations

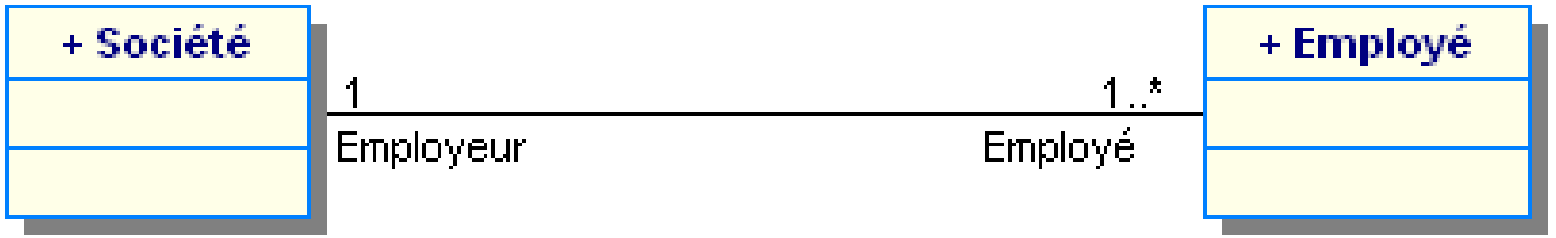
Nom et sens de lecture

- Décrit la nature (signification) de l'association
- Montre la direction de lecture de l'association

Associations

Rôle d'une association

Décrit le rôle d'une classe dans une association

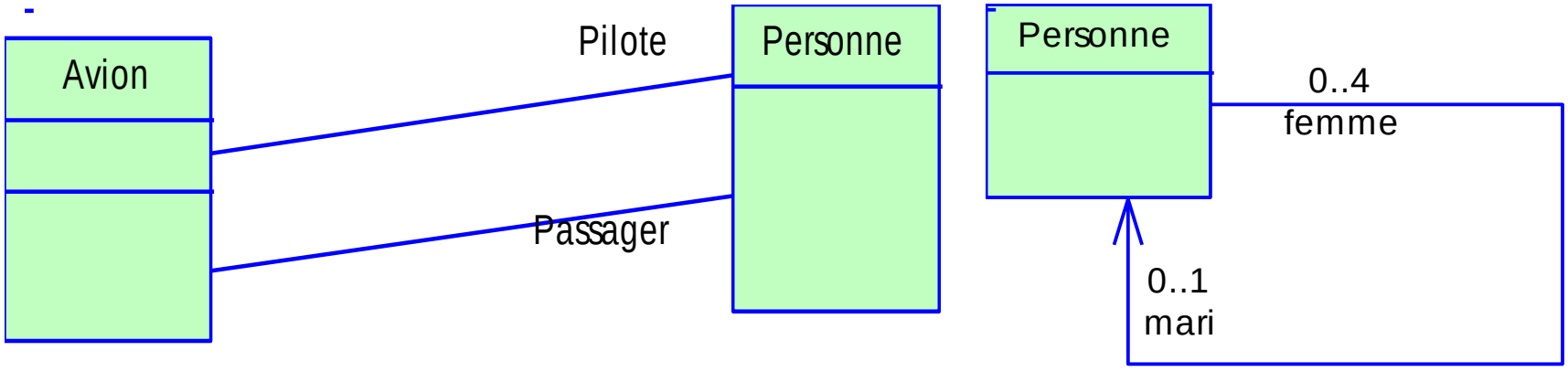


Associations

Rôle d'une association

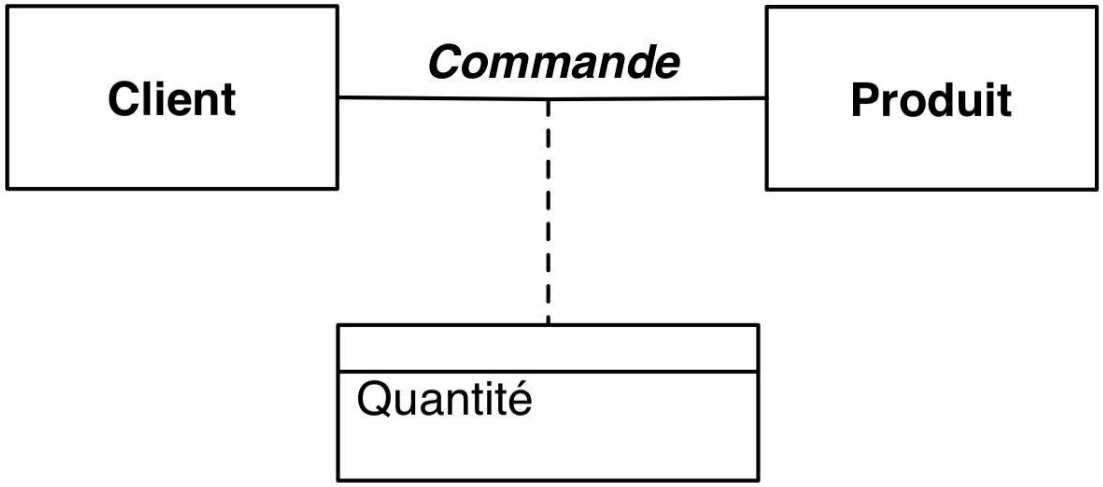
Utile surtout dans deux cas :

- Lorsqu'on a plusieurs associations entre deux classes avec des rôles différents
- une relation réflexive : relation entre deux instances d'une même classe



Associations
Classe association

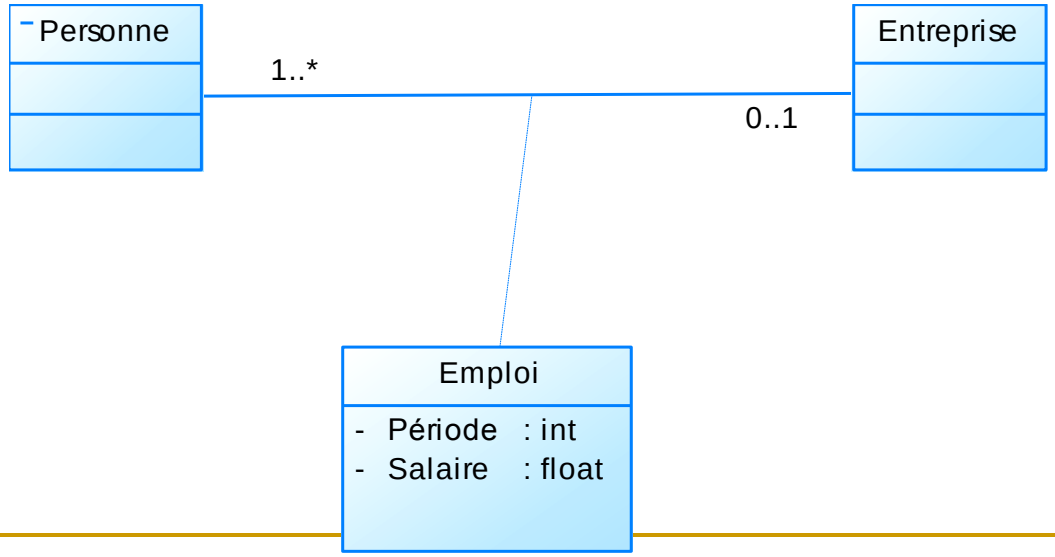
Une association peut avoir des attributs = **classe-association**



Associations

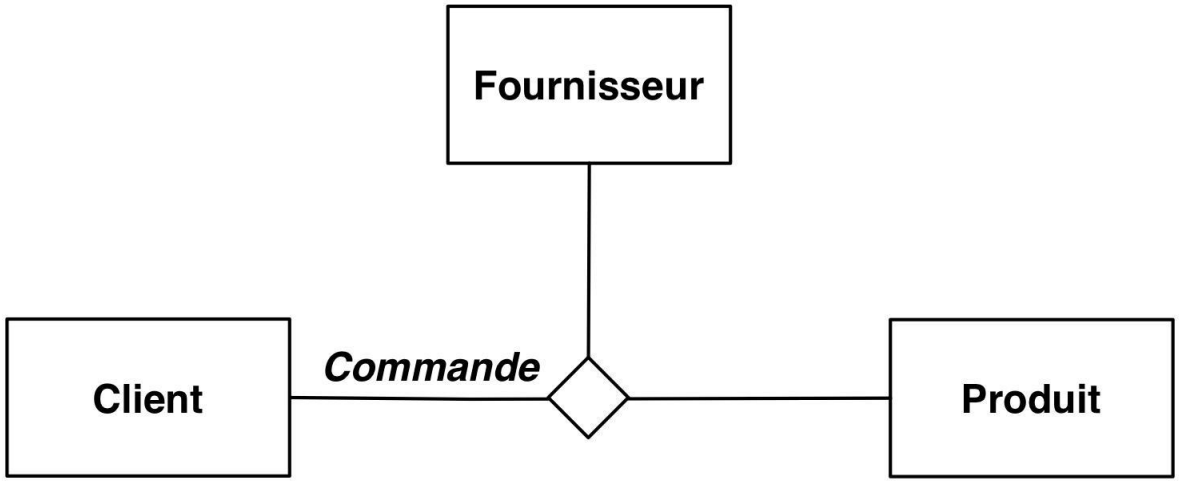
Classe association

- Les classes association sont utiles quand il y a des attributs qui sont pertinents à l'association, mais à aucune des classes impliquées.



Associations

- **degré d'une association** = nombre de classes participantes
  - Association **unaire** : relie 2 instances d'une classe
  - association **binaire** : relie 2 classes
  - association **ternaire** : relie 3 classes
  - association **n-aire** : relie n classes

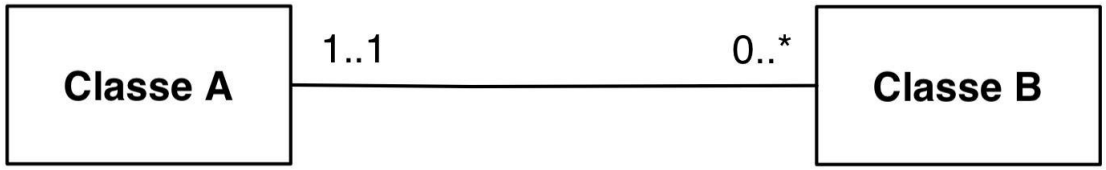


Associations

M**ultiplicité** = nombre de participations d'une classe dans une association

- indiquée **à chaque extrémité** d'une association
- sous la forme **min..max**
- min, max = **0, 1, ***

Exemple général



Exemple concret

Associations

Notation abrégée des multiplicités **:**

1 ⇔ 1..1 (exactement 1)

***** ⇔ 0..* (0 ou plusieurs)

n ⇔ n .. n (exactement n)

1..* ⇔ 1 ou plusieurs (1 ou plus)

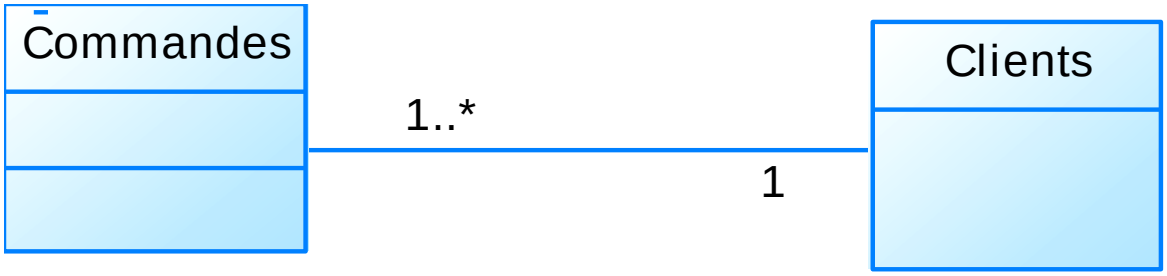
0..1 ⇔ 0 ou 1 (au plus un)

1..100 ⇔ entre 1 et 100

2,4,5 ⇔ 2, 4 ou 5

Association
Navigabilité

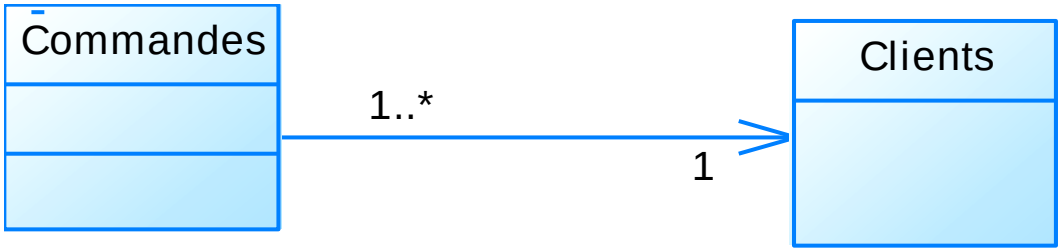
- Une association est par défaut bidirectionnelle.



- Cependant, il peut être utile de se limiter à une seule direction ➔ association navigable

Association
Navigabilité (Exemple)

- Spécification : on doit être en mesure de savoir le client qui a fait la commande et non toutes les commandes d'un client
- Conception :



- Implémentation : la classe commande doit avoir un champ faisant référence à la classe client

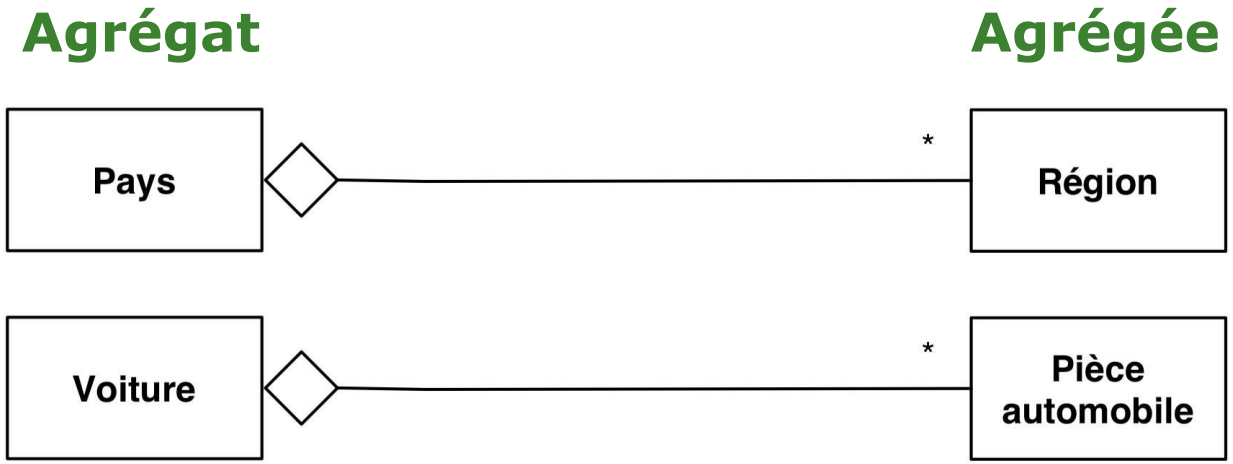
Association
Navigabilité (Exercice)

Un étudiant peut avoir jusqu'à 5 copies d'examens. À un étudiant sont associées ses copies d'examens, mais on ne doit pas autoriser l'accès à l'auteur de la copie (notamment avant la correction des copies)

Agrégation

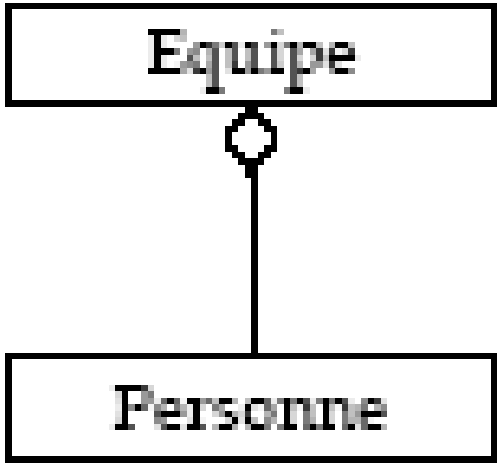
Type particulier d'association dans laquelle :

- Classe agrégat (composé), classes agrégée (composant)
- Entre les deux, il existe une relation de type « est composé de »

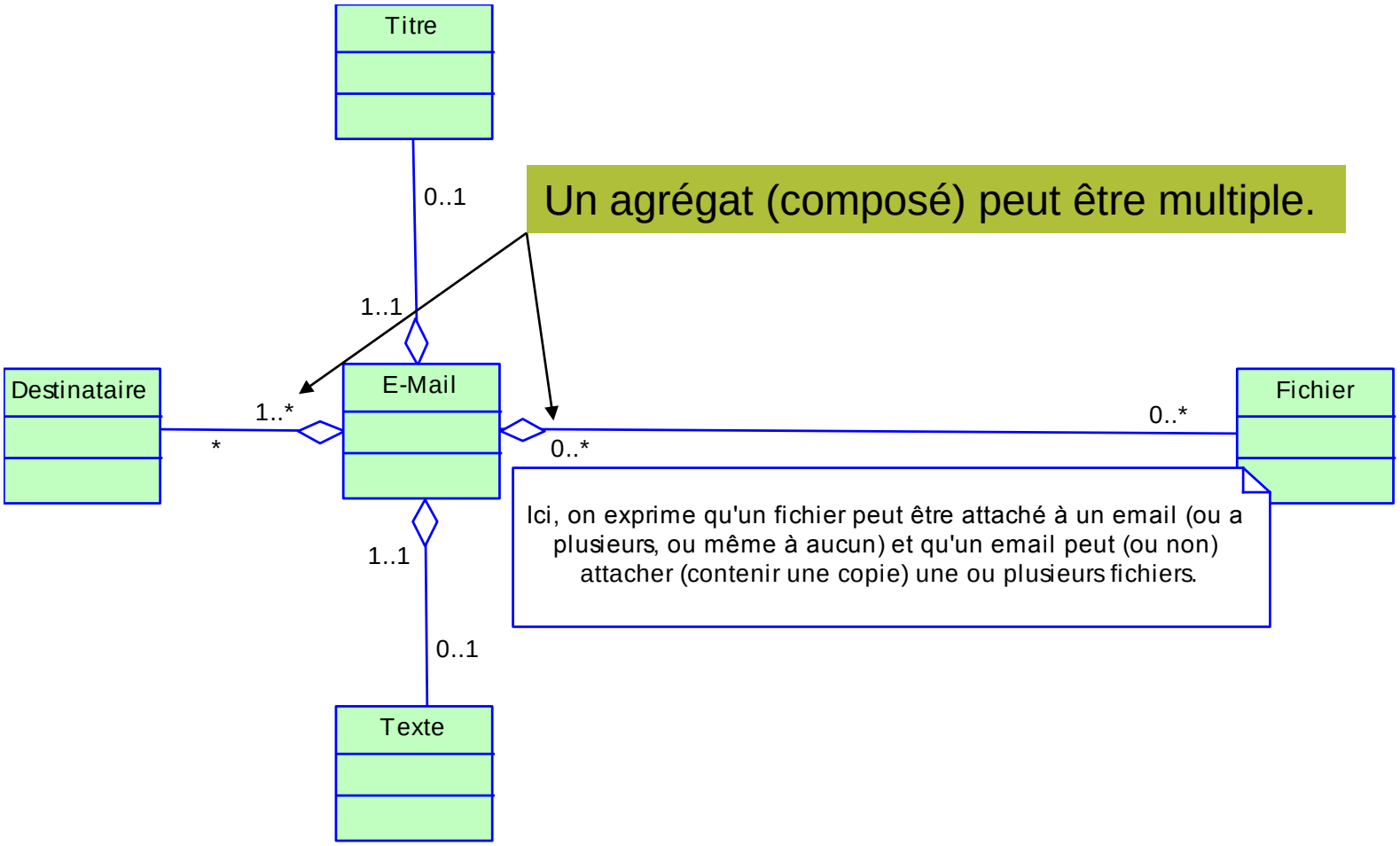


Agrégation

- Les parties (les composants) sont séparables de L'agrégat (le tout)
- La suppression d'une équipe n'implique pas la suppression des personnes qui la composent



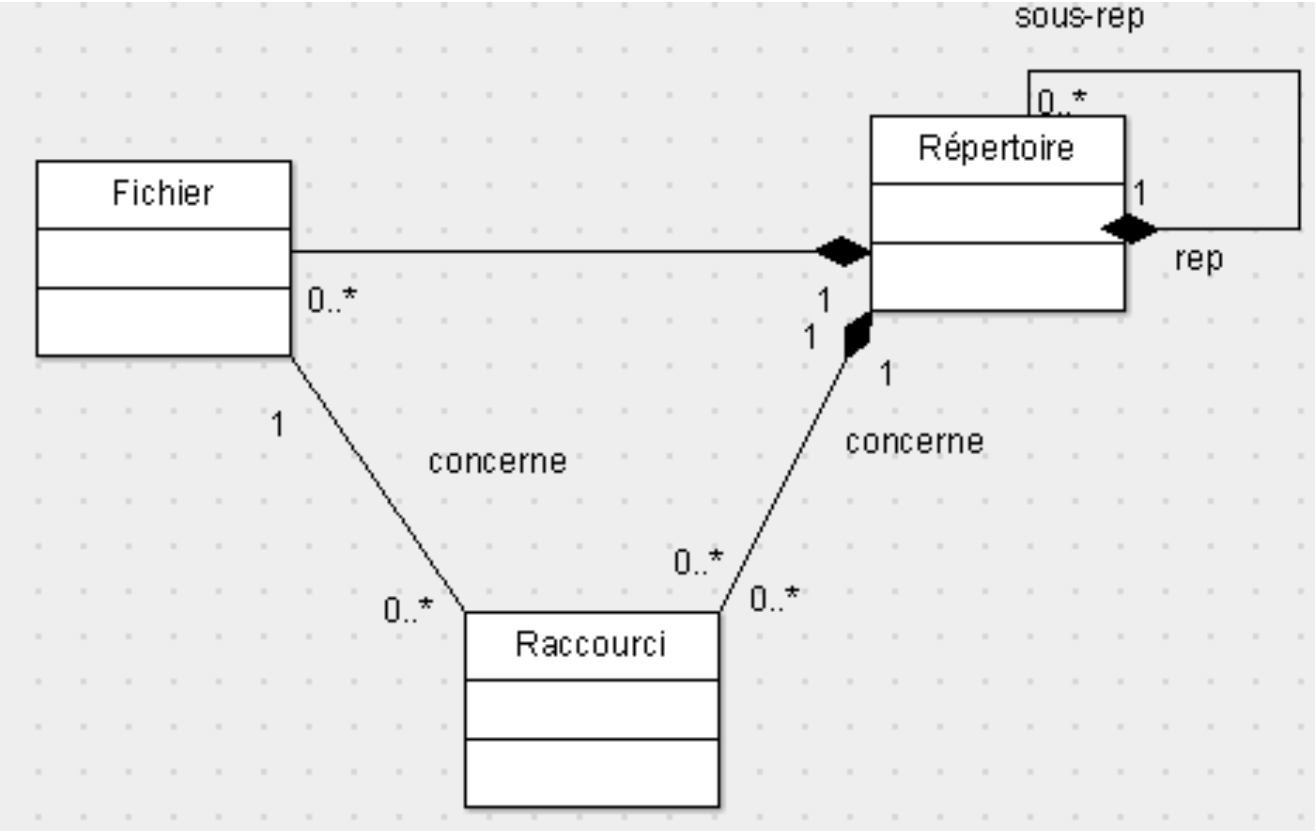
Agrégation


Titre
0..1
Un agrégat (composé) peut être multiple.
1..1
Destinataire
E-Mail
Fichier
1..*
*
0..*
0..*
Ici, on exprime qu'un fichier peut être attaché à un email (ou a plusieurs, ou même à aucun) et qu'un email peut (ou non) attacher (contenir une copie) une ou plusieurs fichiers.
1..1
0..1
Texte

Composition

- La composition est un cas particulier d'une agrégation dans laquelle la vie des composants (élément) est liée à celle de l'agrégat (composé) : si l'agrégat est détruit (ou déplacé), ses composants le sont aussi.
- D'un autre côté, et contrairement à l'agrégation, une instance de composant ne peut être liée qu'a un seul agrégat.
- La composition se représente par un losange noir (plein).

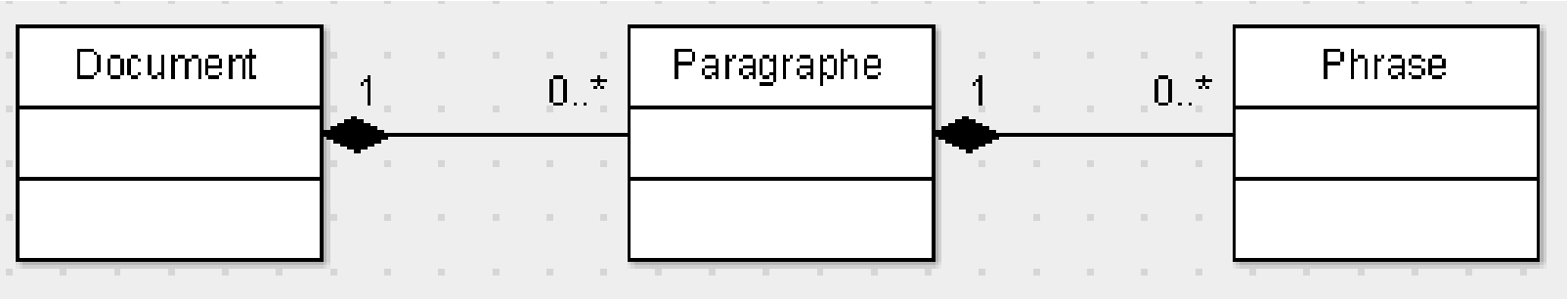
Composition



- la suppression d'un objet agrégat entraîne la suppression des objets agrégés

Composition

- Un document est composé de plusieurs paragraphes, qui, à son tour, est composé de plusieurs phrases
- Remarquer la propagation des opérations (copie, suppression,…)

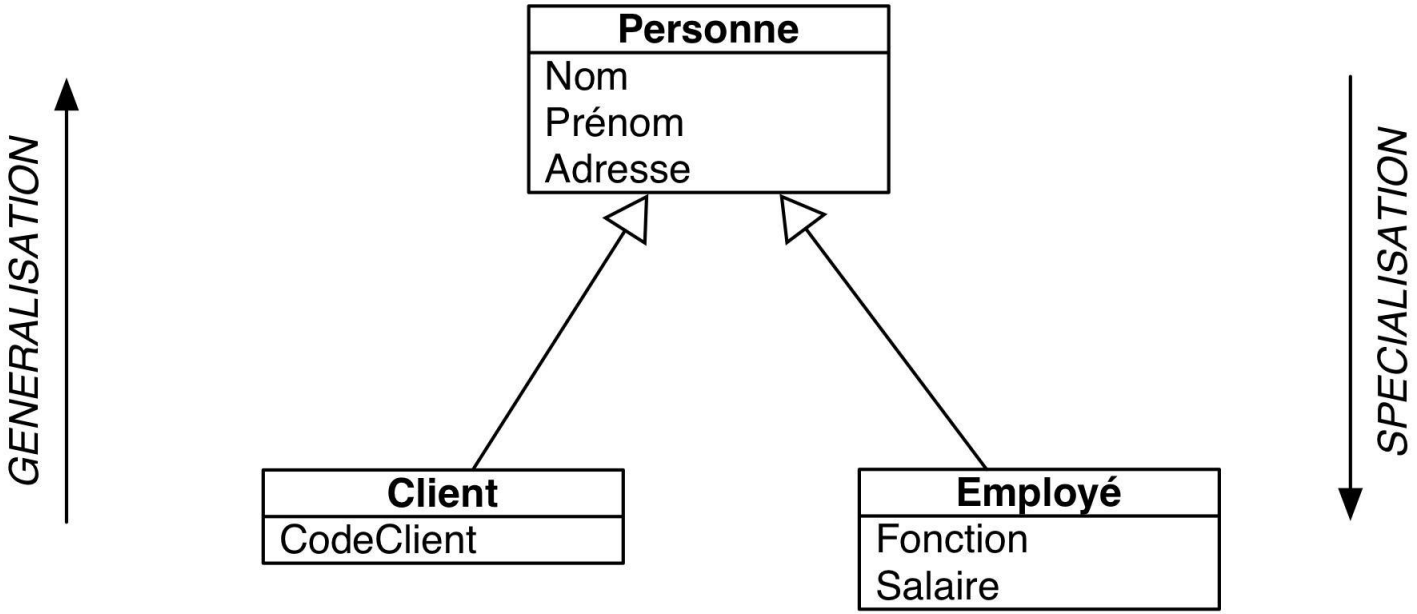


Généralisation / Spécialisation et héritage

- La généralisation est la relation entre une classe et une ou plusieurs de ses versions raffinées.
- On appelle la classe dont on tire les précisions la super-classe et les autres classes les sous-classes.
- C'est une relation de type « est un (is a) » ou « est une sorte de ».
- La notation utilisée pour la généralisation est le triangle

Généralisation / Spécialisation et héritage

- **généraliser** = mettre en facteur des classes → « super-classe »
- **spécialiser** = décrire de nouveaux détails → « sous-classes »



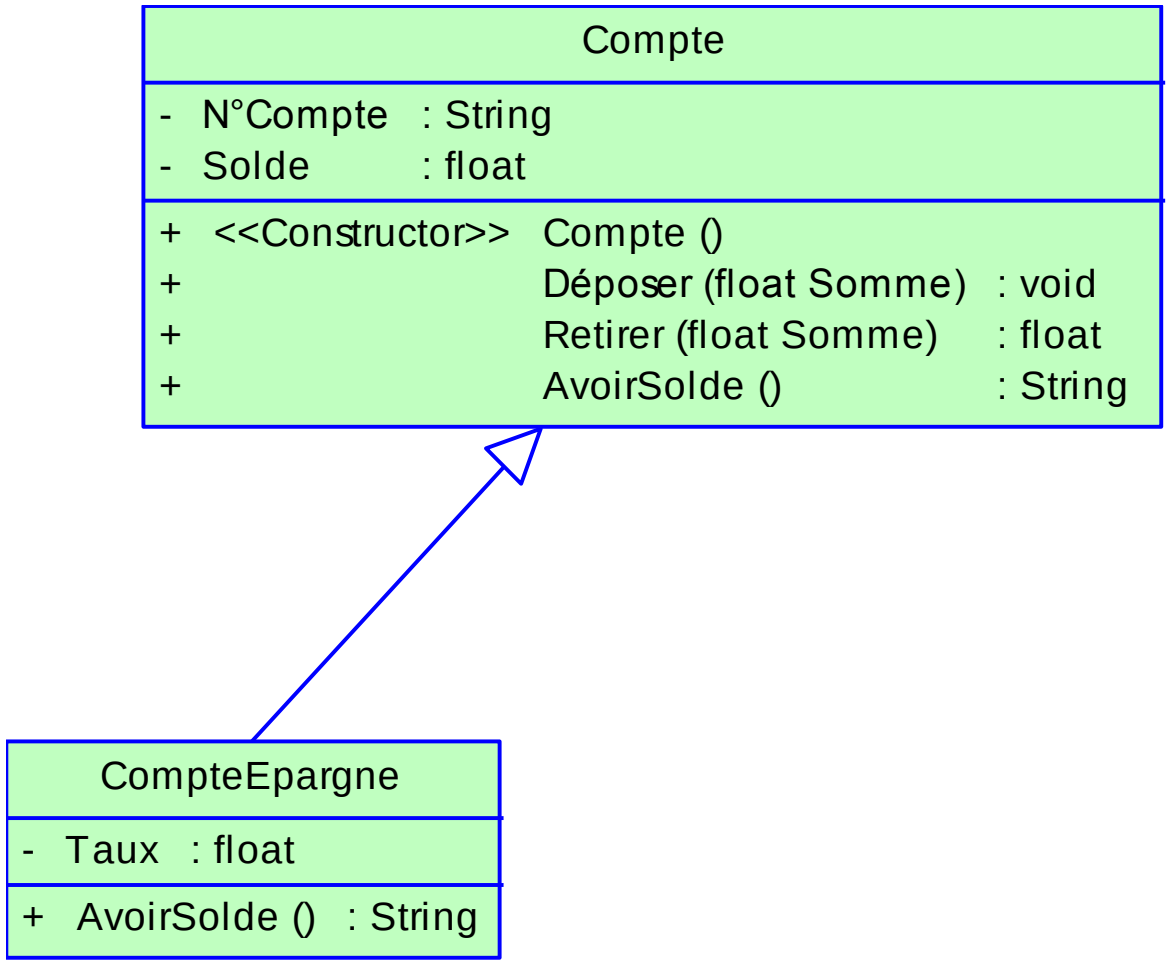
- comparable à une association de type « est un, is a, kind of »
- une sous-classe **hérite** des attributs et opérations de sa super-classe (classe mère)

Généralisation / Spécialisation et héritage

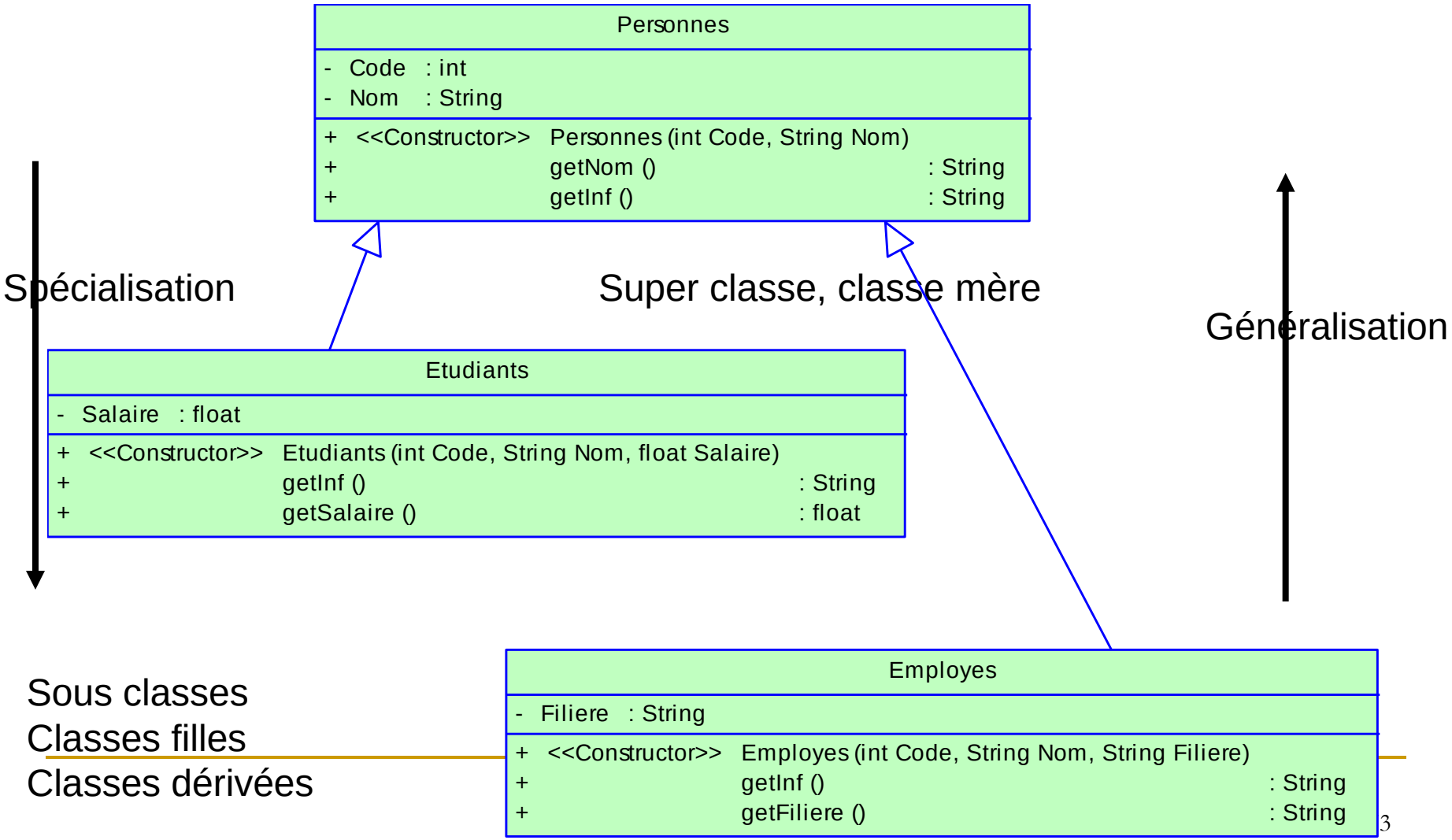
La classe spécialisée (sous-classe)

- hérite les méthodes et les attributs de la classe générale (super-classe)
- peut ajouter ses propres attributs et méthodes.
- peut redéfinir le comportement d'une méthode.

Généralisation / Spécialisation et héritage



Généralisation / Spécialisation et héritage

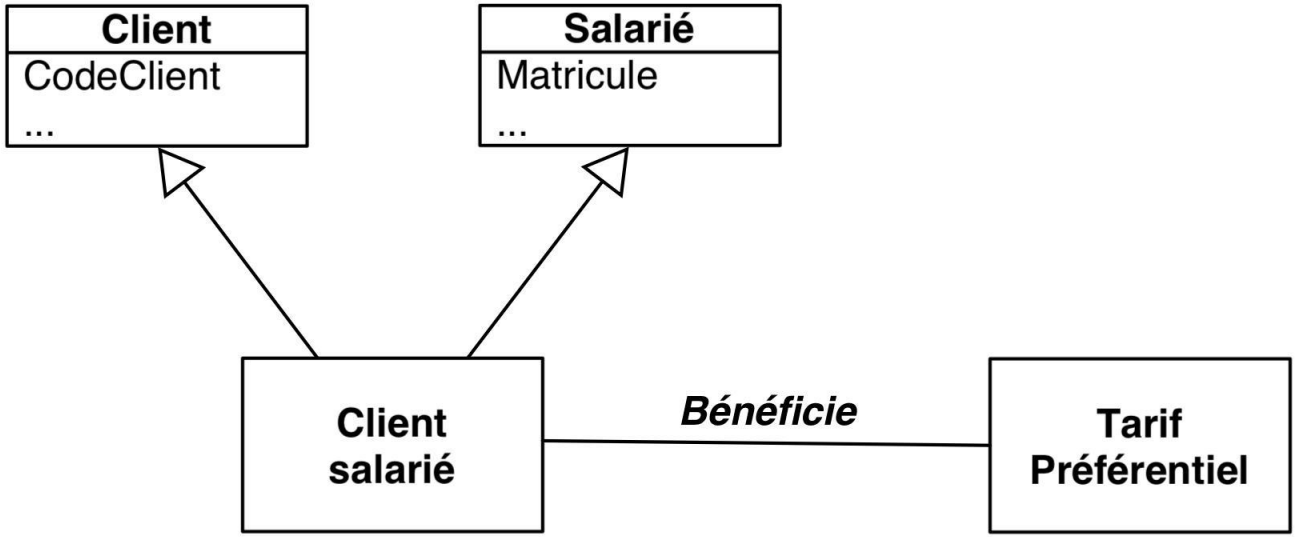


Sous classes
Classes filles
Classes dérivées

Généralisation / Spécialisation

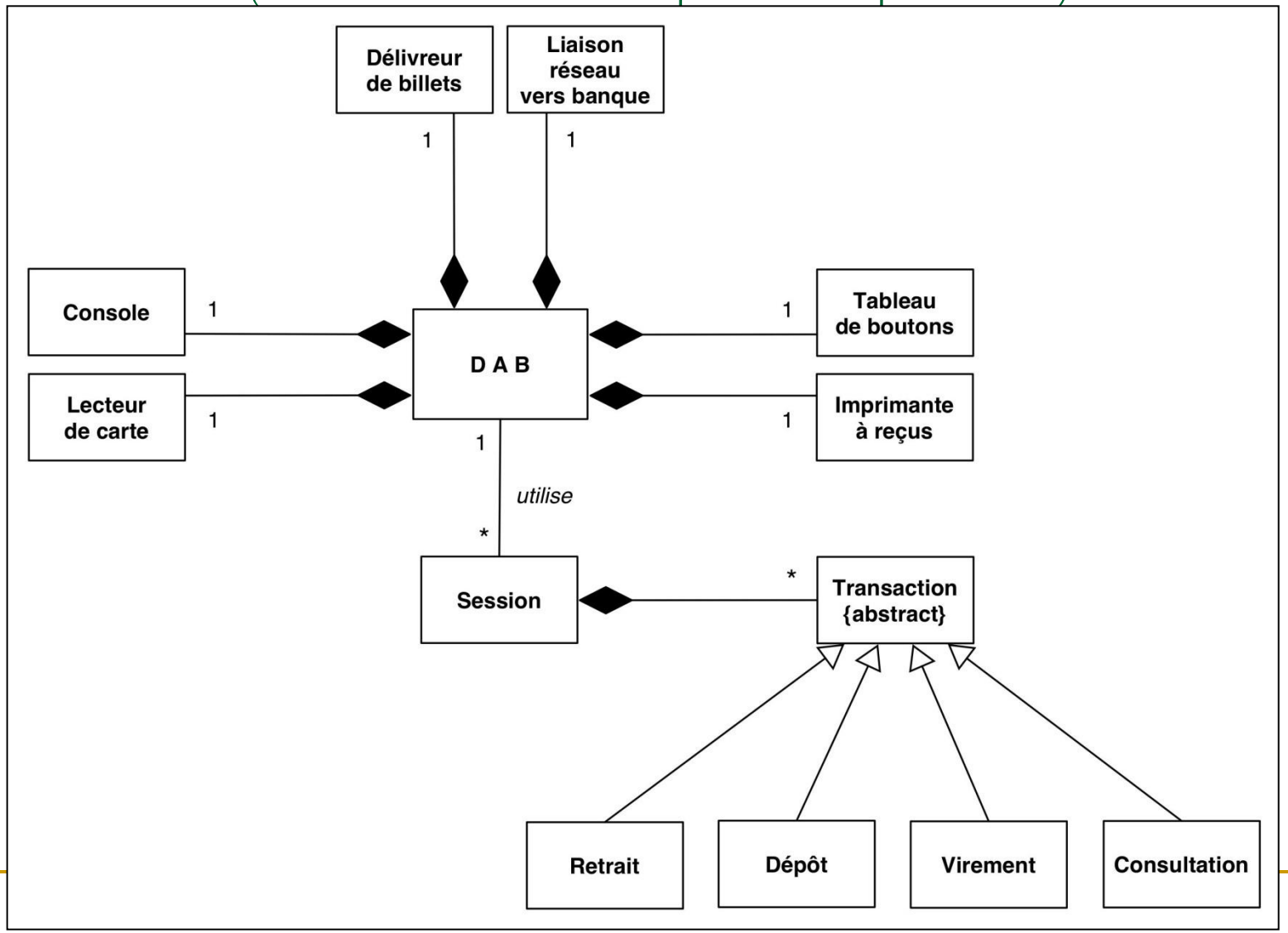
- une classe peut hériter de plusieurs super-classes

= **héritage multiple**



Exemple de diagramme de classes

(Distributeur Automatique de Banque : DAB)



UML

Diagrammes de séquences

Diagramme de séquences

- Représenter les interactions entre objets en précisant la chronologie des échanges de messages
- Représente une instance d'un cas d'utilisation (les scénarios possible d'un cas d'utilisation donné)
- Montre sous forme de scénarios, la chronologie des envoies de messages issus d'un cas d'utilisation
- Le diagramme de séquence fait ressortir :
  - Les acteurs
  - Les objets
  - Les messages

Diagramme de séquences

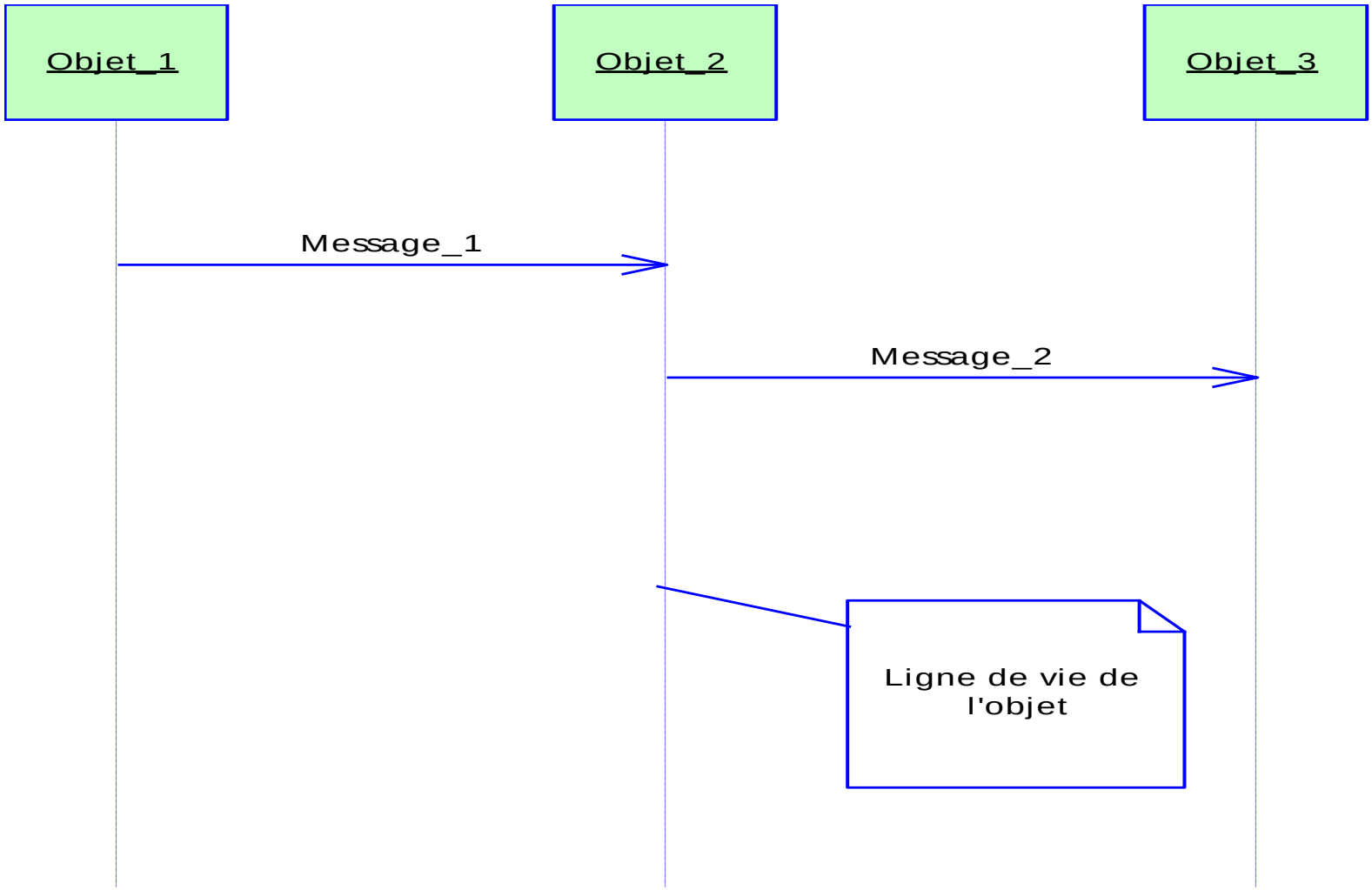


Diagramme de séquences

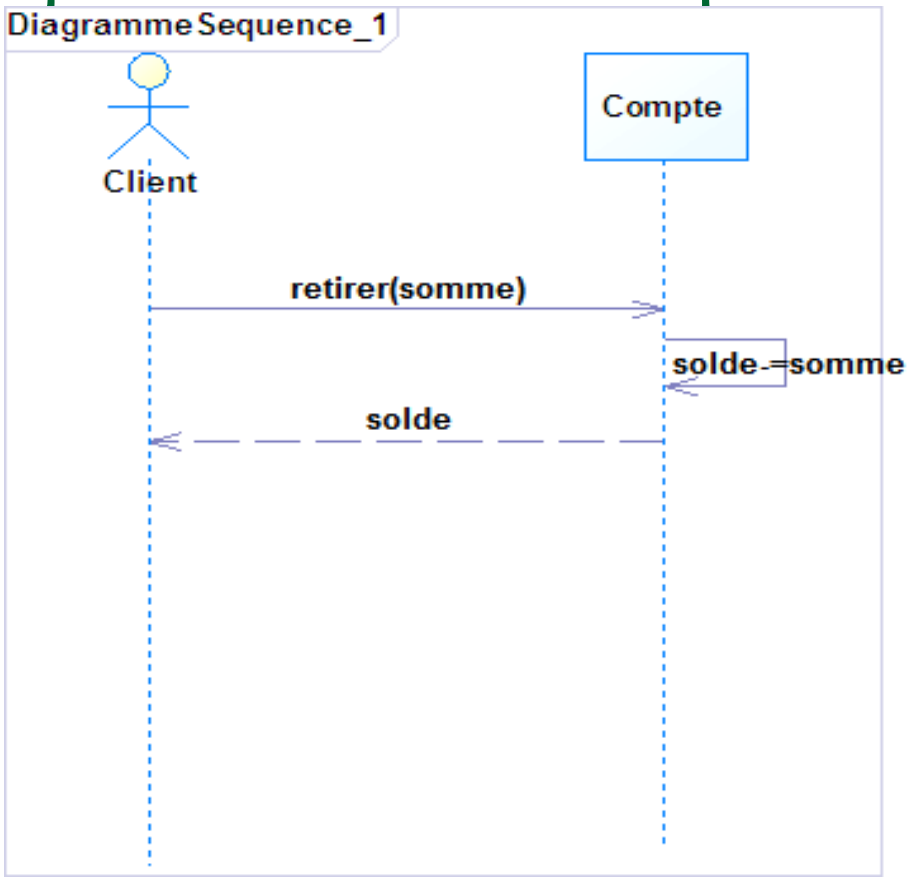
- Un objet est représenté par un rectangle et une ligne verticale (ligne de vie de l'objet)
- Les objets communiquent en échangeant des messages représentés par des flèches orientées de l'émetteur au récepteur
- L'ordonnancement verticale des messages indique la chronologie

Diagramme de séquences

- Un message reçu par un objet déclenche l'exécution d'un opération
- Un message envoyé par objet correspond :
  - Demander un service d'un autre objet
  - Renvoyer le résultat d'un opération

Diagramme de séquences : Exemple

Compte	
- N°Compte	: String
- Solde	: float
+ <<Constructor>>	Compte (int n, float s)
+	déposer (float somme) : void
+	retirer (float somme) : float
+	consulter () : float



- Le client demande un service (déposer de l'argent) à l'objet Compte
- Le compte reçoit le message et déclenche l'opération de même nom
- Le compte retourne le résultat (le solde actuel)

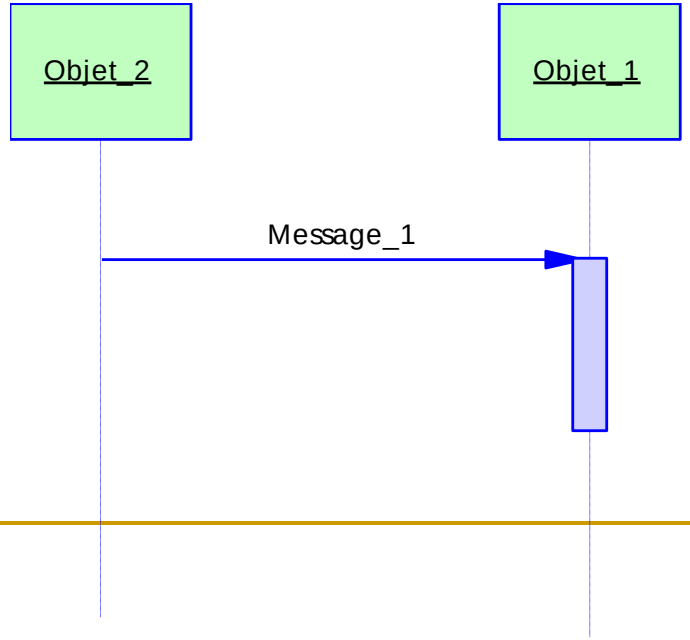
Diagramme de séquences

Plusieurs concepts additionnels :

- Période d'activité
- Types de messages
- Création et destruction d'objets
- Structures de contrôles

Période d'activité

- Correspond au temps pendant lequel un objet fait une action
- Représentée par une bande rectangulaire superposée à la ligne de vie de l'objet

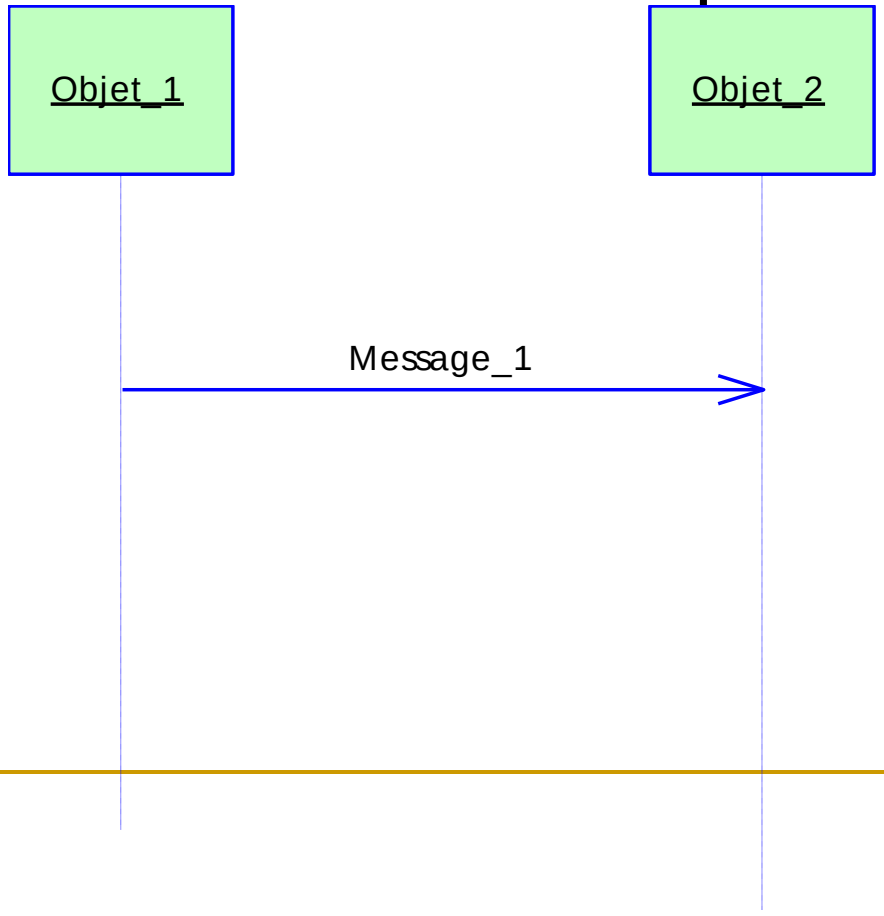


Messages

- Traduisent les interactions (échange d'informations) entre objets
- Représentés par des flèches orientées de l'émetteur au récepteur
- Plusieurs types :
  - Message simple
  - Message minuté (Timeout)
  - Message synchrone
  - Message asynchrone
  - Message récursif

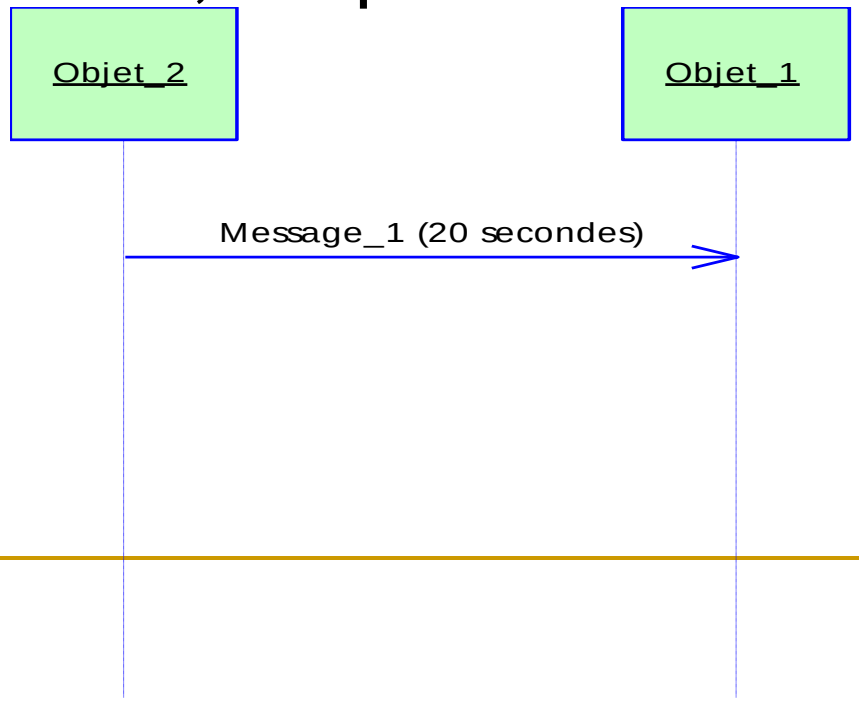
Message simple

Message pour lequel on ne spécifie aucune information d'envoi ou de réception



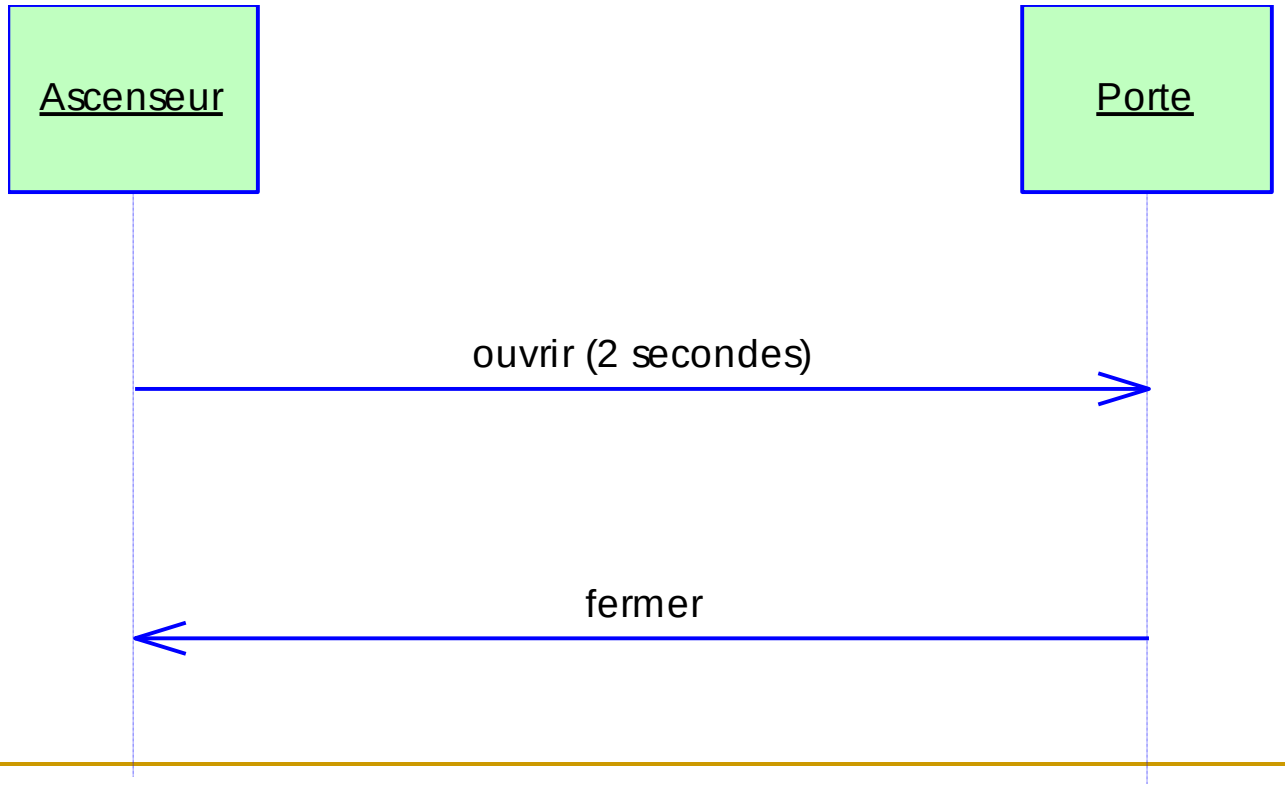
Message minuté (Timeout)

- Bloque l'expéditeur pendant un temps donné, en attendant la prise en compte du message par le récepteur
- Après le délai, l'expéditeur est libéré et peut envoyer



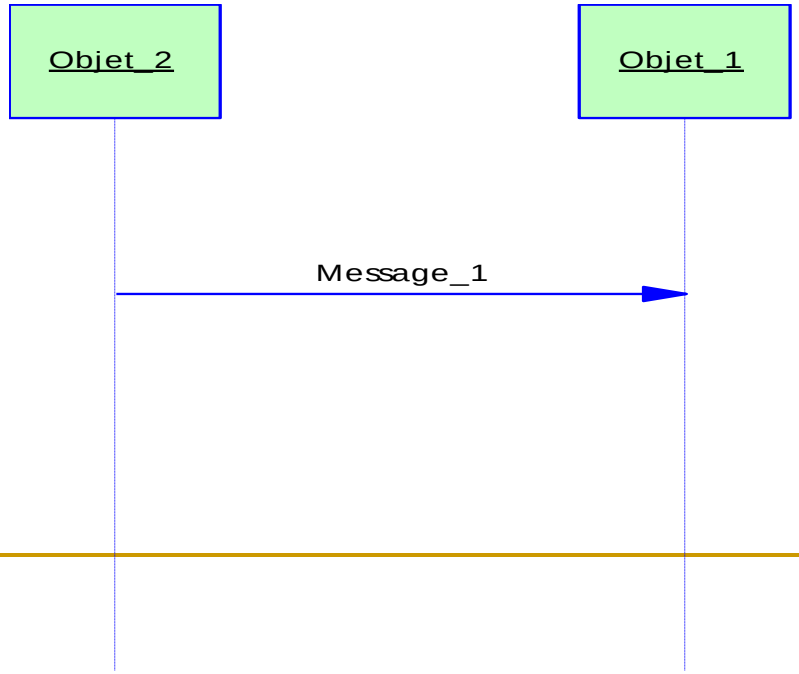
Message minuté (Timeout) : Exemple

La porte d'un ascenseur s'ouvre pendant un certain délai avant d'être refermée.



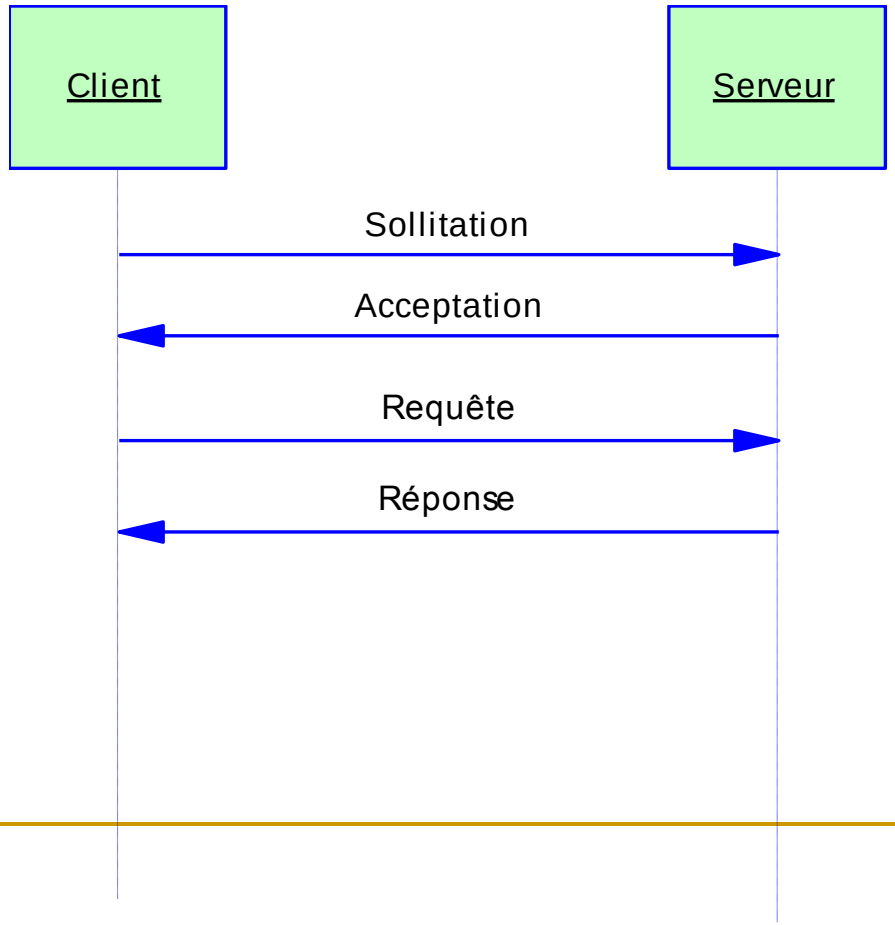
Message synchrone (appel de procédure)

- Bloque l'expéditeur jusqu'à la prise en compte du message par le récepteur
- Le contrôle est passé de l'émetteur au récepteur qui devient à son tour émetteur (actif)



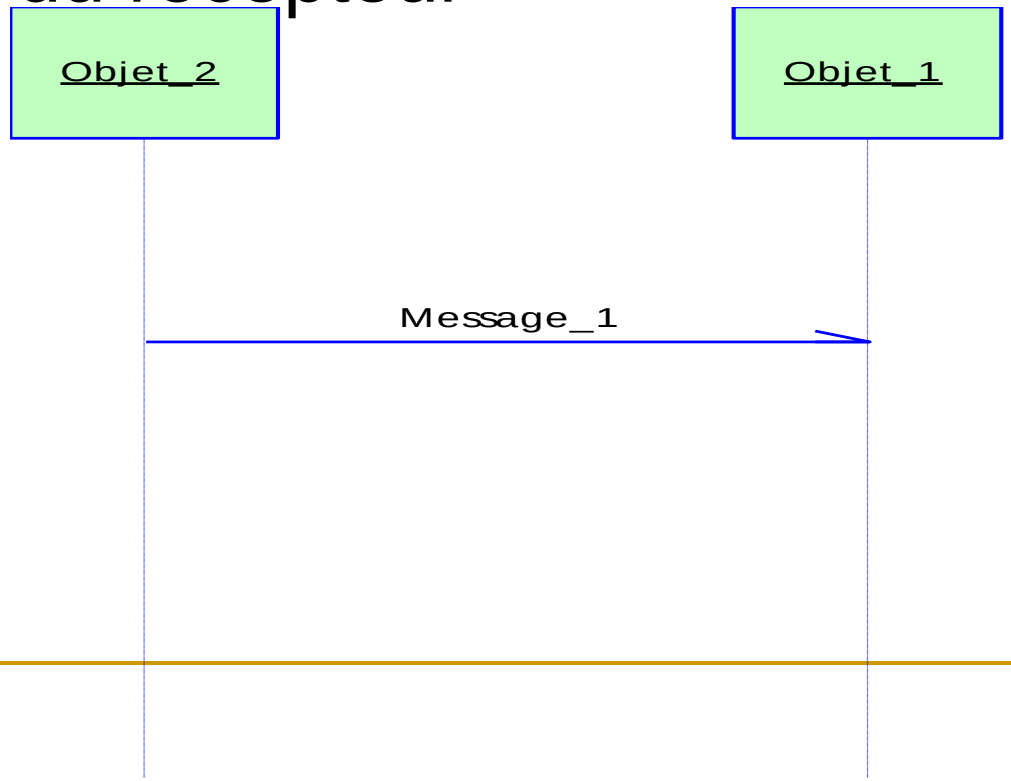
Message synchrone (appel de procédure) : Exemple

Communication client serveur : Sockets



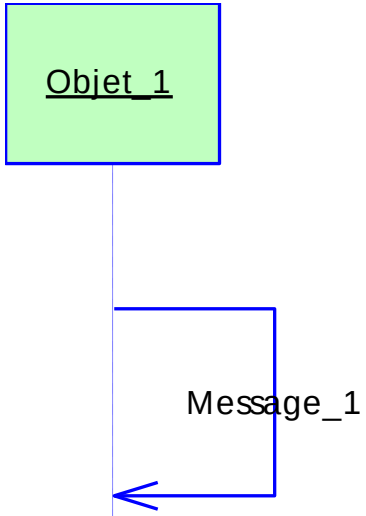
Message asynchrone

- N'interrompt pas l'exécution de l'expéditeur
- L'expéditeur peut émettre sans attendre la réponse du récepteur



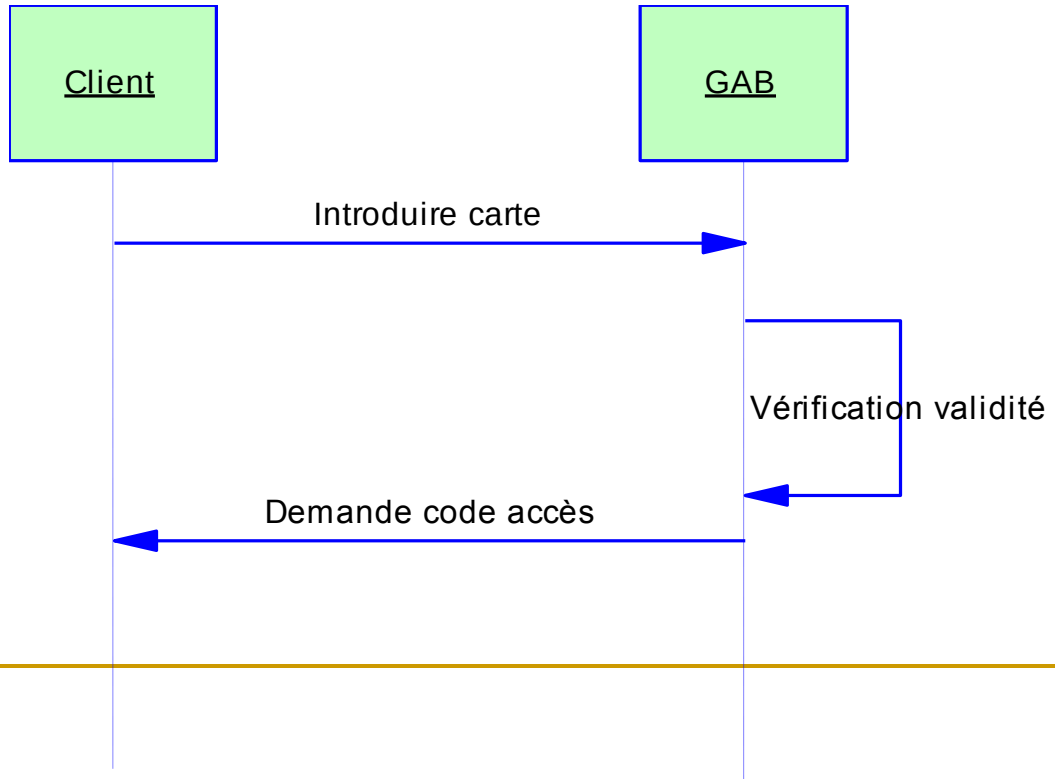
Message récursif

- Appelé aussi message réflexive
- Message envoyé d'un objet vers lui-même.



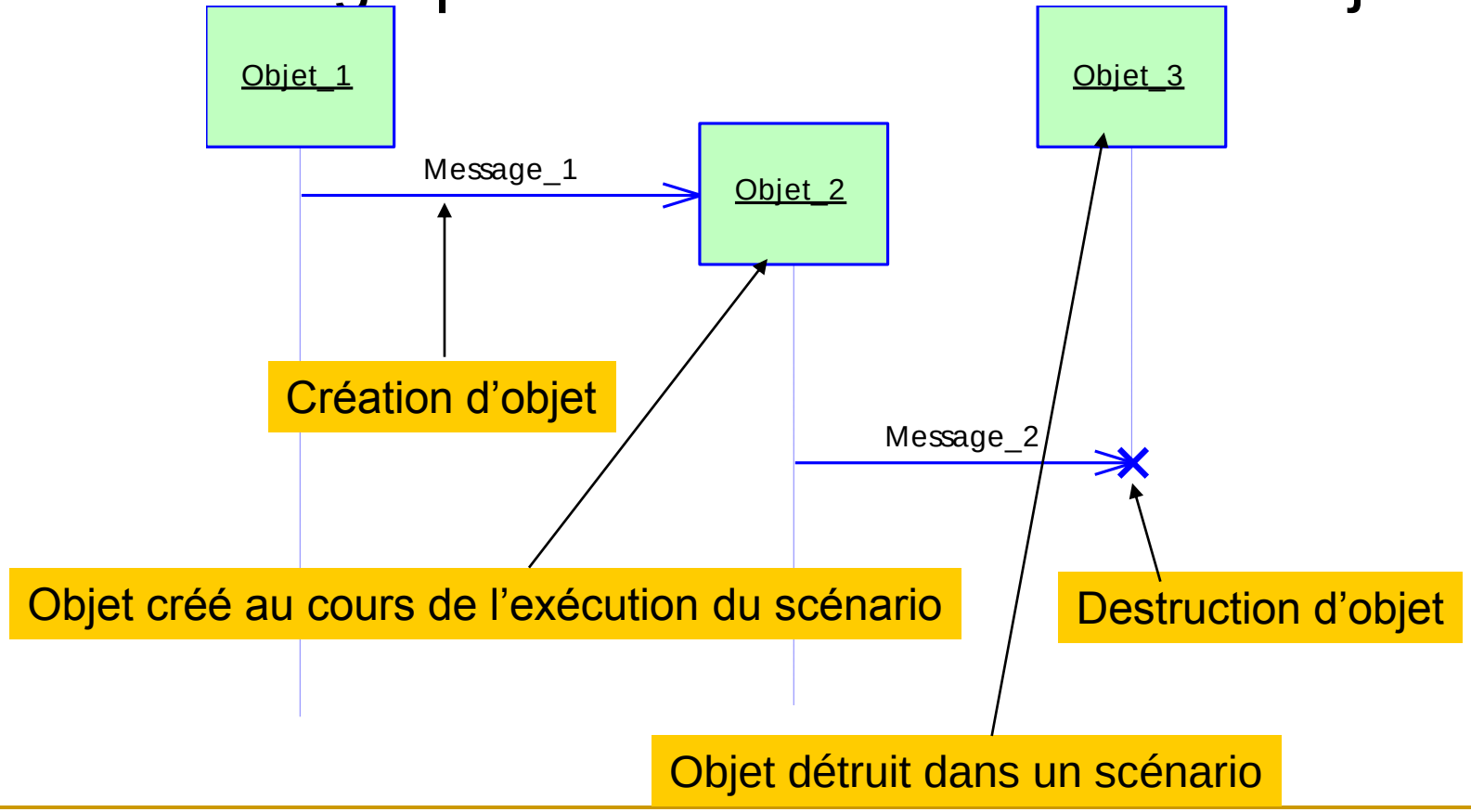
Message récursif : Exemple

Lorsque le client introduit sa carte de guichet, ce dernier vérifie la validité de la carte avant de demander le code d'accès



Création et destruction d'objets

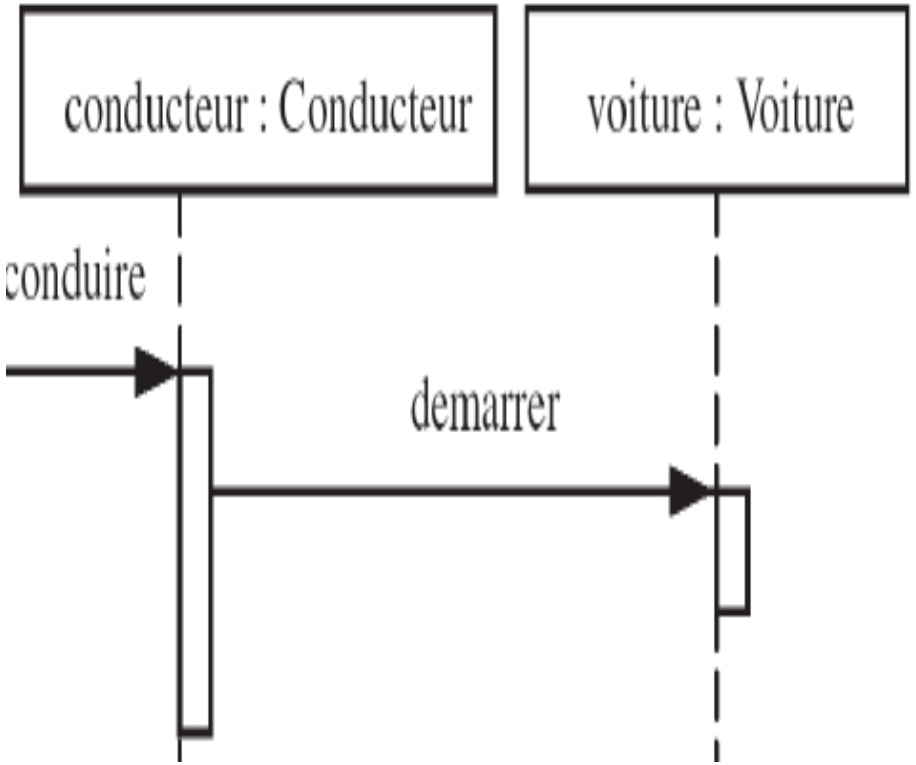
Un message peut créer ou détruire un objet



Traduction des messages

- Envoyer un message c'est demander un service d'un autre objet (sauf le cas d'un message de retour).
- Les messages sont traduits par des opérations dans la classe de l'objet ayant reçu le message

Traduction des messages




```
class Voiture{
    Public void demarrer(){}
}
class Conducteur{
    private Voiture voiture;
    public void conduire(){
        voiture.demarrer();
    }
}
… main(String[] arg){
    conducteur.conduire();
}
```

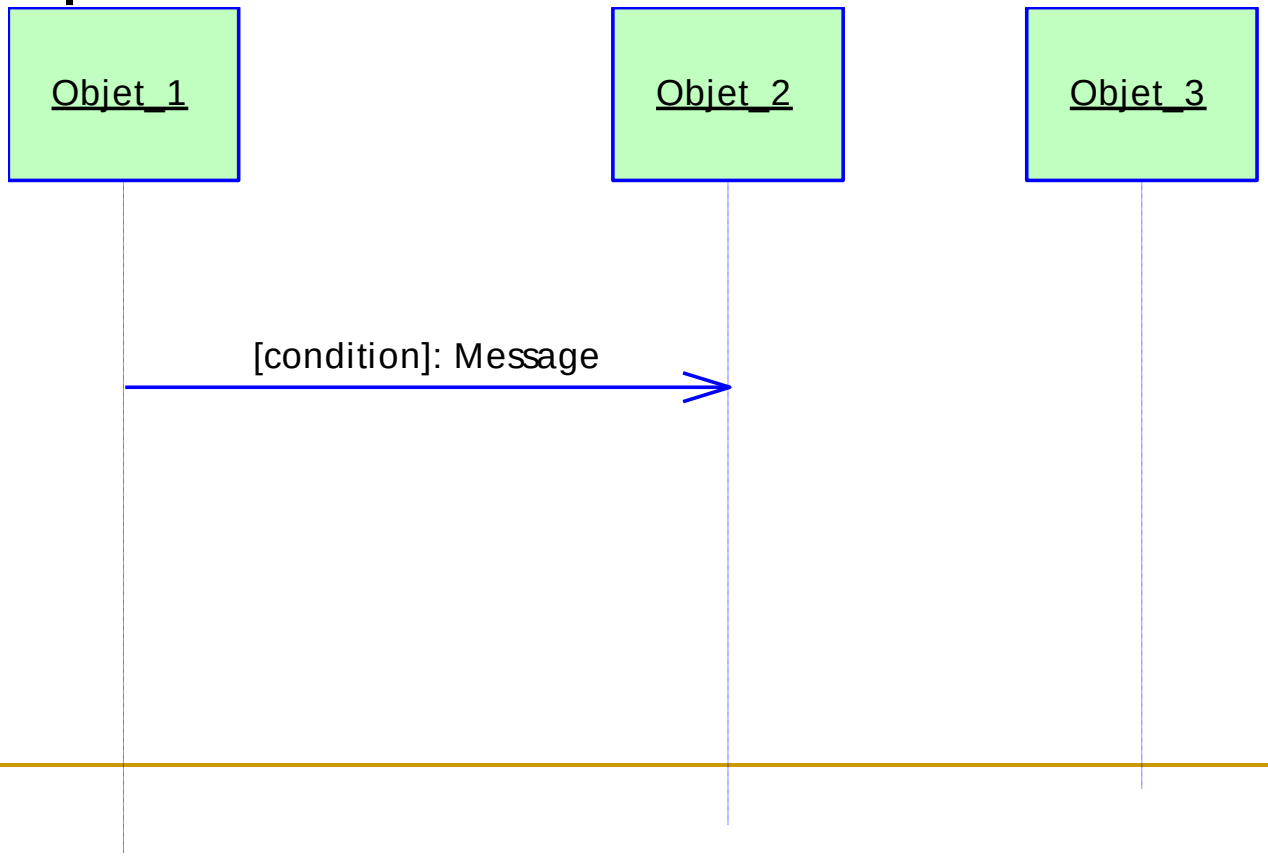
Structures de contrôle

Le diagramme de séquences peut inclure un certain nombre de structures

- Branchements (tests)
- Répétitions (itérations, boucles)

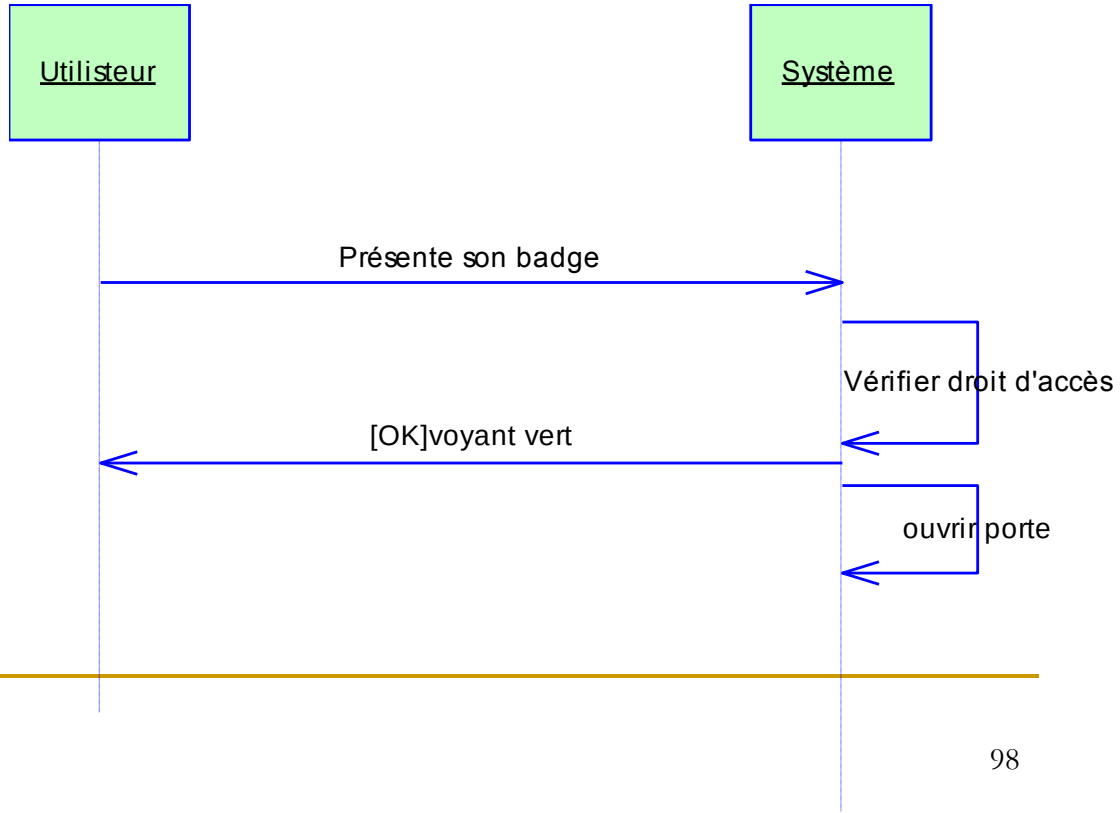
Les test (branchements)

La condition précédée le message et elle est délimitée par des crochets



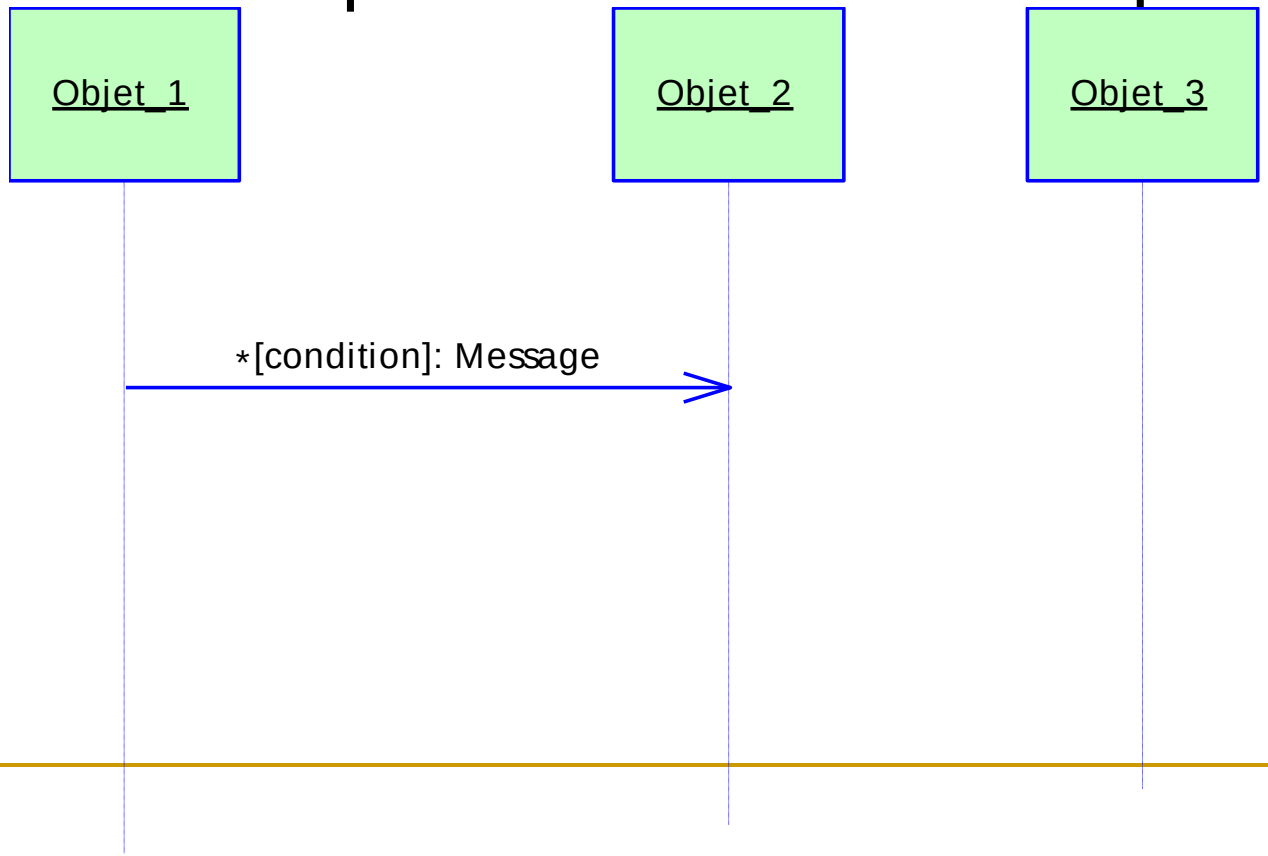
Les test (branchements) : Exemple

Pour accéder au centre de recherche, l'utilisateur doit présenter son badge. S'il a droit d'accès, un voyant vert est allumé et la porte s'ouvre



Les boucles (répétitions)

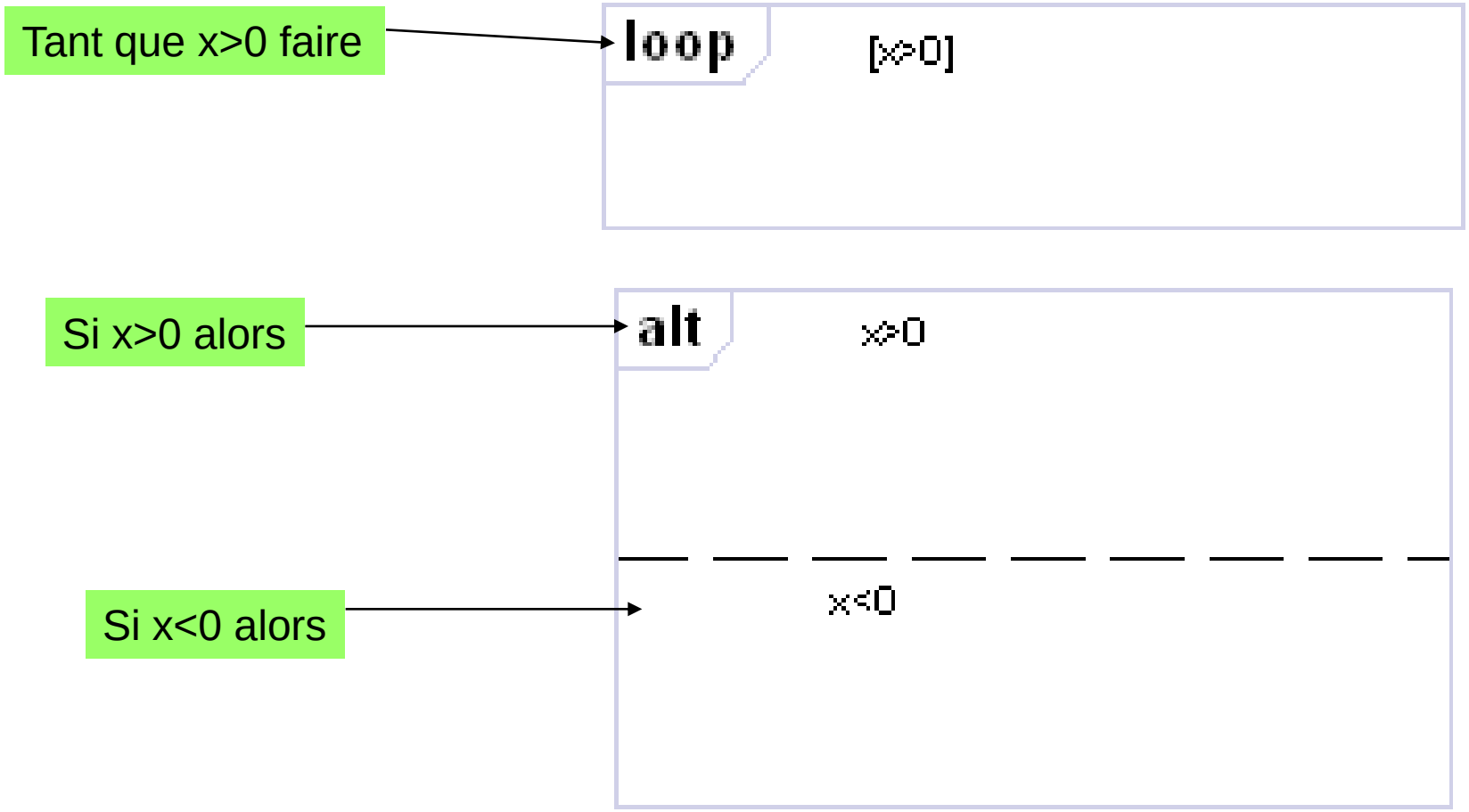
La boucle se note comme le test, mais la condition est précédée d'un astérisque



Fragments

- Permet de décomposer une interaction complexe en fragments simples
- Représenté par un rectangle dont le coin supérieur gauche contient un pentagone
- Dans le pentagone figure le type du fragment
  - loop : boucle
  - alt : alternative
  - ref : référence
  - …

Fragments



UML

Diagrammes de collaboration

Diagramme de collaboration

- Représente les interactions entre objets et relations structurelles permettant celles-ci.
- Permettent la description:
  - Du comportement collectif d'un ensemble d'objets
  - Des connexions entre ces objets
  - Des messages échangés par les objets
- Interaction réalisée par un groupe d'objets qui collaborent en échangeant des messages

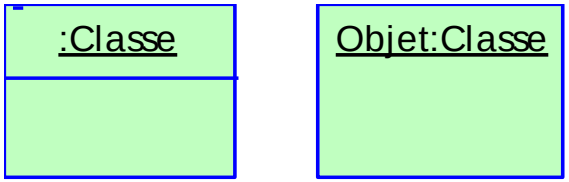
Diagrammes de collaboration

- Représentation graphique de l'évolution d'un ensemble d'objets pour effectuer une action
- Différences avec diagrammes de séquence
  - pas d'axe temporel
  - temps modélisé par numérotation

Diagrammes de collaboration

Éléments d'une interaction

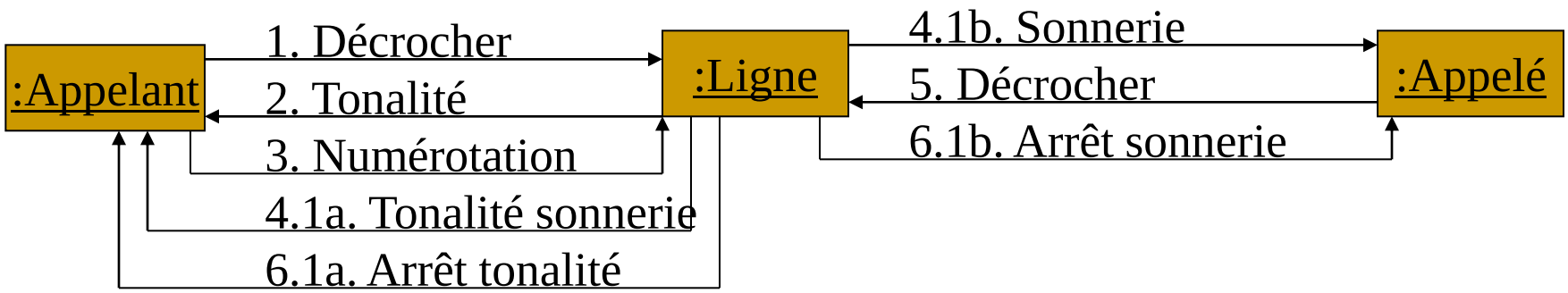
- ***Instances***
  - qui collaborent avec d'autres objets en échangeant des informations
  - Représentés par



- ***liens***
  - qui sont des supports de messages
  - Représentés comme des associations
- ***messages***
  - déclenchant les opérations
  - Indiqués par des flèches

Diagrammes de collaboration

- Exemple : Appel téléphonique



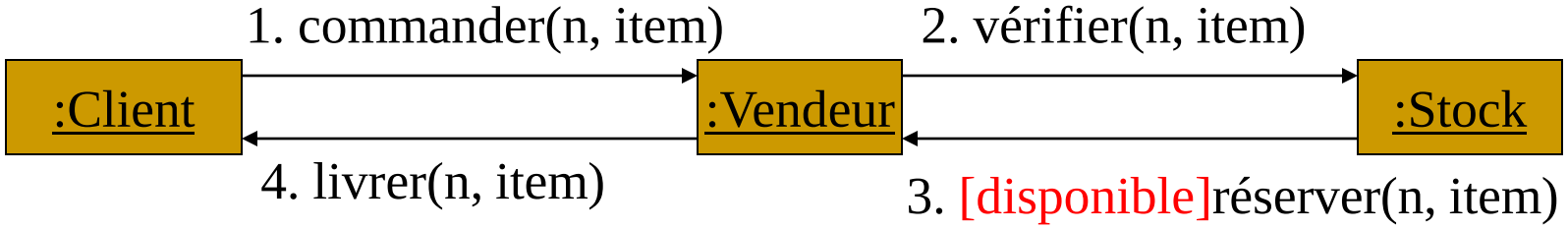
Diagrammes de collaboration

- Aspect temporel
  - modélisé par numérotation des messages
- Type et Sémantique des numérotations
  - 1, 2, 3, 4 : **Numérotation simple**
    - **séquencement** des messages
  - 1, 1.1, 1.2, 1.2.1, 1.2.2, 1.2.3 : **Dot notation**
    - séquencement + un point : ne peut être terminé que si ses sous points le sont aussi
  - 1, 1.1a, 1.1b, 1.2, 1.3 : **Dot notation + concurrence**
    - idem dot notation, mais les points 1.1a et 1.1b peuvent être effectués en parallèle

Diagrammes de collaboration

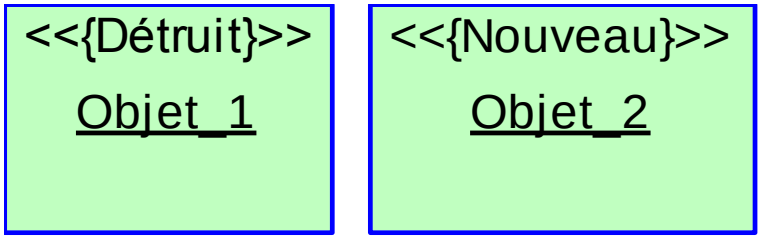
- Mêmes types contraintes que pour les diagrammes de séquence
  - Itération : *[condition]
  - Conditions : [condition]

- Exemple : réservation d'articles



Diagrammes de collaboration

- Les **objets crées** ou **détruits** au cours d'une interaction peuvent **respectivement** porter les contraintes :
- {Nouveau}
- {Détruit}

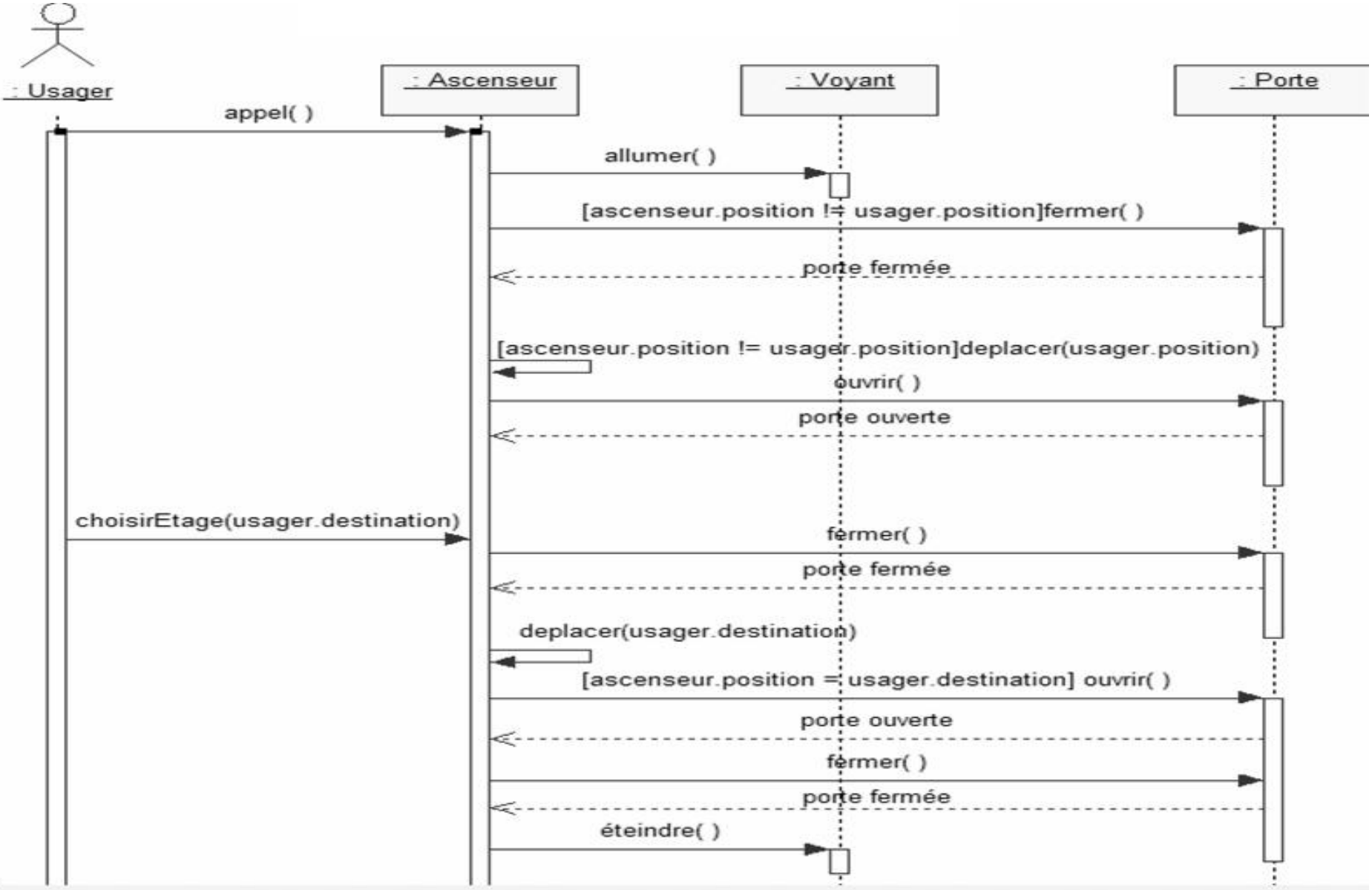


Diagrammes de collaboration

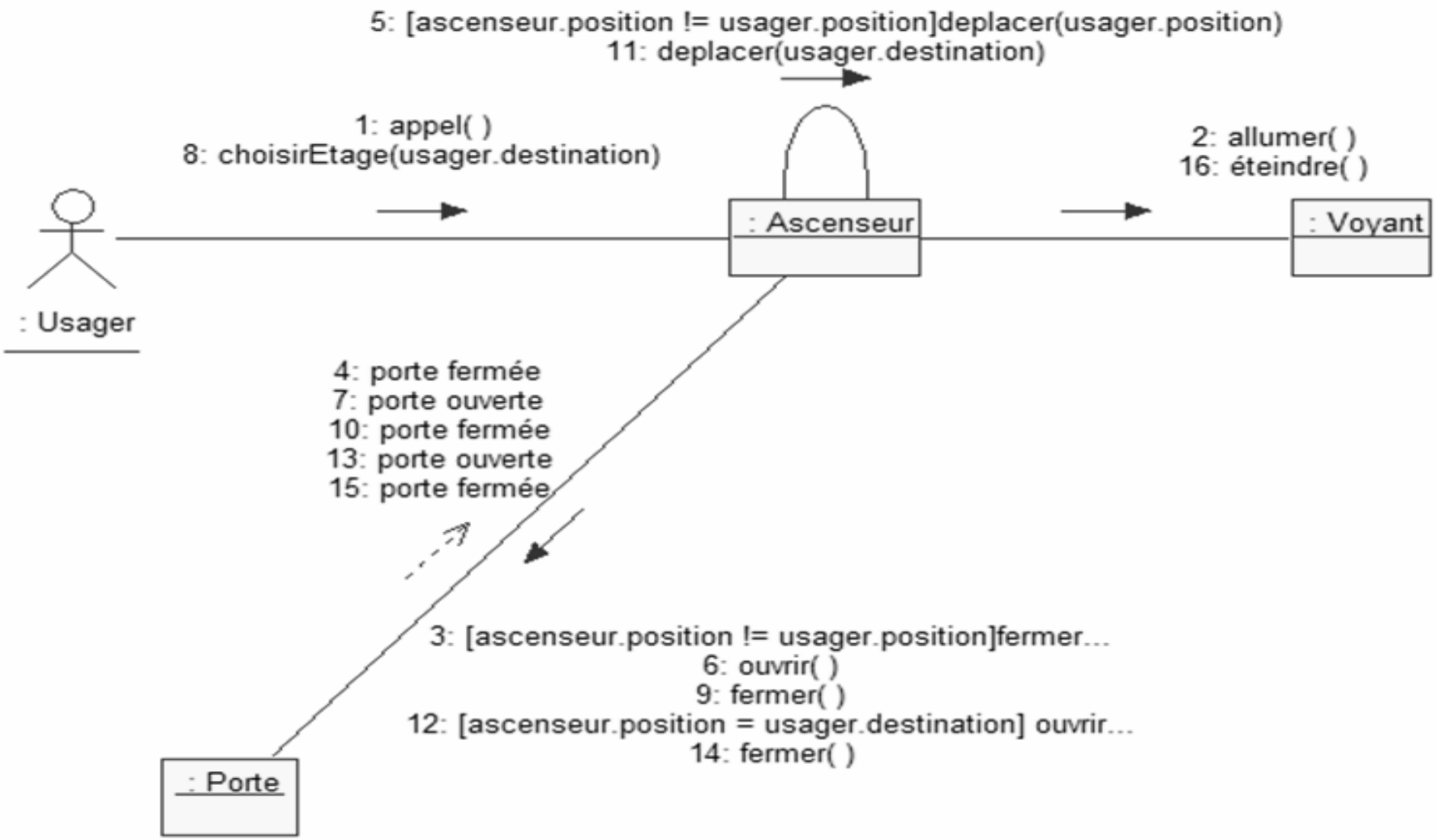
Conclusion

- Représentation spatiale
  - Aspect temporel plus difficile à suivre que pour les Diagramme de séquence
  - Durée d'exécution d'une contrainte difficile à évaluer
- Diagramme niveau instance
  - Limite : taille des diagrammes
    - Plus d'instances peuvent être représentées sur un même diagramme que pour les diagrammes de séquence

Exemple : Ascenseur (Séquence)



Exemple : Ascenseur (Collaboration)



UML

Diagramme état-transition

Diagramme état-transition

Le diagramme état-transition :

- Fait partie des modèles dynamiques
- Décrit l'enchaînement de tous les états d'un objet
- Propre à une classe donnée. Il décrit :
  - Les états des objets de cette classe
  - Les événements auxquels ils réagissent
  - Les transitions qu'ils effectuent

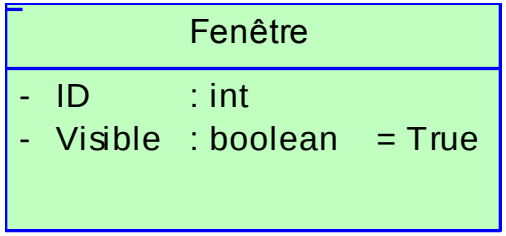
Diagramme état-transition

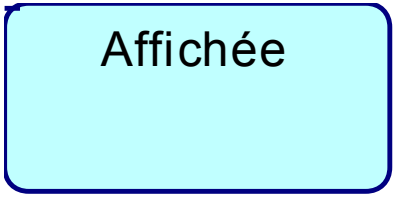
Le diagramme état-transition manipule plusieurs concepts :

- État
- Transition
- Événement
- Garde
- …

État

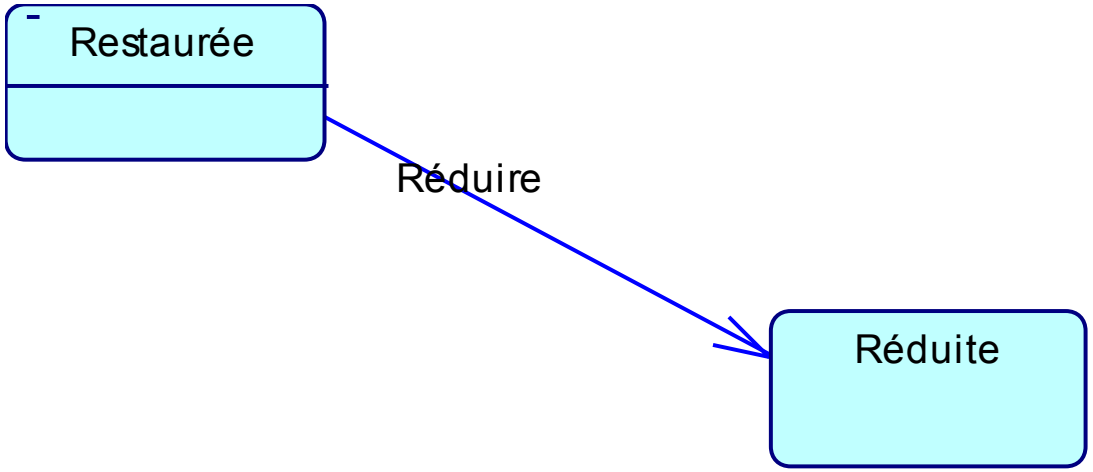
- L'état d'un objet est défini par l'ensemble des valeurs de ses attributs (fenêtre affichée, fenêtre cachée, …)
- Un état dépend de l'état précédent et de l'événement survenu
- Un état est représenté par un rectangle aux coins arrondis





Transition

- C'est le passage d'un état à un autre
- Peut être nommé par un événement
- Représenté par une flèche orientée de l'état source vers l'état cible

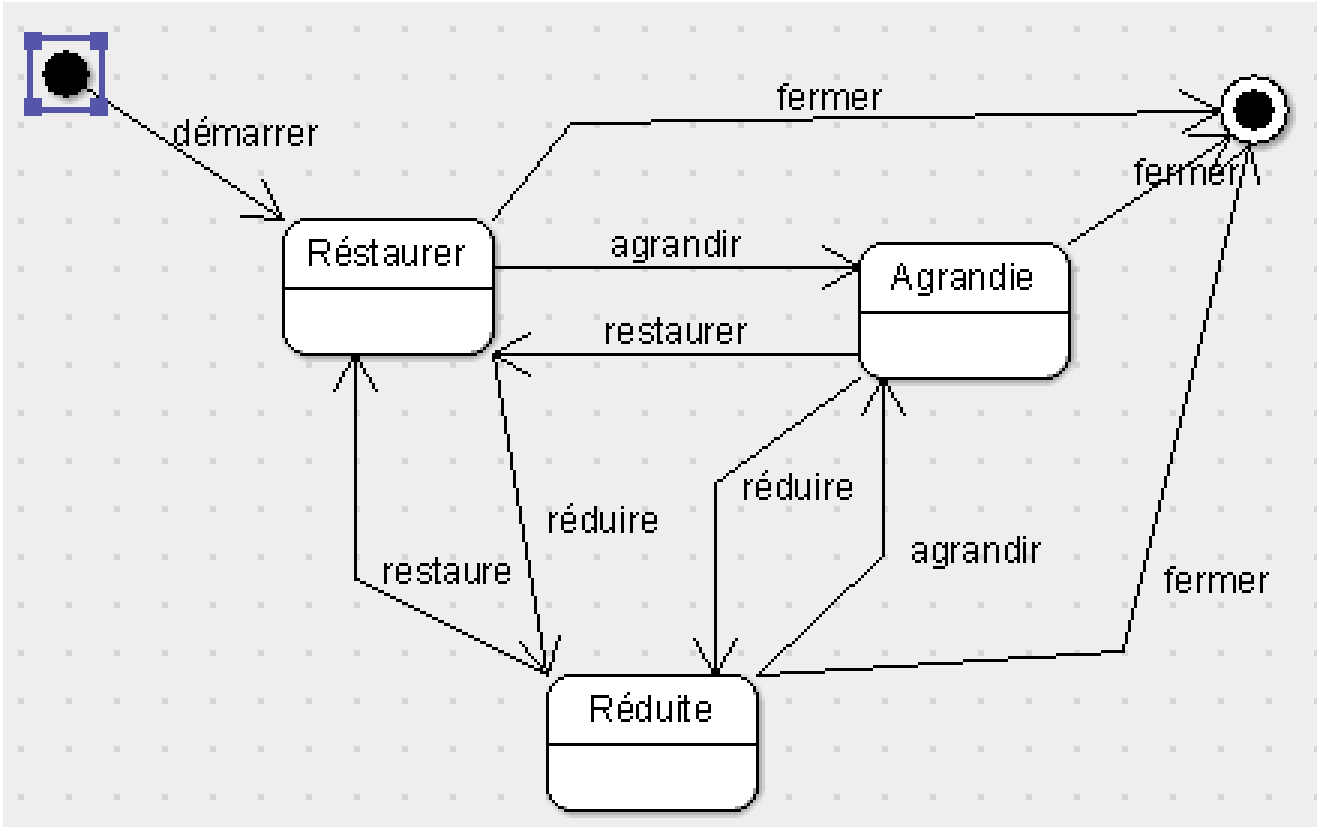


Événement

- Fait (externe) survenu qui déclenche une transition (changement d'états)
- Peut être réflexif et conduire au même état
- Conduit à l'appel d'une méthode de la classe de l'objet
- Peut posséder des attributs :
  - paramètres portés par des événements
  - Représentés entre parenthèses

Exemple

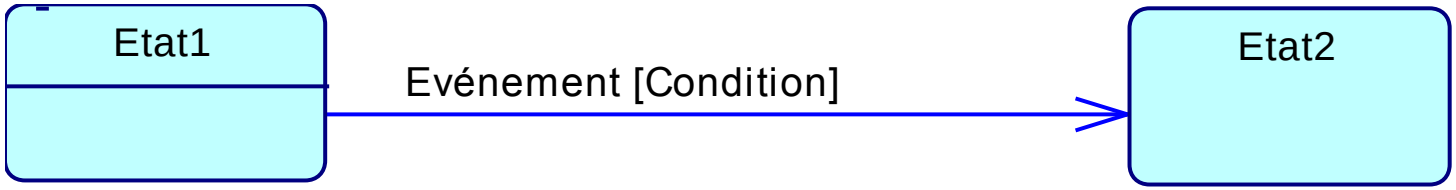
Soit le diagramme d'états/transitions de l'objet 'Fenêtre'



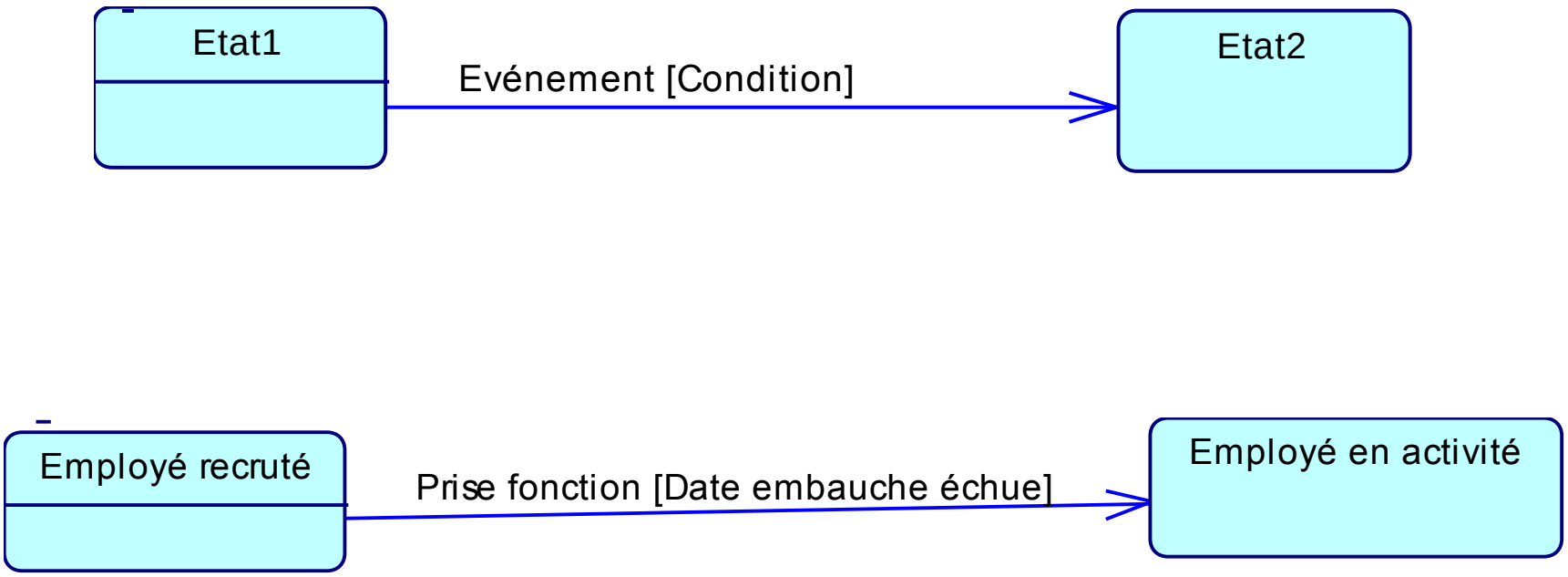
Fenêtre
Id : Integer taille : Rectangle abs : Integer ord : Integer newAttr : Integer
restaure() réduire() fermer() agrandir()

Gardiens

- Conditions ou fonctions booléennes associées à une transition
- Une transition gardée ne peut être effectuée que si le gardien est vérifié
- Un gardien est représenté entre crochets



Formalisme et exemple



Actions et activités

- Un objet qui reçoit un événement déclenche une ou plusieurs opérations
- On distingue deux types d'opérations :
  - Action : associée à un état ou à une transition
  - Activité : associée à un état

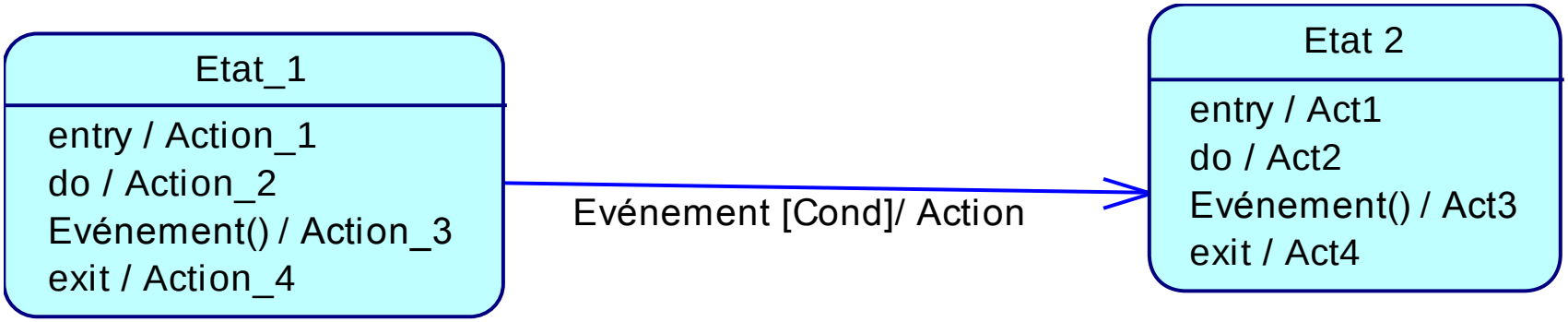
Activité

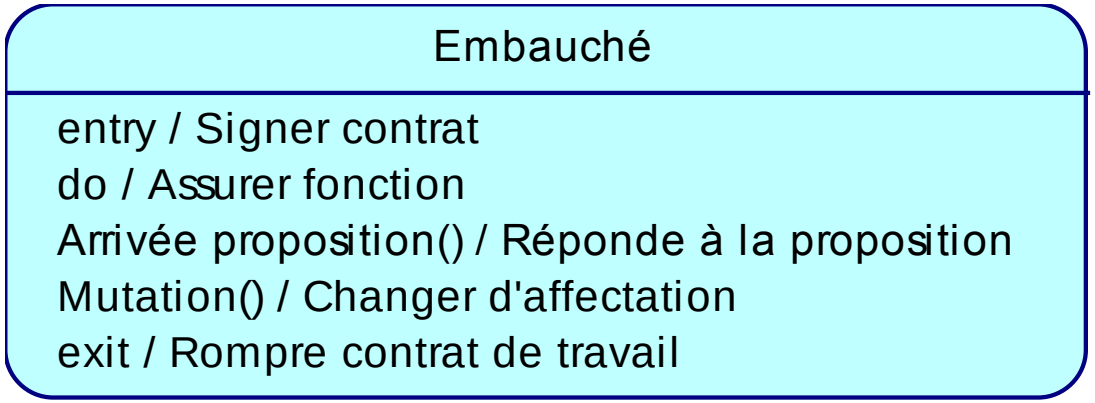
- Opération d'une certaine durée, qui est exécutée tant que l’objet se trouve dans l’état
- Associée à un état d'un objet
- Représentée dans l'état précédée par la notation "do/"

Action

- Opération instantanée non interrompue
- Peut être associée aussi bien à l'état d'un objet qu'a une transition
- Elle peut intervenir soit
  - En entrée de l'état (préfixe : "entry/")
  - En sortie de l'état (préfixe : "exit/")
  - En réponse à un événement (préfixe :"evt/")
  - Au cours d'une transition (préfixe : "evt/")

Formalisme et exemple

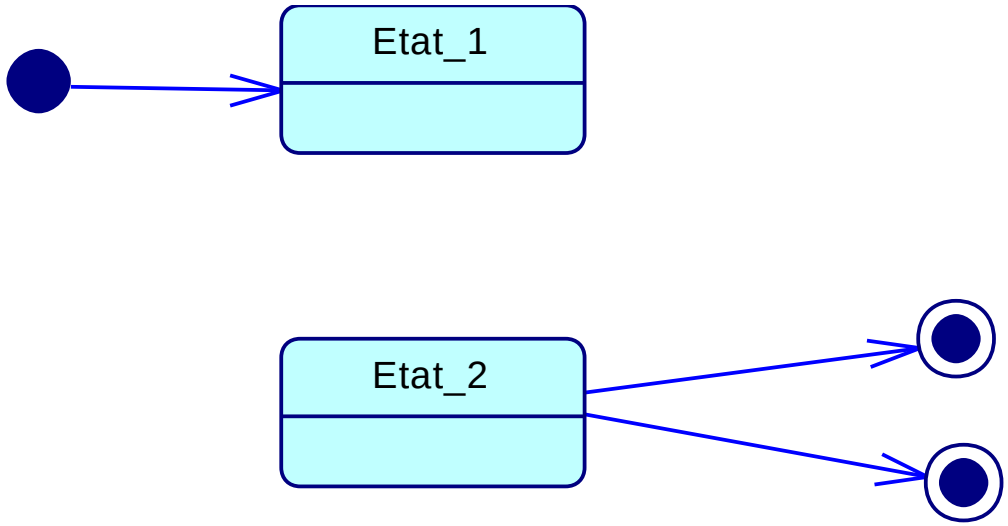




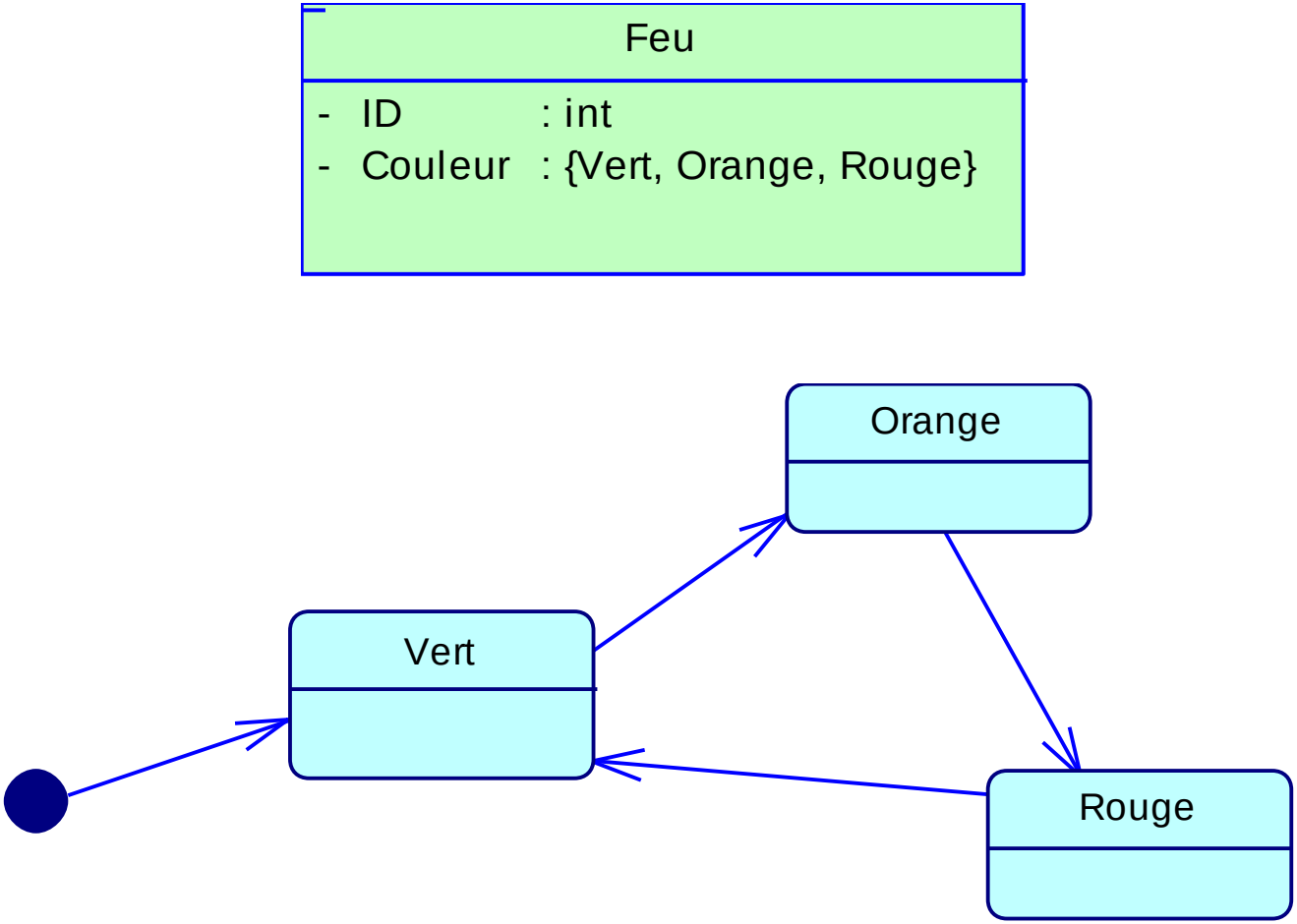
État initial et états finaux

Un diagramme état-transition

- Débute toujours par un état initial
- Se termine par un ou plusieurs états finaux (sauf où le diagramme représente une boucle)



Exemple (Feu de signalisation)



UML

Diagramme d'activités

Introduction

- Variante des diagrammes d'état-transition
- Permet de décrire le flot de contrôle entre les opérations :
  - Choix
  - Séquences
  - Itérations
  - Parallélisme
- Au niveau macroscopique : décrit les enchaînements des opérations
- Au niveau microscopique : décrit l'algorithme d'une action du diagramme d'états

Concepts de base

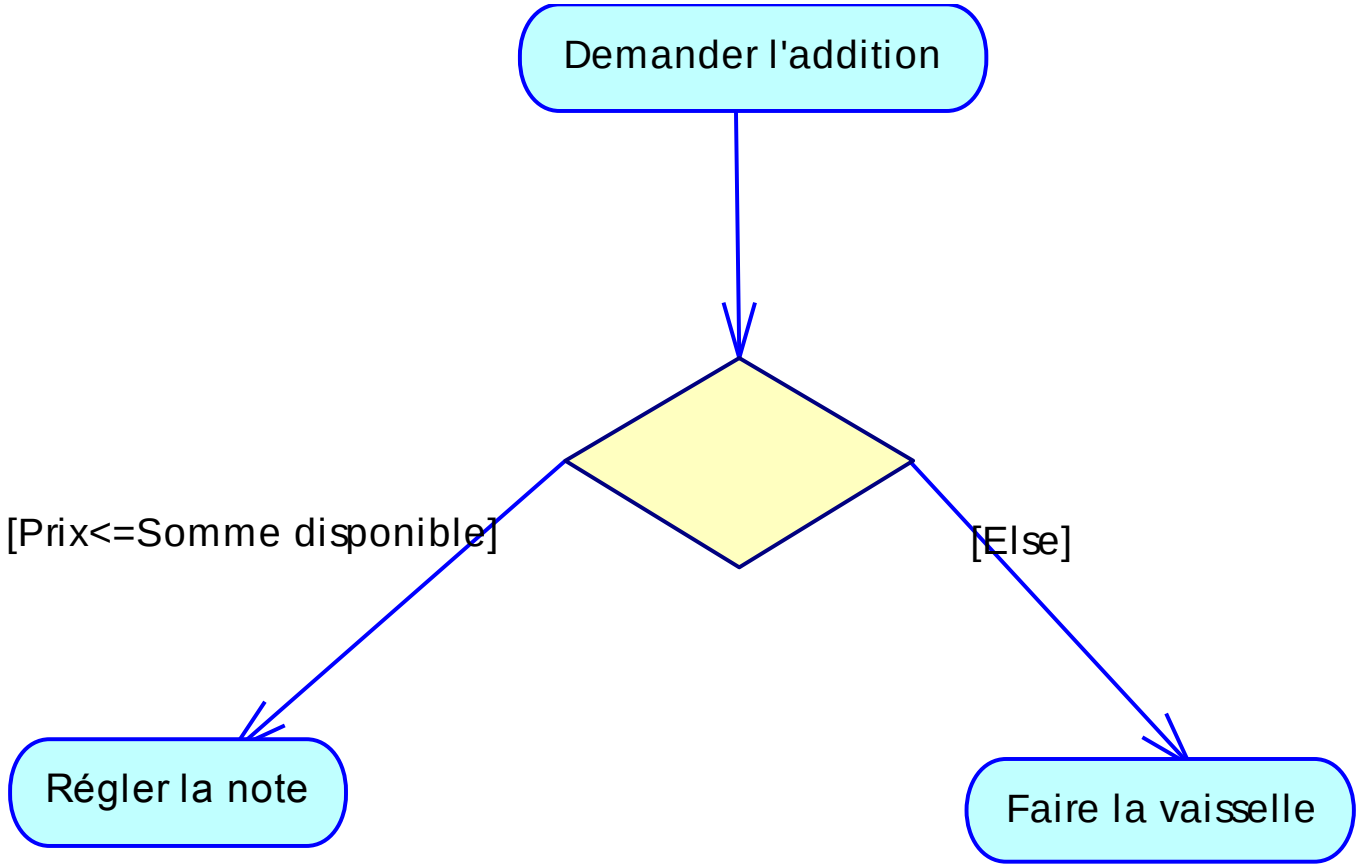
Plusieurs concepts sont manipulés :

- État
- Activité
- *Transition (séquentielle, alternatives ou conditionnelle)*
- *Synchronisation (disjonction et conjonctions d'activités)*
- *Itération*
- *Swimlanes*

Comportement conditionnel

- Appelé aussi le branchement
- Symbolise une transition entrante gardée par une condition et plusieurs transitions sortantes mutuellement exclusives

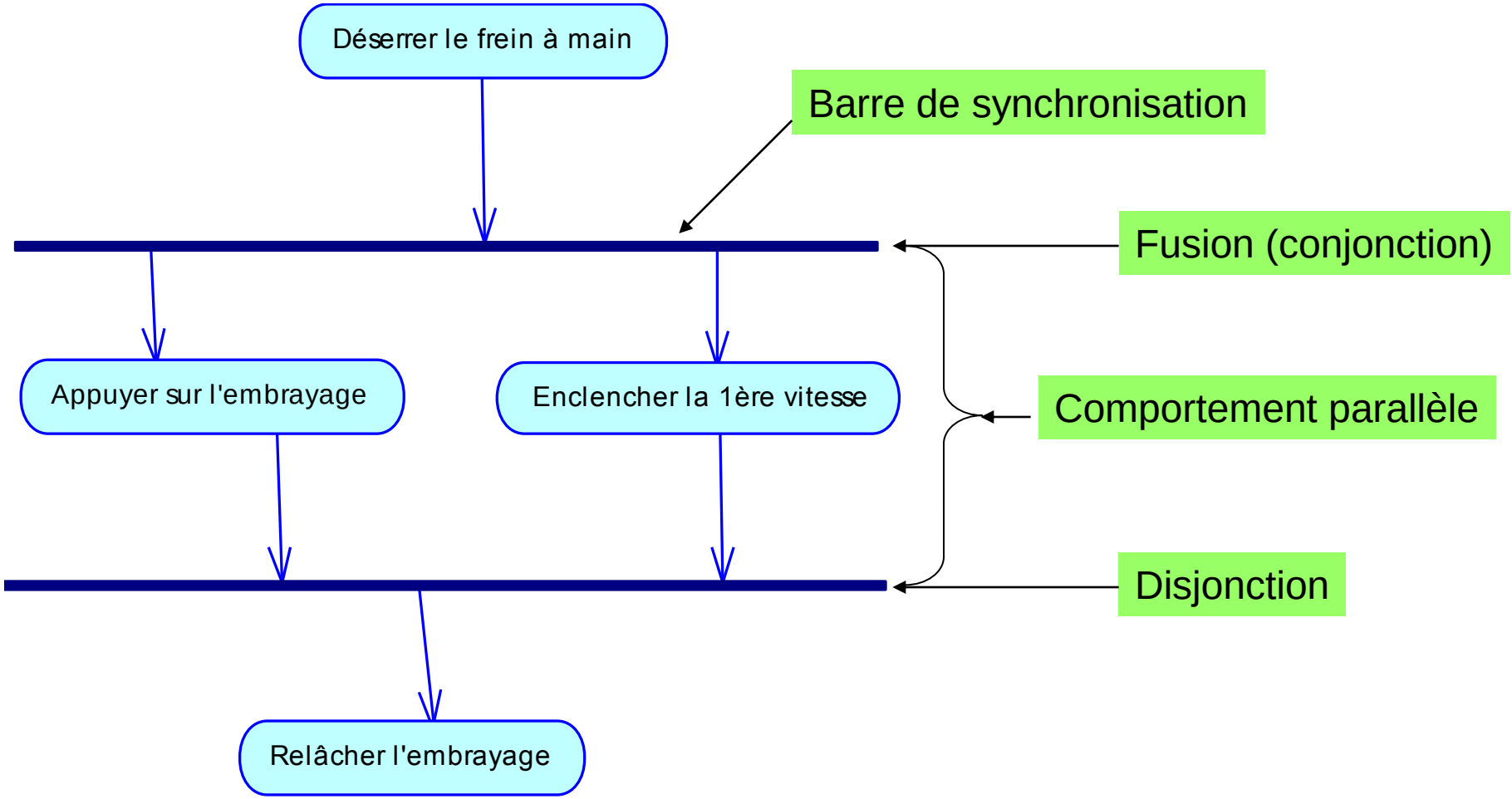
Comportement conditionnel : Exemple



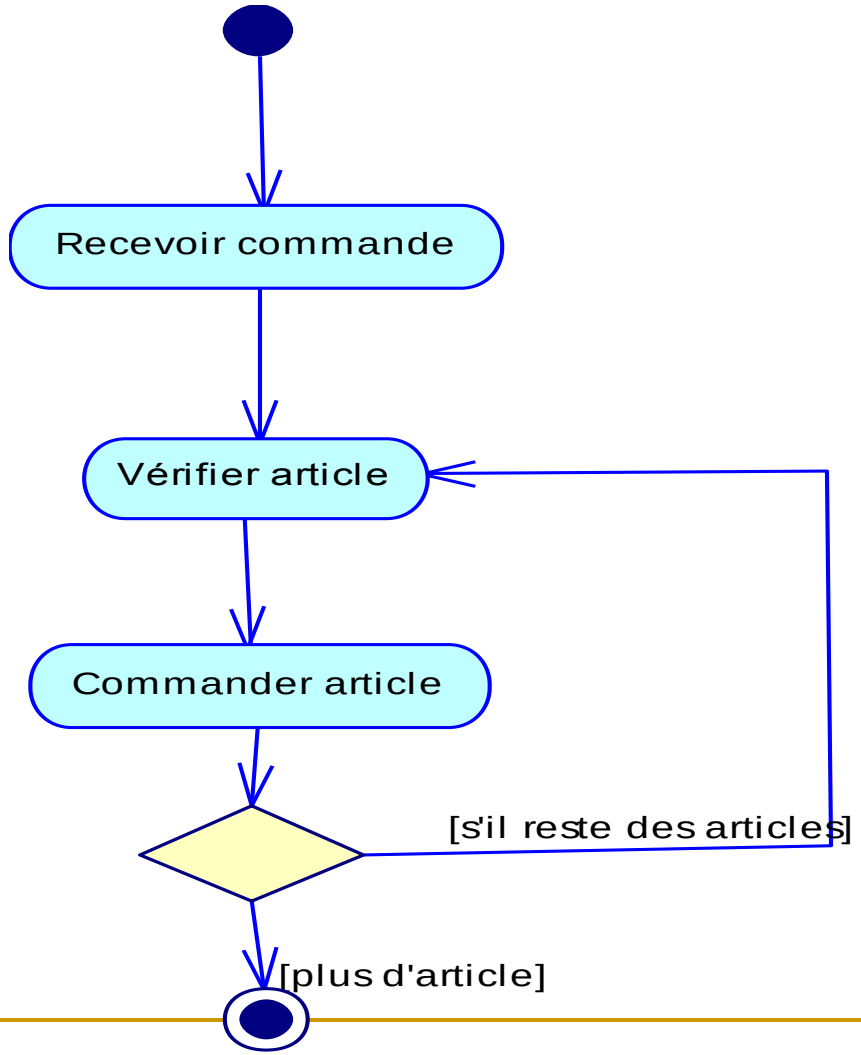
Synchronisation

- Fusion (conjonction) : plusieurs transitions entrantes et une seule sortante
- Comportement parallèle :
  - La barre de synchronisation permet d'ouvrir et de fermer les branches parallèles au sein d'un flot d'exécution
  - Les transitions partantes d'une barre ont lieu en même temps
  - La barre n'est franchie qu'après réalisation de toutes les transitions qui s'y rattachent

Synchronisation : Exemple

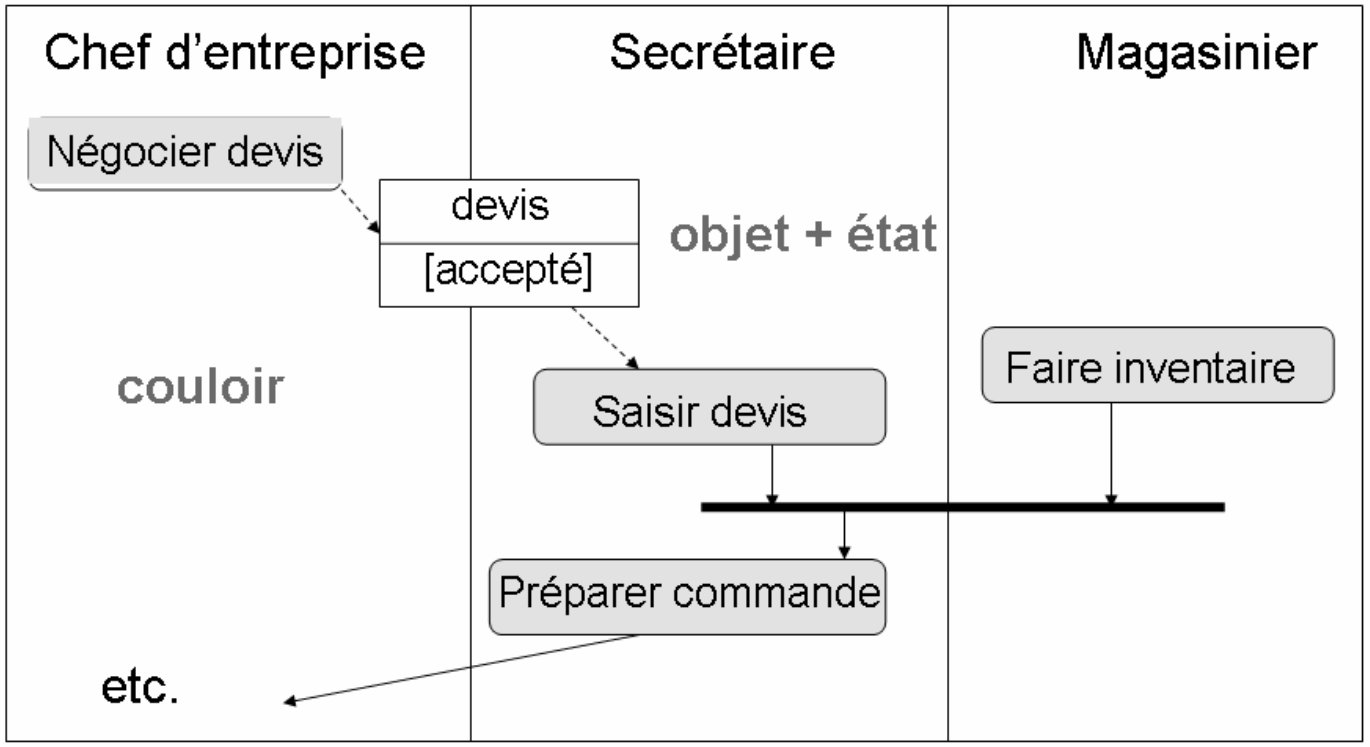


Itération : Exemple

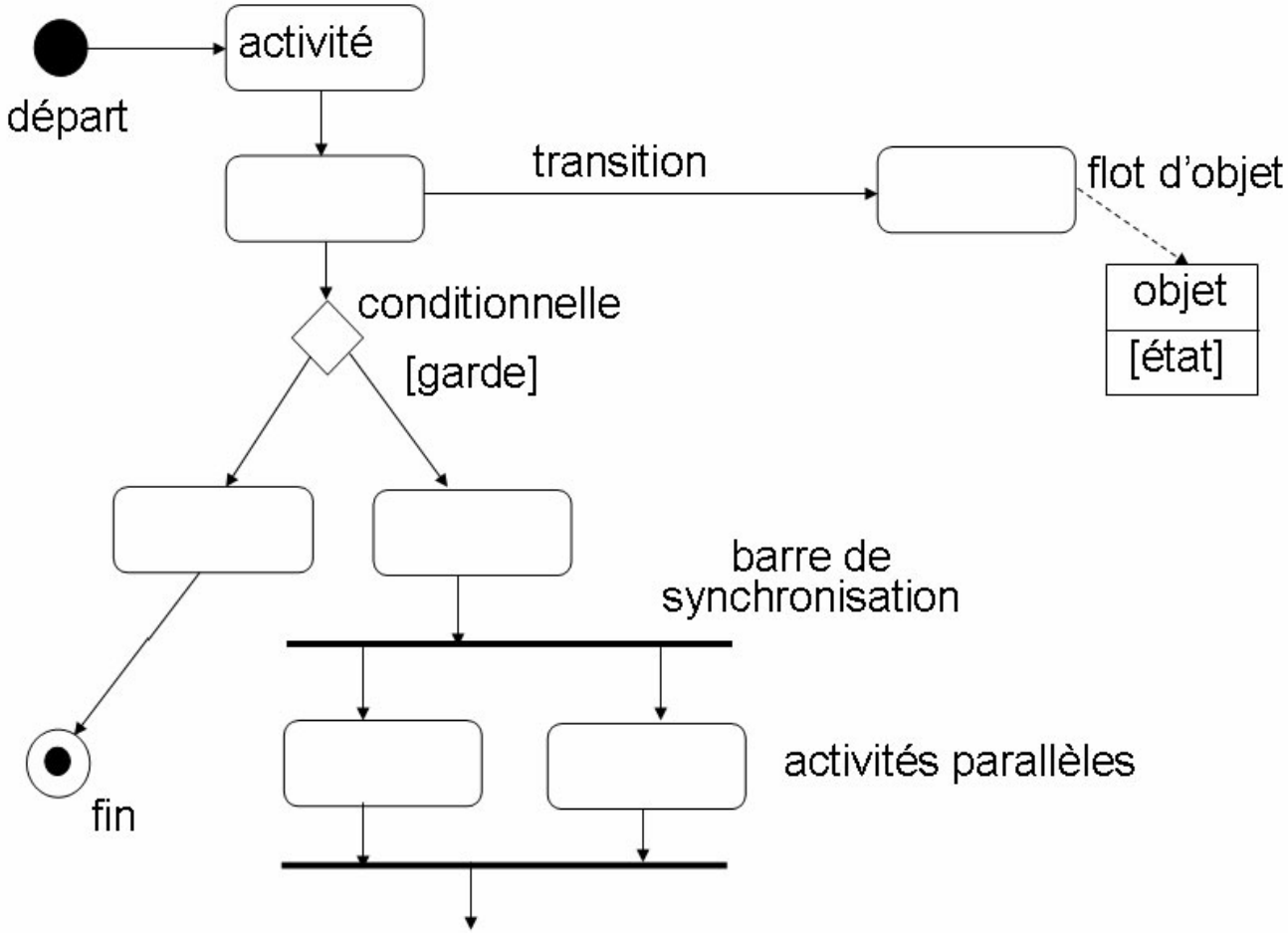


Swimlanes

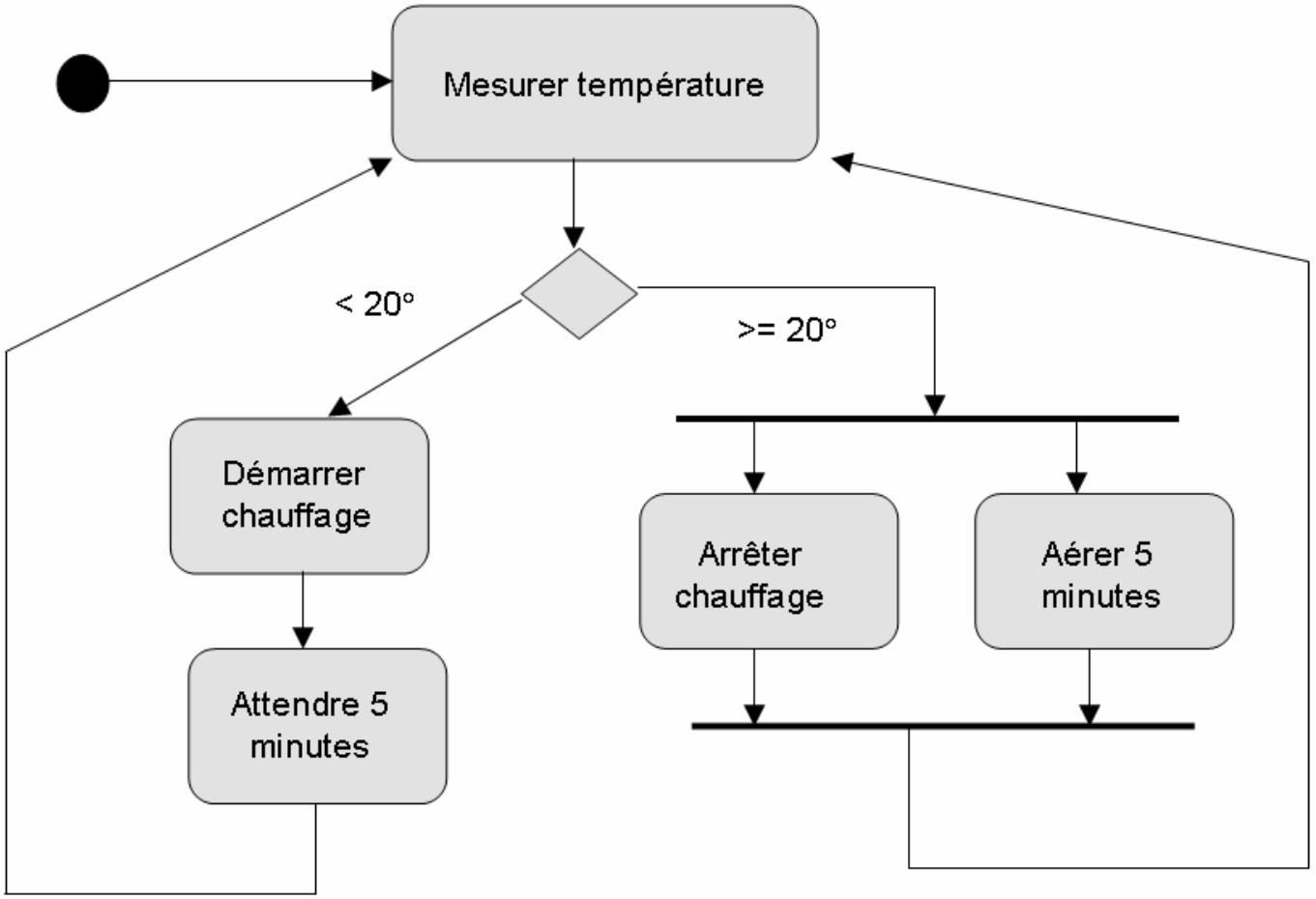
- Extension des diagrammes d'activités permettant de représenter l'organisation.
- Représente le lieu, le responsable des activités.



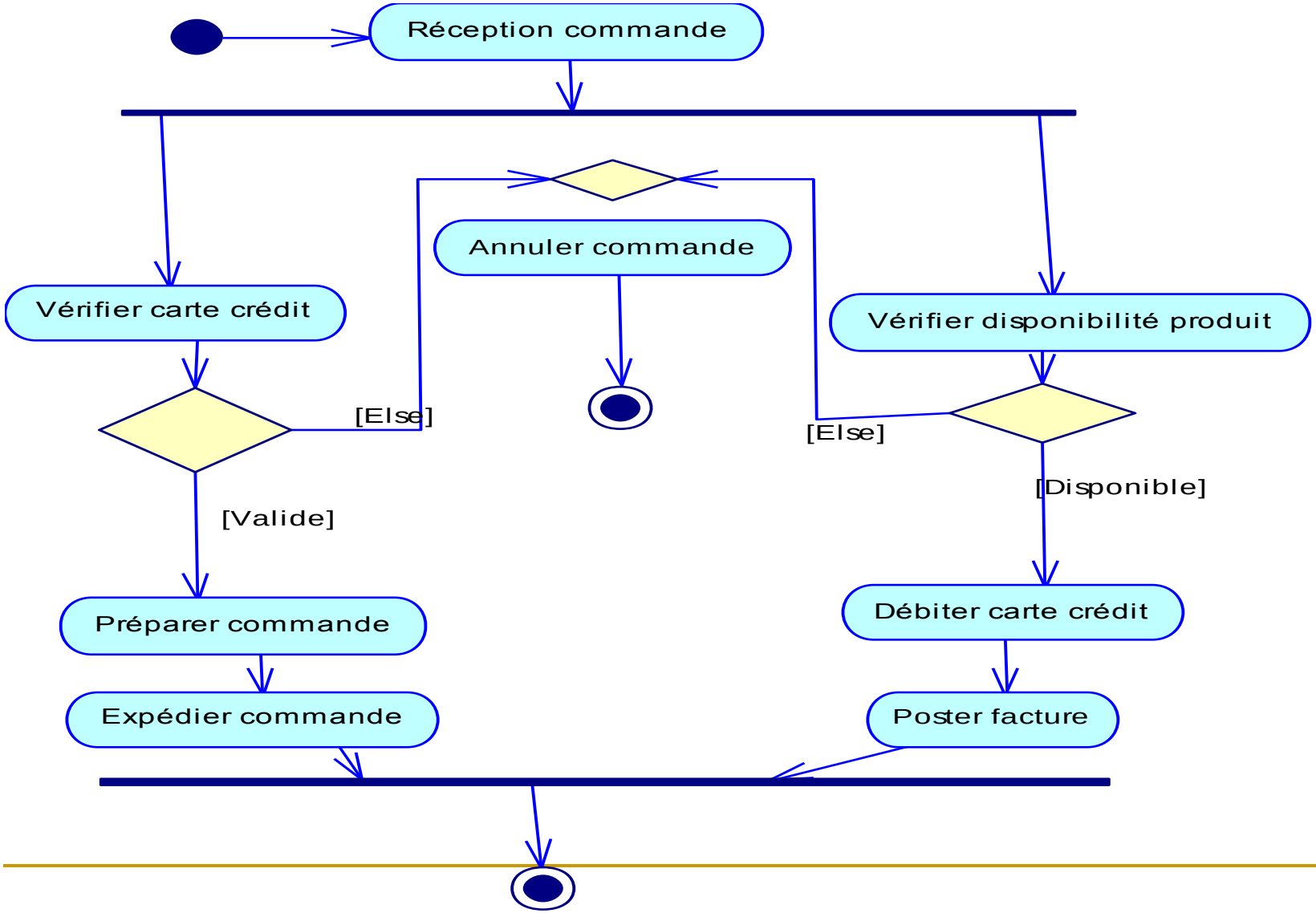
Résumé notation



Exemple récapitulatif



Exemple récapitulatif

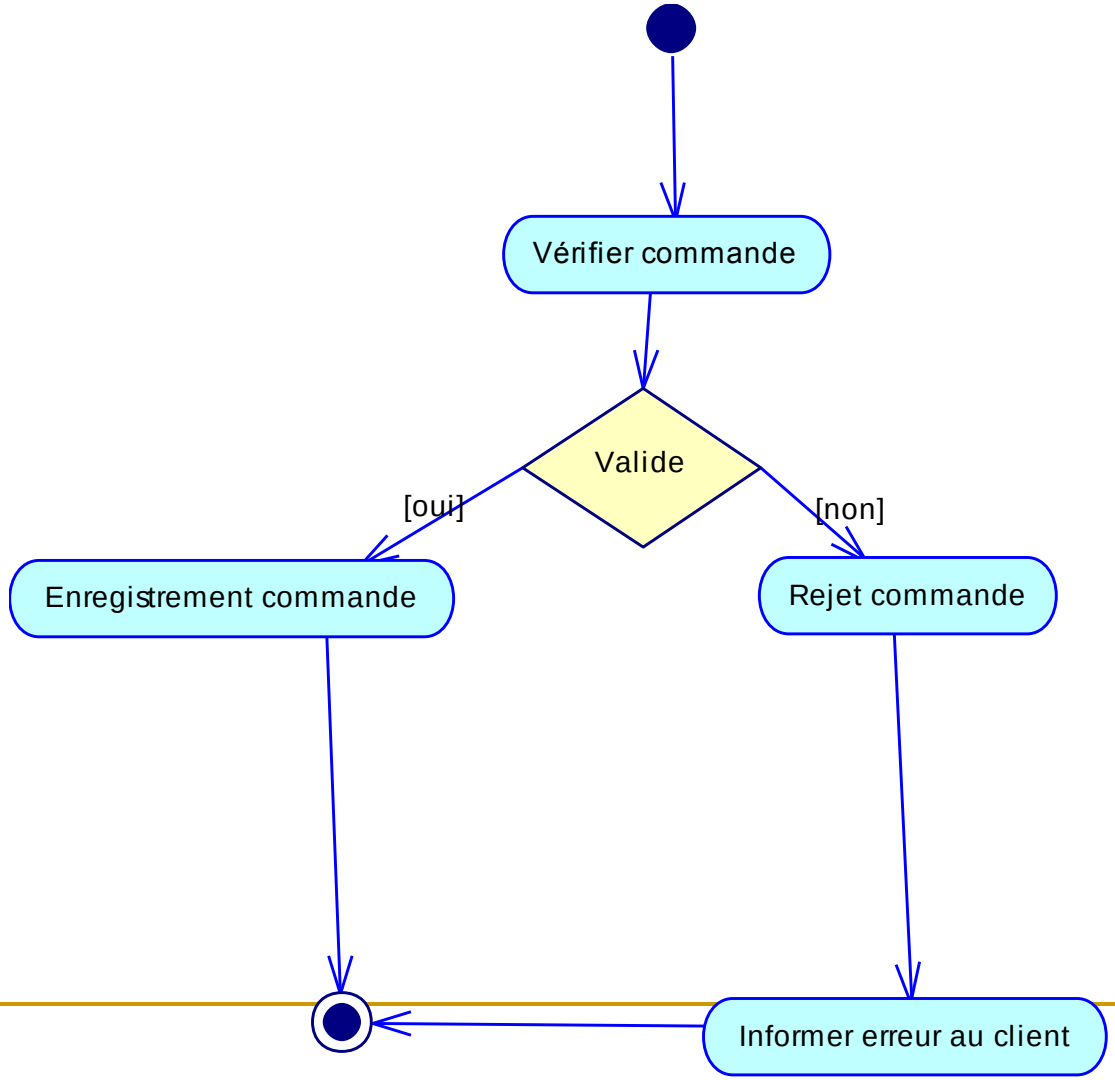


Exercice 1

Représenter les états suivants sous forme de diagramme d'activité :

- Vérification commande
- Enregistrement commande
- Rejet commande
- Informer erreur au client

Exercice 1 : solution



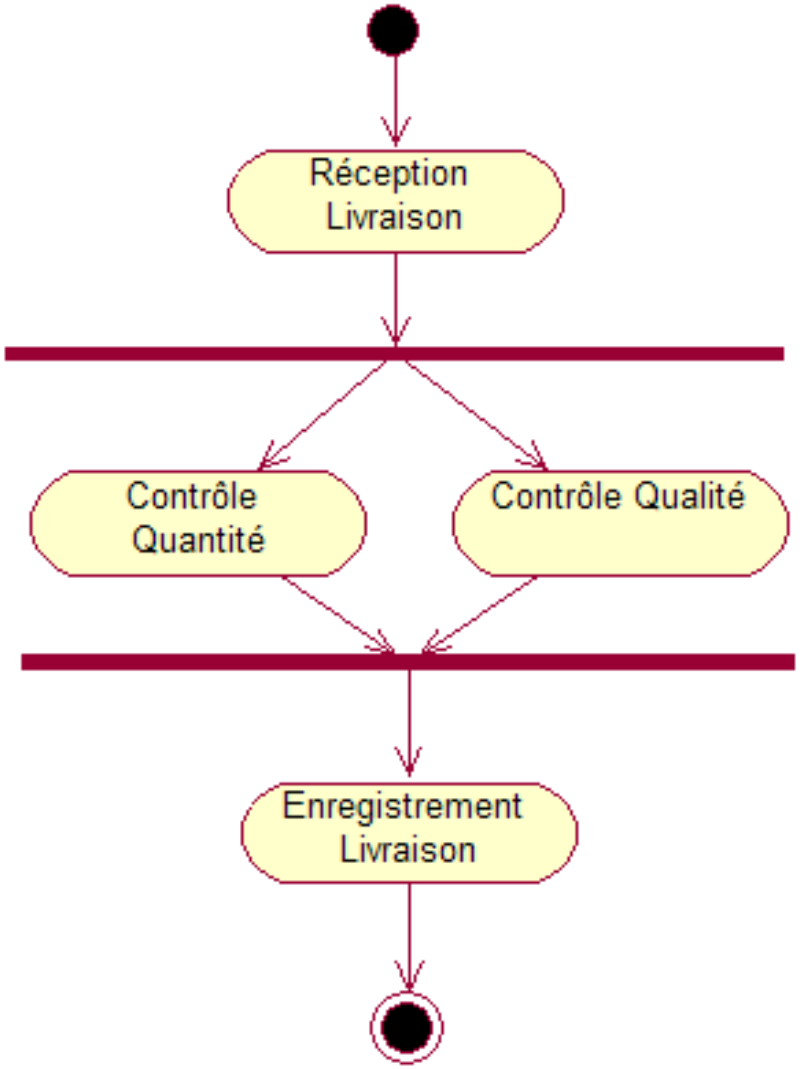
Exercice 2

Dans le domaine de gestion de stock, on considère les états suivants indiquant le flot de contrôle de réception d'une livraison :

Réception livraison, contrôle qualité, contrôle quantité et enregistrement livraison.

Proposez un diagramme d'activité représentant ce flot d'information

Exercice 2 : solution

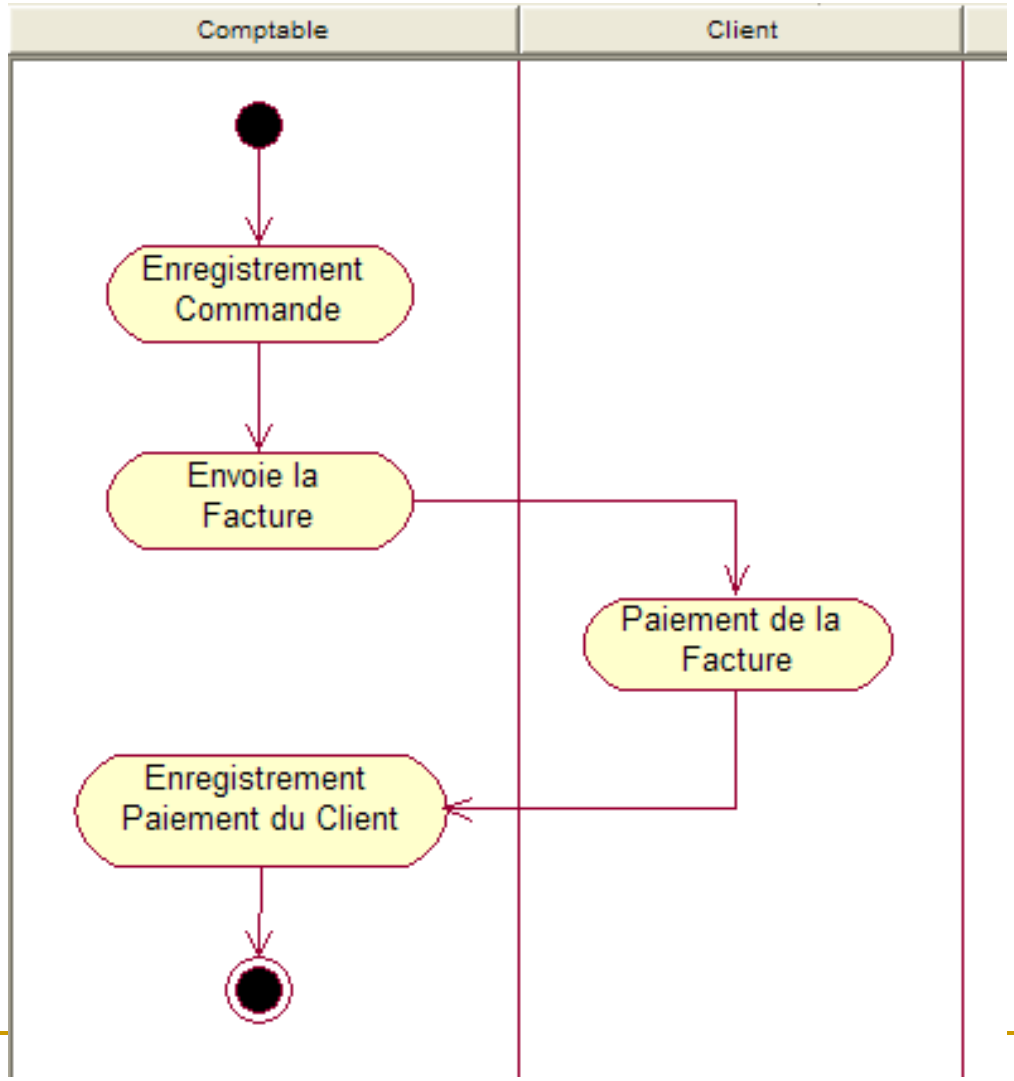


Exercice 3

Construire un diagramme d'activité pour modéliser le processus de commander d'un produit. Le processus concerne les acteurs suivants:

- **Comptable** : enregistrement commande, envoie la facture et enregistrement paiement du client
- **Client** : paiement de la facture

Exercice 3 : solution

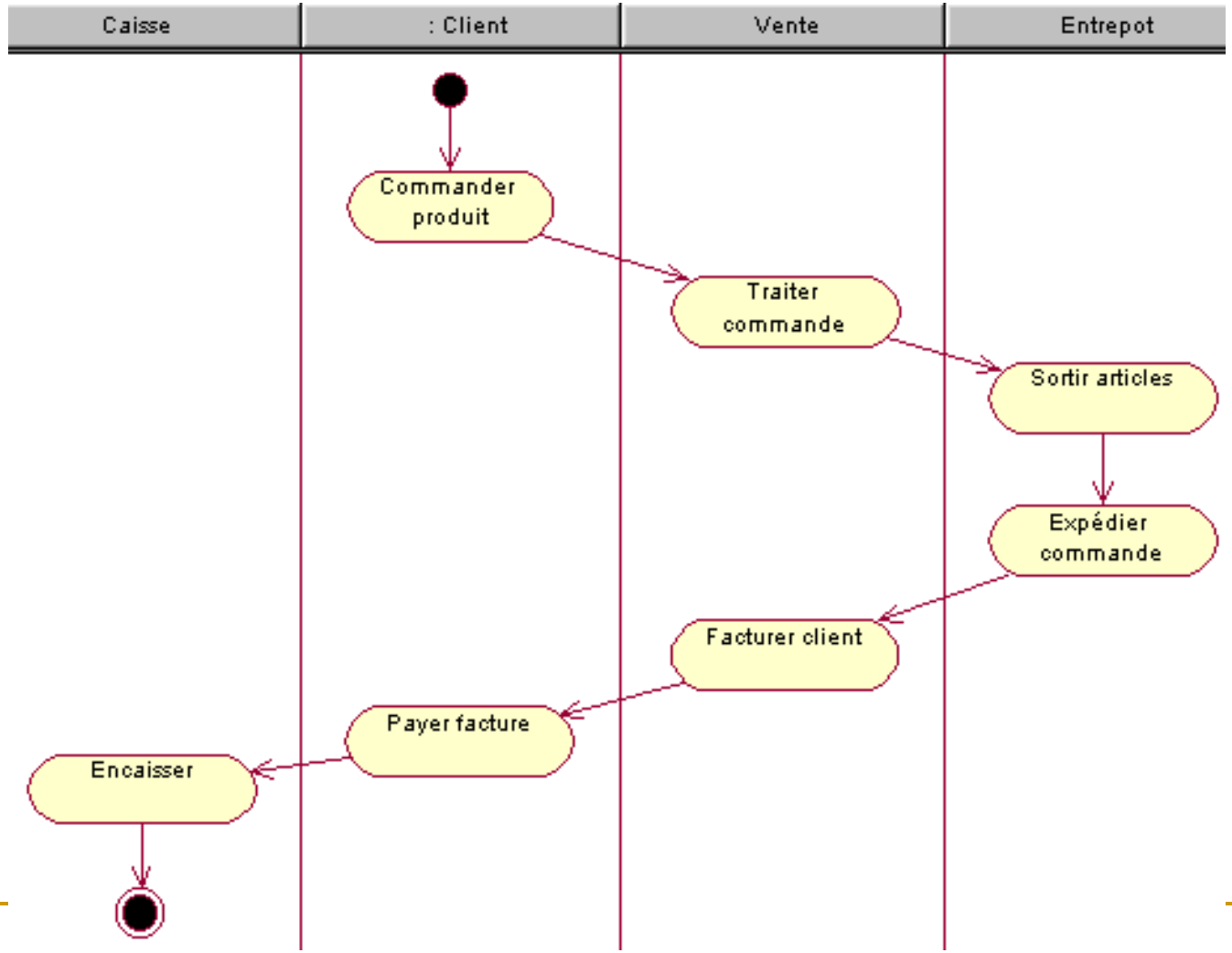


Exercice 4

Construire un diagramme d'activité pour modéliser le processus de commander d'un produit. Le processus concerne les acteurs suivants:

- **Client**: qui commande un produit et qui paie la facture
- **Caisse**: qui encaisse l'argent du client
- **Vente**: qui s'occupe de traiter et de facturer la commande du client
- **Entrepôt**: qui est responsable de sortir les articles et d'expédier la commande.

Exercice 4 : solution



UML

Diagrammes de Composants et de Déploiement

Diag de Composants/ Déploiement

Permettent de modéliser les aspects physiques d'un système orienté-objet

- Diagramme de Composants : se focalise sur l'organisation et les dépendances entre un ensemble de composants
- Diagrammes de Déploiement : se focalise sur la configuration en temps d'exécution des nœuds de traitement et de composants qui sont actifs

Diagramme de composants

- Dans le monde de bâtiment, tout ce qui est proposé par l'architecte (plan) constitue une vue logique : visualiser, spécifier, documenter
- Lors de la construction, on utilise des composants physiques du monde réel : murs, fenêtres, portes, …

Diagramme de composants

- De même, tout ce que nous avons vu jusqu'à présent constitue le modèle logique : visualiser, spécifier et documenter la structure et le comportement des objets
- La construction va s'appuyer sur les composants du monde réel de l'ordinateur : fichiers, tables, librairies, …

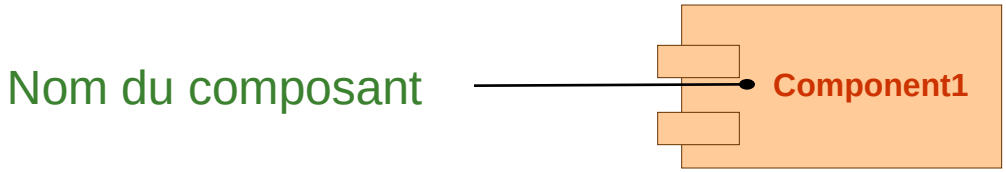
Diagramme de composants

- Permet de décrire l'architecture physique et statique d'une application en terme de composants :
  - code source,
  - bibliothèques,
  - exécutables,
- Il montre la mise en oeuvre physique des modèles de la vue logique dans l'environnement de développement
- Permet de spécifier :
  - Composants
  - Interfaces
  - Relations (dépendance, généralisation, association, réalisation).

Composant

- Un composant est une partie physique et remplaçable d'un système qui sait faire et fournit la réalisation d'un ensemble d'interface
- Les composants peuvent être organisés en paquetages

Composant

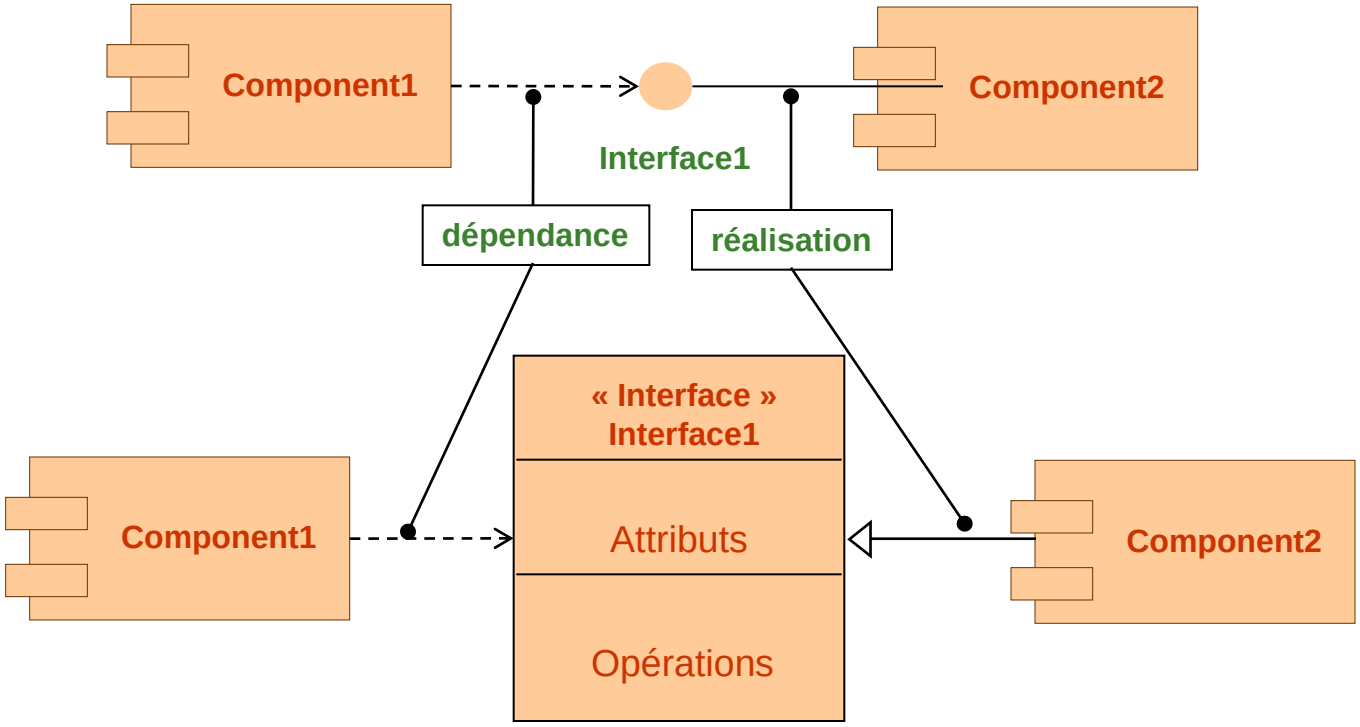


- **Nom du composant** :
  - Permet de distinguer un composant des autres composants
  - Il peut être un nom simple ou un nom composé qui indique le paquetage auquel appartient le composant
- **Stéréotypes** : spécifient un composant qui désigne:
  - « *executable* »: un programme pouvant s'exécuter sur un nœud
  - « *library* » : une bibliothèque statique ou dynamique
  - « *table* »: une table de base de données
  - « *file* » : un fichier contenant du code source ou des données
  - « *document* » : un document

Concepts

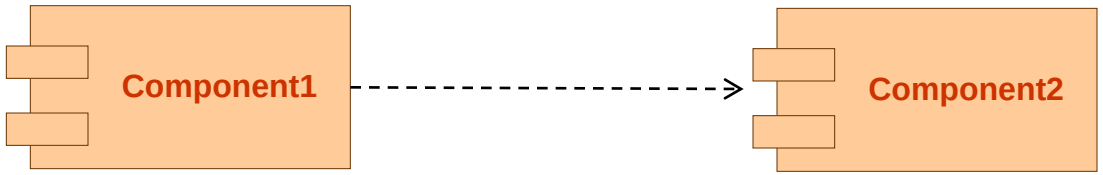
- **Interface** :
  - Est une collection d'opérations utilisées pour spécifier les services d'une classe ou d'un composant
- **Relations avec les interfaces**
  - **Réalisation** :
    - Définie entre l'interface et le composant qui fournit les services pour les autres composants
    - Cette interface est appelée « *interface exportée* »
  - **Dépendance** :
    - Définie entre l'interface et le composant qui utilise les services fournis par un autre composant
    - Cette interface est appelée « *interface importée* ».

Interface


Component1
Component2
Interface1
dépendance
réalisation
« Interface »
Interface1
Attributs
Opérations
Component1
Component2

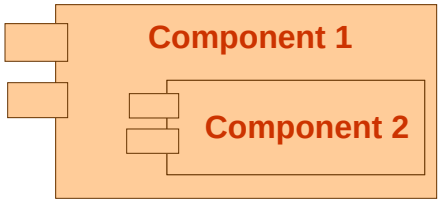
Relations entre les composants

- **Dépendance** :
  - Cela signifie qu'un des éléments d'un composant a besoin des services que les élément de l'autre composant réalisent
  - Notation UML



Relations entre les composants

- **Contenance** :
  - Un composant peut être contenu dans un autre composant
  - Notation UML



Système Vente (diagramme de classes)

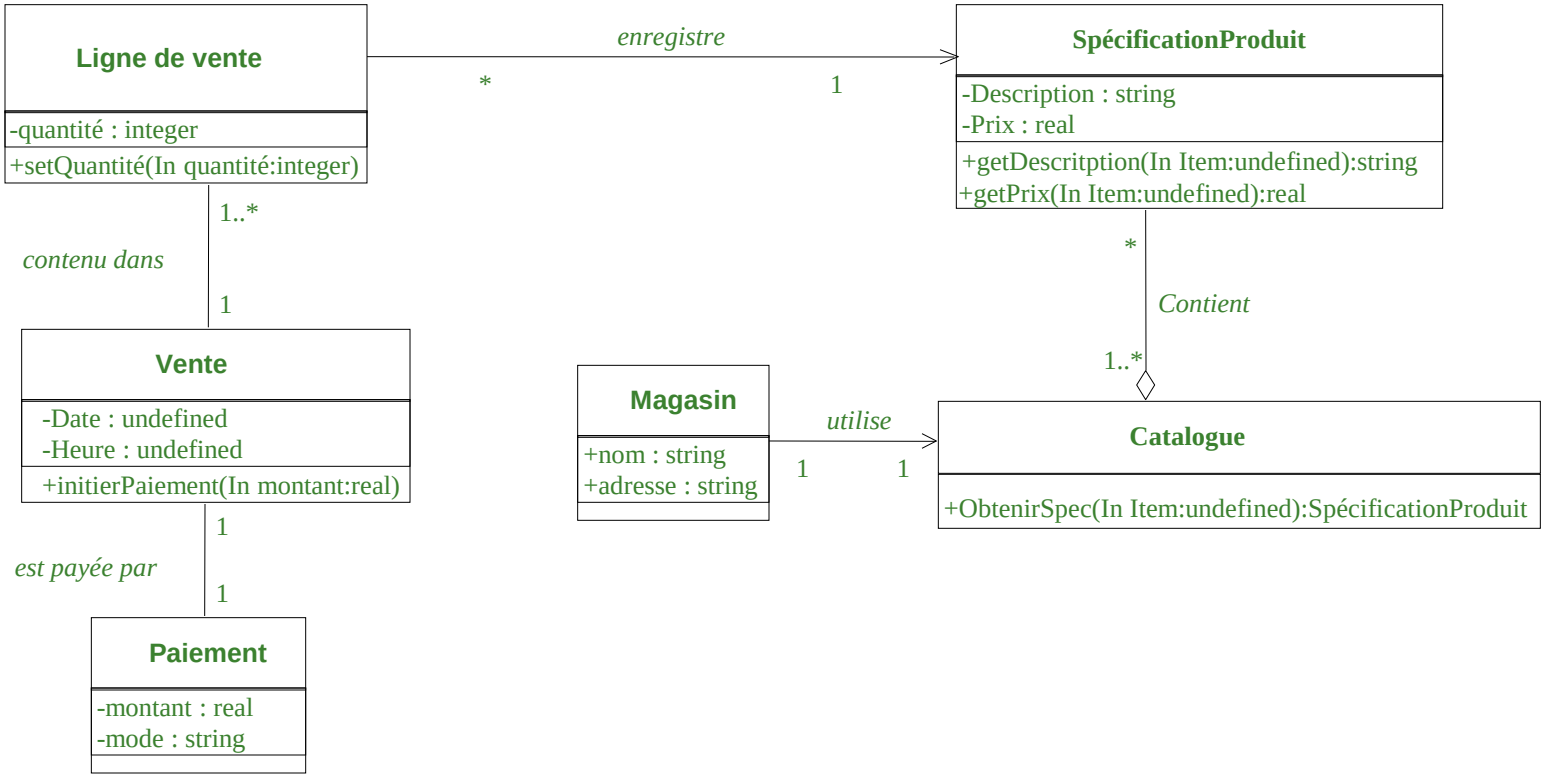


Diagramme de composants (Exemple)

- Système Vente

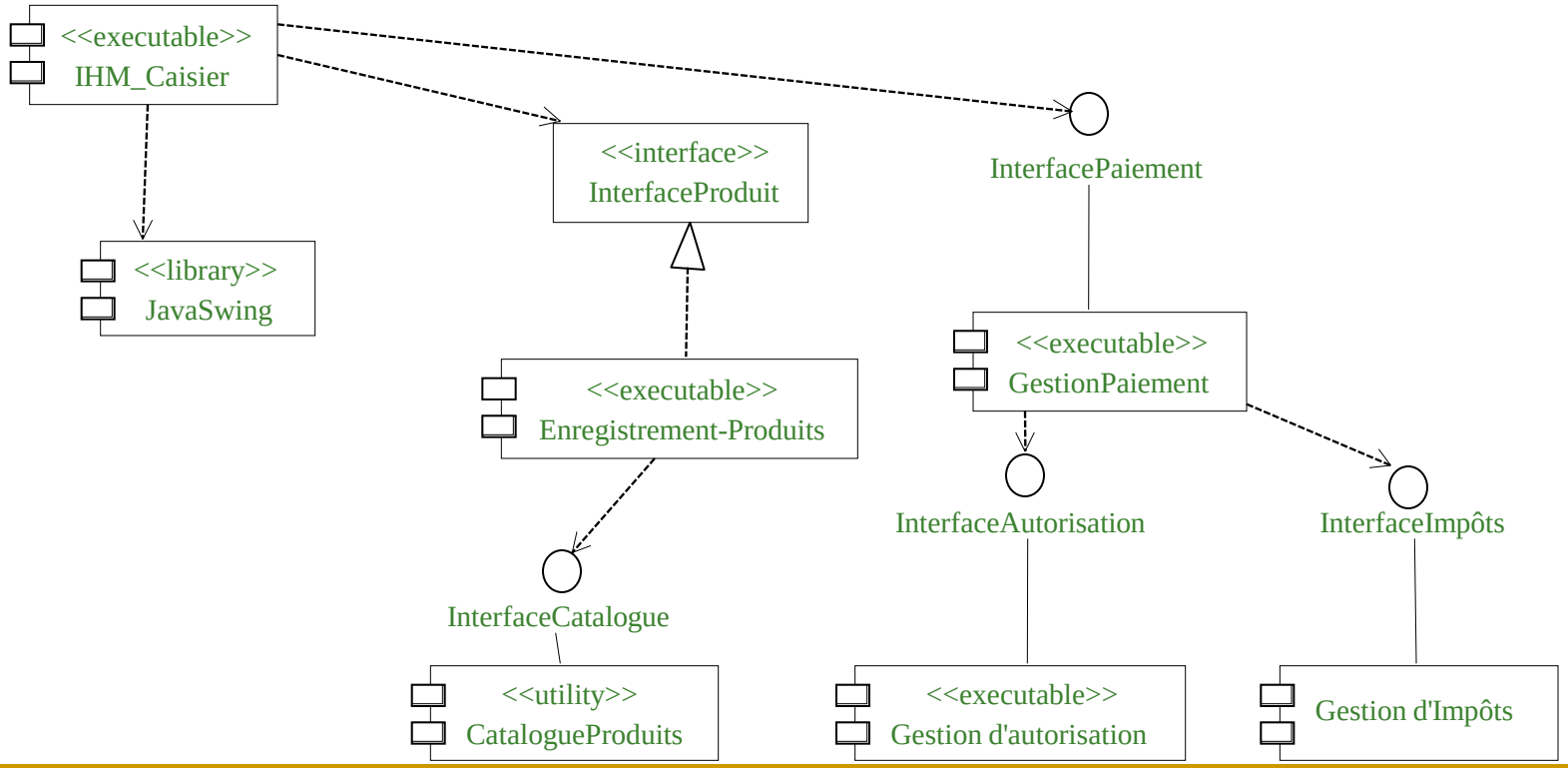


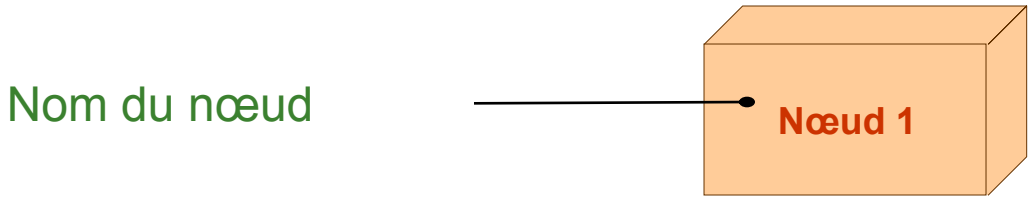
Diagramme de déploiement

- Montre la configuration des nœuds de exécution et des composants qu'y résident
- Montre les relations physiques entre les composants logiciels et matériels d'un système
- Permet de spécifier
  - Nœuds
  - Relations : (dépendance, associations)

Nœud

- Est un élément physique qui existe pendant l'exécution et représente une ressource informatique dans la plupart de cas il s'agit d'un élément matériel
- En général un nœud possède sa propre mémoire et une capacité de traitement
- L'ensemble de composants qui est associé aux nœuds est appelé « *unité de répartition* »
- Les nœuds prennent en charge l'exécution des composants.

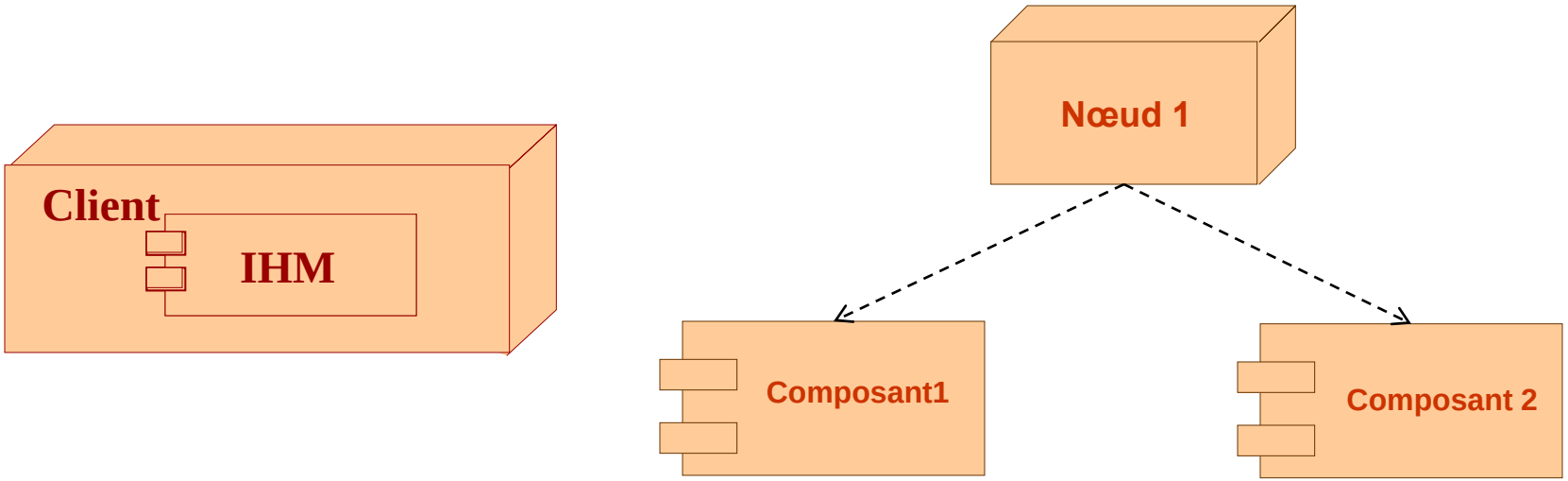
Nœud



- **Nom du nœud** :
  - Permet de distinguer un nœud des autres nœuds
  - Le nom peut être composé du nom de paquetage qui contient le nœud
- **Stéréotypes** : un nœud peut posséder un stéréotype qui permet de le distinguer des autres types de ressources (permettant de spécifier le types de ressources)
  - « *CPU* » : une unité de calcul
  - « *memory* » : une unité de stockage
  - « *device* »: un dispositif tel qu'un capteur

Relations entre nœuds et composants

- **Dépendance**
  - Montre la capacité d'un nœud de supporter un composant
  - Peut être également exprimée entre les composants résidant dans un même nœud
  - Notation UML



Relations entre deux nœuds

- Association
  - Indiquent une voie physique entre deux nœuds
  - Exemple:
    - Une connexion Ethernet
    - Un bus de communication
  - Notation UML

Diagramme de déploiement (Exemple)

- Système Vente

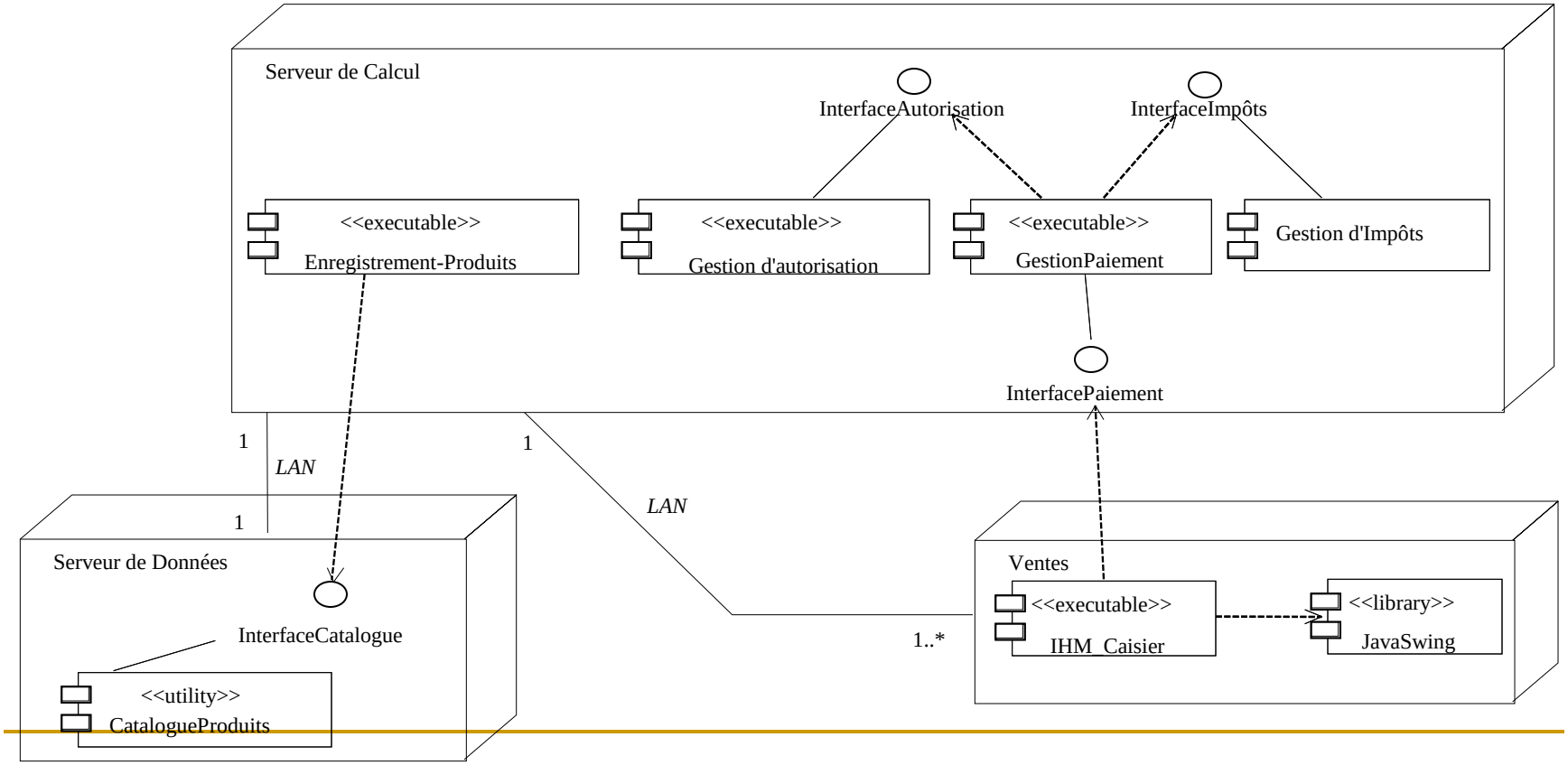


Diagramme de déploiement

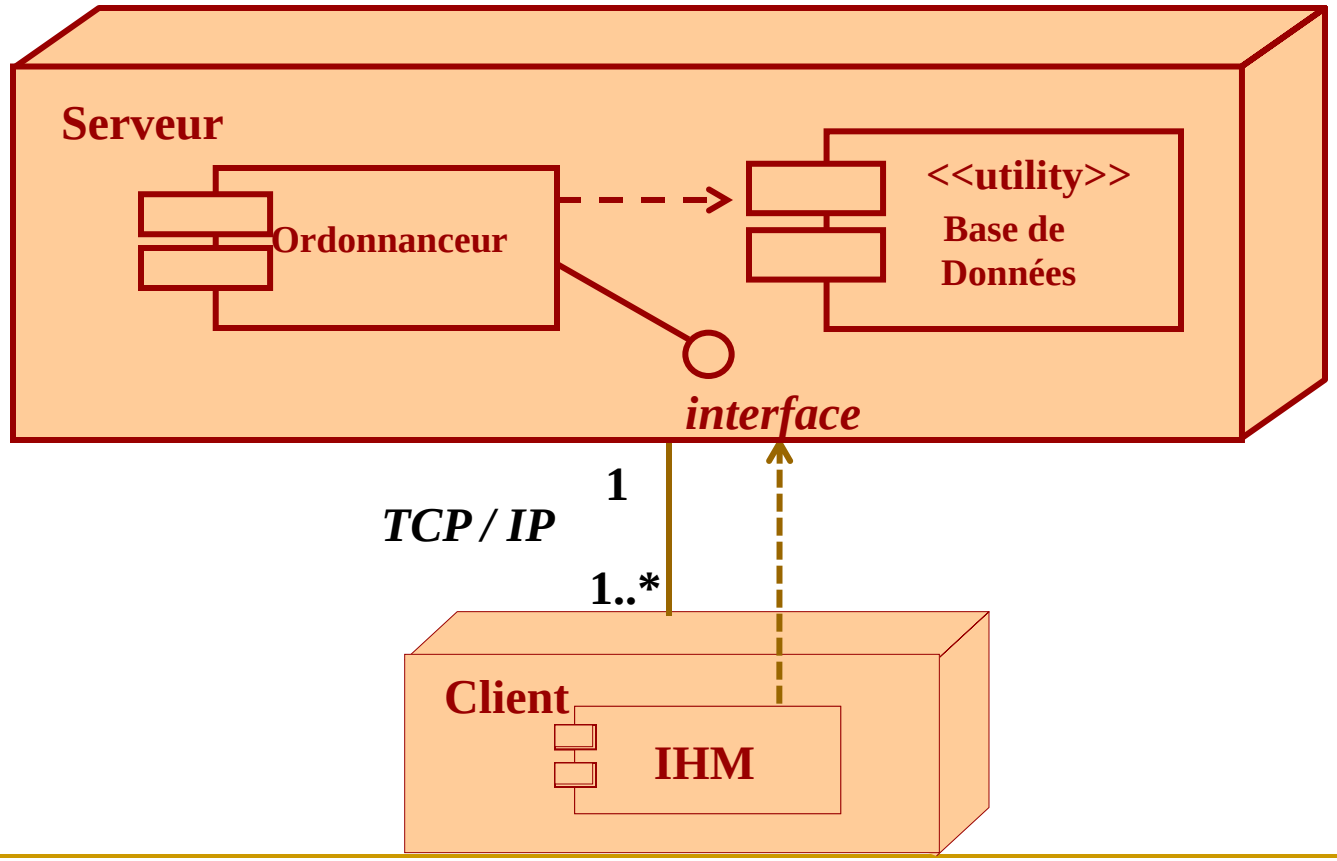


Diagramme de déploiement