

## PLAN DU COURS

### *Méthodes pédagogiques*

Cours magistral, étude de cas, projets

### *Contrôle des connaissances*

Contrôle continu au travers des travaux pratiques et des interrogations.  
Examen final.  
Projet final.

### *Durée*

90 heures dont 60 heures de cours, et 30 heures de travaux pratiques.

Après les heures théoriques corroborer le cours par un exemple englobant les diagrammes nécessaires en passant par Entreprise Architect et le langage C #

### *Contenu*

Objectifs :

Après un rappel des anciennes méthodes d'Analyse et de conception, les objectifs seront :

- Analyser et concevoir un projet Objet avec le formalisme UML
- Comprendre la représentation et l'intérêt d'utilisation de chaque diagramme
- Savoir progresser de l'analyse à la conception et assimiler un raisonnement itératif et incrémental basé sur les cas d'utilisation.

Ce cours permettra de traiter les points suivants :

- Positionnement des méthodes et des méthodologies dans une démarche de production de logiciel.
- La notation UML et les différents diagrammes. Représentation des diagrammes les plus importants et les plus pertinents.
- Concepts avancés de cette notation de type framework, analysis and design patterns...

*Ce cours permettra de :*

*Apprendre à concevoir une application informatique au sein d'une entreprise qui répond effectivement aux besoins des utilisateurs ;*

*Apprendre une manière professionnelle de travailler.*

### **Bibliographie**

**Hugues Bersini , L'orienté objet - Cours et Exercices en UML2**

**GILLE ROY, Conception BD avec UML**

**Joseph Bagay & David Bagay UML 2 Analyse et conception dunod, paris 2008**

**Christian Soutou, UML 2 pour les bases de données, Eyrolles**

**Pascal Roques, les cahiers du programmeur**

**Pascal Roques UML 2 par la pratique**

## INTRODUCTION

### 1. SYSTEME D'INFORMATION

C'est une erreur couramment commise que de considérer les termes informatique et systèmes d'information comme étant des synonymes.

L'angle de **vue système d'information est celui du manager, qui a des besoins** de traitement de données et qui est en position de maîtrise d'ouvrage ou **client**.

A l'opposé, l'angle de **vue informatique est celui de la maîtrise d'œuvre ou fournisseur, qui offre des outils techniques (biens et services), qui doivent être capables de satisfaire les besoins** de la maîtrise d'ouvrage.

Comme sur tout marché où se confrontent l'offre et la demande, c'est l'objectif de **satisfaction des besoins** qui doit être prioritaire (application de l'optique marketing). Actuellement, on constate encore que le marché des outils appartenant au système d'information est dominé par l'offre (maîtrise d'œuvre, représentée par informaticiens de l'organisation ou les éditeurs et intégrateurs de progiciels). Pour rendre les gestionnaires maîtres de leur système d'information et aptes à posséder des outils satisfaisant leurs besoins, il faut tout d'abord bien comprendre le concept de système d'information.

Ce concept de système d'information, loin d'être né avec l'informatique, a son origine dans un courant de pensée de la systémique. Pour comprendre le **concept de système d'information**, il est donc fondamental de se référer au courant philosophique du constructivisme et de le comparer aux courants opposés que sont le cartésianisme et le positivisme.

### 2. MODELISATION

Un modèle est une représentation artificielle de ce que l'on pense avoir compris du monde environnant. Il :

- possède trois propriétés :

- la figuration : les figures sont mises à la place de concepts généraux

- l'imitation : il copie sur un support des relations perçues sur l'environnement

- la formalisation : il propose de mettre de l'ordre dans la diversité observée ;

- sert :

- à communiquer : voir si on a bien compris la même chose que les utilisateurs

- à préparer la réalisation. Un modèle peut dire deux choses :

- ce que l'application devra faire (une spécification)

- comment elle est organisée du point de vue de l'ordinateur (une

réalisation).

- Un modèle est une abstraction de la réalité. L'abstraction est un des piliers de l'approche objet.
  - ✓ Il s'agit d'un processus qui consiste à identifier les caractéristiques intéressantes d'une entité, en vue d'une utilisation précise.
  - ✓ L'abstraction désigne aussi le résultat de ce processus, c'est-à-dire l'ensemble des caractéristiques essentielles d'une entité, retenues par un observateur.
- Un modèle est une vue subjective mais pertinente de la réalité

- ✓ Un modèle définit une frontière entre la réalité et la perspective de l'observateur. Ce n'est pas "la réalité", mais une vue très subjective de la réalité.
  - ✓ Bien qu'un modèle ne représente pas une réalité absolue, un modèle reflète des aspects importants de la réalité, il en donne donc une vue juste et pertinente.
- Caractéristiques fondamentales des modèles

Le caractère abstrait d'un modèle doit notamment permettre :

- ✓ de faciliter la compréhension du système étudié  
→ Un modèle réduit la complexité du système étudié.
- ✓ de simuler le système étudié  
→ Un modèle représente le système étudié et reproduit ses comportements.

Un modèle réduit (décompose) la réalité, dans le but de disposer d'éléments de travail exploitables par des moyens mathématiques ou informatiques :  
modèle / réalité ~ digital / analogique

### *Pourquoi modéliser ?*

**Modéliser un système avant sa réalisation permet de mieux comprendre le fonctionnement du système.** C'est également un bon moyen de maîtriser sa complexité et d'assurer sa cohérence. **Un modèle est un langage commun, précis, qui est connu par tous les membres de l'équipe et il est donc, à ce titre, un vecteur privilégié pour communiquer.** Cette communication est essentielle pour aboutir à une compréhension commune aux différentes parties prenantes (notamment entre la maîtrise d'ouvrage et la maîtrise d'œuvre informatique) et précise d'un problème donné.

Dans le domaine de l'**ingénierie du logiciel**, le modèle permet de mieux répartir les tâches et d'automatiser certaines d'entre elles. C'est également un facteur de réduction des coûts et des délais. Par exemple, les plateformes de modélisation savent maintenant exploiter les modèles pour faire de la génération de code (au moins au niveau du squelette) voire des allers-retours entre le code et le modèle sans perte d'information. Le modèle est enfin indispensable pour assurer un bon niveau de qualité et une maintenance efficace. En effet, une fois mise en production, l'application va devoir être maintenue, probablement par une autre équipe et, qui plus est, pas nécessairement de la même société que celle ayant créé l'application.

**Le choix du modèle a donc une influence capitale sur les solutions obtenues. Les systèmes non-triviaux sont mieux modélisés par un ensemble de modèles indépendants. Selon les modèles employés, la démarche de modélisation n'est pas la même.**

### 3. LES GRANDES LIGNES DE L'HISTOIRE DE LA CONCEPTION DE SI

1. Les premières méthodes d'analyse (années 70)
  - Découpe cartésienne (fonctionnelle et hiérarchique SA/SD, SADT) d'un système.
2. L'approche systémique (années 80)
  - Modélisation des données + modélisation des traitements (Merise, Axial, IE...).

3. L'émergence des méthodes objet (1990-1995)
  - Prise de conscience de l'importance d'une méthode spécifiquement objet : comment structurer un système sans centrer l'analyse uniquement sur les données ou uniquement sur les traitements (mais sur les deux) ?
4. Premiers consensus (1995)
  - **OMT** (James Rumbaugh) : vues statiques, dynamiques et fonctionnelles d'un système
  - **OOD** (GradyBooch) : vues logiques et physiques du système
  - **OOSE** (Ivar Jacobson) : couvre tout le cycle de développement
5. L'unification et la normalisation des méthodes (1995-1997)
  - UML (Unified Modeling Language), la fusion et synthèse des méthodes dominantes.

### Chapitre I : DE SADT A MERISE

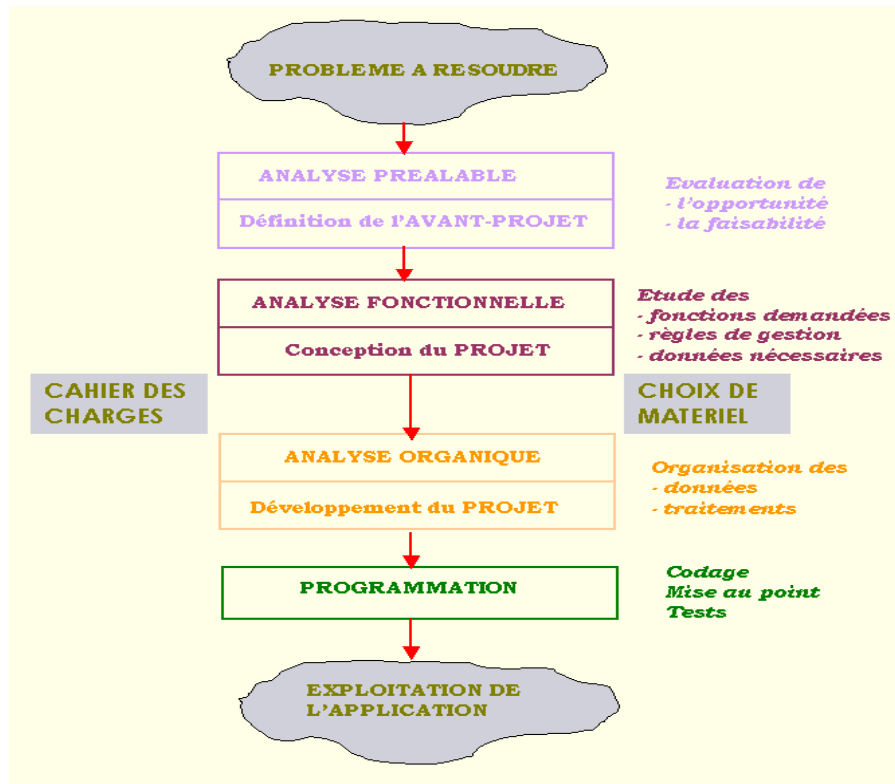
#### I.1. CRITIQUES : les premières méthodes

##### 1. SADT : *Structured Analysis and Design Technique.*

Comme dans tous les métiers de l'ingénieur, une démarche de développement d'un projet est prescrite

- qui doit respecter certaines étapes dans un ordre déterminé
- qui doit produire certains résultats à la fin de chaque étape

Cette démarche SADT est à l'origine d'un vocabulaire encore utilisé aujourd'hui, par exemple la distinction entre le *fonctionnel* et l'*organique* :



## Remarques :

On observera trois présupposés implicites (et tenaces) :

- il s'agit d'une démarche de résolution de problème applicable en toutes circonstances.
- chaque étape est un pas d'un algorithme (simple) conduisant progressivement à la solution.
- le procédé consiste à passer de notions informelles (nébulosité du problème) à quelque chose de strictement formalisé (des programmes mis en exploitation).

### a. Difficultés.

Outre les problèmes techniques de coopération entre tâches très divisées qui ont déjà été mentionnées, le développement des systèmes informatiques par empilement d'applications indépendantes va devenir d'une complexité très difficile à maîtriser.

### Les caractéristiques principales sont :

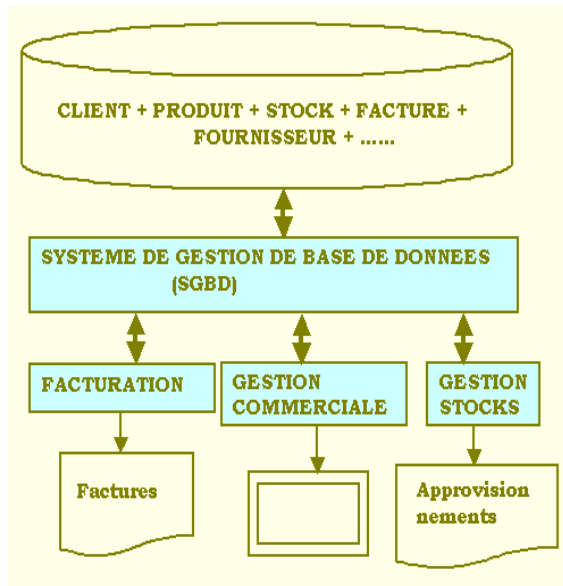
- **la redondance des informations.** Dans l'exemple précédent, la notion de produit est connue dans 3 applications différentes, sous des facettes variables.
- **La lourdeur des mises à jour :** un changement de prix de vente par exemple va concerner l'application de facturation et l'application commerciale. Au moins 2 fichiers seront concernés.
- **L'incohérence des informations** résulte de la difficulté précédente. Si la mise à jour n'est pas faite simultanément partout où elle devrait l'être dans le système d'information, celui-ci devient globalement incohérent. Par exemple un client supprimé en Facturation rentre toujours dans les calculs de ratio de CA parce qu'il n'a pas été supprimé en Gestion Commerciale.

De graves problèmes de maintenance des applications vont se manifester :

### b. Solution :

#### L'idée d'une indispensable modélisation et la solution de la base de données centrale

Pour éviter les complications qui résultent de l'empilement d'applications indépendantes on va proposer une architecture radicalement différente :



#### Caractéristiques :

- Une donnée ne figure qu'une seule fois (et une fois pour toutes) dans la base
- Les données sont partagées par toutes les applications et par voie de conséquence par les utilisateurs

#### Contrainte :

Il faut intercaler une couche logicielle, le SGBD, chargé d'assurer l'accès concurrent aux données, l'intégrité et le maintien de cohérence de celles-ci lors des mises à jour (cf la notion d'atomicité de transaction).

#### Avantages :

On peut modifier les applications, en ajouter sans remettre en cause l'architecture existante. Les données sont partagées par les utilisateurs (selon les droits d'accès qui leur sont attribués).

Mais se pose alors un problème de conception tout à fait nouveau :

- Comment construire une base qui soit indépendante des applications (actuelles ou à venir) ?
- Comment définir des procédures d'exploitation appropriées ?

#### Solution

#### La préconisation du rapport ANSI-SPARC (1975) relative à la structuration des données :

Selon ce rapport, 3 niveaux de structuration doivent être distingués :

- un niveau conceptuel, indépendant des besoins et de l'état de l'art technologique.

## CONCEPTION DES SYSTEMES D'INFORMATION L1 ISC-GOMA 2016-2017

- un niveau externe qui concerne le point de vue de l'utilisateur sur les données.
- un niveau interne qui concerne la représentation physique des données pour une technologie particulière.

On connaît aussi quelques modèles théoriques :  
- Le modèle relationnel : Codd (1970)  
- Le modèle Entité-Association : Chen (1976)  
- Le formalisme des Réseaux de Petri.  
(Se souvenir aussi que les BD n'ont pas commencé avec le modèle relationnel. On a développé antérieurement des SGBD navigationnels).

### 1.2.MERISE

#### Historique

1977 : Le Ministère français de l'Industrie finance le travail de définition d'une méthode d'intérêt national auquel participent :  
des SSCI,  
- le CETE (Centre d'Etudes Techniques du Ministère de l'Equipeement),  
- des universitaires et ingénieurs d'Aix-en-Provence ;

1979-1980 : Lancement dans le domaine public : entreprises et administrations

1988 : Selon une étude du cabinet A. Young :  
50% des entreprises françaises utilisent une méthode  
60% d'entre elles utilisent Merise

1994 : Dernier Congrès officiel de la méthode à Versailles

***Très marquée par des fondements théoriques issus de la systémique (ses initiateurs : Hubert Tardieu, D. Nanci, H. Heckenroth ont travaillé étroitement avec Jean-Louis Le Moigne à Aix).***

#### Définitions d'un système

Une définition courante mais excessivement vague est celle d'un ensemble d'éléments en inter-relation les uns avec les autres.

Cette définition met l'accent sur le fait qu'un système est une structure. J.-L. Le Moigne propose une définition en plusieurs facettes et plus complète :

Un système est :

**QUELQUE CHOSE** : n'importe quoi, identifiable

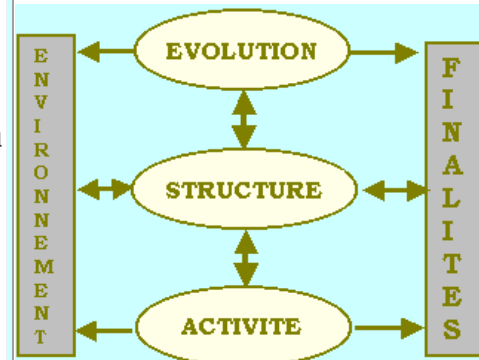
**DANS** autre chose : un environnement

**POUR** quelque chose : une finalité ou projet ou objectif

**QUI FAIT** quelque chose : activité, fonctionnement

**PAR** quelque chose : une structure, une forme stable

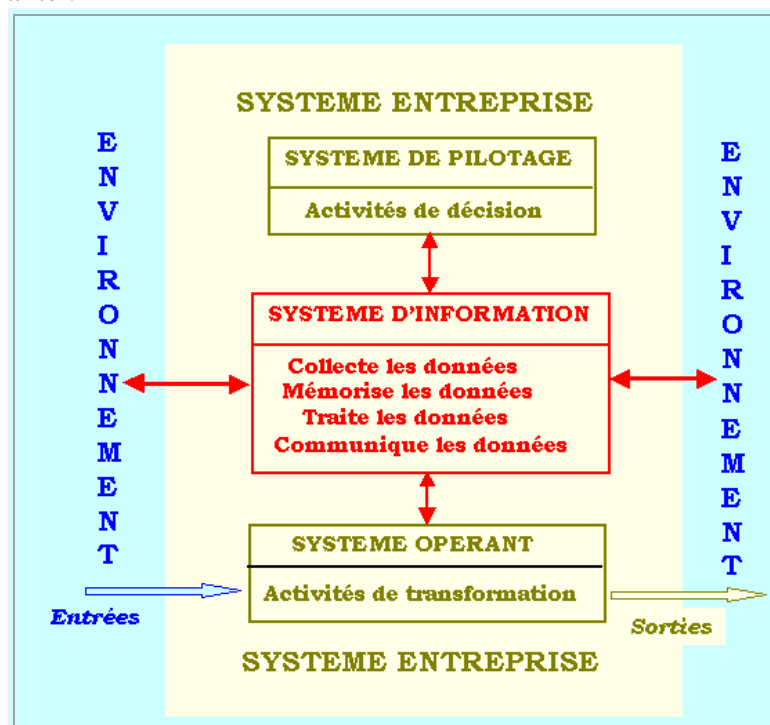
**QUI se transforme** : évolution dans le temps



Et toujours selon J.-L. Le Moigne, un système se construit

### Système d'information d'entreprise

La décomposition d'une Entreprise en sous-systèmes permet de situer son système d'information de la manière suivante :

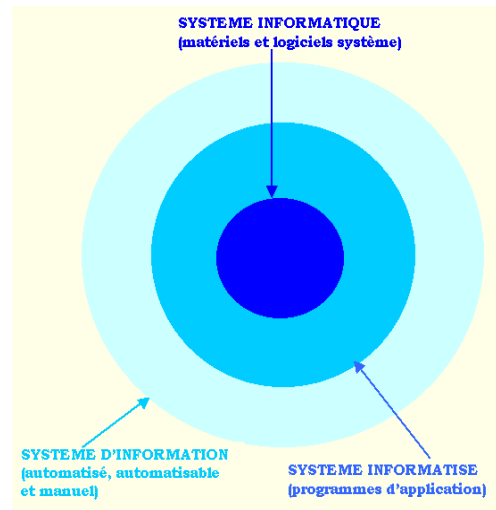


Cette présentation du système d'information peut être discutée :



## CONCEPTION DES SYSTEMES D'INFORMATION L1 ISC-GOMA 2016-2017

- il est défini par différence, "coincé" entre 2 autres sous-systèmes, **pilotage et opérant**. Or, il existe des systèmes d'information de pilotage (aide à la décision) ainsi que des systèmes d'information de l'opérant.
- il ne rend pas compte de l'aspect représentation, pourtant essentiel dans la notion d'information, ce qui expliquerait que différentes vues puissent cohabiter.
- En tout état de cause, un système d'information est plus qu'un système informatique avec lequel il ne doit pas être confondu.



### Merise, le cycle d'abstraction et le cycle de vie

Comme méthode de construction d'un système d'information, Merise développe 3 aspects de manière conjointe :

- le cycle d'abstraction : une réponse à l'objectif de modélisation
- le cycle de vie : la gestion d'un projet d'informatisation
- le cycle de décision : l'insertion des systèmes d'information dans une organisation.

**Le cycle d'abstraction** : concevoir c'est distinguer des couches ou "niveaux d'abstraction" au moyen de modèles. L'objectif est d'isoler l'invariant du système d'information (sa part la plus stable, celle du niveau conceptuel).

NIVEAU	PORTE SUR	UTILISE	MODELISE PAR
CONCEPTUEL	De QUOI s'agit-il ? L'UNIVERS du DISCOURS ou le MONDE REEL	Des règles de GESTION	Un <u>Graphe des Flux</u> et des Acteurs Un <u>Modèle Conceptuel</u> des Données ( <u>MCD</u> ) Un <u>Modèle Conceptuel</u> des Traitements ( <u>MCT</u> )
ORGANISATIONNEL ou LOGIQUE	QUI fait quoi ? OU ?	Des règles d'ORGANISATION	Un <u>Modèle</u> Organisationnel des

## CONCEPTION DES SYSTEMES D'INFORMATION L1 ISC-GOMA 2016-2017

	QUAND ?		Traitements (MOT) Un Modèle Logique des Données (MLD)
OPERATIONNEL PHYSIQUE	ou COMMENT ?	La technologie du moment	Un Modèle Opérationnel des Traitements (MOpT) Un modèle Physique des Données (MPD)

Dans une démarche de construction partiellement itérative : La description séparée des données et des traitements permet de confronter les vues externes (Modèles externes et MCTA, MOTA de Merise 2) issues des traitements pour corriger le modèle de données conçu antérieurement :

**Le cycle de vie** : est une préconisation de la manière de conduire le projet de développement d'un système informatique.

A ce titre, il fournit un procédé (au sens où l'on parle de génie des procédés dans une industrie) et représente un savoir-faire.

Il est remarquable que cette préconisation (empruntée aux méthodes d'informatisation qui ont précédé Merise) se rencontre sous une forme semblable dans la plupart des disciplines de l'ingénieur (par ex. en Génie Civil).

Le projet est découpé en **6 étapes** séparées et qui se succèdent dans le temps :

**1- Le schéma directeur** consiste à définir les finalités du système d'information en fonction de la stratégie de l'organisation. Il définit des priorités qui s'inscrivent dans un découpage en domaines (que certains ont aussi appelé une cartographie cf. Urbanisation) du système d'information.

**2- L'étude préalable** modélise les solutions techniques et organisationnelles susceptible de répondre au problème posé. **Elle doit donc passer en revue tous les modèles du cycle d'abstraction (MCD, MCT, MLD, MOT,...), mais selon une portée particulière : un sous-ensemble représentatif du domaine.**

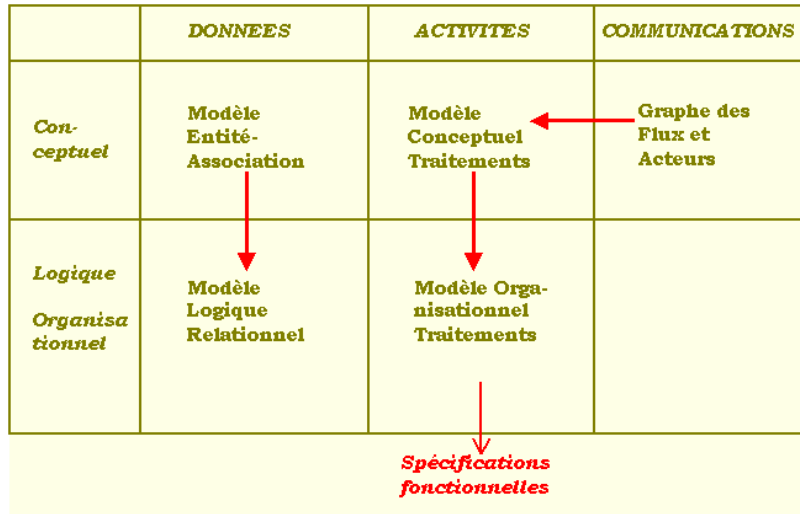
Par exemple, s'il s'agit de développer un système de commerce électronique, l'étude préalable pourra consister à étudier de manière approfondie un modèle physique des communications sous l'angle de la sécurité mais se contenter d'un MCD sommaire. **Il ne faut donc pas confondre une étude préalable avec un dégrossissement plus ou moins vague du niveau conceptuel. Elle doit produire les éléments qui permettront de choisir entre différentes solutions organisationnelles, mais aussi techniques.**

**3- L'étude détaillée** consiste à fournir les spécifications fonctionnelles détaillées du futur système. Soit quelque chose qui ressemble à **un Cahier des Charges**.

**Rétroaction des spécifications sur les modèles de données**

Reprenons l'ensemble des modèles exposés jusqu'à présent et examinons leurs relations. Dans la méthode Merise, c'est ce qu'on appelle le *cycle d'abstraction*.

*Articulation des modèles :*



On peut dériver un MLD d'un MCD au moyen de règles, dériver un MOT d'un MCT par raffinement mais ce qui est caractéristique est l'indépendance des deux colonnes DONNEES / ACTIVITES. D'où l'apport de **Merise 2** par ces modèles analytiques (cf Camille Moine). Dans ce cours, nous resterons dans le cadre stricte de **Merise classique**.

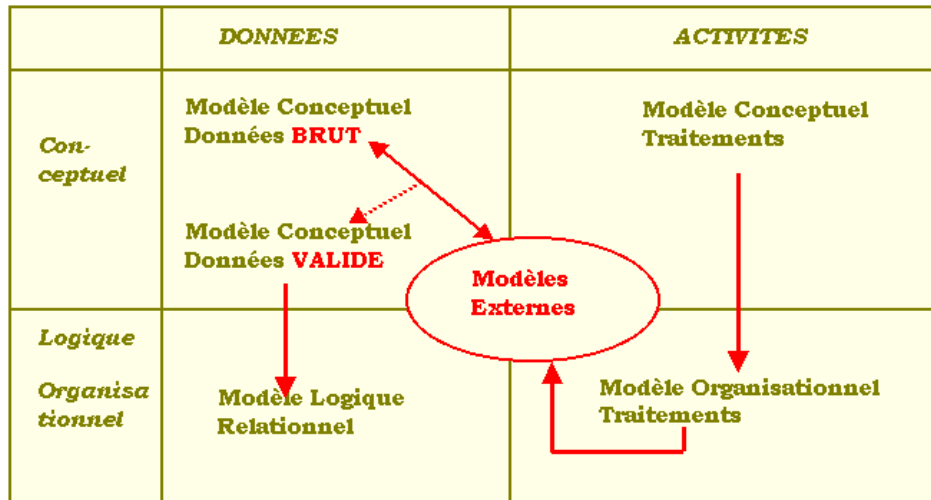
Cette caractéristique est commune à toute une famille de méthodes, pas seulement à Merise, dont le but est le plus souvent un logiciel architecturé autour d'une Base de Données. On peut citer par exemple SSADM en Grande Bretagne.

*Validation :*

Ayant étudié séparément les aspects statiques (données) et les aspects dynamiques (activités) du système d'information, on va pouvoir procéder à une référence croisée entre les deux pour valider que :

- 1) les données nécessaires aux traitements se trouvent en bonne et due forme dans le MCD,
- 2) elles sont bien toutes utilisées par une procédure fonctionnelle au moins.

On obtient ainsi le rebouclage suivant :



L'objectif est celui d'une mise en cohérence des deux approches d'un système d'information global : les données et les traitements.

D'un point de vue Génie Logiciel, on peut paraphraser en disant que les spécifications fonctionnelles rétro-agissent sur la définition de la structure des données.

Le moyen de l'opération est le concept de "**modèle externe**".

**4- L'étude technique** consiste à produire le logiciel : codage des programmes, leur test et leur mise au point.

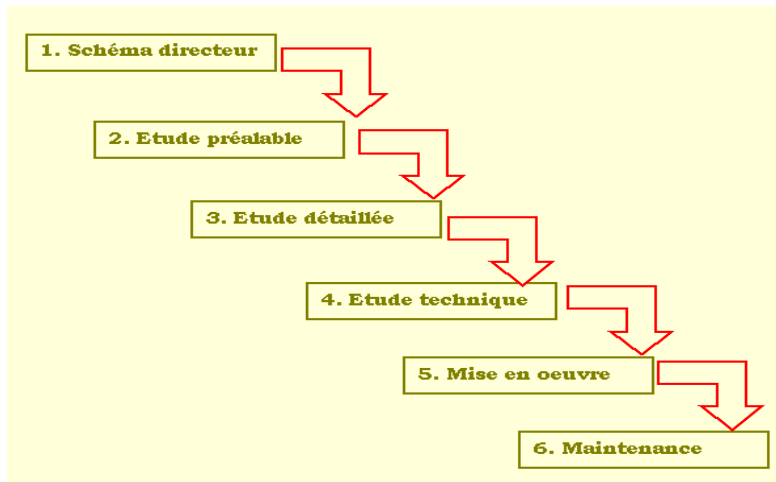
**5- La mise en œuvre** consiste à déployer l'application auprès des utilisateurs, mettre en place les moyens humains (formation, recrutement), organisationnels et techniques.

**6- La maintenance** consiste à corriger les erreurs qui n'auraient pas été détectées par les tests et à faire évoluer l'application pour répondre à d'éventuels nouveaux besoins des utilisateurs.  
**Remarque :** que cette étape soit explicitement préconisée par la méthode revient à reconnaître que malgré l'effort important de modélisation et d'abstraction, tout n'est pas définitivement réglé pour autant. Cf Cours de Génie logiciel.

## I.3. Critique de Merise

La forme générale de ce cycle de vie est celui de la cascade que l'on peut résumer ainsi :

**Le modèle dit de la cascade :**



### *Caractéristiques*

Le projet est découpé en étapes qui se succèdent dans le temps : on passe à l'étape suivante quand la précédente est terminée. C'est probablement ce cycle de vie qui a fait de Merise une méthode aujourd'hui obsolète.

Il ne faut cependant pas en oublier les avantages :

- chaque étape étant clairement définie, on peut lui associer une production documentable et vérifiable: l'étude préalable doit produire un dossier d'avant projet, l'étude détaillée doit produire un cahier des charges, etc.
- on peut associer à chaque étape une responsabilité humaine (division du travail)
- on peut planifier et suivre l'état d'avancement du travail, mesurer des écarts

L'inconvénient reconnu est l'extrême difficulté de faire "marche-arrière". S'il est certes possible à l'intérieur d'une étape de corriger une erreur, lorsque cette étape est terminée elle l'est définitivement. Par conséquent si l'on détecte une erreur dans les étapes terminales et que celle-ci conduit à remettre en cause une étape amont, il faut refaire tout le travail depuis cette étape amont à travers toutes les étapes intermédiaires qui ont été construites sur la base de la précédente.

**La raison de l'obsolescence de cette manière de faire n'est probablement pas spécifiquement informatique à trois points de vue :**

**1) On s'est aperçu dans nombre de secteurs de l'ingénierie qu'il était possible d'être aussi efficace et rapide en adoptant une ingénierie concourante.** Par exemple, pour construire une usine, commencer l'installation des machines avant que les murs ne soient terminés. **Donc ce qui est en cause ici, est un mode d'organisation du travail en équipe : passer du travail à la chaîne aux îlots de production.**(Cf Génie logiciel :Méthodes Agiles et XP).

**2) Le cycle de vie en cascade s'appuie sur l'hypothèse que l'avenir est prévisible avec certitude.** C'est en particulier la fonction du schéma directeur de Merise lorsqu'il définit des priorités sur une période de l'ordre de trois à cinq ans. Il faut aujourd'hui raisonner en avenir incertain : que devient une étude en cours de développement pour des besoins internes lorsqu'elle doit tout d'un coup répondre aussi aux besoins d'une filiale qui vient d'être absorbée par l'entreprise ?

3) le cycle de vie de Merise est en phase avec une époque où l'informatique était propriété exclusive des informaticiens, représentés au plus haut niveau de l'organisation dans une direction informatique. Ils pouvaient s'engager sur des (gros) projets à financement spécifique. L'informatique est devenue propriété des utilisateurs et de leurs services. Les informaticiens doivent développer des (plus petits) projets financés au même titre que d'autres au sein de ces services. La question devient donc plutôt de faire coopérer un grand nombre de projets diversifiés, et de moins en moins d'en planifier un petit nombre.

### I.4. UML

#### 1. Présentation générale des diagrammes UML 2

UML 2 propose **treize diagrammes** qui peuvent être utilisés dans la description d'un système. Ces diagrammes sont regroupés dans deux grands ensembles.

- **Les diagrammes structurels**

Ces diagrammes, au nombre de six, ont vocation à représenter l'aspect statique d'un système (classes, objets, composants...).

- *Diagramme de classe* – Ce diagramme représente la description statique du système en intégrant dans chaque classe la partie dédiée aux données et celle consacrée aux traitements. C'est le diagramme pivot de l'ensemble de la modélisation d'un système.

- *Diagramme d'objet* – Le diagramme d'objet permet la représentation d'instances des classes et des liens entre instances.

- *Diagramme de composant (modifié dans UML 2)* – Ce diagramme représente les différents constituants du logiciel au niveau de l'implémentation d'un système.

- *Diagramme de déploiement (modifié dans UML 2)* – Ce diagramme décrit l'architecture technique d'un système avec une vue centrée sur la répartition des composants dans la configuration d'exploitation.

- *Diagramme de paquetage (nouveau dans UML 2)* – Ce diagramme donne une vue d'ensemble du système structuré en paquetage. Chaque paquetage représente un ensemble homogène d'éléments du système (classes, composants...).

- *Diagramme de structure composite (nouveau dans UML 2)* – Ce diagramme permet de décrire la structure interne d'un ensemble complexe composé par exemple de classes ou d'objets et de composants techniques. Ce diagramme met aussi l'accent sur les liens entre les sous-ensembles qui collaborent.

- **Les diagrammes de comportement** – Ces diagrammes représentent la partie dynamique d'un système réagissant aux événements et permettant de produire les résultats attendus par les utilisateurs. Sept diagrammes sont proposés par UML :

- *Diagramme des cas d'utilisation* – Ce diagramme est destiné à représenter les besoins des utilisateurs par rapport au système. Il constitue un des diagrammes les plus structurants dans l'analyse d'un système.

- *Diagramme d'état-transition (machine d'état)* – Ce diagramme montre les différents états des objets en réaction aux événements.

- *Diagramme d'activités (modifié dans UML 2)* – Ce diagramme donne une vision des enchaînements des activités propres à une opération ou à un cas d'utilisation. Il permet aussi de représenter les flots de contrôle et les flots de données.

- *Diagramme de séquence (modifié dans UML 2)* – Ce diagramme permet de décrire les scénarios de chaque cas d'utilisation en mettant l'accent sur la chronologie des opérations en interaction avec les objets.

- *Diagramme de communication (anciennement appelé collaboration)* – Ce diagramme est une autre représentation des scénarios des cas d'utilisation qui met plus l'accent sur les objets et les messages échangés.
- *Diagramme global d'interaction (nouveau dans UML 2)* – Ce diagramme fournit une vue générale des interactions décrites dans le diagramme de séquence et des flots de contrôle décrits dans le diagramme d'activités.
- *Diagramme de temps (nouveau dans UML 2)* – Ce diagramme permet de représenter les états et les interactions d'objets dans un contexte où le temps a une forte influence sur le comportement du système à gérer.

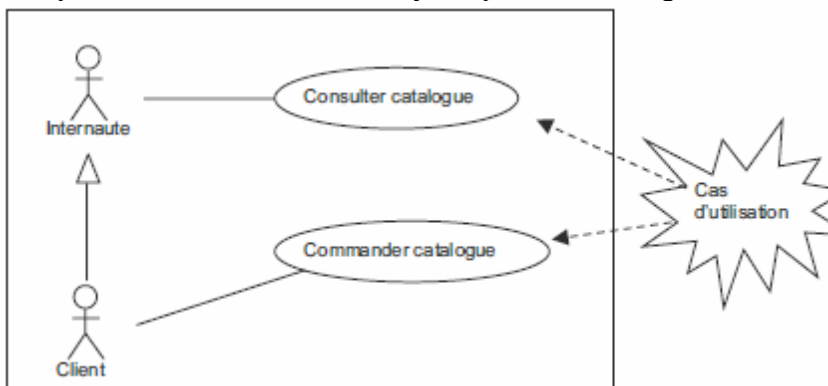
UML 2 décrit les concepts et le formalisme de ces treize diagrammes mais ne propose pas de démarche de construction couvrant l'analyse et la conception d'un système.

### Formalisme et exemple

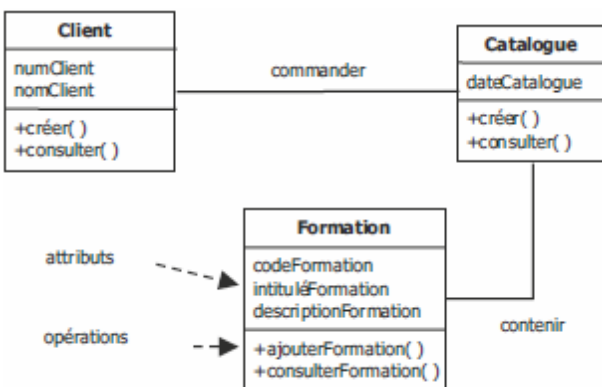
Afin de donner un premier aperçu des principaux diagrammes tant sur l'aspect du formalisme que sur leur usage, nous proposons à titre introductif un petit exemple très simple.

*Considérons une nouvelle société de formation qui souhaite développer un premier niveau de site web dans lequel elle présente succinctement les formations proposées et enregistre en ligne les demandes de catalogue.*

Nous pouvons dès ce stade de l'analyse représenter le diagramme des cas d'utilisation

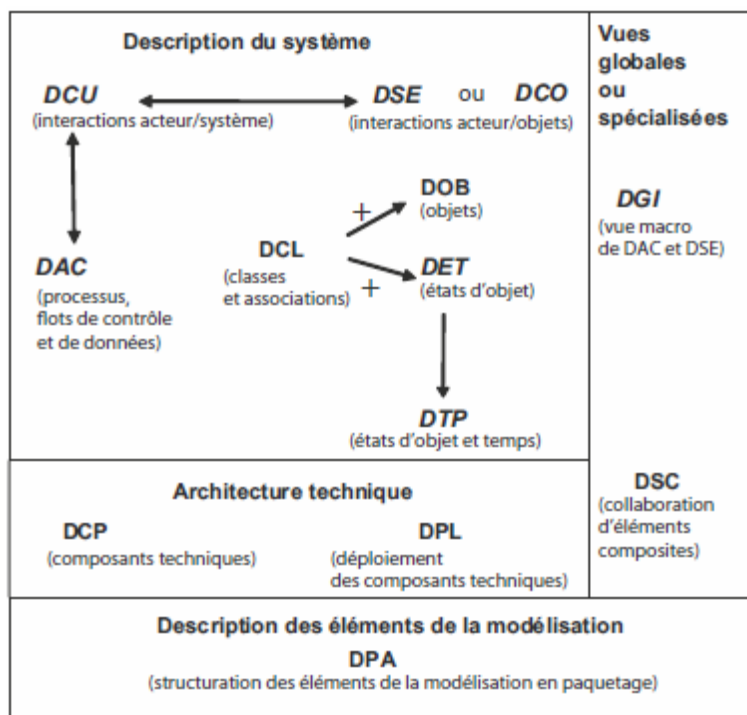
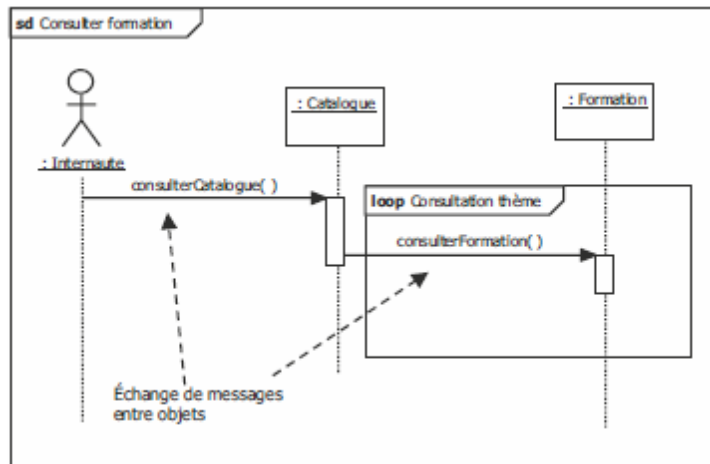


Le diagramme de classe va nous permettre de décrire les concepts manipulés, à savoir : Client, Catalogue et Formation.



## CONCEPTION DES SYSTEMES D'INFORMATION L1 ISC-GOMA 2016-2017

Le diagramme de séquence va nous permettre de décrire les scénarios des cas d'utilisation du diagramme des cas d'utilisation. À titre d'exemple nous montrons le scénario correspondant à la consultation du catalogue.





## CHAPITRE II : LES CAS D'UTILISATION

### Introduction

UML permet de construire **plusieurs modèles d'un système** : certains montrent le système du **point de vue des utilisateurs**, d'autres montrent sa **structure interne**, d'autres encore en donnent une **vision globale ou détaillée**. Les modèles se complètent et peuvent être assemblés. Ils sont élaborés tout au long du cycle de vie du développement d'un système (depuis le recueil des besoins jusqu'à la phase de conception). Dans ce chapitre, nous allons étudier un des modèles, en l'occurrence le premier à construire : **le diagramme de cas d'utilisation**. Il permet de **recueillir, d'analyser et d'organiser les besoins**. Avec lui débute l'étape d'analyse d'un système.

### 2.1. Concepts de base

#### 1. L'expression des besoins

Le **développement d'un nouveau système, ou l'amélioration d'un système existant**, doit répondre à un ou à plusieurs besoins. Par exemple, une banque a besoin d'un guichet automatique pour que ses clients puissent retirer de l'argent même en dehors des heures d'ouverture de la banque. Celui qui commande le logiciel est le maître d'ouvrage. Celui qui réalise le logiciel est le maître d'œuvre.

Le maître d'ouvrage intervient constamment au cours du projet, notamment pour :

- définir et exprimer les besoins ;
- valider les solutions proposées par le maître d'œuvre ;
- valider le produit livré.

Le maître d'œuvre est, par exemple, une société de services en informatique (SSII). Il a été choisi, avant tout, pour ses compétences techniques. Mais son savoir-faire va bien au-delà.

Au **début du projet**, il est capable de **recueillir les besoins** auprès du maître d'ouvrage. Le recueil des besoins implique une bonne compréhension des métiers concernés. Réaliser un logiciel pour une banque, par exemple, implique la connaissance du domaine bancaire et l'intégration de toutes les contraintes et exigences de ce métier. Cette condition est nécessaire pour bien cerner les **besoins exprimés par le client** afin d'apporter les solutions adéquates.

Chaque **cas a ses particularités liées au métier** du client. Le recueil des besoins peut s'opérer de différentes façons. Cela dit, il est recommandé de compléter le cahier des charges par des discussions approfondies avec le maître d'ouvrage et les futurs utilisateurs du système. Il convient également d'**utiliser tous les documents** produits à propos du sujet (rapports techniques ,étude de marché...) et d'**étudier les procédures administratives des fonctions de l'entreprise** qui seront prises en charge par le système. La question que doit se poser le maître d'œuvre durant le recueil des besoins est la suivante : *ai-je toutes les connaissances et les informations pour définir ce que doit faire le système ?*

#### 2. Les cas d'utilisation

Comme le maître d'ouvrage et les utilisateurs ne sont pas des informaticiens, il leur faut donc un moyen simple d'**exprimer leurs besoins**. C'est précisément le rôle des diagrammes de cas d'utilisation. Ils permettent de recenser les grandes fonctionnalités d'un système.

Les cas d'utilisation ont été définis initialement par Ivar Jacobson en 1992 dans sa méthode OOSE. **Les cas d'utilisation constituent un moyen de recueillir et de décrire les besoins des acteurs du système**. Ils peuvent être aussi utilisés ensuite comme moyen d'organisation du développement du logiciel, notamment pour la structuration et le déroulement des tests du logiciel.

### a) Définitions

**Un cas d'utilisation est une manière spécifique d'utiliser un système.** Les acteurs sont à l'extérieur du système ; ils modélisent tout ce qui interagit avec lui. Un cas d'utilisation réalise un service de bout en bout, avec un déclenchement, un déroulement et une fin, pour l'acteur qui l'initie.

**Un acteur est un utilisateur type qui a toujours le même comportement vis-à-vis d'un cas d'utilisation.** Ainsi les utilisateurs d'un système appartiennent à une ou plusieurs classes d'acteurs selon les rôles qu'ils tiennent par rapport au système. Une même personne physique peut se comporter en autant d'acteurs différents que le nombre de rôles qu'elle joue vis-à-vis du système.

Un acteur peut aussi être un système externe avec lequel le cas d'utilisation va interagir.

Un stéréotype représente une variation d'un élément de modèle existant.

### b) Notations et spécification UML

#### ACTEUR

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

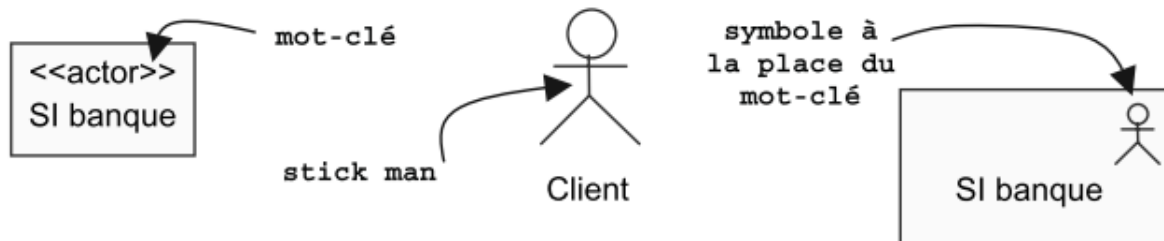
Un acteur peut consulter et/ou modifier directement l'état du système, en émettant et/ou en recevant des messages susceptibles d'être porteurs de données.

#### Comment les identifier ?

Les acteurs candidats sont systématiquement :

- les utilisateurs humains directs : faites donc en sorte d'identifier tous les profils possibles, sans oublier l'administrateur, l'opérateur de maintenance, etc. ;
- les autres systèmes connexes qui interagissent aussi directement avec le système étudié, souvent par le biais de protocoles bidirectionnels.

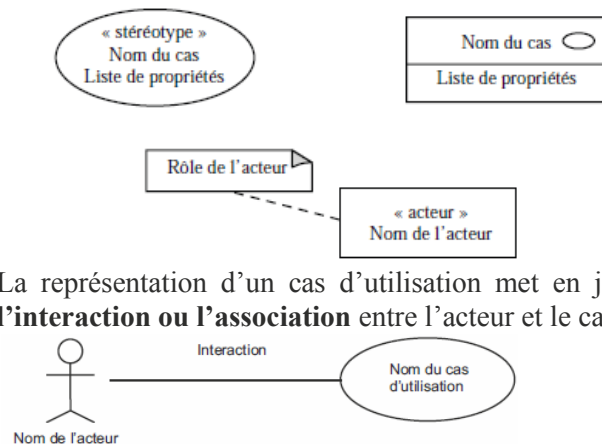
Un acteur peut se représenter symboliquement par un « bonhomme » et être identifié par son nom. Il peut aussi être formalisé par une classe stéréotypée « acteur ».



Un cas d'utilisation se représente par une ellipse. Le nom du cas est inclus dans l'ellipse ou bien il figure dessous. Un stéréotype peut être ajouté optionnellement au-dessus du nom, et une liste de propriétés placée au-dessous.

Un cas d'utilisation se représente aussi sous la forme d'un rectangle à deux compartiments :

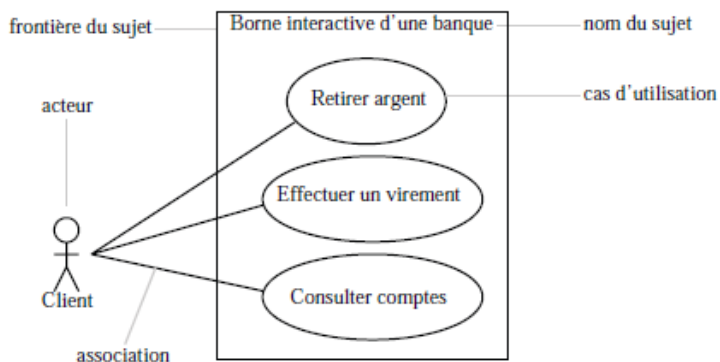
- celui du haut contient le nom du cas ainsi qu'une ellipse (symbole d'un cas d'utilisation) ;
- celui du bas est optionnel et peut contenir une liste de propriétés.



La représentation d'un cas d'utilisation met en jeu trois concepts : **l'acteur**, le **cas d'utilisation** et **l'interaction ou l'association** entre l'acteur et le cas d'utilisation.

## Exemple :

La figure ci-après modélise une borne interactive qui permet d'accéder à une banque. Le système à modéliser apparaît dans un cadre (cela permet de séparer le système à modéliser du monde extérieur). Les utilisateurs sont représentés par des petits bonshommes, et les grandes fonctionnalités (les cas d'utilisation) par des ellipses.



L'ensemble des cas d'utilisation contenus dans le cadre constitue « un sujet ». Les petits bonshommes sont appelés « acteurs ». Ils sont connectés par de simples traits (appelés « associations ») aux cas d'utilisation et mettent en évidence les interactions possibles entre le système et le monde extérieur.

## c) Relations entre cas d'utilisation

Afin d'optimiser la formalisation des besoins en ayant recours notamment à la réutilisation de cas d'utilisation.

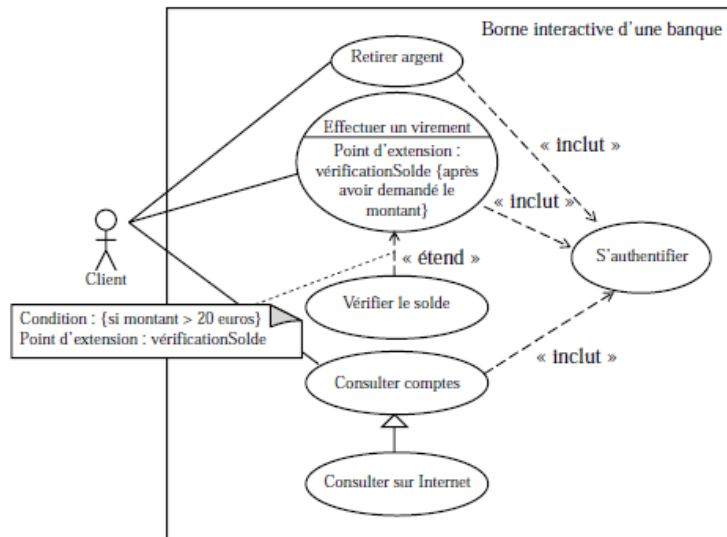
Il existe principalement **deux types de relations** : **les dépendances stéréotypées et la généralisation/spécialisation**. Les dépendances stéréotypées sont des dépendances dont la portée est explicitée par le nom du stéréotype. Les stéréotypes les plus utilisés sont l'inclusion « *include* » et l'extension « *extend* ».

• **La relation d'inclusion** : Un cas A est inclus dans un cas B si le comportement décrit par le cas A est inclus dans le comportement du cas B : on dit alors que le cas B dépend de A.

Cette dépendance est symbolisée par le stéréotype *include*. Par exemple, l'accès aux informations d'un compte bancaire inclut nécessairement une phase d'authentification avec un mot de passe.

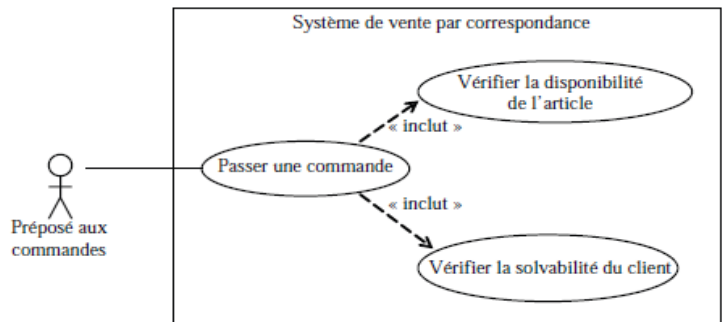
• **La relation d'extension** : Si le comportement de B peut être étendu par le comportement de A, on dit alors que A étend B. Une extension est souvent soumise à condition. Graphiquement, la condition est exprimée sous la forme d'une note. La figure présente l'exemple d'une banque où la vérification du solde du compte n'intervient que si la demande de retrait d'argent dépasse 20 dollars.

• **La relation de généralisation** : Un cas A est une généralisation d'un cas B si B est un cas particulier de A. À la figure, la consultation d'un compte bancaire *via* Internet est un cas particulier de la consultation. Cette relation de généralisation/spécialisation est présente dans la plupart des diagrammes UML et se traduit par le **concept d'héritage** dans les langages orientés objet.



## Remarque

Attention à l'orientation des flèches : si le cas A inclut B on trace la flèche de A vers B, mais si B étend A, la flèche est dirigée de B vers A. Les inclusions permettent aussi de décomposer un cas complexe en sous-cas plus simples.



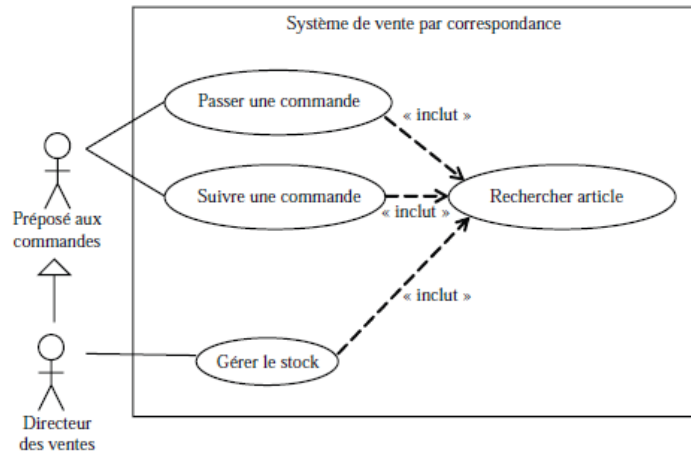
Un cas relié à un autre cas peut ne pas être directement accessible à un acteur. Un tel cas est appelé « cas interne ».

Un cas d'utilisation est dit « interne » s'il n'est pas relié directement à un acteur.

## d) Relations entre acteurs

La seule relation possible entre deux acteurs est la généralisation : un acteur A est une généralisation d'un acteur B si l'acteur A peut-être substitué par l'acteur B (tous les cas d'utilisation accessibles à A le sont aussi à B, mais l'inverse n'est pas vrai).

**Exemple :** Le directeur des ventes est un préposé aux commandes avec un pouvoir supplémentaire (en plus de pouvoir passer et suivre une commande, il peut gérer le stock). Le préposé aux commandes ne peut pas gérer le stock.

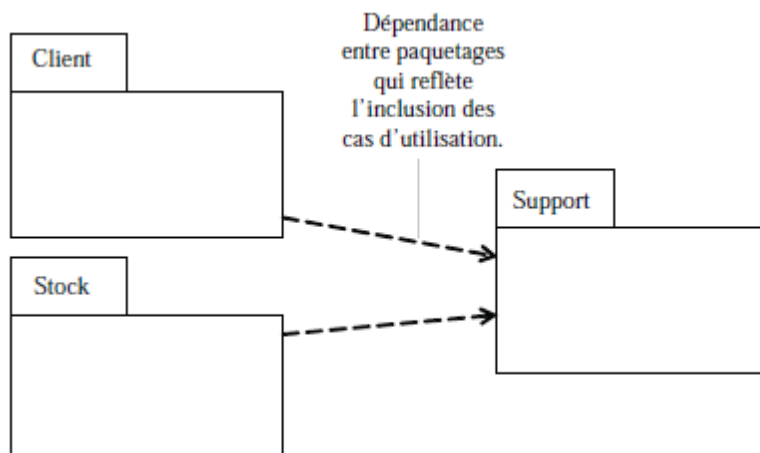


## e) Regroupement des cas d'utilisation en paquetages

UML permet de regrouper des cas d'utilisation dans une entité appelée « paquetage ». Le regroupement peut se faire par acteur ou par domaine fonctionnel. Un diagramme de cas d'utilisation peut contenir plusieurs paquetages et des paquetages peuvent être inclus dans d'autres paquetages.

Un paquetage ( ou package) permet d'organiser des éléments de modélisation en groupe. Un paquetage peut contenir des classes, des cas d'utilisations, des interfaces, etc.

**Exemple :** Trois paquetages ont été créés : Client, Stock et Support. Ces paquetages contiennent les cas d'utilisation du diagramme précédent (Client contient les cas« Passer une commande » et « Suivre une commande », Stock contient le cas « Gérer le stock », tandis que le cas « Rechercher article » est inclus dans le paquetage Support.



## 2.2. Mise en œuvre : Modélisation des besoins avec UML

### 1. Comment identifier les acteurs.

Les principaux acteurs sont les utilisateurs du système. Ils sont en général faciles à repérer. Chaque acteur doit être nommé, mais attention, pour trouver son nom, il faut penser à son **rôle** : **un acteur représente un ensemble cohérent de rôles joués vis-à-vis d'un système**. Plusieurs personnes peuvent avoir le même rôle. C'est le cas des clients d'une banque par exemple. Il n'y aura alors qu'un seul acteur. Réciproquement, une même personne physique peut jouer des rôles différents vis-à-vis du système et donc correspondre à plusieurs acteurs.

Il faut veiller à ne pas oublier les personnes responsables de l'exploitation et de la maintenance du système. Il ne s'agit pas ici des personnes qui installent et paramètrent le logiciel avant sa mise en production, mais des utilisateurs du logiciel dans son fonctionnement nominal.

En plus des utilisateurs, les acteurs peuvent être :

- des périphériques manipulés par le système (imprimantes, robots...) ;
- des logiciels déjà disponibles à intégrer dans le projet ;
- des systèmes informatiques externes au système mais qui interagissent avec lui, etc.

Pour faciliter la recherche des acteurs, on peut imaginer les frontières du système : **Diagramme de contexte**. Tout ce qui est à l'extérieur et qui interagit avec le système est un acteur ; tout ce qui est à l'intérieur est une fonctionnalité du système à réaliser.

### Acteur principal et acteur secondaire

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

**Par exemple**, l'acteur principal d'un cas de *retrait d'argent* dans un distributeur automatique de billets est la personne qui fait le retrait, tandis que la banque qui vérifie le solde du compte est un acteur secondaire.

En général, l'acteur principal initie le cas d'utilisation par ses sollicitations.

## 2. Comment recenser les cas d'utilisation

Il n'y a pas une façon unique de repérer les cas d'utilisation. Il faut se placer du point de vue de chaque acteur et déterminez comment il se sert du système, dans quels cas il l'utilise, et à quelles fonctionnalités il doit avoir accès.

Il faut éviter les redondances et **limiter le nombre de cas** en se situant au bon niveau d'abstraction (par exemple, ne pas réduire un cas à une action).

### Exemple

Considérons un système de réservation et d'impression de billets d'avion via des agences de voyage. En prenant pour acteur une personne qui souhaite obtenir un billet, on peut obtenir la liste suivante des cas d'utilisation :

- rechercher un voyage ;
- réserver une place dans un avion ;
- acheter son billet.

L'ensemble des cas d'utilisation doit couvrir exhaustivement tous les besoins fonctionnels du système. L'étape de recueil des besoins est souvent longue et fastidieuse car les utilisateurs connaissent l'**existant** et n'ont qu'une vague idée de ce que leur apportera un **futur système**. L'élaboration du diagramme de cas d'utilisation permet de pallier ces problèmes en recentrant le système sur les besoins fonctionnels et ce, dès le début du projet.

On peut se demander pourquoi adopter un point de vue fonctionnel, et non technique ?

Avec les diagrammes de cas d'utilisation, on se place clairement du côté des utilisateurs. Le côté technique n'est pas oublié mais abordé différemment : les besoins sont affinés lors de l'écriture des spécifications – on parle de spécifications techniques des besoins (voir la section « Description textuelle des cas d'utilisation »).

### 3. Remarque

L'intérêt des cas d'utilisation ne se limite pas au recueil des besoins. La description des cas d'utilisation peut servir de base de travail pour établir les tests de vérification du bon fonctionnement du système, et orienter les travaux de rédaction de la documentation à l'usage des utilisateurs.

Les spécifications peuvent être divisées en deux catégories selon qu'elles sont fonctionnelles ou techniques. Les spécifications fonctionnelles concernent les fonctions du système, tandis que les spécifications techniques permettent de préciser le contexte d'exécution du système.

Les spécifications fonctionnelles découlent directement du diagramme de cas d'utilisation.

Il s'agit de reprendre chaque cas et de le décrire très précisément. En d'autres termes, vous devez décrire comment les acteurs interagissent avec le système.

### 4. Description des cas d'utilisation

Le diagramme de cas d'utilisation décrit les grandes fonctions d'un système du point de vue des acteurs, mais n'expose pas de façon détaillée le dialogue entre les acteurs et les cas d'utilisation.

UML n'impose rien quant à la façon de décrire un cas d'utilisation. Au lieu d'utiliser une séquence de messages, il est possible d'utiliser les diagrammes d'activités d'UML. Cette liberté de choix peut être déroutante, mais comme UML est censé pouvoir modéliser tout type de système, une manière unique de décrire un cas ne suffirait pas.

#### Description textuelle des cas d'utilisation

Bien que de nombreux diagrammes d'UML permettent de décrire un cas, il est recommandé de rédiger une description textuelle car c'est une forme souple qui convient dans bien des situations. Une description textuelle couramment utilisée se compose de **trois parties**.

**La première partie permet d'identifier** le cas. Elle doit contenir :

- le **nom** du cas ;
- un **résumé** de son objectif ;
- les **acteurs impliqués (principaux et secondaires)** ;
- les **dates de création et de mise à jour** de la description courante ;
- le **nom des responsables** ;
- un **numéro de version**.

**La deuxième partie contient la description du fonctionnement** du cas sous la forme d'une séquence de messages échangés entre les acteurs et le système : c'est le **scénario**. Elle contient toujours une **séquence nominale** qui correspond au fonctionnement nominal du cas (par exemple, un retrait d'argent qui se termine par l'obtention des billets demandés par l'utilisateur). Cette séquence nominale commence par préciser l'événement qui déclenche le cas (l'utilisateur introduit sa carte bancaire par exemple) et se développe en trois points :

- Les **pré-conditions**. Elles indiquent dans quel état est le système avant que se déroule la séquence (le distributeur est alimenté en billets par exemple).
- L'**enchaînement des messages**. Le scénario.
- Les **post-conditions**. Elles indiquent dans quel état se trouve le système après le déroulement de la séquence nominale (une transaction a été enregistrée par la banque par exemple).

Parfois la séquence correspondant à un cas a besoin d'être appelée dans une autre séquence (par exemple quand une relation d'inclusion existe entre deux cas d'utilisation). Signifier l'appel d'une autre séquence se fait de la façon suivante : « appel du cas X », où X est le nom du cas.

Les acteurs n'étant pas sous le contrôle du système, ils peuvent avoir des comportements imprévisibles. La séquence nominale ne suffit donc pas pour décrire tous les comportements possibles.

À la séquence nominale s'ajoutent fréquemment des **séquences alternatives** et des **séquences d'exceptions**. Ces deux types de séquences se décrivent de la même façon que la séquence nominale mais il ne faut pas les confondre. Une séquence alternative diverge de la séquence nominale (c'est un embranchement dans une séquence nominale) mais y revient toujours, alors qu'une séquence d'exception intervient quand une erreur se produit (le séquençement nominal s'interrompt, sans retour à la séquence nominale).

**La troisième partie est une description formelle** par le diagramme de séquence ou le diagramme d'activité.

### 1.2. Exemples

1. *Dans le cas d'un retrait d'argent, des séquences alternatives se produisent par exemple dans les situations suivantes :*

- *Le client choisit d'effectuer un retrait en euros ou en dollars.*
- *Le client a la possibilité d'obtenir un reçu.*

*Une exception se produit si la connexion avec le système central de la banque qui doit vérifier la validité du compte est interrompue.*

La survenue des erreurs dans les séquences doit être signalée de la façon suivante : « appel de l'exception Y » où Y est le nom de l'exception.

La dernière partie de la description d'un cas d'utilisation est une rubrique optionnelle. Elle contient généralement des spécifications non fonctionnelles (ce sont le plus souvent des spécifications techniques, par exemple pour préciser que l'accès aux informations bancaires doit être sécurisé). Cette rubrique contient aussi d'éventuelles contraintes liées aux interfaces homme-machine (par exemple, pour donner la possibilité d'accéder à tous les comptes d'un utilisateur à partir de l'écran principal).

### Description d'un retrait d'argent

#### Identification

Nom du cas : *RetirerArgent en espèces en euros.*

But : détaille les étapes permettant à un guichetier d'effectuer l'opération de retrait d'euros demandé par un client.

Acteur principal : Guichetier.

Acteur secondaire : Système central.

Date : le 03/03/2017.

Responsable : M. Cadet.

Version : 1.0.



## Séquencement

Le cas d'utilisation commence lorsqu'un client demande le retrait d'espèces en euros.

### Pré-conditions

Le client possède un compte (donne son numéro de compte).

### Enchaînement nominal

1. Le guichetier saisit le numéro de compte client.
2. L'application valide le compte auprès du système central.
3. L'application demande le type d'opération au guichetier.
4. Le guichetier sélectionne un retrait d'espèces de 200 euros.
5. L'application demande au système central de débiter le compte.
6. Le système notifie au guichetier qu'il peut délivrer le montant demandé.

### Post-conditions

Le guichetier ferme le compte.

Le client récupère l'argent.

## Rubriques optionnelles

### *Contraintes non fonctionnelles*

Fiabilité : les accès doivent être extrêmement sûrs et sécurisés.

Confidentialité : les informations concernant le client ne doivent pas être divulguées.

Contraintes liées à l'interface homme-machine

Donner la possibilité d'accéder aux autres comptes du client.

Toujours demander la validation des opérations de retrait.

La séquence nominale, les séquences alternatives, les exceptions, etc., font qu'il existe unemultitude de chemins depuis le début du cas jusqu'à la fin. Chaque chemin est appelé« scénario ». Un système donné génère peu de cas d'utilisation, mais, en général, beaucoup de scénarii.

## Exercice 1 : Cas d'utilisation

*Considérons le système informatique qui gère une station-service de distribution d'essence.*

*On s'intéresse à la modélisation de la prise d'essence par un client.*

*1. Le client se sert de l'essence de la façon suivante. Il prend un pistolet accroché à une pompe et appuie sur la gâchette pour prendre de l'essence. Qui est l'acteur du système ?*

*Est-ce le client, le pistolet ou la gâchette ?*

*2. Le pompiste peut se servir de l'essence pour sa voiture. Est-ce un nouvel acteur ?*

*3. La station a un gérant qui utilise le système informatique pour des opérations de gestion.*

*Est-ce un nouvel acteur ?*

*4. La station-service a un petit atelier d'entretien de véhicules dont s'occupe un mécanicien.*

*Le gérant est remplacé par un chef d'atelier qui, en plus d'assurer la gestion, est aussi mécanicien.*

*Comment modéliser cela ?*

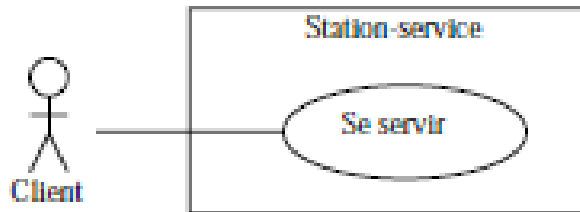
### Solution :

1. Pour le système informatique qui pilote la station-service, le pistolet et la gâchette sont des périphériques matériels. De ce point de vue, ce sont des acteurs ; néanmoins, il est nécessaire de consigner

dans le système informatique l'état de ces périphériques : Pistolet et gâchette doivent donc faire partie du système à modéliser puisqu'au service du client. On opte pour le point de vue du client. Le client agit sur le système informatique quand il se sert l'essence. L'action de se servir constitue une transaction bien isolée des autres fonctionnalités de la station-service. Nous disons donc que « Se servir » est un cas d'utilisation.

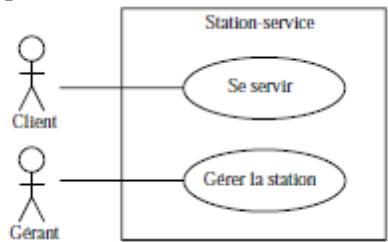
Le client, qui est en dehors du système, devient alors l'acteur principal. Ce cas englobe la prise du pistolet et l'appui sur la gâchette.

Le client est donc l'acteur principal du système.

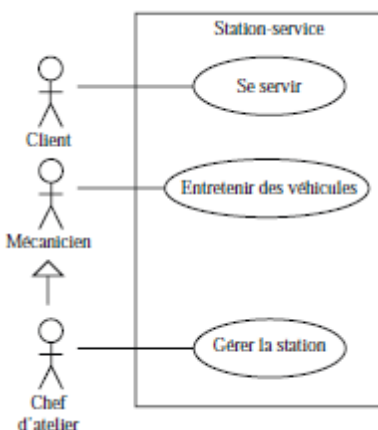


2. Un acteur est caractérisé par le rôle qu'il joue vis-à-vis du système. Le pompiste, bien qu'étant une personne différente du client, joue un rôle identique quand il se sert de l'essence. Pour le cas « Se servir », il n'est pas nécessaire de créer un acteur supplémentaire représentant le pompiste.

3. La gestion de la station-service définit une nouvelle fonctionnalité à modéliser. Le gérant prend le rôle principal ; c'est donc un nouvel acteur.



4. La station offre un troisième service : l'entretien des véhicules. Le système informatique doit prendre en charge cette fonctionnalité supplémentaire. Un nouvel acteur apparaît alors : le mécanicien. Le gérant est à présent un chef d'atelier qui est un mécanicien ayant la capacité de gérer la station. Il y a ainsi une relation de généralisation entre les acteurs *Mécanicien* et *Chef d'atelier* signifiant que le chef d'atelier peut, en plus d'assurer la gestion, entretenir des véhicules.



### Exercice 2 : Identification des acteurs et recensement de cas d'utilisation

*Une bibliothèque universitaire souhaite automatiser sa gestion. Cette bibliothèque est gérée par un gestionnaire chargé des inscriptions et des relances des lecteurs quand ceux-ci n'ont pas rendu leurs ouvrages au-delà du délai autorisé. Les bibliothécaires sont chargés de gérer les emprunts et la restitution des ouvrages ainsi que l'acquisition de nouveaux ouvrages.*

*Il existe trois catégories d'abonné. Tout d'abord les étudiants qui doivent seulement s'acquitter d'une somme forfaitaire pour une année afin d'avoir droit à tous les services de la bibliothèque. L'accès à la bibliothèque est libre pour tous les enseignants.*

*Enfin, il est possible d'autoriser des étudiants d'une autre université à s'inscrire exceptionnellement comme abonné moyennant le versement d'une cotisation.*

*Le nombre d'abonné externe est limité chaque année à environ 10 % des inscrits. Un nouveau service de consultation du catalogue général des ouvrages doit être mis en place.*

*Les ouvrages, souvent acquis en plusieurs exemplaires, sont rangés dans des rayons de la bibliothèque. Chaque exemplaire est repéré par une référence gérée dans le catalogue et le code du rayon où il est rangé.*

*Chaque abonné ne peut emprunter plus de trois ouvrages. Le délai d'emprunt d'un ouvrage est de trois semaines, il peut cependant être prolongé exceptionnellement à cinq semaines.*

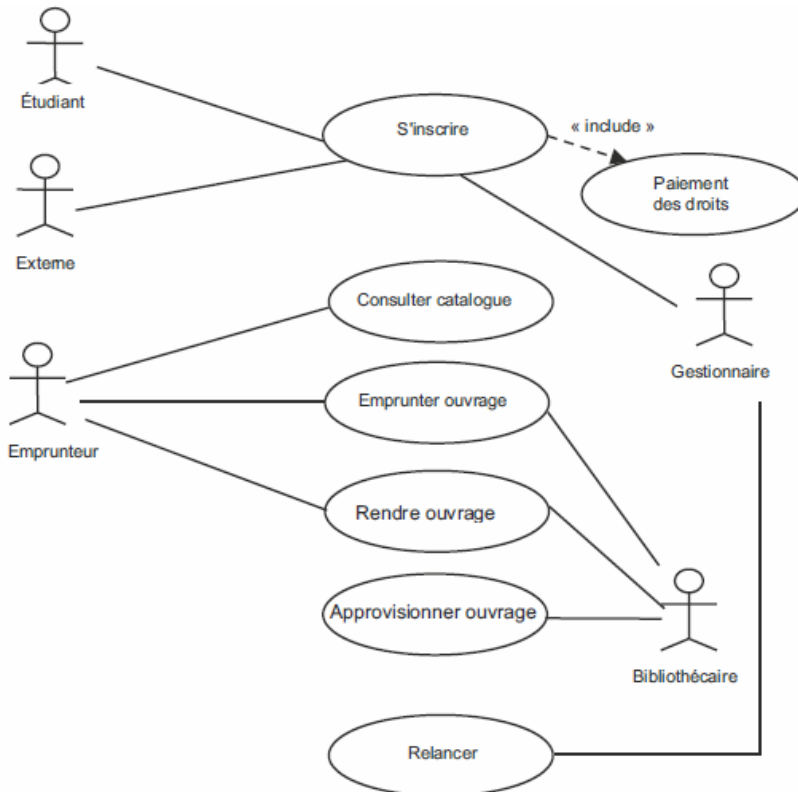
### Solution

Six cas d'utilisation peuvent être identifiés :

- inscription à la bibliothèque,
- consultation du catalogue,
- emprunt d'ouvrages,
- restitution d'ouvrages,
- approvisionnement d'ouvrages,
- relance emprunteur.

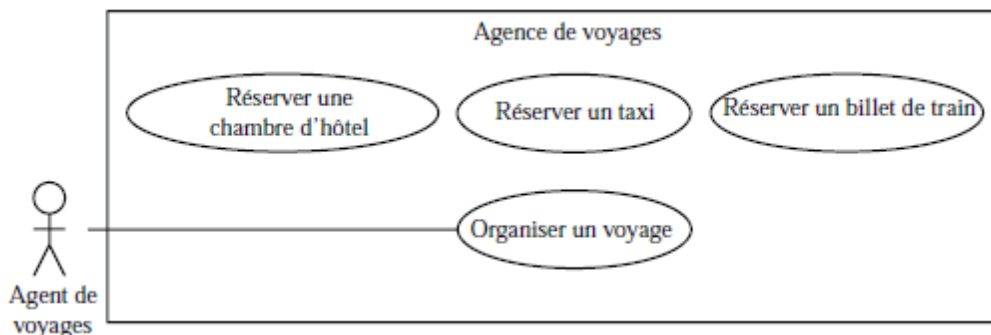
Cinq types d'acteurs peuvent être identifiés :

- étudiant,
- externe,
- emprunteur,
- gestionnaire,
- bibliothécaire.



## Exercice 3 : Relations entre cas d'utilisation – cas internes. Modélisation d'une agence de voyage

1. Une agence de voyages organise des voyages où l'hébergement se fait en hôtel. Le client doit disposer d'un taxi quand il arrive à la gare pour se rendre à l'hôtel.



**Diagramme incomplet des cas d'utilisation d'une agence de voyages.**

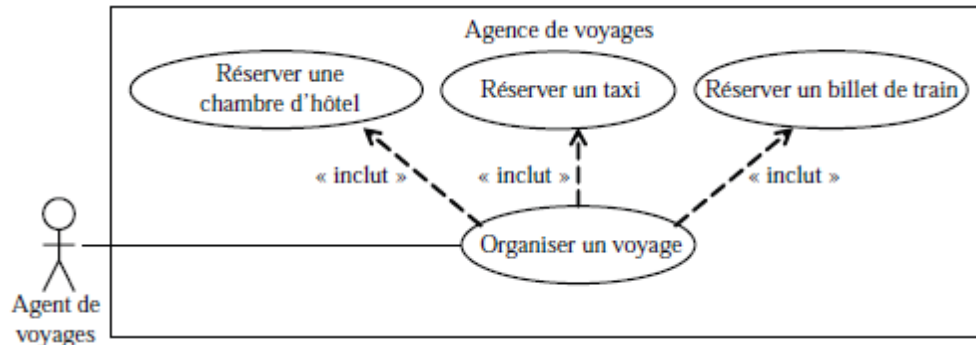
2. Certains clients demandent à l'agent de voyages d'établir une facture détaillée. Cela donne lieu à un nouveau cas d'utilisation appelé « Établir une facture détaillée ». Comment mettre ce cas en relation avec les cas existants ?

3. Le voyage se fait soit par avion, soit par train. Comment modéliser cela ?

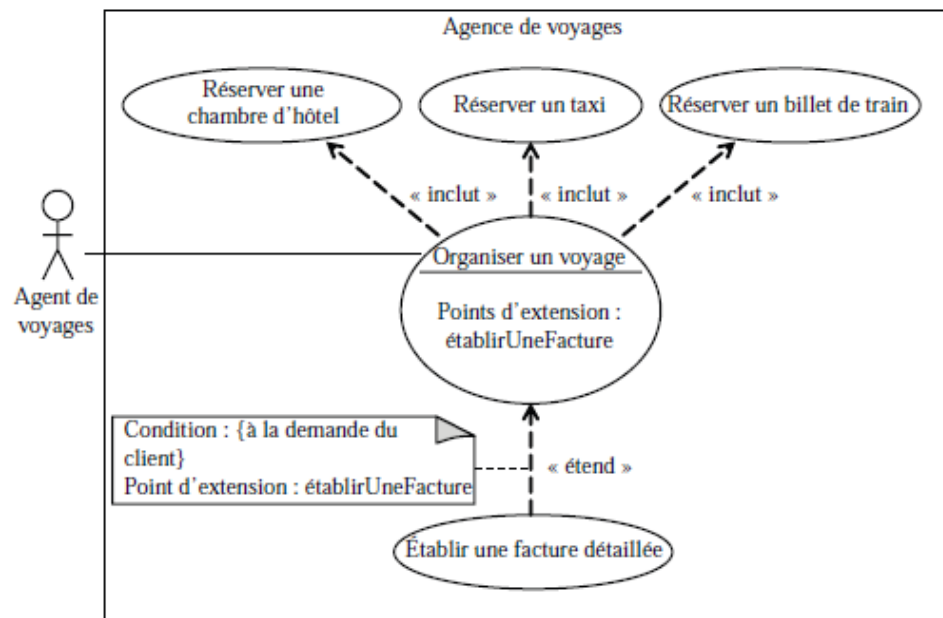
## Solution

1. Le modélisateur a considéré que l'organisation d'un voyage est trop complexe pour être représentée par un seul cas d'utilisation. Il l'a donc décomposée en trois tâches modélisées par les trois cas d'utilisation « Réserver une chambre d'hôtel », « Réserver un taxi » et « Réserver un

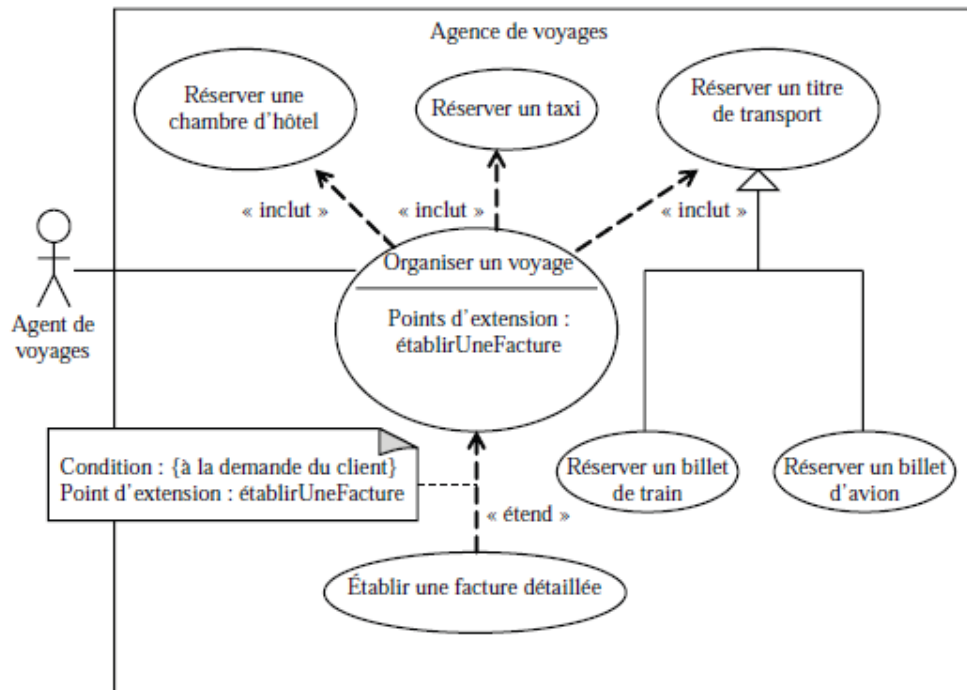
billet de train ». Ces trois tâches forment des transactions suffisamment isolées les unes des autres pour être des cas d'utilisation. De plus, ces cas sont mutuellement indépendants. Quel est le défaut de ce diagramme de CU ?



2. L'établissement d'une facture détaillée se fait uniquement sur demande du client. Ce caractère optionnel est modélisé par une relation d'extension entre les cas « Organiser un voyage » et « Établir une facture détaillée ». L'extension porte la condition « à la demande du client ».



3. Il y a maintenant deux cas particuliers : le voyage se fait en train ou en avion. Ces cas particuliers sont modélisés par les cas « Réserver un billet de train » et « Réserver un billet d'avion ». Ceux-ci sont liés à un cas plus général appelé « Réserver un titre de transport ».



#### Exercice 4 : Relations entre acteurs, extensions conditionnelles entre cas d'utilisation

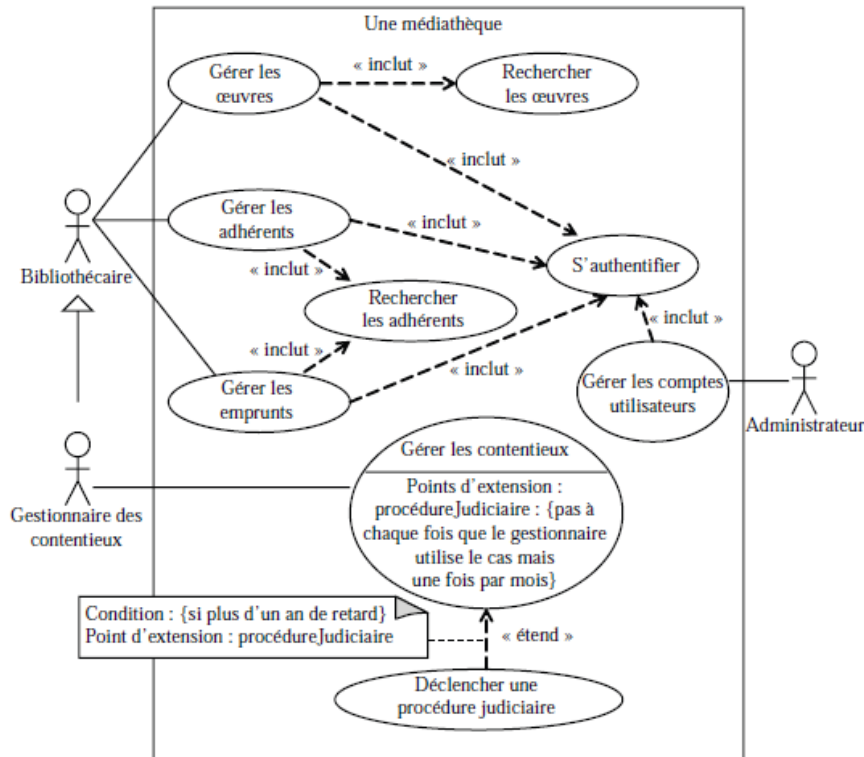
Modélisez à l'aide d'un diagramme de cas d'utilisation une médiathèque dont le fonctionnement est décrit ci-après.

Une petite médiathèque n'a qu'une seule employée qui assume toutes les tâches :

- la gestion des œuvres de la médiathèque ;
- la gestion des adhérents.

Le prêt d'un exemplaire d'une œuvre donnée est limité à trois semaines. Si l'exemplaire n'est pas rapporté dans ce délai, cela génère un contentieux. Si l'exemplaire n'est toujours pas rendu au bout d'un an, une procédure judiciaire est déclenchée.

L'accès au système informatique est protégé par un mot de passe.



## Exercice 5 : Identification des acteurs, recensement des cas d'utilisation et leur description

Modélisez avec un diagramme de cas d'utilisation le fonctionnement d'un distributeur de cassettes vidéo dont la description est donnée ci-après.

Une personne souhaitant utiliser le distributeur doit avoir une carte magnétique spéciale. Les cartes sont disponibles au magasin qui gère le distributeur. Elles sont créditées d'un certain montant en euros et rechargeables au magasin. Le prix de la location est fixé par tranches de 6 heures (1 euro par tranche). Le fonctionnement du distributeur est le suivant:

- le client introduit sa carte ; si le crédit est supérieur ou égal à 1 euro, le client est autorisé à louer une cassette (il est invité à aller recharger sa carte au magasin sinon) ;
- le client choisit une cassette et part avec ; quand il la ramène, il l'introduit dans le distributeur puis insère sa carte ; celle-ci est alors débitée ; si le montant du débit excède le crédit de la carte, le client est invité à régulariser sa situation au magasin et le système mémorise le fait qu'il est débiteur ;
- la gestion des comptes débiteurs est prise en charge par la personne du magasin.

On ne s'intéresse ici qu'à la location des cassettes, et non à la gestion du distributeur par le personnel du magasin (ce qui exclut la gestion du stock des cassettes).

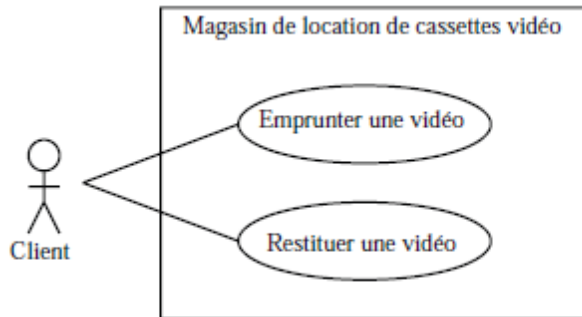
Décrivez sous forme textuelle les cas d'utilisation « Emprunter une vidéo » et « Rechercher une vidéo ». La recherche d'une vidéo peut se faire par genres ou par titres de film. Les différents genres sont action, aventure, comédie et drame. Quand une liste de films s'affiche, le client peut trier les films par titres ou par dates de sortie en salles.

### Solution :

1°

Le seul acteur est le client.

L'acquisition d'une carte et sa recharge ne se font pas via le distributeur : il faut aller au magasin. Ces fonctions ne donnent pas lieu à des cas d'utilisation.



Il n'y a donc que deux cas : « Emprunter une vidéo » et « Restituer une vidéo ».

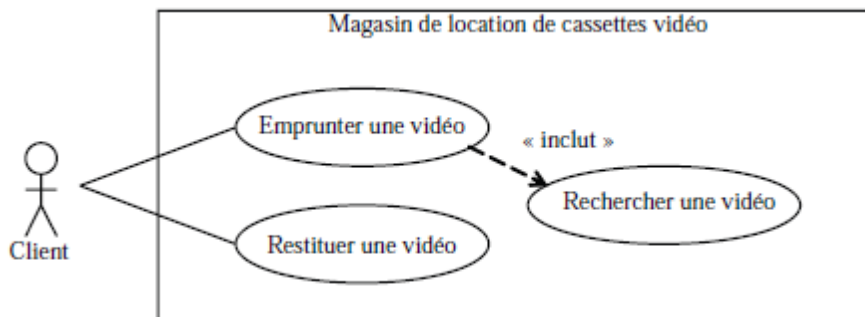
Pour décrire le cas « Emprunter une vidéo », imaginons un scénario possible. Le client introduit sa carte.

Il doit ensuite pouvoir choisir une vidéo. Quels sont les critères de choix ?

L'énoncé ne précise pas ces critères. Ce problème arrive fréquemment en situation réelle.

Choisir une vidéo peut être complexe : la recherche se fait-elle par genres, par titres ou par dates de sortie des films en salles ? Si la recherche se fait par genres, quels sont-ils ? Rechercher un film semble plus complexe qu'on ne l'imaginait au premier abord. De plus, cette fonctionnalité peut être isolée de la location proprement dite, qui concerne plutôt la gestion de la carte.

Ces remarques incitent à créer le cas supplémentaire « Rechercher une vidéo ». L'emprunt d'une vidéo inclut sa recherche. Une relation d'inclusion intervient donc entre les cas « Emprunter une vidéo » et « Rechercher une vidéo ».



2°

Avant de présenter la description, donnons quelques indications :

- Tous les cas d'utilisation d'un système doivent être décrits sous forme textuelle (dans la suite de ce chapitre, nous omettrons éventuellement de le faire pour des raisons de place ou d'intérêt).
- Quand une erreur (exception) est détectée dans un cas, une séquence d'erreurs est activée (par exemple, voir la séquence E1 dans la description suivante). La séquence nominale n'est pas reprise et le cas s'interrompt aussitôt.



## 1. Description du cas « Emprunter une vidéo »

### Identification

Nom du cas : « Emprunter une vidéo ».

But : décrire les étapes permettant au client du magasin d'emprunter une cassette vidéo via le distributeur.

Acteur principal : Client.

Acteur secondaire : néant.

Date de création : le 03/03/2017.

Date de mise à jour : le 15/03/2017.

Responsable : .

Version : 1.1.

### Description textuelle

Le cas d'utilisation commence lorsqu'un client introduit sa carte.

#### Pré-conditions

Le client possède une carte qu'il a achetée au magasin.

Le distributeur est alimenté en cassettes.

#### Enchaînement nominal

1. Le système vérifie la validité de la carte.
2. Le système vérifie que le crédit de la carte est supérieur ou égal à 1 euro.
3. Appel du cas « Rechercher une vidéo ».
4. Le client a choisi une vidéo.
5. Le système indique, d'après la valeur de la carte, pendant combien de temps (tranches de 6 heures) le client peut garder la cassette.
6. Le système délivre la cassette.
7. Le client prend la cassette.
8. Le système rend la carte au client.
9. Le client prend sa carte.

#### Enchaînements alternatifs

##### A1 : Le crédit de la carte est inférieur à 1 euro.

L'enchaînement démarre après le point 2 de la séquence nominale :

3. Le système indique que le crédit de la carte ne permet pas au client d'emprunter une vidéo.

4. Le système invite le client à aller recharger sa carte au magasin.

La séquence nominale reprend au point 8.

#### Enchaînements d'exception

##### E1 : La carte introduite n'est pas valide.

L'enchaînement démarre après le point 1 de la séquence nominale :

2. Le système indique que la carte n'est pas reconnue.

3. Le distributeur éjecte la carte.

##### E2 : La cassette n'est pas prise par le client.

L'enchaînement démarre après le point 6 de la séquence nominale :

7. Au bout de 15 secondes le distributeur avale la cassette.

8. Le système annule la transaction (toutes les opérations mémorisées par le système sont défaites).

9. Le distributeur éjecte la carte.

### **E3 : La carte n'est pas reprise par le client.**

L'enchaînement démarre après le point 8 de la séquence nominale :

9. Au bout de 15 secondes le distributeur avale la carte.

10. Le système consigne cette erreur (date et heure de la transaction, identifiant du client, identifiant du film).

### **E4 : Le client a annulé la recherche (il n'a pas choisi de vidéo).**

L'enchaînement démarre au point 4 de la séquence nominale :

5. Le distributeur éjecte la carte.

### **Post-conditions**

Le système a enregistré les informations suivantes :

- La date et l'heure de la transaction, à la minute près : les tranches de 6 heures sont calculées à la minute près.
- L'identifiant du client.
- L'identifiant du film emprunté.

### **Rubriques optionnelles**

#### **Contraintes non fonctionnelles**

Le distributeur doit fonctionner 24 heures sur 24 et 7 jours sur 7.

La vérification de la validité de la carte doit permettre la détection des contrefaçons.

#### **Contrainte liée à l'interface homme-machine**

Avant de délivrer la cassette, demander confirmation au client.

## **2. Description du cas « Rechercher une vidéo »**

### **Identification**

Nom du cas : « Rechercher une vidéo ».

But : décrire les étapes permettant au client de rechercher une vidéo *via* le distributeur automatique.

Acteur principal : néant (cas interne inclus dans le cas « Emprunter une vidéo »).

Acteur secondaire : néant.

Date de création : le 31/12/2011.

Responsable :

Version : 1.0.

### **Description textuelle**

Le cas démarre au point 3 de la description du cas « Emprunter une vidéo ».

#### **Enchaînement nominal (le choix du film se fait par genres)**

1. Le système demande au client quels sont ses critères de recherche pour un film (les choix possibles sont : par titres ou par genres de film).
2. Le client choisit une recherche par genres.
3. Le système recherche les différents genres de film présents dans le distributeur.
4. Le système affiche une liste des genres (les choix possibles sont action, aventure, comédie et drame).
5. Le client choisit un genre de film.
6. Le système affiche la liste de tous les films du genre choisi présents dans le distributeur.
7. Le client sélectionne un film.

#### **Enchaînements alternatifs**

**A1 : Le client choisit une recherche par titres.**

L'enchaînement démarre après le point 1 de la séquence nominale :

2. Le client choisit une recherche par titres.

3. Le système affiche la liste de tous les films classés par ordre alphabétique des titres.

La séquence nominale reprend au point 7.

### **Enchaînements d'exception**

#### **E1 : Le client annule la recherche.**

L'enchaînement peut démarrer aux points 2, 5 et 7 de la séquence nominale :

Appel de l'exception E4 du cas « Emprunter une vidéo ».

### **Post-conditions**

Le système a mémorisé le film choisi par le client.

### **Rubriques optionnelles**

### **Contraintes non fonctionnelles**

#### **Contraintes liées à l'interface homme-machine**

Quand une liste de films s'affiche, le client peut trier la liste par titres ou par dates de sortie en salles.

Le client peut se déplacer dans la liste et la parcourir de haut en bas et de bas en haut.

Ne pas afficher plus de 10 films à la fois dans la liste.

## Chapitre III : La modélisation statique

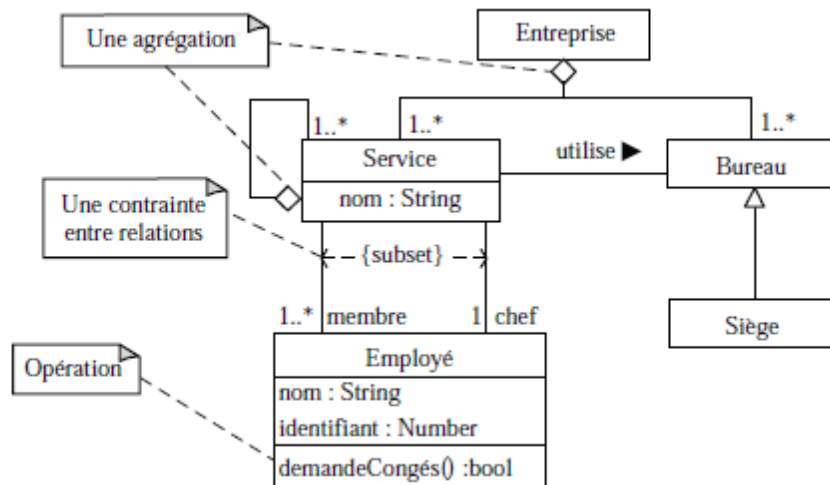
### 3.1. Diagramme de classes

#### 1. Introduction

Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet. Alors que le diagramme de cas d'utilisation montre un système du point de vue des acteurs, le diagramme de classes en montre la structure interne. Il contient principalement des classes. Une classe contient des attributs et des opérations. Le diagramme de classes n'indique pas comment utiliser les opérations : c'est

Le diagramme de classes présente un ensemble de classeurs. Il décrit les classes et leurs relations. Il peut également décrire les **regroupements de classes en paquetages**, les interfaces et les objets, les classes qui participent à une collaboration ou qui réalisent un cas d'utilisation, etc.

Par exemple le diagramme suivant modélise une entreprise à l'aide de cinq classes : *Entreprise*, *Service*, *Bureau*, *Employé* et *Siège*. Les classes possèdent éventuellement des *attributs* (la classe *Service* est caractérisée par un nom, un employé se caractérise par un nom et un identifiant). Elles contiennent aussi des *opérations* (demandeCongés pour la classe *Employé*). Les classes sont reliées entre elles par des *associations* symbolisées par des traits. Plusieurs types d'associations existent, qui se représentent différemment (par des losanges, des flèches...).



Pour créer un diagramme de classes, vous avez besoin d'abord de définir les classes et leurs responsabilités, les paquetages ainsi que les *relations* (**association, composition, agrégation, héritage, dépendance...**) possibles entre ces éléments. D'autres éléments peuvent également apparaître dans le diagramme de classes, comme les objets et les interfaces.

Nous étudierons les éléments indispensables pour bien réussir une modélisation statique par le diagramme de classes.

#### 2. Concepts de base

Beaucoup d'objets naturels ou informatiques se ressemblent tant au niveau de leur description que de leur comportement. Afin de limiter la complexité de la modélisation, vous pouvez regrouper les objets ayant

les mêmes caractéristiques en classes. Ainsi, une classe décrit les états (ou attributs) et le comportement à haut niveau d'abstraction d'objets similaires.

De cette façon, vous pouvez définir **la classe comme le regroupement des données et des traitements** applicables aux objets qu'elle représente. Dans l'approche objet, il faut, avant tout, pouvoir identifier les objets, les différencier des autres avant de les définir. L'identité associée à une classe est essentielle car elle conditionne la perception des structures et des comportements des objets qu'elle représente.

En UML, la classe est représentée graphiquement par un rectangle divisé en plusieurs compartiments. Le premier compartiment indique le nom de la classe, le deuxième ses attributs, le troisième ses méthodes. En fonction de l'état de la modélisation et des éléments à mettre en valeur, d'autres compartiments, comme les compartiments de responsabilités et d'exceptions, peuvent être ajoutés à la description de la classe.

Un objet d'une classe doit avoir des valeurs uniques pour tous les attributs définis dans la classe et doit pouvoir exécuter (sans exception) toutes les méthodes de la classe.

### 2.1. Classe

Une classe est une description d'un ensemble d'objets ayant une sémantique, des attributs, des méthodes et des relations en commun. Un objet est une instance d'une classe.

#### Remarque

Dans les langages de programmation objet, on confond souvent la notion d'instance et la notion d'objet alors que la première est plus générale que la seconde. Un objet est une instance d'une classe. Mais une instance peut être une instance de plusieurs autres éléments du langage UML. Par exemple, un lien est une instance d'une association.

Une **classe** décrit un groupe d'objets ayant les mêmes propriétés (attributs), un même comportement (méthodes ou opérations), et une sémantique commune (domaine de définition).

Un objet est une **instance** d'une classe. La classe représente l'abstraction de ses objets. Au niveau de l'implémentation, c'est-à-dire au cours de l'exécution d'un programme, l'identificateur d'un objet correspond à une adresse mémoire.

### 2.2. Formalisme général et exemple

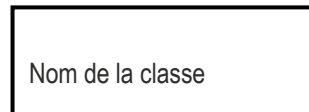
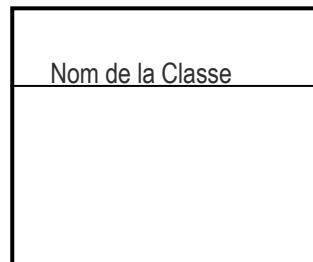
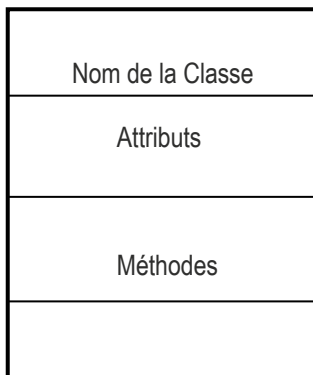
Une classe se représente à l'aide d'un rectangle comportant plusieurs compartiments.

Les trois compartiments de base sont :

- la désignation de la classe,
- la description des attributs,
- la description des opérations.

Deux autres compartiments peuvent être aussi indiqués :

- la description des responsabilités de la classe,
- la description des exceptions traitées par la classe.



## Responsabilités

Il est possible de manipuler les classes en limitant le niveau de description à un nombre réduit de compartiments selon les objectifs poursuivis par le modélisateur.

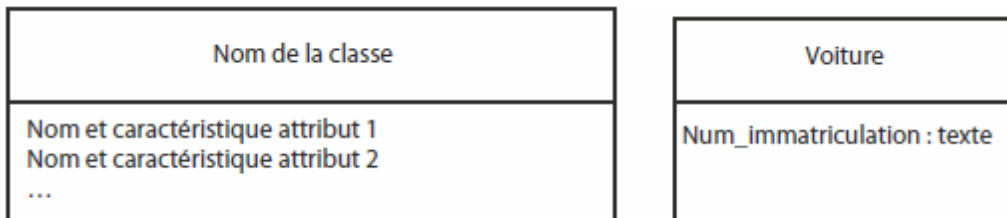
Ainsi les situations suivantes sont possibles pour la manipulation d'une description restreinte de classe :

- description uniquement du nom et des caractéristiques générales de la classe,
- description du nom de la classe et de la liste d'attributs.

## Attribut

Un **attribut** est une propriété élémentaire d'une classe. Pour chaque objet d'une classe, l'attribut prend une valeur (sauf cas d'attributs multivalués).

## Formalisme et exemple



*Formalisme d'attributs de classe et exemple Voiture*

## Caractéristiques

Le nom de la classe peut être qualifié par un « stéréotype ». La description complète des attributs d'une classe comporte un certain nombre de caractéristiques qui doivent respecter le formalisme suivant :

- **Visibilité/Nom attribut : type [= valeur initiale {propriétés}]**
  - *Visibilité* : se reporter aux explications données plus loin sur ce point.
  - *Nom d'attribut* : nom unique dans sa classe.
  - *Type* : type primitif (entier, chaîne de caractères...) dépendant des types disponibles dans le langage d'implémentation ou type classe matérialisant un lien avec une autre classe.
  - *Valeur initiale* : valeur facultative donnée à l'initialisation d'un objet de la classe.
  - *{propriétés}* : valeurs marquées facultatives (ex. : « interdit » pour mise à jour interdite).

Un attribut peut avoir des valeurs multiples. Dans ce cas, cette caractéristique est indiquée après le nom de l'attribut (ex. : prénom [3] pour une personne qui peut avoir trois prénoms).

Un attribut dont la valeur peut être calculée à partir d'autres attributs de la classe est un **attribut dérivé** qui se note « /nom de l'attribut dérivé ».

## Opération

Une **opération** est une fonction applicable aux objets d'une classe. Une opération permet de décrire le comportement d'un objet. Une **méthode** est l'implémentation d'une opération.

## Formalisme et exemple

Chaque opération est désignée soit seulement par son nom soit par son nom, sa liste de paramètres et son type de résultat. La signature d'une méthode correspond au nom de la méthode et la liste des paramètres en entrée.

Nom de la classe
Nom et caractéristique attributs ...
Nom et caractéristique opération 1 Nom et caractéristique opération 2 ...

Voiture
marque : texte
rouler (vitesse)

Formalisme et exemple d'opérations de classe

### Caractéristiques

La description complète des opérations d'une classe comporte un certain nombre de caractéristiques qui doivent respecter le formalisme suivant :

- **Visibilité**

Chaque attribut ou opération d'une classe peut être de type public, protégé, privé ou paquetage. Les symboles + (public), # (protégé), - (privé) et ~ (paquetage) sont indiqués devant chaque attribut ou opération pour signifier le type de visibilité autorisé pour les autres classes.

Les droits associés à chaque niveau de confidentialité sont :

- **Public (+)** – Attribut ou opération visible par tous.
- **Protégé (#)** – Attribut ou opération visible seulement à l'intérieur de la classe et pour toutes les sous-classes de la classe.
- **Privé (-)** – Attribut ou opération seulement visible à l'intérieur de la classe.
- **Paquetage (~)** – Attribut ou opération ou classe seulement visible à l'intérieur du paquetage où se trouve la classe.

Voiture
- marque - puissance - cylindrée - année - chiffreAffaire
+ démarrer () - rouler () + freiner () # arrêter ()

Dans cet exemple, tous les attributs sont déclarés de type privé, les opérations « démarrer » et « freiner » sont de type public, l'opération « rouler » est de type privé et l'opération « arrêter » est de type protégé.

- **Nom d'opération (paramètres) [:[type résultat] {propriétés}]**

– *Nom d'opération* : utiliser un verbe représentant l'action à réaliser.

– *Paramètres* : liste de paramètres (chaque paramètre peut être décrit, en plus de son nom, par son type et sa valeur par défaut). L'absence de paramètre est indiquée par ( ).

– *Type résultat* : type de (s) valeur(s) retourné(s) dépendant des types disponibles dans le langage d'implémentation. Par défaut, une opération ne retourne pas de valeur, ceci est indiqué par exemple par le mot réservé « void » dans le langage C++ ou Java.

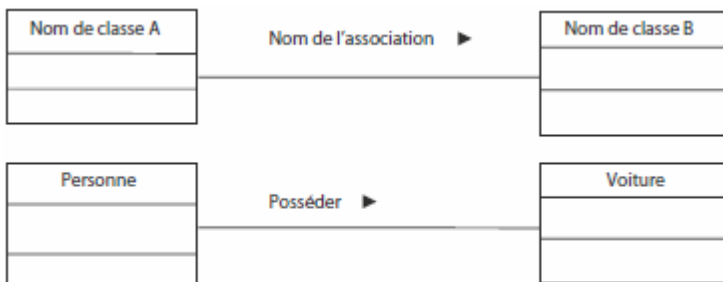
– {*propriétés*} : valeurs facultatives applicables (ex. : {query} pour un comportement sans influence sur l'état du système).

## 2.3. Association, multiplicité, navigabilité et contraintes

### 1. Définitions : Lien et association

Un **lien** est une connexion physique ou conceptuelle entre instances de classes donc entre objets. Une **association** décrit un groupe de liens ayant une même structure et une même sémantique. Un lien est une instance d'une association. Chaque association peut être identifiée par son nom.

Une **association entre classes** représente les liens qui existent entre les instances de ces classes.



Le symbole (facultatif) indique le sens de lecture de l'association. Dans cette figure est donné aussi un exemple de représentation d'une association.

### 2. Rôle d'association

Le **rôle** tenu par une classe vis-à-vis d'une association peut être précisé sur l'association.



### 3. Multiplicité

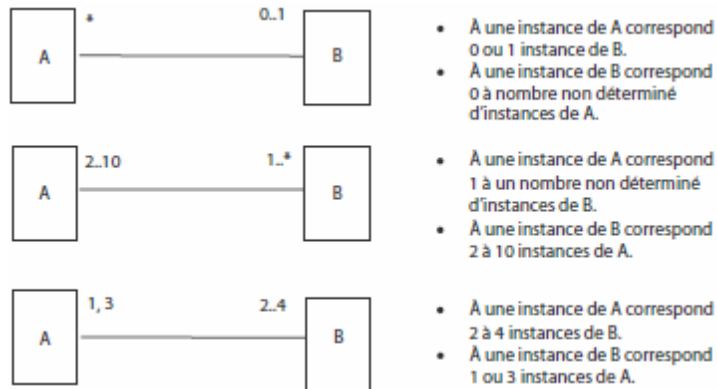
La **multiplicité** indique un domaine de valeurs pour préciser le nombre d'instance d'une classe vis-à-vis d'une autre classe pour une association donnée. La multiplicité peut aussi être utilisée pour d'autres usages comme par exemple un attribut multivalué.

Voici quelques exemples de multiplicité :

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

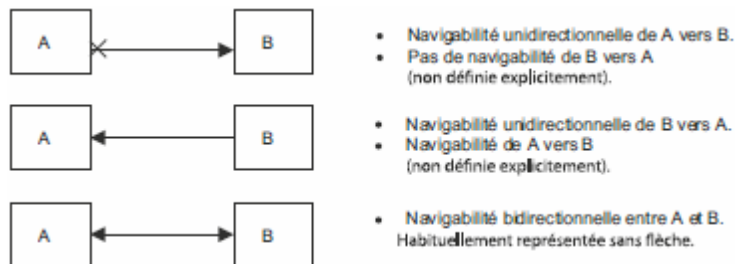
Dans une association binaire, la multiplicité sur la terminaison cible contraint le nombre d'objets de la classe cible pouvant être associés à un seul objet donné de la classe source (la classe de l'autre terminaison de l'association).





## 4. Navigabilité

La **navigabilité** indique si l'association fonctionne de manière unidirectionnelle ou bidirectionnelle, elle est matérialisée par une ou deux extrémités fléchées. La non navigabilité se représente par un « X ».



Par défaut, on admet qu'une navigabilité non définie correspond à une navigabilité implicite.

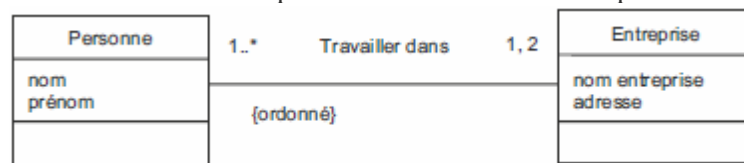
## 5. Contraintes

D'autres propriétés particulières (contraintes) sont proposées dans UML pour préciser la sémantique d'une association.

### - Ordre de tri

Pour une association de multiplicité supérieure à 1, les liens peuvent être :

- non ordonnés (valeur par défaut),
- ordonnés ou triés lorsque l'on est au niveau de l'implémentation (tri sur une valeur interne).



Dans cet exemple, pour une entreprise donnée, les personnes seront enregistrées suivant un ordre qui correspondra à un des attributs de Personne.

### - Propriétés de mise à jour de liens

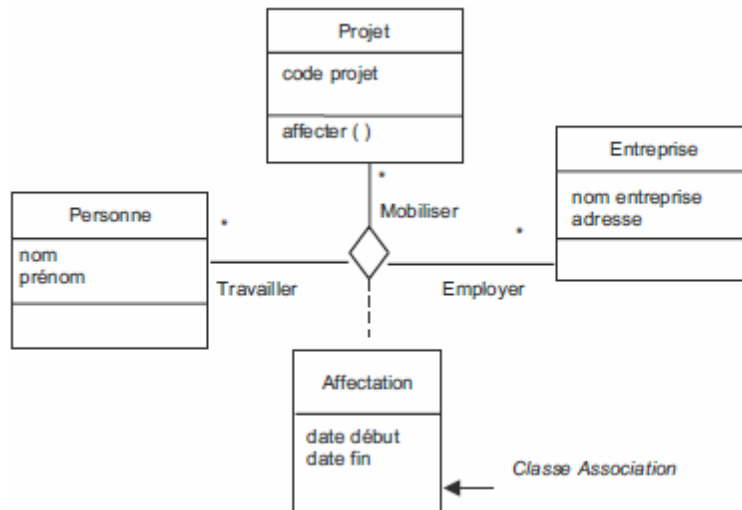
Il est possible d'indiquer des contraintes particulières relatives aux conditions de mise à jour des liens.

- {interdit} : interdit l'ajout, la suppression ou la mise à jour des liens.
- {ajout seul} : n'autorise que l'ajout de liens.

## 6. Association de dimension supérieure à 2 et classe-association

Une association de dimension supérieure à 2 se représente en utilisant un losange permettant de relier toutes les classes concernées.

Une **classe-association** permet de décrire soit des attributs soit des opérations propres à l'association. Cette classe-association est elle-même reliée par un trait en pointillé au losange de connexion. Une classe-association peut être reliée à d'autres classes d'un diagramme de classes.

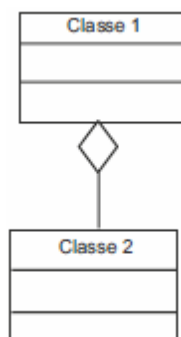


La classe-association Affectation permet de décrire les attributs propres à l'association de dimension 3 représentée.

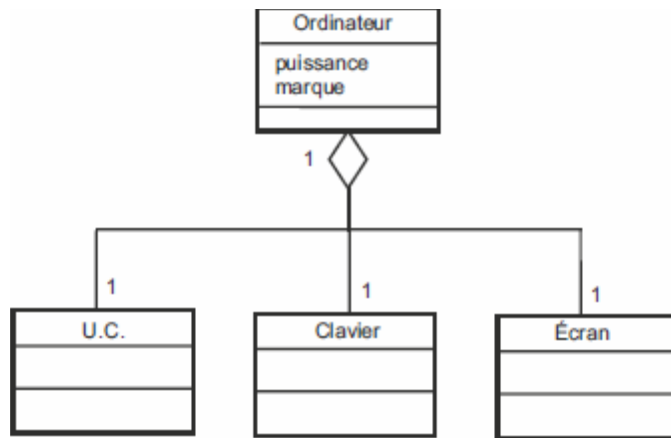
## 2.4. Agrégation et composition entre classes

### 1. Agrégation

L'**agrégation** est une association qui permet de représenter un lien de type « ensemble » comprenant des « éléments ». Il s'agit d'une relation entre une classe représentant le niveau « ensemble » et 1 à  $n$  classes de niveau « éléments ». L'agrégation représente un lien structurel entre une classe et une ou plusieurs autres classes.



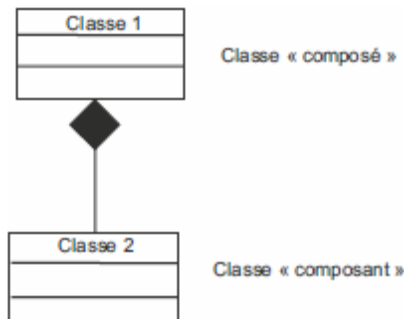
Exemple



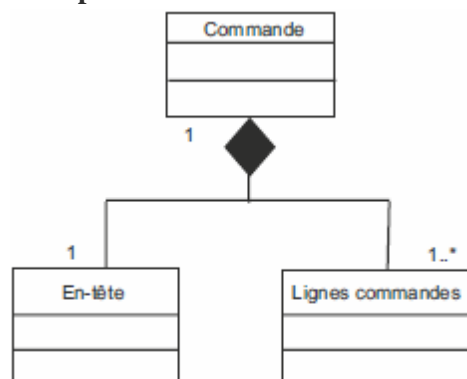
Dans cet exemple, nous avons modélisé le fait qu'un ordinateur comprend une UC, un clavier et un écran.

## 2. Composition

La **composition** est une relation d'agrégation dans laquelle il existe une contrainte de durée de vie entre la classe « composant » et la ou les classes « composé ». Autrement dit la suppression de la classe « composé » implique la suppression de la ou des classes « composant ».



### Exemple



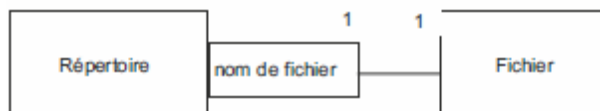
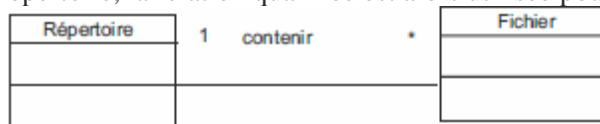
## 2.5. Association qualifiée, dépendance et classe d'interface

### 1. Qualification

La **qualification** d'une relation entre deux classes permet de préciser la sémantique de l'association et de qualifier de manière restrictive les liens entre les instances.

Seules les instances possédant l'attribut indiqué dans la qualification sont concernées par l'association. Cet attribut ne fait pas partie de l'association.

Soit la relation entre les répertoires et les fichiers appartenant à ces répertoires. À un répertoire est associé 0 à  $n$  fichiers. Si l'on veut restreindre cette association pour ne considérer qu'un fichier associé à son répertoire, la relation qualifiée est alors utilisée pour cela.

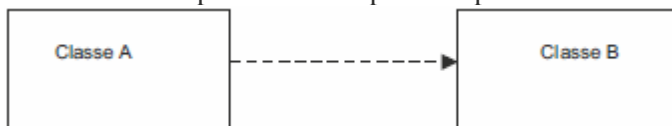


### 2. Dépendance

La **dépendance** entre deux classes permet de représenter l'existence d'un lien sémantique.

Une classe B est en dépendance de la classe A si des éléments de la classe A sont nécessaires pour construire la classe B.

La relation de dépendance se représente par une flèche en pointillé entre deux classes.



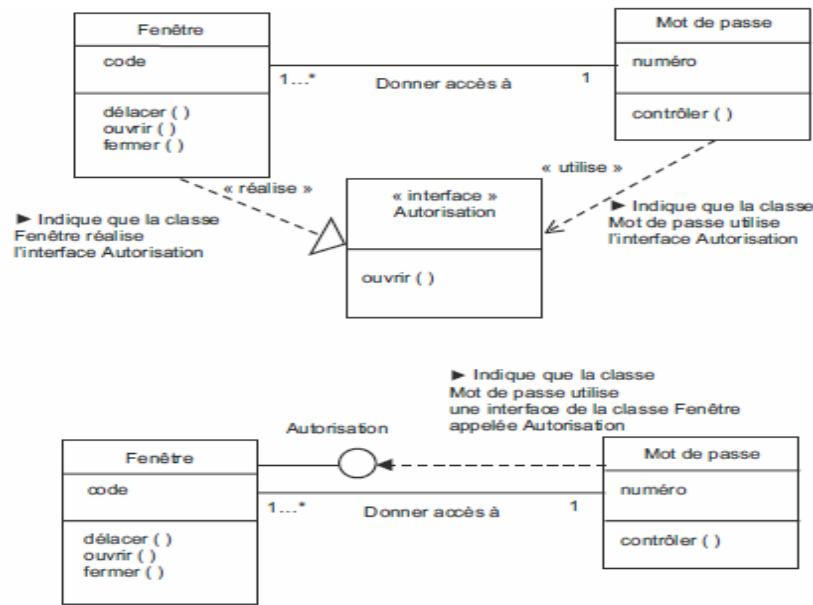
### 3. Interface

Une classe d'**interface** permet de décrire la vue externe d'une classe. La classe d'interface, identifiée par un nom, comporte la liste des opérations accessibles par les autres classes. Le compartiment des attributs ne fait pas partie de la description d'une interface.

L'interface peut être aussi matérialisée plus globalement par un petit cercle associé à la classe source.

La classe utilisatrice de l'interface est reliée au symbole de l'interface par une flèche en pointillé. La classe d'interface est une spécification et non une classe réelle.

Une classe d'interface peut s'assimiler à une classe abstraite.

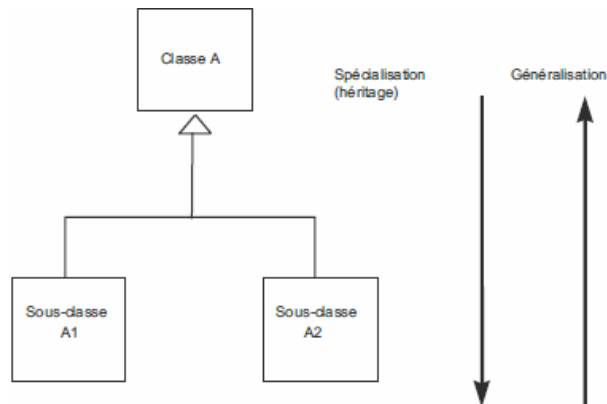


## 2.6. Généralisation et spécialisation

### 1. La généralisation/spécialisation et l'héritage simple

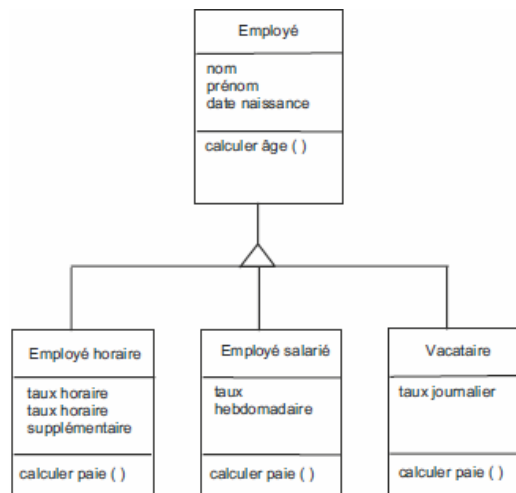
La **généralisation** est la relation entre une classe et deux autres classes ou plus partageant un sous-ensemble commun d'attributs et/ou d'opérations.

La classe qui est affinée s'appelle **super-classe**, les classes affinées s'appellent **sous-classes**. L'opération qui consiste à créer une super-classe à partir de classes s'appelle la généralisation. Inversement la **spécialisation** consiste à créer des sous classes à partir d'une classe.



- la sous-classe A1 hérite de A, c'est une spécialisation de A ;
- la sous-classe A2 hérite de A, c'est une spécialisation de A.

L'héritage permet à une sous-classe de disposer des attributs et opérations de la classe dont elle dépend. Un discriminant peut être utilisé pour exploiter le critère de spécialisation entre une classe et ses sous-classes. Le discriminant est simplement indiqué sur le schéma, puisque les valeurs prises par ce discriminant correspondent à chaque sous-classe.



Dans cet exemple, les attributs nom, prénom et date de naissance et l'opération « calculer âge » de « Employé » sont hérités par les trois sous-classes : Employé horaire, Employé salarié, Vacataire.

## 2. Classe abstraite

Une **classe abstraite** est une classe qui n'a pas d'instance directe mais dont les classes descendantes ont des instances. Dans une relation d'héritage, la super-classe est par définition une classe abstraite. C'est le cas de la classe Employé dans l'exemple ci-dessus.

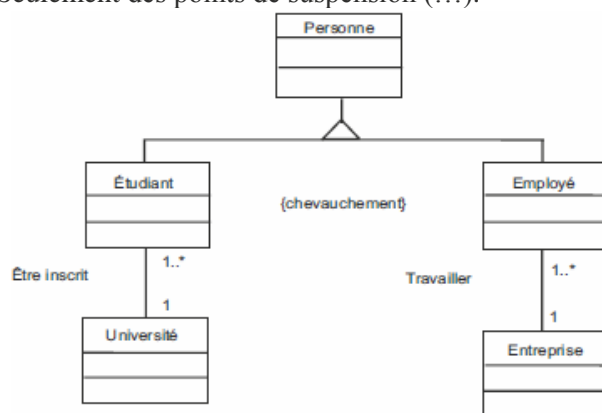
## 3. L'héritage avec recouvrement

Par défaut, les sous-classes ont des instances disjointes les unes par rapport aux autres. Dans certains cas, il existe un recouvrement d'instances entre les sous-classes.

D'une manière générale, quatre situations peuvent se rencontrer et se représentent sous forme de contraintes :

- **{chevauchement}** : deux sous-classes peuvent avoir, parmi leurs instances, des instances identiques ;
- **{disjoint}** : les instances d'une sous-classe ne peuvent être incluses dans une autre sous-classe de la même classe ;
- **{complète}** : la généralisation ne peut pas être étendue ;
- **{incomplète}** : la généralisation peut être étendue.

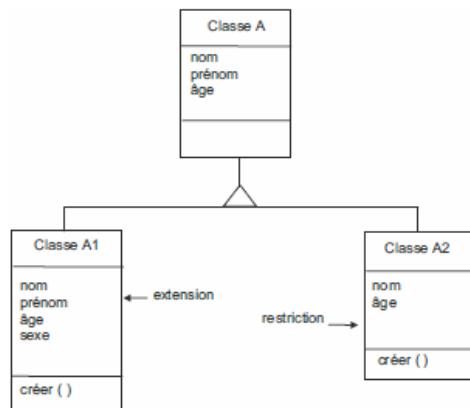
Dans certains cas, il est possible de ne pas citer toutes les sous-classes mais d'indiquer Seulement des points de suspension (...).



Exemple d'héritage avec recouvrement d'instances entre les classes Étudiant et Employé. En effet, une même personne peut être à la fois étudiante dans une université et employée dans une entreprise.

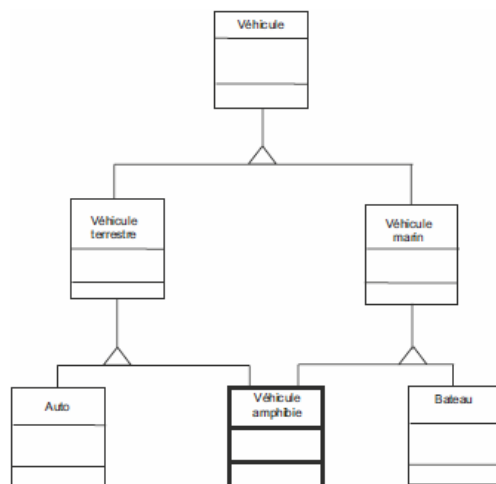
### 4. Extension et restriction de classe

L'ajout de propriétés dans une sous-classe correspond à une **extension de classe**. Le masquage de propriétés dans une sous-classe correspond à une **restriction de classe**.



### 5. L'héritage multiple

Dans certains cas, il est nécessaire de faire hériter une même classe de deux classes « parentes » distinctes. Ce cas correspond à un **héritage multiple**.



Un exemple classique d'héritage multiple où la classe « Véhicule amphibie » hérite des classes « Véhicule terrestre » et « Véhicule marin ».

### 2.7. Stéréotype de classe

UML propose un certain nombre de **stéréotypes** qui permettent de qualifier les profils d'utilisation.

Parmi ces stéréotypes, nous présentons ci-après quatre d'entre eux :

- « **Classe d'implémentation** » – Ce stéréotype est utilisé pour décrire des classes de niveau physique.

- « **Type** » – Ce stéréotype permet de spécifier des opérations applicables à un domaine d'objets. Exemple : Type *Integer* d'un langage de programmation.
- « **Utilitaire** » – Ce stéréotype qualifie toutes les fonctions utilitaires de base utilisées par les objets.
- « **MétaClasse** » – Ce stéréotype permet de regrouper des classes dans une famille de classe.

### 3.3. Exercices

#### Exercice 1 :

*Soient les phrases suivantes :*

- *Un répertoire contient des fichiers*
- *Une pièce contient des murs*
- *Les modems et claviers sont des périphériques d'entrée / sortie*
- *Une transaction boursière est un achat ou une vente*
- *Un compte bancaire peut appartenir à une personne physique ou morale*

*Elaborez les diagrammes de classe correspondants en choisissant le type de relation approprié.*

NB : Pour le compte bancaire, on aurait également pu modéliser 2 associations entre « compte bancaire » et « personne physique » et « personne morale » en y incluant une contrainte d'exclusion.

#### Exercice 2 : Académie à plusieurs collèges

*Une académie souhaite gérer les cours dispensés dans plusieurs collèges. Pour cela, on dispose des renseignements suivants :*

- *Chaque collège possède d'un site Internet*
- *Chaque collège est structuré en départements, qui regroupent chacun des enseignants spécifiques. Parmi ces enseignants, l'un d'eux est responsable du département.*
- *Un enseignant se définit par son nom, prénom, tél, mail, date de prise de fonction et son indice.*
- *Chaque enseignant ne dispense qu'une seule matière.*
- *Les étudiants suivent quant à eux plusieurs matières et reçoivent une note pour chacune d'elle.*
- *Pour chaque étudiant, on veut gérer son nom, prénom, tél, mail, ainsi que son année d'entrée au collège.*
- *Une matière peut être enseignée par plusieurs enseignants mais a toujours lieu dans la même salle de cours (chacune ayant un nombre de places déterminé).*
- *On désire pouvoir calculer la moyenne par matière ainsi que par département*
- *On veut également calculer la moyenne générale d'un élève et pouvoir afficher les matières dans lesquelles il n'a pas été noté*
- *Enfin, on doit pouvoir imprimer la fiche signalétique (, prénom, tél, mail) d'un enseignant ou d'un élève.*

*Elaborez le diagramme de classes correspondant. Pour simplifier l'exercice, on limitera le diagramme à une seule année d'étude*

#### Exercice 3 : Agence de voyage

*On souhaite gérer les réservations de vols effectués dans une agence. D'après les interviews réalisées avec les membres de l'agence, on sait que :*

- *Les compagnies aériennes proposent différents vols*
- *Un vol est ouvert à la réservation et refermé sur ordre de la compagnie*



- Un client peut réserver un ou plusieurs vols, pour des passagers différents
- Une réservation concerne un seul vol et un seul passager
- Une réservation peut être confirmée ou annulée
- Un vol a un aéroport de départ et un aéroport d'arrivée
- Un vol a un jour et une heure de départ, et un jour et une heure d'arrivée
- Un vol peut comporter des escales dans un ou plusieurs aéroport(s)
- Une escale a une heure de départ et une heure d'arrivée
- Chaque aéroport dessert une ou plusieurs villes

A partir des éléments qui vous sont fournis ci-dessus, élaborer le diagramme de classes (en y ajoutant tout attribut que vous jugez pertinent et qui n'a pas été décrit ci-dessus).

### Exercice 4 : Agence de voyage (suite)

Un avion assure plusieurs vols et un vol est assuré par un seul avion. Un vol peut être un vol cargo ou un vol de passagers. Les avions utilisés pour ces deux types de vols ne sont pas les mêmes.

- Donnez le diagramme de classes exprimant cette situation.
- Ajoutez les contraintes du langage OCL pour exprimer le fait qu'un vol cargo soit assuré par un avion-cargo et qu'un vol de passagers soit assuré par un avion de passagers.

Les vols sont proposés par des compagnies aériennes. Pour un meilleur remplissage des avions, un vol est partagé par plusieurs affrêteurs. Un des affrêteurs assure l'ouverture et la fermeture de chaque vol. Un vol est caractérisé par une date et un lieu de départ, une date et un lieu d'arrivée.

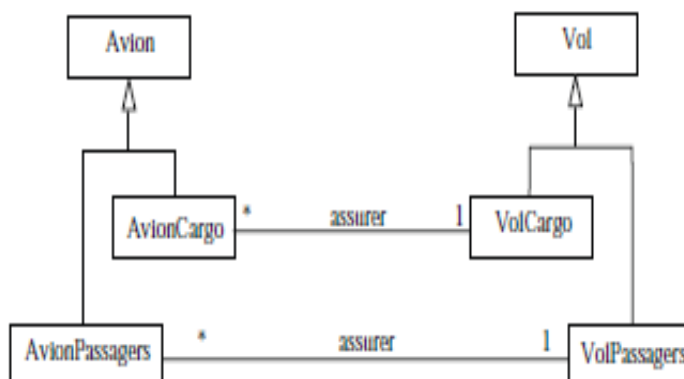
- Donnez la modélisation des classes de cette situation.

Quand un client fait une réservation auprès d'une compagnie aérienne pour un ou plusieurs passagers, il est informé du fait que le vol compte plusieurs escales. Une escale est caractérisée par des dates d'arrivée et de départ ainsi qu'un aéroport qui dessert plusieurs villes.

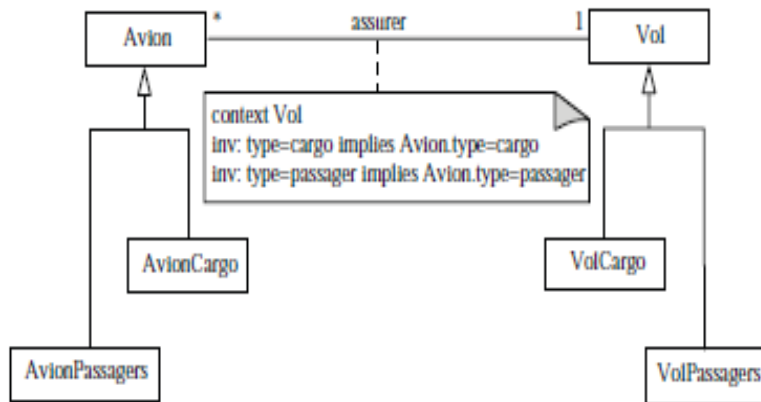
- Proposez le diagramme de classes pour cette situation.
- À partir de ces diagrammes partiels, proposez un diagramme de classes modélisant cette application.

**Solution :**

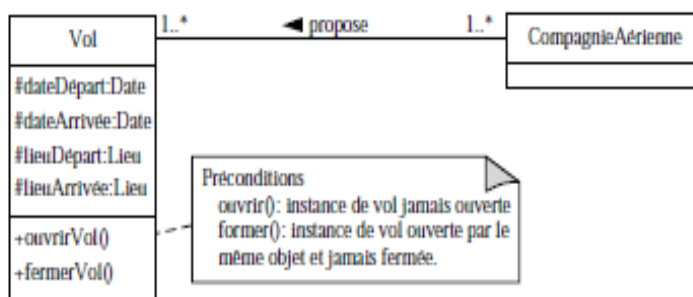
1°



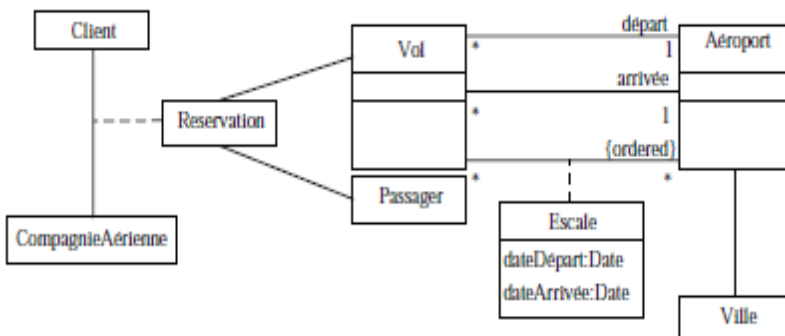
2°



3°



4°



## Exercice 5

Une entreprise nationale de vente d'appareil électroménager souhaite réaliser une première expérience d'analyse objet avec la méthode UML sur un petit sous-ensemble de son SI. Ce sous-ensemble concerne le suivi des personnels des agences locales implantées dans les régions. Chaque région est pilotée par une direction régionale qui a en charge un certain nombre d'agences locales. Une direction régionale est caractérisée par un code et un libellé. Chaque agence est caractérisée par un code, un intitulé, une date de création et une date de fermeture.

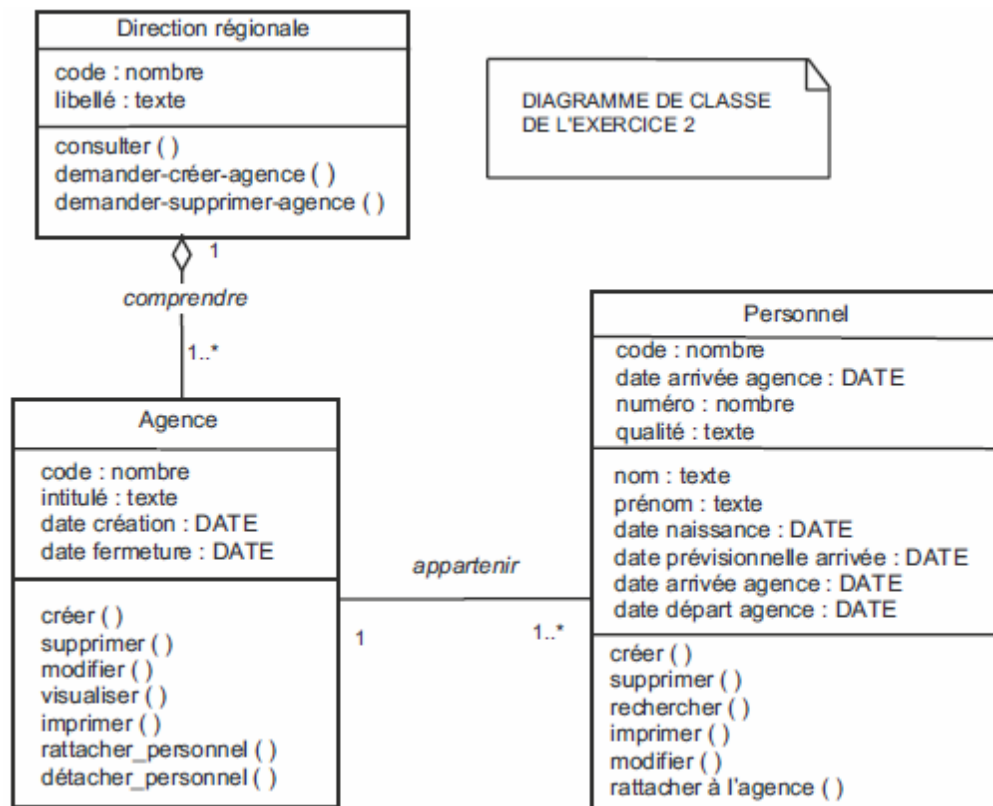
À une agence sont rattachées une à plusieurs personnes. Chaque personne est caractérisée par les données : numéro, qualité (M., Mme, Mlle), nom, prénom, date de naissance, date prévisionnelle d'arrivée, date d'arrivée et date de départ.

Il est demandé d'élaborer le diagramme de classe de ce premier sous-ensemble du SI de cette entreprise.

## Solution :

Les trois classes constituant ce système sont évidentes puisque déjà bien identifiées dans l'énoncé : *Direction régionale*, *Agence* et *Personnel*.

L'association entre *Direction régionale* et *Agence* est une agrégation qui matérialise une relation structurante entre ces classes. La relation entre *Agence* et *Personnel* est une association de un à plusieurs. Les opérations mentionnées dans chaque classe correspondent aux opérations élémentaires nécessaires à la gestion du personnel des agences.



## Exercice 6

La société Forma possède un service qui gère la formation interne. Sa mission comporte plusieurs fonctions :

- Élaborer les catalogues qui décrivent les cours et donnent les dates prévisionnelles des sessions.
- Inscrire les personnes qui désirent participer aux sessions et leur envoyer leur convocation.
- Déterminer les formateurs qui vont animer les sessions et leur envoyer leur convocation (ces personnes sont choisies parmi celles qui peuvent enseigner un cours). Certaines sessions peuvent être animées par une personne d'un organisme extérieur.
- Faire le bilan des participations réelles aux formations.

Les cours sont déterminés afin de répondre aux besoins de formation internes.

Certains cours sont organisés en filières, c'est-à-dire qu'ils doivent être suivis dans un certain ordre. Exemple : le cours ITE 16 (la démarche ITEOR OO) ne peut être suivi avant ITE 03 (UML). Les cours utilisent des documents référencés.

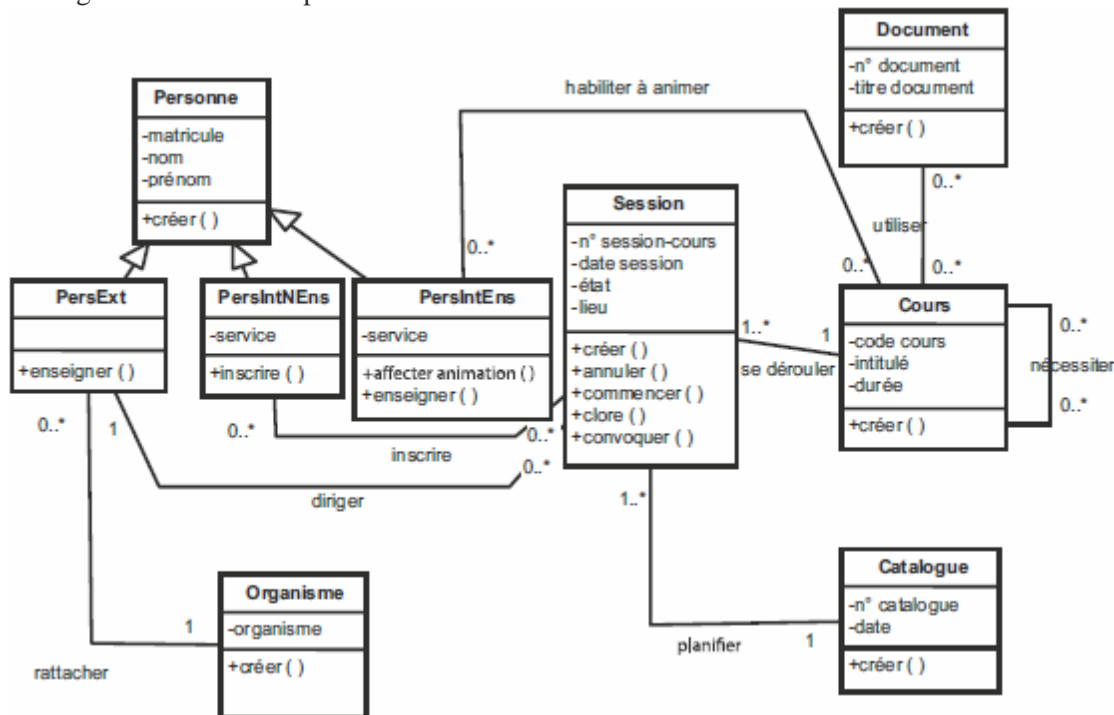
## Liste des attributs

Code cours	N° catalogue
Date catalogue	N° document
Date session	N° session
Durée cours	Nom
État de la session (prévue, annulée, en cours, close)	Organisme extérieur
Intitulé du cours	Prénom
Lieu session	Service
Matricule	Titre document

## Solution

La lecture du sujet et en particulier l'analyse des attributs indiqués conduisent à identifier rapidement les classes suivantes : *Session*, *Cours*, *Catalogue*, *Document*, *Personne* et *Organisme*.

Une réflexion complémentaire menée sur la classe *Personne* permet de distinguer en fait trois sous-classes spécialisées : *PersonneExterne*, *PersonneIntNe* (non enseignante) et *PersonneIntEn* (enseignante). Le diagramme de classes peut être élaboré ensuite sans difficulté.



## 3.4. DIAGRAMME DE PAQUETAGE (Package : DPA)

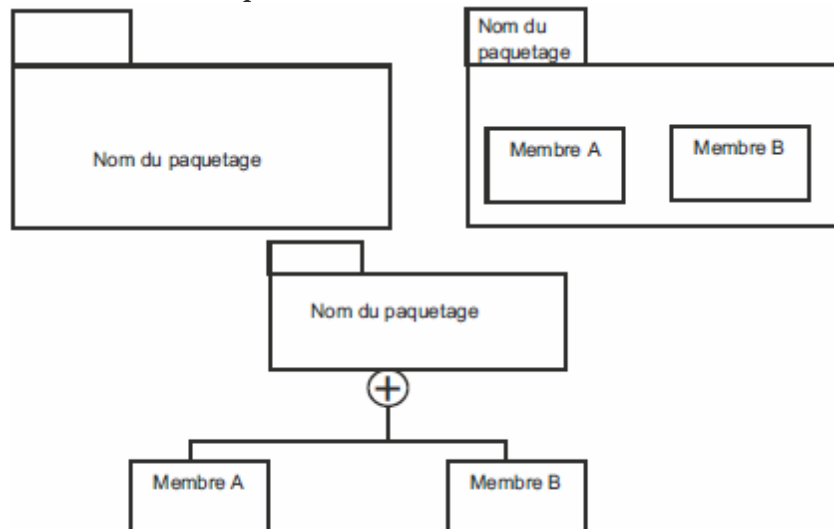
### 3.4.1. Paquetage

Un **paquetage** regroupe des éléments de la modélisation appelés aussi membres, portant sur un sous-ensemble du système. Le découpage en paquetage doit traduire un découpage logique du système à construire qui corresponde à des espaces de nommage homogènes.

Les éléments d'un paquetage peuvent avoir une visibilité déclarée soit de type public (+) soit privé (-).

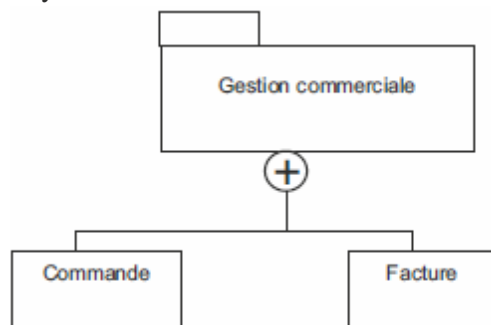
Un paquetage peut importer des éléments d'un autre paquetage. Un paquetage peut être fusionné avec un autre paquetage.

### Formalisme et exemple



Formalisme général d'un paquetage et trois manières de présenter un paquetage.

- **Représentation globale** – Le nom du paquetage se trouve à l'intérieur du grand rectangle.
- **Représentation détaillée** – Les membres du paquetage sont représentés et le nom du paquetage d'ensemble s'inscrit dans le petit rectangle.
- **Représentation éclatée** – Les membres du paquetage sont reliés par un lien connecté au paquetage par le symbole +.



Exemple de représentation éclatée d'un paquetage

### 3.4.2. Dépendance entre paquetages

La **dépendance entre paquetages** peut être qualifiée par un niveau de visibilité qui est soit public soit privé. Par défaut le type de visibilité est public.

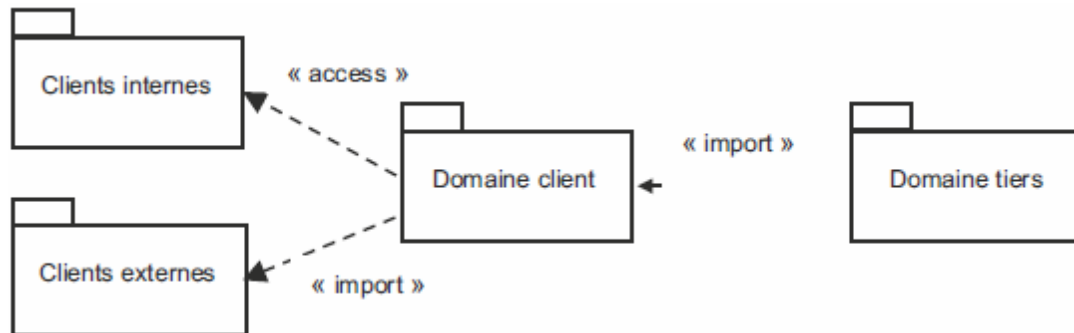
À chaque type de visibilité est associé un lien de dépendance. Les deux types de dépendances entre paquetages sont :

- « **import** » – Ce type de dépendance permet, pour un paquetage donné, d'importer l'espace de nommage d'un autre paquetage. Ainsi tous les membres du paquetage donné ont accès à tous les noms des membres du paquetage importé sans avoir à utiliser explicitement le nom du paquetage concerné. Ce type de dépendance correspond à un lien ayant une visibilité « public ».
- « **access** » – Ce type de dépendance permet, pour un paquetage donné, d'avoir accès à l'espace de nommage d'un paquetage cible. L'espace de nommage n'est donc pas importé et ne peut être transmis à

d'autres paquetages par transitivité. Ce type de dépendance correspond à un lien ayant une visibilité « privé ».

*Exemple :*

Un exemple de dépendance entre paquetages mettant en jeu les niveaux de visibilité :



Dans cet exemple, les éléments de Clients externes sont importés dans Domaine client et ensuite dans Domaine tiers. Cependant, les éléments de Clients internes sont seulement accessibles par le paquetage Domaine client et donc pas à partir du paquetage Domaine tiers.

### 3.4.3. Représentation et exemple

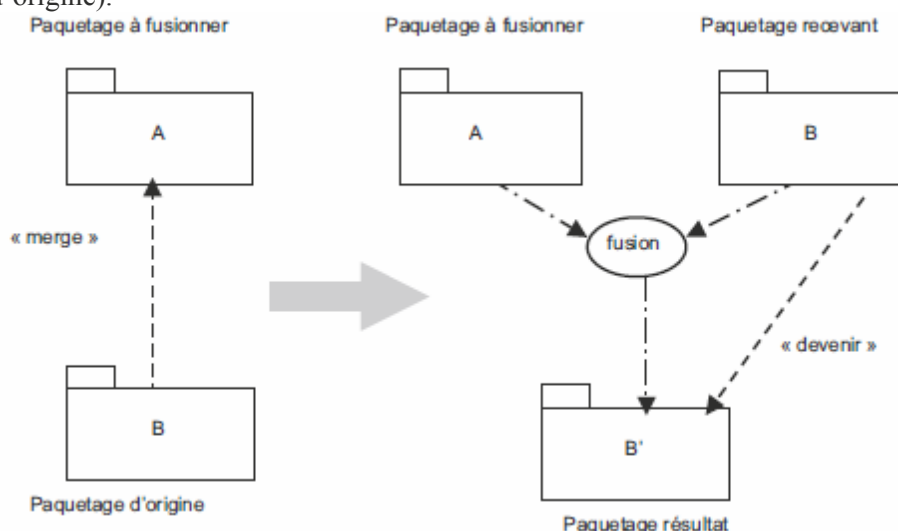
Exemple du cours : Demande de formation.

Enfin, UML propose aussi une opération de fusion entre deux paquetages. Le lien de dépendance comporte dans ce cas le mot-clé « merge ».

Ce type de lien permet de fusionner deux paquetages et d'obtenir ainsi un paquetage contenant la fusion des deux paquetages d'origine. Pour bien clarifier cette opération, il est important de qualifier par des rôles les paquetages dans cette fusion.

Ainsi trois rôles sont à distinguer :

- le paquetage à fusionner (entrant dans la fusion, mais préservé après la fusion) ;
- le paquetage recevant (paquetage d'origine avant la fusion, mais non conservé après la fusion) ;
- le paquetage résultat (paquetage contenant le résultat de la fusion et écrasant le contenu du paquetage d'origine).



## 3.5. DIAGRAMME DE STRUCTURE COMPOSITE (DSC)

Le **diagramme de structure composite** permet de décrire des collaborations d'instances (de classes, de composants...) constituant des fonctions particulières du système à développer.

### 3.5.1. Collaboration

Une **collaboration** représente un assemblage de rôles d'éléments qui interagissent en vue de réaliser une fonction donnée. Il existe deux manières de représenter une collaboration :

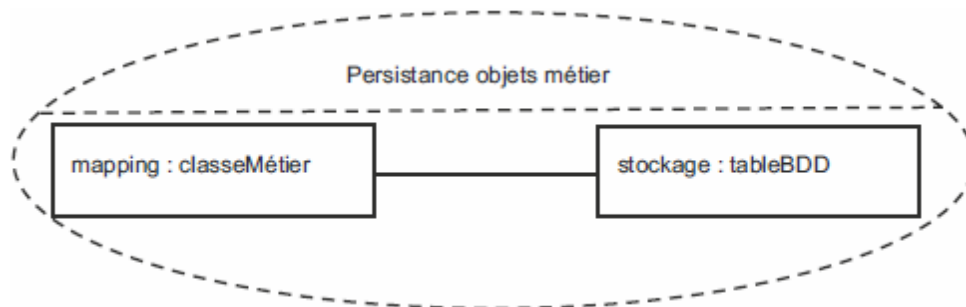
- représentation par une collaboration de rôles,
- représentation par une structure composite : le diagramme de structure composite.

### 3.5.2. Représentation et exemples

*Représentation par une collaboration de rôles*

Dans ce cas, une collaboration est formalisée par une ellipse en pointillé dans laquelle on fait figurer les rôles des éléments qui interagissent en vue de réaliser la fonction souhaitée. Dans cet exemple, la fonction *Persistence objets métier* résulte d'une collaboration entre deux rôles d'éléments :

- mapping : classeMétier,
- stockage : tableBDD.



Exemple de représentation d'une structure composite à l'aide d'une collaboration de rôles  
Persistence objets métier mapping : classeMétier stockage : tableBDD

Une nouvelle représentation permet de montrer plus explicitement les éléments de la collaboration :

- la collaboration représentée par une ellipse en pointillé ;
- les éléments participant à la collaboration (classe, composant...) représentés à l'extérieur de la collaboration ;
- les rôles considérés dans chaque participation représentés sur les liens entre les éléments participants et la collaboration.



Exemple de représentation de collaboration d'instances par un diagramme de structure composite

Dans cet exemple, la fonction Persistance objets métier résulte d'une collaboration entre la classe ClasseMétier considérée suivant le rôle mapping et la classe TableBDD considérée suivant le rôle stockage.

Cette représentation permet aussi de préciser les seuls attributs des classes participantes qui sont considérés suivant les rôles pris en compte.

**Exercices :** Voir Cours

### 3.6. DIAGRAMME DE COMPOSANT (DCP)

Le **diagramme de composant** permet de représenter les composants logiciels d'un système ainsi que les liens existant entre ces composants.

Les composants logiciels peuvent être de deux origines : soit des composants métiers propres à une entreprise soit des composants disponibles sur le marché comme par exemple les composants EJB, CORBA, .NET, WSDL.

#### 1. Composant

Chaque **composant** est assimilé à un élément exécutable du système. Il est caractérisé par :

- un nom ;
- une spécification externe sous forme soit d'une ou plusieurs interfaces requises, soit d'une ou plusieurs interfaces fournies ;
- un port de connexion.

Le port d'un composant représente le point de connexion entre le composant et une interface.

L'identification d'un port permet d'assurer une certaine indépendance entre le composant et son environnement extérieur.

*Formalisme général*



Une interface fournie se représente à l'aide d'un trait et d'un petit cercle et une interface requise à l'aide d'un trait et d'un demi-cercle. Ce sont les **connecteurs d'assemblage**.

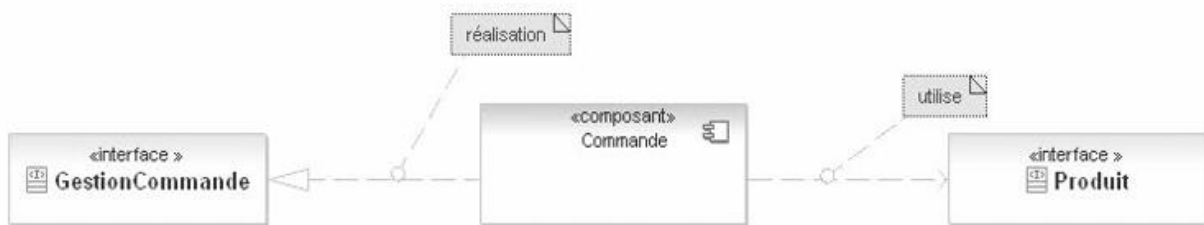


Une autre représentation peut être aussi utilisée en ayant recours aux **dépendances d'interfaces** *utilise* et *réalise* :

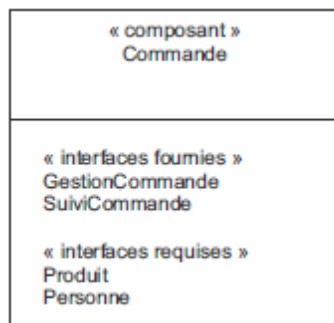
- pour une interface fournie, c'est une relation de réalisation partant du composant et allant vers l'interface ;



- pour une interface requise, c'est une dépendance avec le mot-clé « utilise » partant du composant et allant vers l'interface.

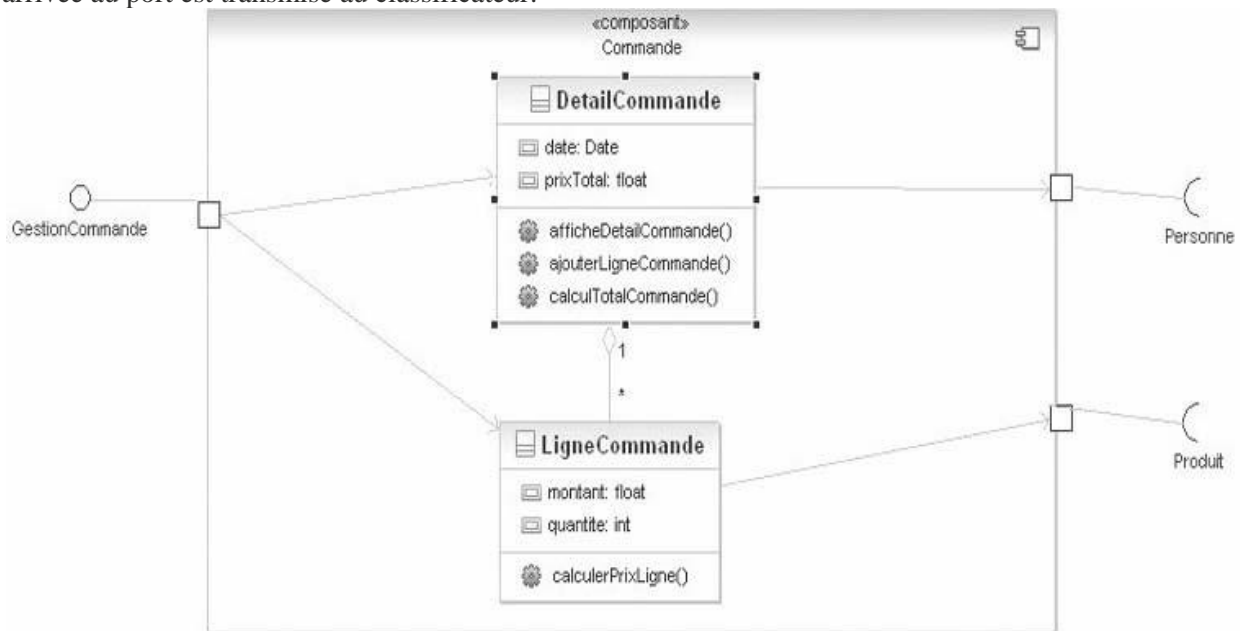


Une dernière manière de modéliser un composant avec une représentation boîte noire est de décrire sous forme textuelle les interfaces fournies et requises à l'intérieur d'un second **compartiment**.



## 2. Ports et connecteurs

Le **port** est représenté par un petit carré sur le composant. Les connecteurs permettent de relier les ports aux classificateurs. Ils sont représentés par une association navigable et indiquent que toute information arrivée au port est transmise au classificateur.



Le composant *Commande* est constitué de deux classes (classificateur) reliées par une agrégation : *DetailCommande* et *LigneCommande*.

L'interface fournie GestionCommande est accessible de l'extérieur *via* un port et permet d'accéder *via* les connecteurs aux opérations des deux classes *DetailCommande* et *LigneCommande*(ajouterLigneCommande, calculTotal-Commande, calculPrixLigne).

L'interface requise *Personne* est nécessaire pour l'affichage du détail de la commande et est accessible *via* un port du composant Commande.

L'interface requise *Produit* est nécessaire pour le calcul du prix de la ligne de commande et est accessible *via* un port du composant Commande.

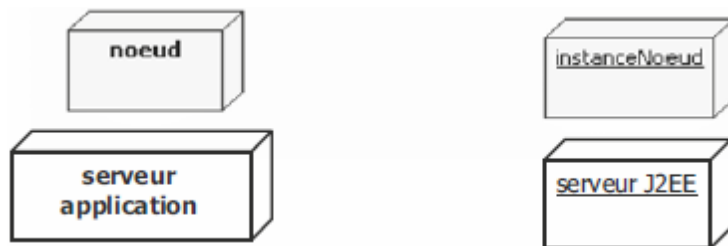
### 3.7. DIAGRAMME DE DÉPLOIEMENT (DPL)

Le **diagramme de déploiement** permet de représenter l'architecture physique supportant l'exploitation du système. Cette architecture comprend des nœuds correspondant aux supports physiques (serveurs, routeurs...) ainsi que la répartition des artefacts logiciels (bibliothèques, exécutables...) sur ces nœuds. C'est un véritable réseau constitué de nœuds et de connexions entre ces nœuds qui modélise cette architecture.

#### 1. Nœuds

Un **nœud** correspond à une **ressource matérielle de traitement sur laquelle des artefacts seront mis en œuvre pour l'exploitation du système**. Les nœuds peuvent être interconnectés pour former un réseau d'éléments physiques.

*Formalisme et exemple*



Il est possible de représenter des nœuds spécialisés. UML propose en standard les deux types de nœuds suivants :

- **Unité de traitement** – Ce nœud est une unité physique disposant de capacité de traitement sur laquelle des artefacts peuvent être déployés.

Une unité de traitement est un nœud spécialisé caractérisé par le mot-clé « device ».

- **Environnement d'exécution** – Ce nœud représente un environnement d'exécution particulier sur lequel certains artefacts peuvent être exécutés.

Un environnement d'exécution est un nœud spécialisé caractérisé par le mot-clé « executionEnvironment ».

#### 2. Artefact

Un **artefact** est la spécification d'un élément physique qui est utilisé ou produit par le processus de développement du logiciel ou par le déploiement du système. C'est donc un élément concret comme par exemple : un fichier, un exécutable ou une table d'une base de données.

Un artefact peut être relié à d'autres artefacts par notamment des liens de dépendance.

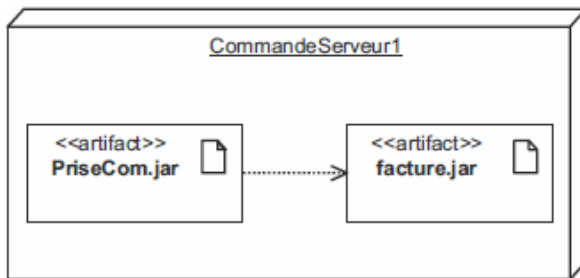
*Formalisme et exemple*



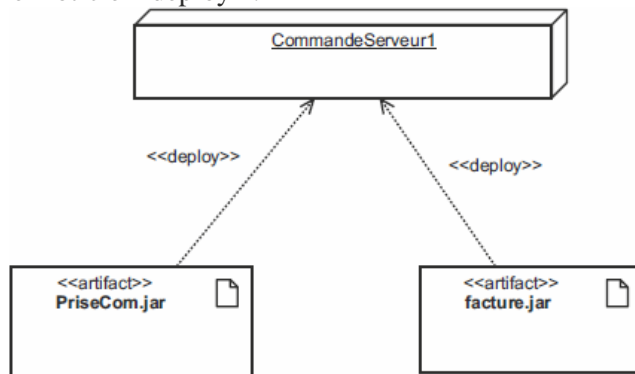
### 3. Liens entre un artefact et les autres éléments du diagramme

Il existe deux manières de représenter le lien entre un artefact et son nœud d'appartenance :

- **Représentation inclusive** – Dans cette représentation, un artefact est représenté à l'intérieur du nœud auquel il se situe physiquement.



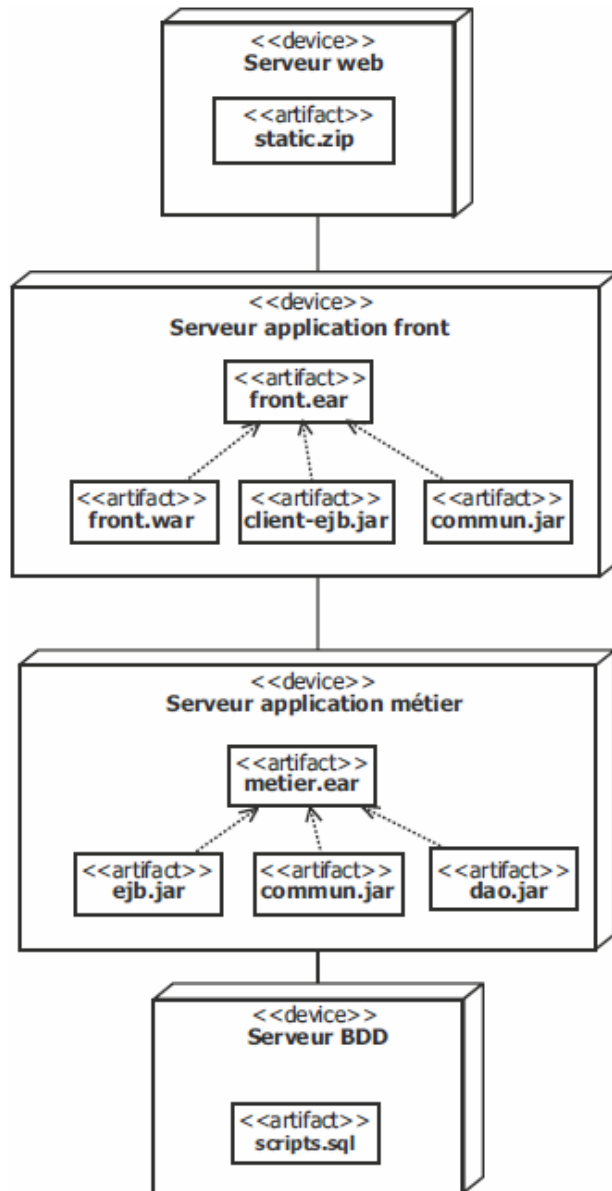
- **Représentation avec un lien de dépendance typé « deploy »** – Dans ce cas l'artefact est représenté à l'extérieur du nœud auquel il appartient avec un lien de dépendance entre l'artefact et le nœud typé avec le mot-clé « deploy ».



Un exemple relatif à une implémentation d'une architecture J2EE avec quatre nœuds est donné ci-dessous.

Dans cet exemple, plusieurs composants sont déployés.

- Un serveur web où se trouvent les éléments statiques du site dans une archive : images, feuilles de style, pages html (static.zip).
- Un serveur d'application « front » sur le quel est déployée l'archive « front.ear » composée de l'application web « front.war » et d'autres composants nécessaires au fonctionnement de cette archive web comme « clientejb.jar » (classes permettant l'appel aux EJB) et « commun.jar » (classes communes aux deux serveurs d'application).
- Un serveur d'application métier sur lequel sont déployés les composants : « ejb.jar ». Ils sont packagés dans l'archive « metier.ear ». Deux autres archives sont nécessaires au fonctionnement des EJB : « dao.jar » (classes qui permettent l'accès à la base de données) et « commun.jar » (classes communes aux deux serveurs d'application).
- Un serveur BDD (base de données) sur lequel sont stockées des procédures stockées T-SQL : « scripts.sql ».



## Chapitre IV : Modélisation dynamique ou de comportement

### 4.1. Introduction

Le diagramme de cas d'utilisation montre des acteurs qui interagissent avec les grandes fonctions d'un système. C'est une vision fonctionnelle et externe d'un système.

Le diagramme de classes, quant à lui, décrit le cœur d'un système et montre des classes et la façon dont elles sont associées. C'est une vision statique et structurelle.

Les diagrammes comportementaux sont focalisés sur la description de la **partie dynamique** du système à modéliser. Sept diagrammes sont proposés par UML 2 pour assurer cette description :

- le diagramme d'état-transition (machine d'état, DET),
- le diagramme d'activité (DAC),
- le diagramme de séquence (DSE), (diagramme d'interaction),
- le diagramme de communication (DCO), (diagramme d'interaction),
- le diagramme global d'interaction (DGI), (diagramme d'interaction),
- le diagramme de temps (DTP), (diagramme d'interaction).

Les **diagrammes d'interaction** permettent d'établir un pont entre ces deux approches : ils montrent comment des instances au cœur du système communiquent pour réaliser une certaine fonctionnalité. Les interactions sont nombreuses et variées. Il faut un langage riche pour les exprimer.

UML propose plusieurs diagrammes : **diagramme de séquence**, **diagramme de communication**, **diagramme de temps**. Ils apportent un **aspect dynamique** à la modélisation du système.

### 4.2. Diagrammes d'interaction

#### 4.2.1. Diagramme de séquence

##### 1. Concepts de base

Les diagrammes de cas d'utilisation montrent des interactions entre des acteurs et les grandes fonctions d'un système. Cependant, les interactions ne se limitent pas aux acteurs : par exemple, des objets interagissent lorsqu'ils s'échangent des messages.

L'objectif du **diagramme de séquence** est de représenter les interactions entre objets en indiquant la chronologie des échanges. Cette représentation peut se réaliser par cas d'utilisation en considérant les différents scénarios associés.

##### Définitions

Une **interaction** décrit le comportement d'un classeur en se focalisant sur l'échange d'informations entre les éléments du classeur. Les participants à une interaction sont appelés « lignes de vie ». Une ligne de vie représente un participant unique à une interaction.

UML propose principalement deux diagrammes pour illustrer une interaction : **le diagramme de séquence et celui de communication**. Une même interaction peut être représentée aussi bien par l'un que par l'autre.

Les diagrammes de communication et de séquence représentent des interactions entre des lignes de vie. Un diagramme de séquence montre des interactions sous un angle temporel, et plus particulièrement le

séquencement temporel de messages échangés entre des lignes de vie, tandis qu'un diagramme de communication montre une représentation spatiale des lignes de vie.

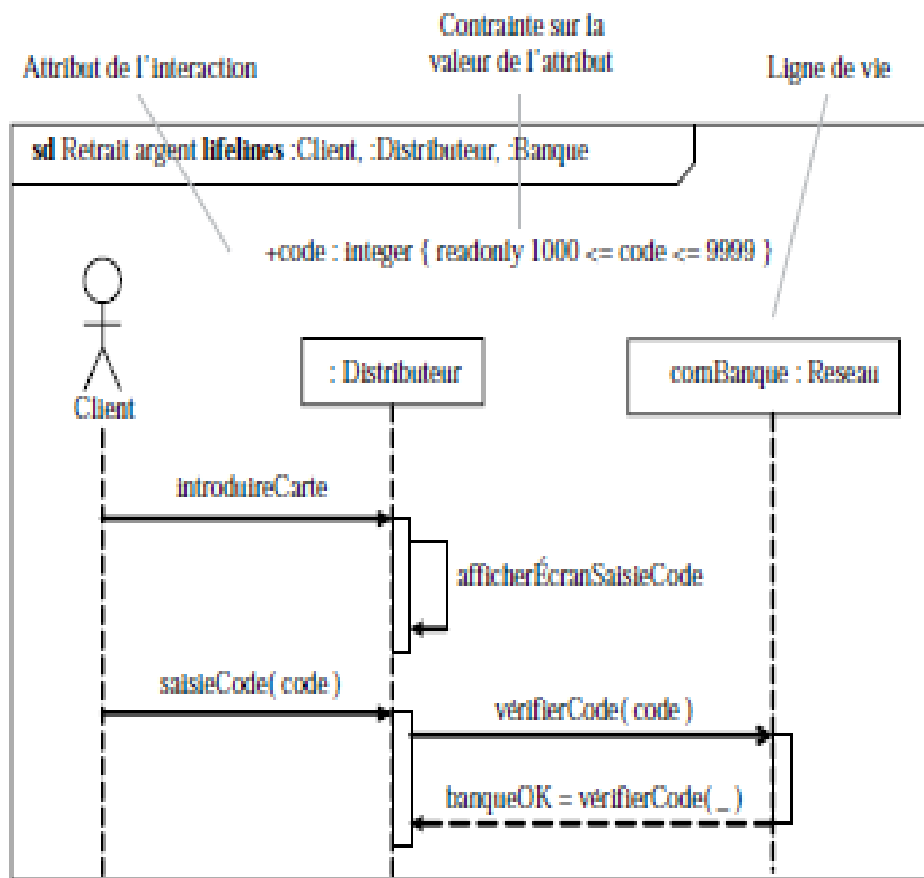
## Remarque

Aux diagrammes de séquence et de communication s'ajoute un troisième type de diagramme d'interaction : **le diagramme de temps**. Son usage est limité à la modélisation des systèmes qui s'exécutent sous de fortes contraintes de temps, comme les systèmes temps réel. Cependant, même dans ces situations extrêmes, les diagrammes de séquence conviennent bien souvent.

## Notation

Une ligne de vie se représente par un rectangle auquel est accrochée une ligne verticale pointillée. Le rectangle contient un identifiant dont la syntaxe est la suivante :

[<nomDeLaLigneDeVie> ['[sélecteur]']] : <NomDeClasseur> [décomposition]  
où le *sélecteur* permet de choisir un objet parmi n (par exemple objet[2]).



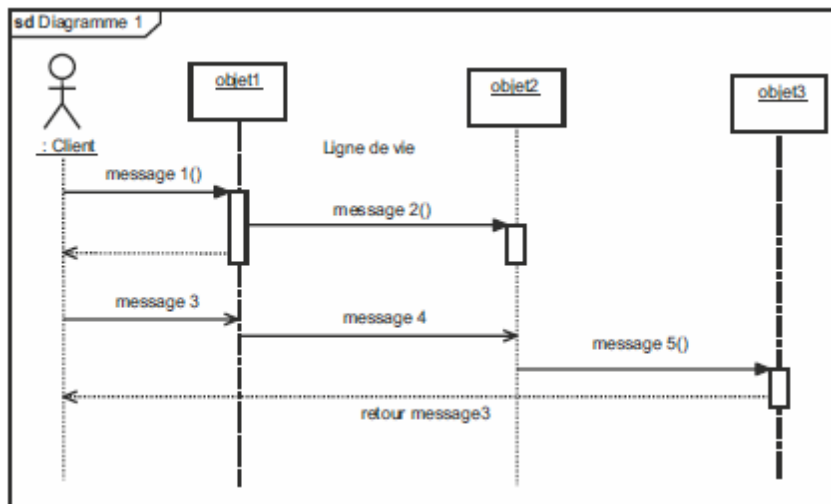
## Ligne de vie

Une **ligne de vie** représente l'ensemble des opérations exécutées par un objet. Un message reçu par un objet déclenche l'exécution d'une opération. Le retour d'information peut être implicite (cas général) ou explicite à l'aide d'un message de retour.

## Message synchrone et asynchrone

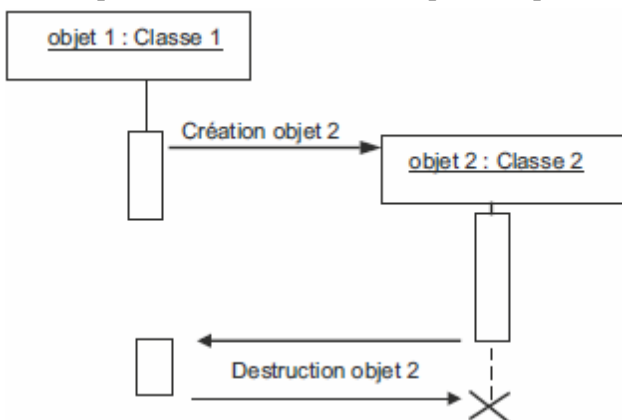
Dans un diagramme de séquence, deux types de messages peuvent être distingués :

- **Message synchrone** – Dans ce cas l'émetteur reste en attente de la réponse à son message avant de poursuivre ses actions. La flèche avec extrémité pleine symbolise ce type de message.
- **Message asynchrone** – Dans ce cas, l'émetteur n'attend pas la réponse à son message, il poursuit l'exécution de ses opérations. C'est une flèche avec une extrémité non pleine qui symbolise ce type de message.



## Création et destruction d'objet

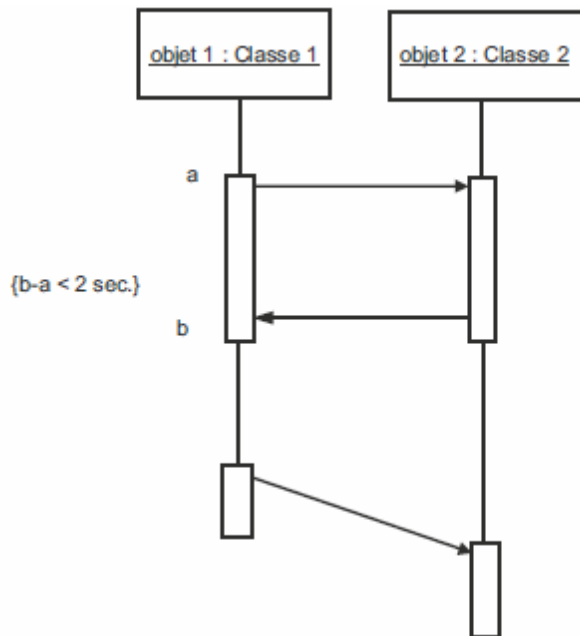
Si un objet est créé par une opération, celui-ci n'apparaît qu'au moment où il est créé. Si l'objet est détruit par une opération, la destruction se représente par « X ».



Il est aussi possible plus simplement la création d'une nouvelle instance d'objet en utilisant le mot-clé « create ».

## Contrainte temporelle

Des contraintes de chronologie entre les messages peuvent être spécifiées. De plus lorsque l'émission d'un message requiert une certaine durée, il se représente sous la forme d'un trait oblique.



## Fragment d'interaction

### Types de fragments d'interaction

Dans un diagramme de séquence, il est possible de distinguer des sous-ensembles d'interactions qui constituent des fragments.

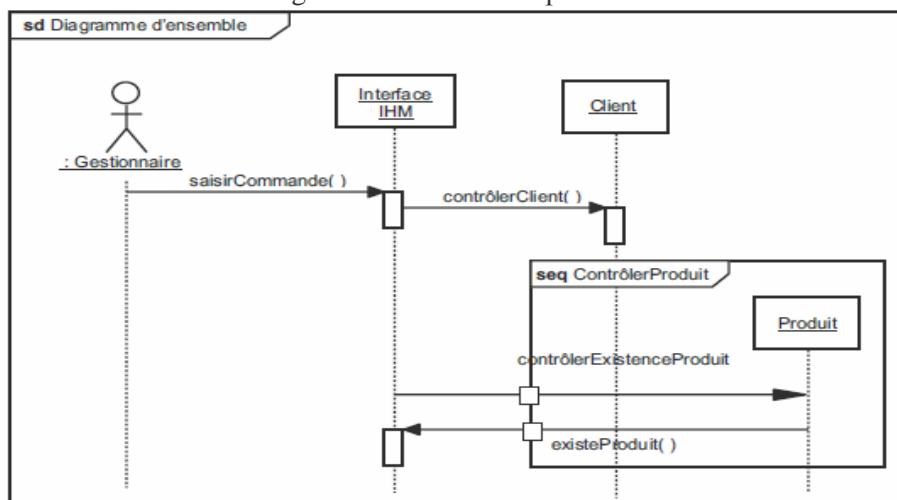
Un **fragment d'interaction** se représente globalement comme un diagramme de séquence dans un rectangle avec indication dans le coin à gauche du nom du fragment.

Treize opérateurs ont été définis dans UML : *alt*, *opt*, *loop*, *par*, *strict/weak*, *break*, *ignore/consider*, *critical*, *negative*, *assertion* et *ref*.

### Formalisme et exemple

Dans l'exemple proposé, le fragment « ContrôlerProduit » est représenté avec un port d'entrée et un port de sortie.

Un fragment d'interaction dit combiné correspond à un ensemble d'interaction auquel on applique un opérateur. Un fragment combiné se représente globalement comme un diagramme de séquence avec indication dans le coin à gauche du nom de l'opérateur.

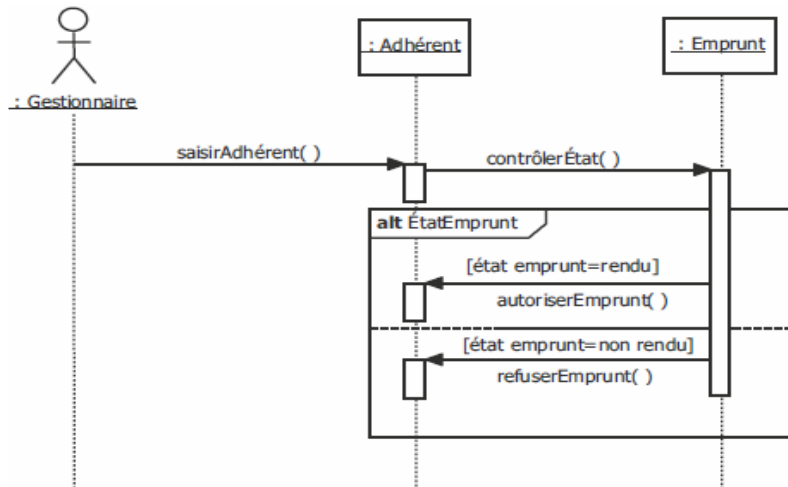




## Opérateur alt

L'opérateur **alt** correspond à une instruction de test avec une ou plusieurs alternatives possibles. Il est aussi permis d'utiliser les clauses de type sinon.

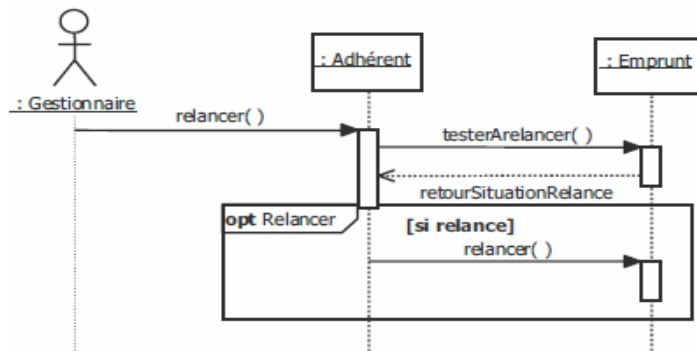
### Formalisme et exemple



## Opérateur opt

L'opérateur **opt** (optional) correspond à une instruction de test sans alternative (sinon).

### Formalisme et exemple

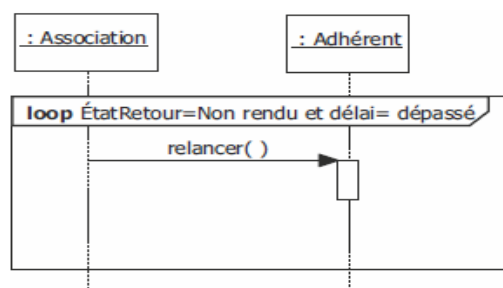


## Opérateur loop

L'opérateur **loop** correspond à une instruction de boucle qui permet d'exécuter une séquence d'interaction tant qu'une condition est satisfaite.

Il est possible aussi d'utiliser une condition portant sur un nombre minimum et maximum d'exécution de la boucle en écrivant : loop min, max. Dans ce cas, la boucle s'exécutera au minimum min fois et au maximum max fois. Il est aussi possible de combiner l'option min/max avec la condition associée à la boucle.

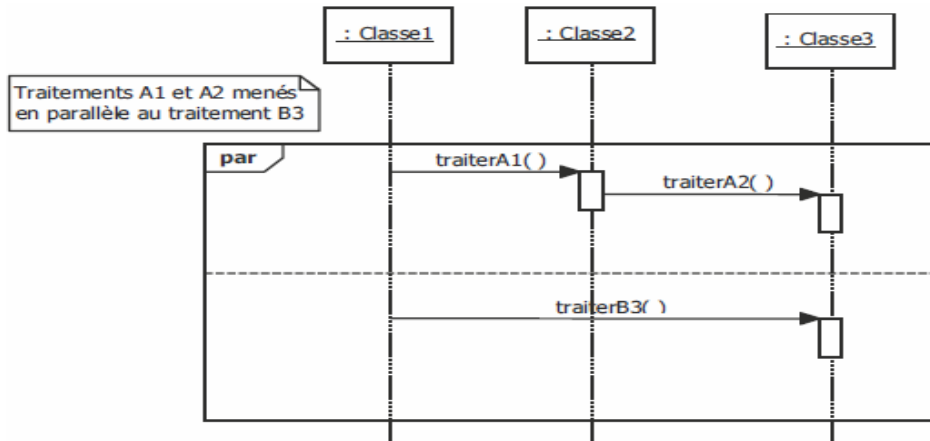
### Formalisme et exemple



## Opérateur par

L'opérateur **par** (parallel) permet de représenter deux séries d'interactions qui se déroulent en parallèle.

### Formalisme et exemple

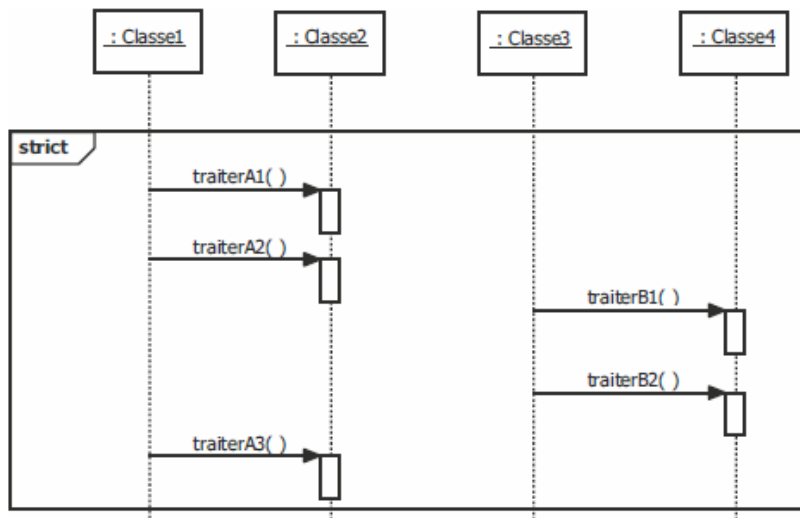


## Opérateurs strict et weak sequencing

Les opérateurs **strict** et **weak** permettent de représenter une série d'interactions dont certaines s'opèrent sur des objets indépendants :

- L'opérateur strict est utilisé quand l'ordre d'exécution des opérations doit être strictement respecté.
- L'opérateur weak est utilisé quand l'ordre d'exécution des opérations n'a pas d'importance.

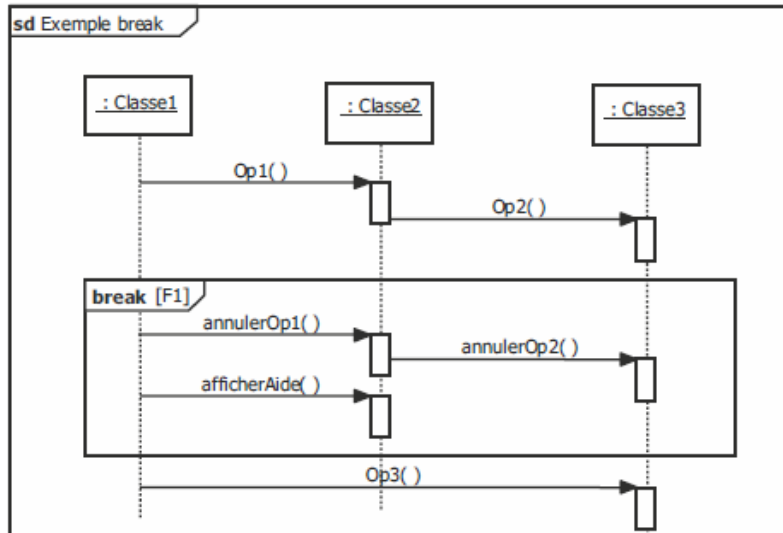
### Formalisme et exemple



## Opérateur break

L'opérateur **break** permet de représenter une situation exceptionnelle correspondant à un scénario de rupture par rapport au scénario général. Le scénario de rupture s'exécute si la condition de garde est satisfaite.

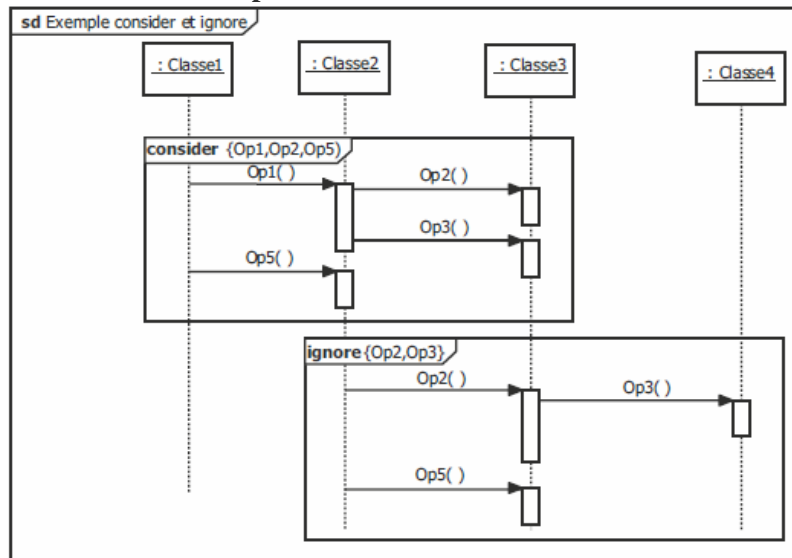
### Formalisme et exemple



## Opérateurs ignore et consider

Les opérateurs **ignore** et **consider** sont utilisés pour des fragments d'interactions dans lesquels on veut montrer que certains messages peuvent être soit absents sans avoir d'incidence sur le déroulement des interactions (ignore), soit obligatoirement présents (consider).

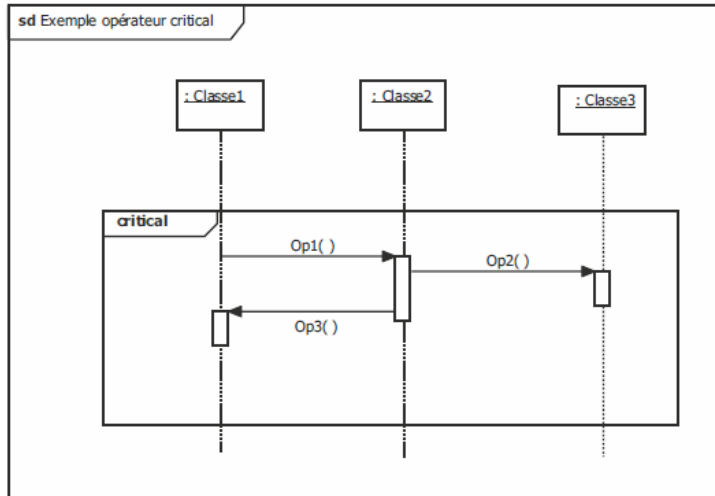
## Formalisme et exemple



## Opérateur critical

L'opérateur **critical** permet d'indiquer qu'une séquence d'interactions ne peut être interrompue compte tenu du caractère critique des opérations traitées. On considère que le traitement des interactions comprises dans la séquence critique est atomique.

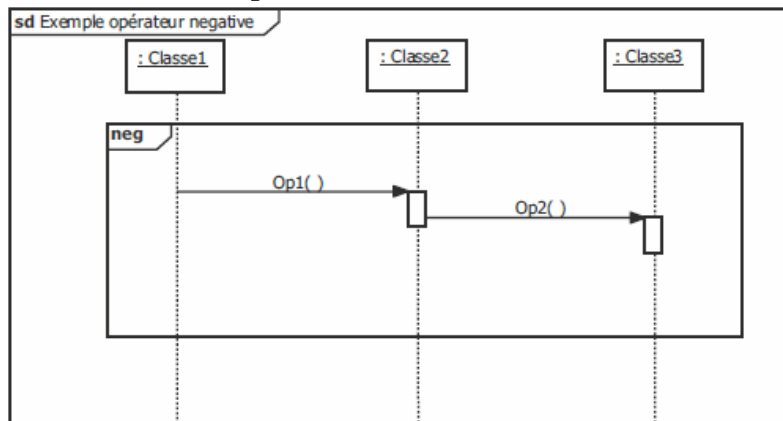
## Formalisme et exemple



## Opérateur negative

L'opérateur **neg** (negative) permet d'indiquer qu'une séquence d'interactions est invalide.

## Formalisme et exemple

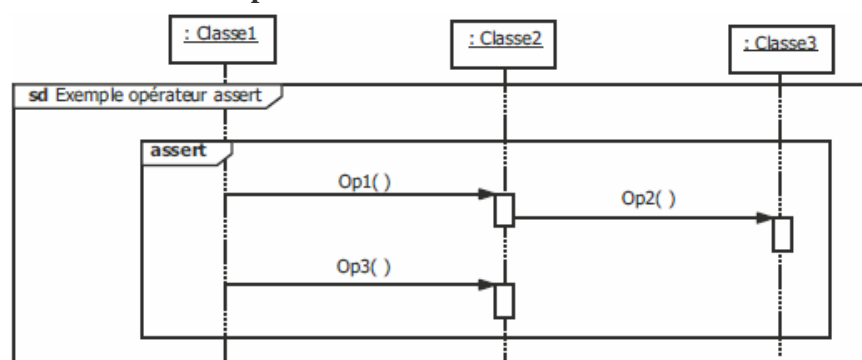


## Opérateur assertion

L'opérateur **assert** (assertion) permet d'indiquer qu'une séquence d'interactions est l'unique séquence possible en considérant les messages échangés dans le fragment.

Toute autre configuration de message est invalide.

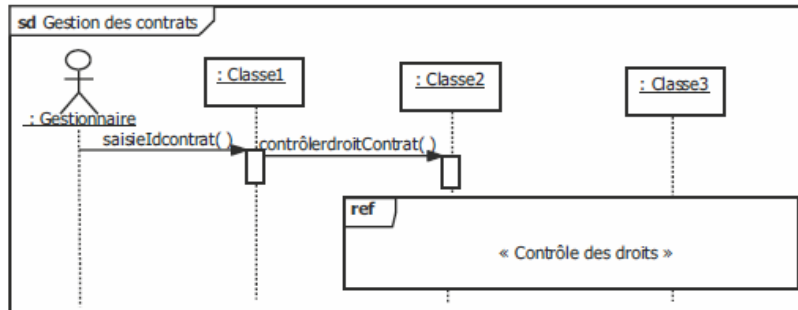
## Formalisme et exemple



## Opérateur ref

L'opérateur **ref** permet d'appeler une séquence d'interactions décrite par ailleurs constituant ainsi une sorte de sous-diagramme de séquence.

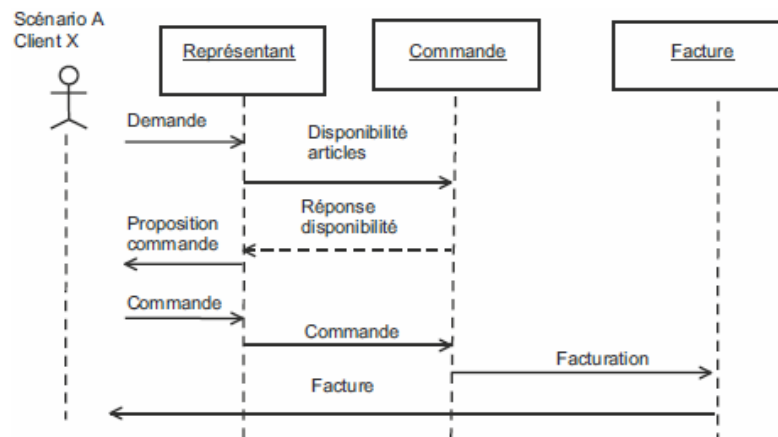
## Formalisme et exemple



## Autre utilisation du diagramme de séquence

Le diagramme de séquence peut être aussi utilisé pour documenter un cas d'utilisation.

Les interactions entre objets représentent, dans ce cas, des flux d'informations échangés et non pas de véritables messages entre les opérations des objets.



## Exercices

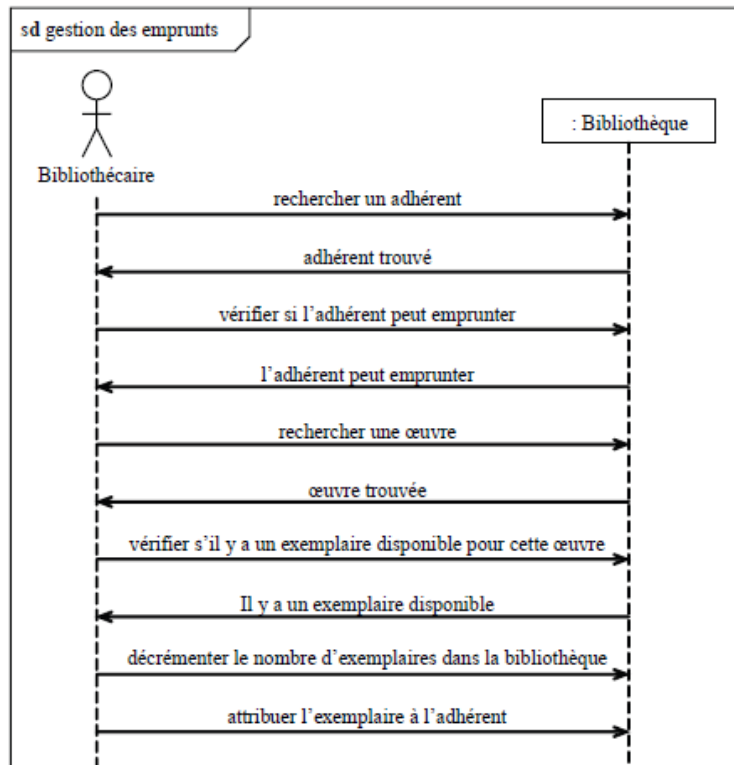
### 1. Diagrammes de séquence pour illustrer des cas d'utilisation

Le diagramme de cas d'utilisation présenté modélise la gestion simplifiée d'une bibliothèque.

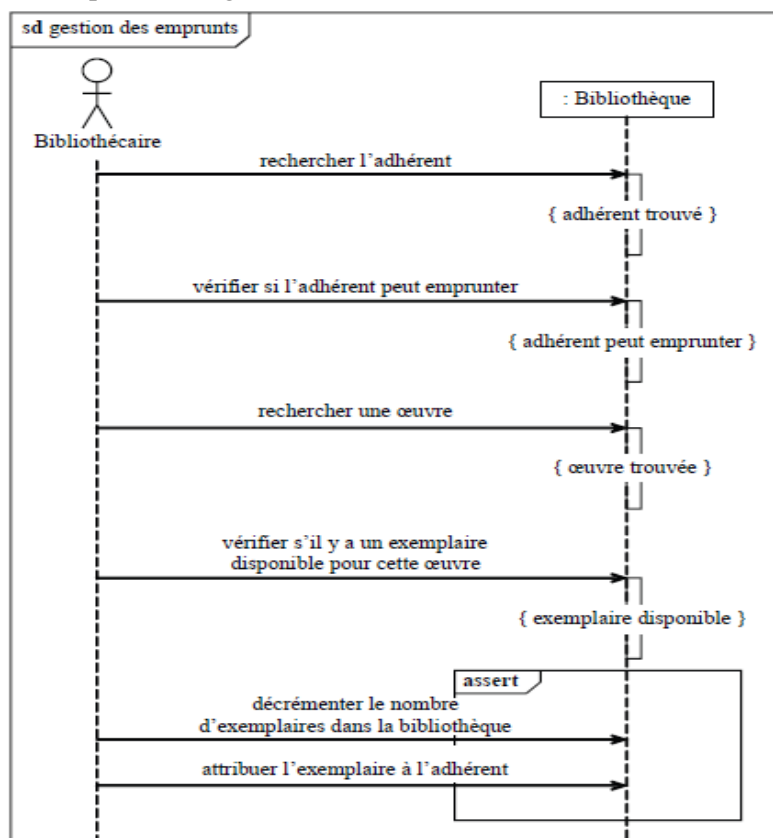


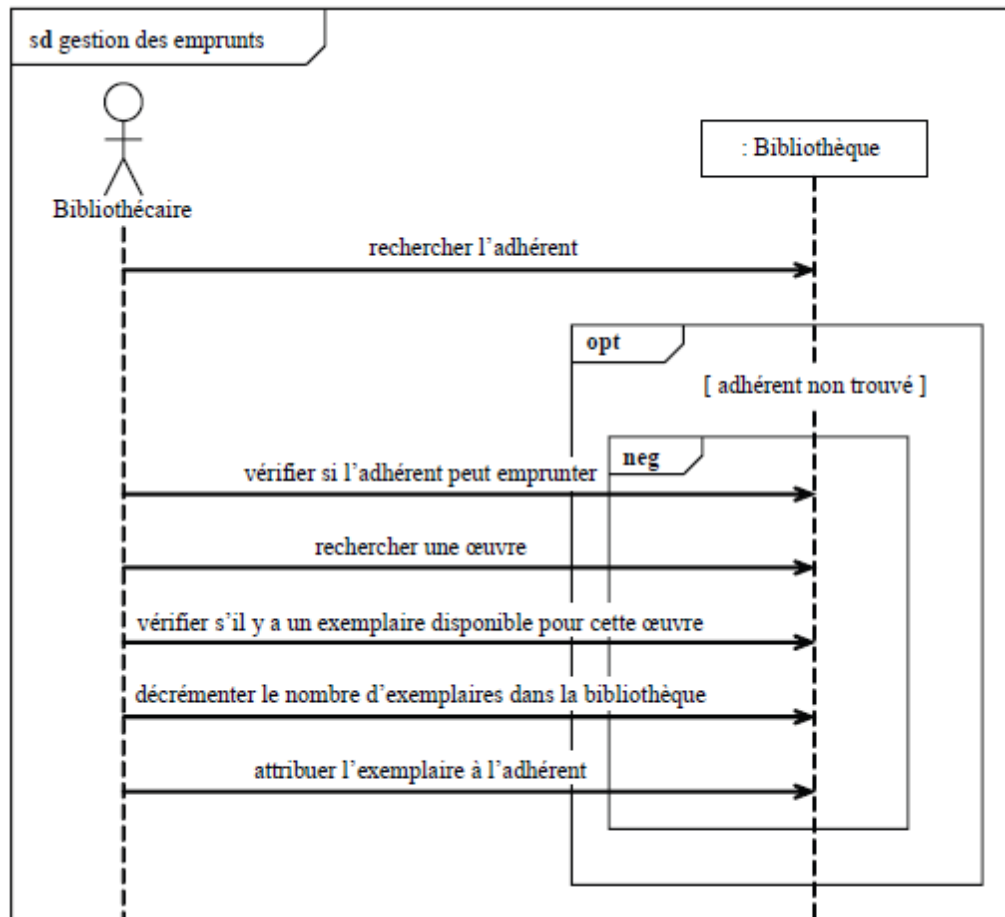
Le fonctionnement de la bibliothèque est le suivant : une bibliothèque propose à ses adhérents des œuvres littéraires ; les œuvres peuvent être présentes en plusieurs exemplaires ; un adhérent peut emprunter jusqu'à trois livres.

1. Écrivez à l'aide d'un diagramme de séquence un scénario nominal d'emprunt.
2. Écrivez à l'aide de diagrammes de séquence des scénarios d'emprunt alternatifs et d'exceptions.



On complète ce diagramme des contraintes à vérifier.





#### 4.3. DIAGRAMME DE COMMUNICATION (DCO)

##### 4.3.1. Concepts de base

Le **diagramme de communication** constitue une autre représentation des interactions que celle du diagramme de séquence. En effet, le diagramme de communication met plus l'accent sur l'aspect spatial des échanges que l'aspect temporel.

##### Rôle

Chaque participant à un échange de message correspondant à une ligne de vie dans le diagramme de séquence se représente sous forme d'un **rôle** dans le diagramme de communication. Un rôle est identifié par :

<nom de rôle> : <nom du type>

Une des deux parties de cette identification est obligatoire ainsi que le séparateur

« : ». Le **nom du rôle** correspond au nom de l'objet dans le cas où l'acteur ou la classe ont un rôle unique par rapport au système. Le **nom du type** correspond au nom de la classe lorsque l'on manipule des objets.

##### Exemple

administrateur : Utilisateur

Pour un utilisateur qui est vu au travers de son rôle d'administrateur.

## Message

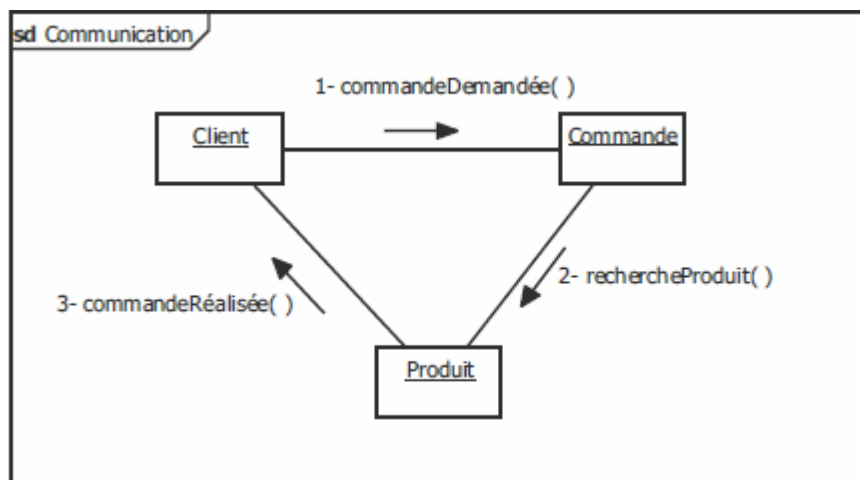
Un **message** correspond à un appel d'opération effectué par un rôle émetteur vers un rôle récepteur. Le sens du message est donné par une flèche portée au-dessus du lien reliant les participants au message (origine et destinataire). Chaque message est identifié par :

<numéro> : nom ( )

Plus précisément l'identification d'un message doit respecter la syntaxe suivante :

**[n° du message préc. reçu] « . » n° du message [clause d'itération] [condition] « : » nom du message.**

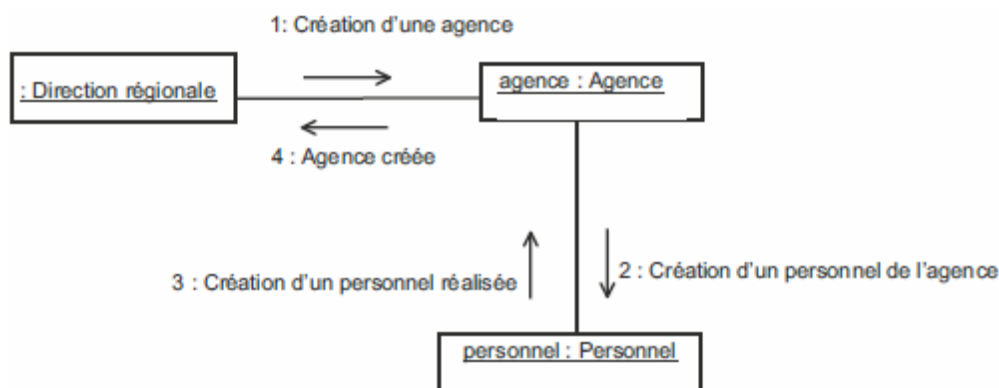
- Numéro du message précédent reçu : permet d'indiquer la chronologie des messages.
- Numéro du message : numéro hiérarchique du message de type 1.1, 1.2... avec utilisation de lettre pour indiquer la simultanéité d'envoi de message.
- Clause d'itération : indique si l'envoi du message est répété. La syntaxe est \*[spécification de l'itération].
- Condition : indique si l'envoi du message est soumis à une condition à satisfaire.



## Exercices

### Exercice 1

En reprenant le sujet de l'exercice 1 du diagramme de séquence, nous donnons son équivalent en diagramme de communication.





## 4.4. DIAGRAMME D'ETAT

### 4.4.1. Introduction au formalisme

#### Présentation

Les diagrammes d'états 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).

Ils spécifient habituellement le comportement d'une instance de classeur (classe ou composant), mais parfois aussi le comportement interne d'autres éléments tels que les cas d'utilisation, les sous-systèmes, les méthodes.

Le diagramme d'états est le seul diagramme, de la norme UML, à offrir une vision complète et non ambiguë de l'ensemble des comportements de l'élément auquel il est attaché. En effet, un diagramme d'interaction n'offre qu'une vue partielle correspondant à un scénario sans spécifier comment les différents scénarios interagissent entre eux.

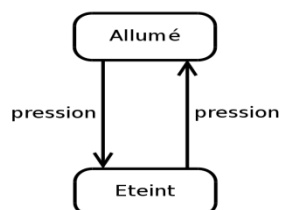
La vision globale du système n'apparaît pas sur ce type de diagramme puisqu'ils ne s'intéressent qu'à un seul élément du système indépendamment de son environnement.

Concrètement, un diagramme d'états est un graphe qui représente un *automate à états finis*, c'est-à-dire une machine dont le comportement des sorties ne dépend pas seulement de l'état de ses entrées, mais aussi d'un historique des sollicitations passées.

#### Notion et exemple d'automate à états finis

Comme nous venons de le dire, un automate à états finis est un automate dont le comportement des sorties ne dépend pas seulement de l'état de ses entrées, mais aussi d'un historique des sollicitations passées. Cet historique est caractérisé par un *état global*.

Un état global est un jeu de valeurs d'objet, pour une classe donnée, produisant la même réponse face aux événements. Toutes les instances d'une même classe ayant le même état global réagissent de la même manière à un événement. Il ne faut pas confondre les notions d'état global et d'état. La section suivante donne plus d'information sur ces deux acceptions du terme *état*.



Un diagramme d'états simple.

La figure montre un exemple simple d'automate à états finis. Cet automate possède deux états (*Allumé* et *Eteint*) et deux transitions correspondant au même évènement : la pression sur un bouton d'éclairage domestique.


### Diagrammes d'états

Un diagramme d'états-transitions rassemble et organise les états et les transitions d'un classeur donné. Bien entendu, le modèle dynamique du système comprend plusieurs diagrammes d'états-transitions. Il est souhaitable de construire un diagramme d'états-transitions pour chaque classeur (qui, le plus souvent, est une classe) possédant un comportement dynamique important. Un diagramme d'états-transitions ne peut être associé qu'à un seul classeur. Tous les automates à états finis des diagrammes d'états-transitions d'un système s'exécutent concurremment et peuvent donc changer d'état de façon indépendante.

### État

#### Les deux acceptions du terme *état*

#### État dans un diagramme d'états



état simple

Exemple d'état simple.

Un état, que l'on peut qualifier informellement d'*élémentaire*, se représente graphiquement dans un diagramme d'états par un rectangle aux coins arrondis.

Certains états, dits *composites*, peuvent contenir (*i.e.* envelopper) des sous-états.

Le nom de l'état peut être spécifié dans le rectangle et doit être unique dans le diagramme d'états-transitions, ou dans l'état enveloppant. On peut l'omettre, ce qui produit un état anonyme. Il peut y avoir un nombre quelconque d'états anonymes distincts. Un état imbriqué peut être identifié par son nom qualifié si tous les états enveloppant ont des noms.

Un état peut être partitionné en plusieurs compartiments séparés par une ligne horizontale. Le premier compartiment contient le nom de l'état et les autres peuvent recevoir des transitions internes, ou des sous-états, quand il s'agit d'un état composite. Dans le cas d'un état simple (*i.e.* sans transitions interne ou sous-état), on peut omettre toute barre de séparation.

#### État initial et final

##### État initial



Représentation graphique de l'état initial.

L'état initial est un pseudo état qui indique l'état de départ, par défaut, lorsque le diagramme d'états-transitions, ou l'état enveloppant, est invoqué. Lorsqu'un objet est créé, il entre dans l'état initial.

## État final



Représentation graphique de l'état final.

L'état final est un pseudo état qui indique que le diagramme d'états-transitions, ou l'état enveloppant, est terminé.

## Événement

### Notion d'évènement

Un événement est quelque chose qui se produit pendant l'exécution d'un système et qui mérite d'être modélisé. Les diagrammes d'états permettent justement de spécifier les réactions d'une partie du système à des événements discrets. Un événement se produit à un instant précis et est dépourvu de durée. Quand un événement est reçu, une transition peut être déclenchée et faire basculer l'objet dans un nouvel état. On peut diviser les événements en plusieurs types explicites et implicites : signal, appel, changement et temporel.

### Transition

#### Définition et syntaxe

Une transition définit la réponse d'un objet à l'occurrence d'un événement. Elle lie, généralement, deux états  $E1$  et  $E2$  et indique qu'un objet dans un état  $E1$  peut entrer dans l'état  $E2$  et exécuter certaines activités, si un événement déclencheur se produit et que la condition de garde est vérifiée.

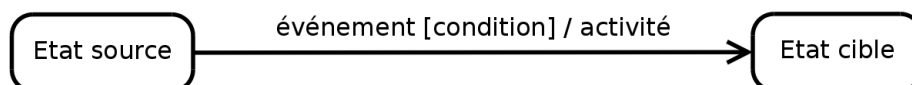
La syntaxe d'une transition est la suivante :

[ <événement> ] [ '[' <garde> ']' ] [ '/' <activité> ]

La syntaxe de <événement> a été déjà définie

Le même événement peut être le déclencheur de plusieurs transitions quittant un même état. Chaque transition avec le même événement doit avoir une condition de garde différente. En effet, une seule transition peut se déclencher dans un même flot d'exécution. Si deux transitions sont activées en même temps par un même événement, une seule se déclenche et le choix n'est pas prévisible (*i.e.* pas déterministe).

### Transition externe



Représentation graphique d'une transition externe entre deux états.

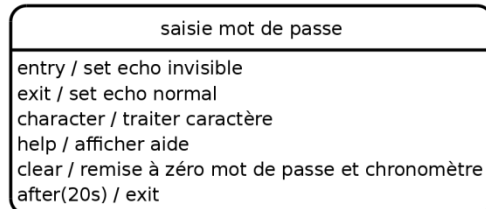
Une transition externe est une transition qui modifie l'état actif. Il s'agit du type de transition le plus répandu. Elle est représentée par une flèche allant de l'état source vers l'état cible.

### Transition d'achèvement

Une transition dépourvue d'événement déclencheur explicite se déclenche à la fin de l'activité contenue dans l'état source (y compris les états imbriqués). Elle peut contenir une condition de garde qui est évaluée au moment où l'activité contenue dans l'état s'achève, et non pas ensuite.

Les transitions de garde sont, par exemple, utilisées pour connecter les états initiaux et les états historiques avec leur état successeurs puisque ces pseudo-états ne peuvent rester actifs.

### Transition interne



Représentation de la saisie d'un mot de passe dans un état unique en utilisant des transitions internes.

Les règles de déclenchement d'une transition interne sont les mêmes que pour une transition externe excepté qu'une transition interne ne possède pas d'état cible et que l'état actif reste le même à la suite de son déclenchement. La syntaxe d'une transition interne reste la même que celle d'une transition classique. Par contre, les transitions internes ne sont pas représentées par des arcs mais sont spécifiées dans un compartiment de leur état associé.

Les transitions internes possèdent des noms d'événement prédéfinis correspondant à des déclencheurs particuliers : *entry*, *exit*, *do* et *include*. Ces mots clefs réservés viennent prendre la place du nom de l'événement dans la syntaxe d'une transition interne.

**entry** –*entry* permet de spécifier une activité qui s'accomplit quand on entre dans l'état.

**exit** –*exit* permet de spécifier une activité qui s'accomplit quand on sort de l'état.

**do** –Une activité *do* commence dès que l'activité *entry* est terminée. Lorsque cette activité est terminée, une transition d'achèvement peut être déclenchée, après l'exécution de l'activité *exit* bien entendu. Si une transition se déclenche pendant que l'activité *do* est en cours, cette dernière est interrompue et l'activité *exit* de l'état s'exécute.

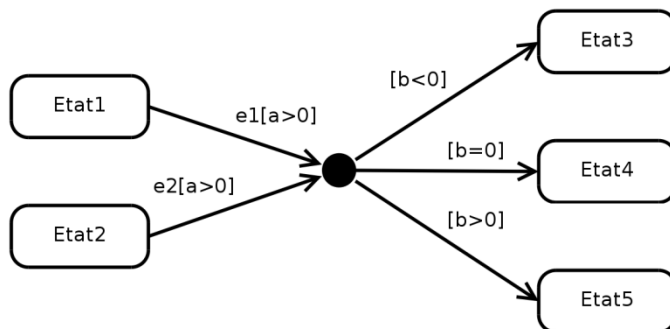
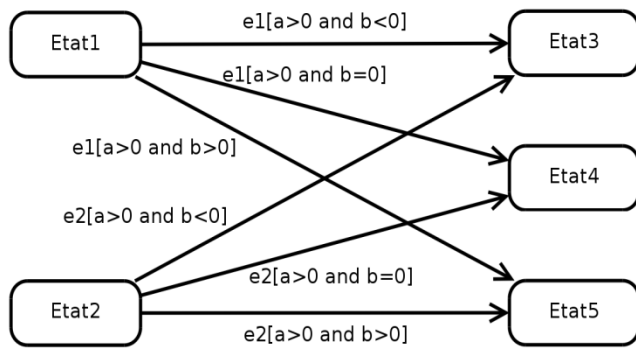
**include** –permet d'invoquer un sous-diagramme d'états-transitions.

Le déclenchement d'une transition interne ne modifie pas l'état actif et n'entraîne donc pas l'activation des activités *entry* et *exit*.

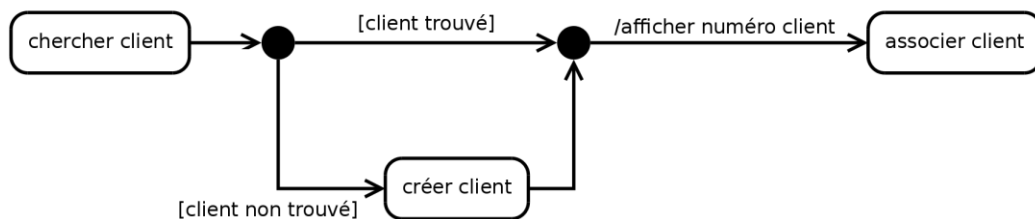
### Point de choix

Il est possible de représenter des alternatives pour le franchissement d'une transition. On utilise pour cela des pseudo-états particuliers : les points de jonction (représentés par un petit cercle plein) et les points de décision (représenté par un losange).

### Point de jonction



En haut, un diagramme sans point de jonction. En bas, son équivalent utilisant un point de jonction.



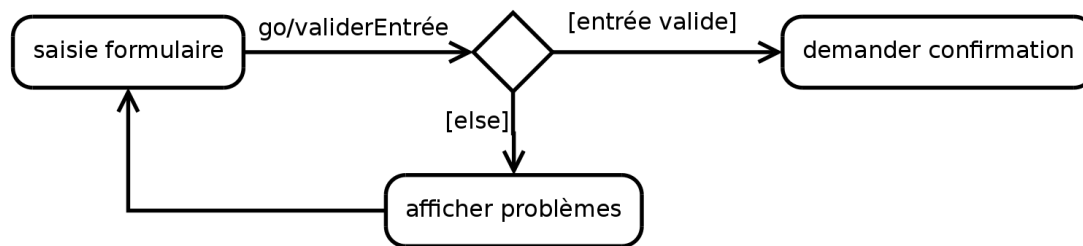
Exemple d'utilisation de deux points de jonction pour représenter une alternative.

Les points de jonction sont un artefact graphique (un pseudo-état en l'occurrence) qui permet de partager des segments de transition, l'objectif étant d'aboutir à une notation plus compacte ou plus lisible des chemins alternatifs.

Un point de jonction peut avoir plusieurs segments de transition entrante et plusieurs segments de transition sortante. Par contre, il ne peut avoir d'activité interne ni des transitions sortantes dotées de déclencheurs d'événements.

Il ne s'agit pas d'un état qui peut être actif au cours d'un laps de temps fini. Lorsqu'un chemin passant par un point de jonction est emprunté (donc lorsque la transition associée est déclenchée) toutes les gardes le long de ce chemin doivent s'évaluer à vrai dès le franchissement du premier segment.

## Point de décision

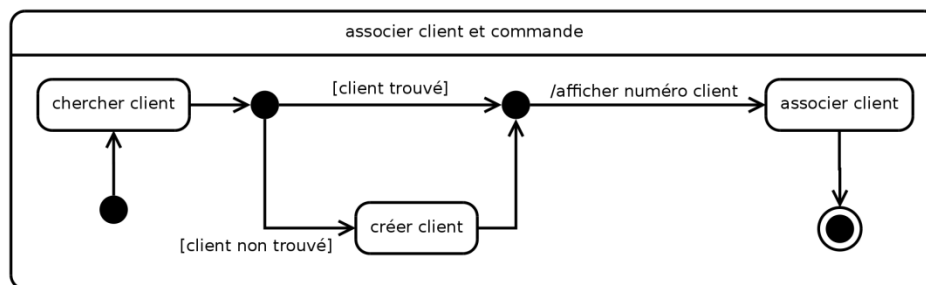


Exemple d'utilisation d'un point de décision.

Un point de décision possède une entrée et au moins deux sorties. Contrairement à un point de jonction, les gardes situées après le point de décision sont évaluées au moment où il est atteint. Cela permet de baser le choix sur des résultats obtenus en franchissant le segment avant le point de choix. Si, quand le point de décision est atteint, aucun segment en aval n'est franchissable, c'est que le modèle est mal formé.

## États composites

### Présentation

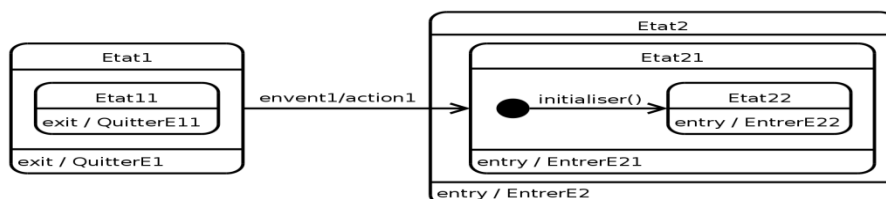


Exemple d'état composite modélisant l'association d'une commande à un client.

Un état simple ne possède pas de sous-structure mais uniquement, le cas échéant, un jeu de transitions internes. Un état composite est un état décomposé en régions contenant chacune un ou plusieurs sous-états.

### Transition

Les transitions peuvent avoir pour cible la frontière d'un état composite et sont équivalentes à une transition ayant pour cible l'état initial de l'état composite.



Exemple de configuration complexe de transition. Depuis l'état *État 1*, la réception de l'événement *event1* produit la séquence d'activités *QuitterE11*, *QuitterE1*, *action1*, *EntrerE2*, *EntrerE21*, *initialiser()*, *EntrerE22*, et place le système dans l'état *État22*.

### 5.5. Diagramme d'activités (*Activitydiagram*)

#### *Introduction au formalisme*

##### **Présentation**

**Les diagrammes d'activités permettent de mettre l'accent sur les traitements.** Ils sont donc particulièrement adaptés à la modélisation du cheminement de flots de contrôle et de flots de données. Ils permettent ainsi de représenter graphiquement 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 leur interprétation est sensiblement différente. Les diagrammes d'états-transitions sont orientés vers des systèmes réactifs, mais ils ne donnent pas une vision satisfaisante d'un traitement faisant intervenir plusieurs classeurs et doivent être complétés, par exemple, par des diagrammes de séquence. Au contraire, les diagrammes d'activités ne sont pas spécifiquement rattachés à un classeur particulier. 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.

La différence principale entre les diagrammes d'interaction et les diagrammes d'activités est que les premiers mettent l'accent sur le flot de contrôle d'un objet à l'autre, tandis que les seconds insistent sur le flot de contrôle d'une activité à l'autre.

##### **Utilisation courante**

Dans la phase de conception, les diagrammes d'activités sont particulièrement adaptés à la description des cas d'utilisation. Plus précisément, ils viennent illustrer et consolider la description textuelle des cas d'utilisation. De plus, leur représentation sous forme d'organigrammes les rend facilement intelligibles et beaucoup plus accessibles que les diagrammes d'états. On parle généralement dans ce cas de modélisation de *workflow*. On se concentre ici sur les activités telles que les voient les acteurs qui collaborent avec le système dans le cadre d'un processus métier. La modélisation du flot d'objets est souvent importante dans ce type d'utilisation des diagrammes d'activités.

#### *Activité et Transition*

##### **Action (*action*)**

**Une action est le plus petit traitement qui puisse être exprimé en UML.** Une action a une incidence sur l'état du système ou en extrait une information. Les actions sont des étapes discrètes à partir desquelles se construisent les comportements. La notion d'action est à rapprocher de la notion d'instruction élémentaire d'un langage de programmation (comme C++ ou Java). Une action peut être, par exemple :

- une affectation de valeur à des attributs ;
- un accès à la valeur d'une propriété structurelle (attribut ou terminaison d'association) ;
- la création d'un nouvel objet ou lien ;

- un calcul arithmétique simple ;
- l'émission d'un signal ;
- la réception d'un signal ;

Nous décrivons ci-dessous les types d'actions les plus courants prédéfinis dans la notation UML.

**Action appeler (*call operation*)** – L'action *call operation* correspond à l'invocation d'une opération sur un objet de manière synchrone ou asynchrone. Lorsque l'action est exécutée, les paramètres sont transmis à l'objet cible. Si l'appel est asynchrone, l'action est terminée et les éventuelles valeurs de retour seront ignorées. Si l'appel est synchrone, l'appelant est bloqué pendant l'exécution de l'opération et, le cas échéant, les valeurs de retour pourront être réceptionnées.

**Action comportement (*call behavior*)** – L'action *call behavior* est une variante de l'action *call operation* car elle invoque directement une activité plutôt qu'une opération.

**Action envoyer (*send*)** – Cette action crée un message et le transmet à un objet cible, où elle peut déclencher un comportement. Il s'agit d'un appel asynchrone (*i.e.* qui ne bloque pas l'objet appelant) bien adapté à l'envoi de signaux (*send signal*).

**Action accepter événement (*acceptevent*)** – L'exécution de cette action bloque l'exécution en cours jusqu'à la réception du type d'événement spécifié, qui généralement est un signal. Cette action est utilisée pour la réception de signaux asynchrones.

**Action accepter appel (*accept call*)** – Il s'agit d'une variante de l'action *accept event* pour les appels synchrones.

**Action répondre (*reply*)** – Cette action permet de transmettre un message en réponse à la réception d'une action de type *accept call*.

**Action créer (*create*)** – Cette action permet d'instancier un objet.

**Action détruire (*destroy*)** – Cette action permet de détruire un objet.

**Action lever exception (*raise exception*)** – Cette action permet de lever explicitement une exception. Graphiquement, les actions apparaissent dans des nœuds d'action.

### Activité (*activity*)

Une activité définit un comportement décrit par un séquençement organisé d'unités dont les éléments simples sont les actions. Le flot d'exécution est modélisé par des nœuds reliés par des arcs (transitions). Le flot de contrôle reste dans l'activité jusqu'à ce que les traitements soient terminés.

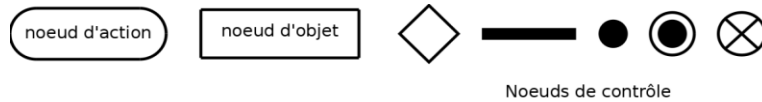
Une activité est un comportement (*behavior* en anglais) et à ce titre peut être associée à des paramètres.



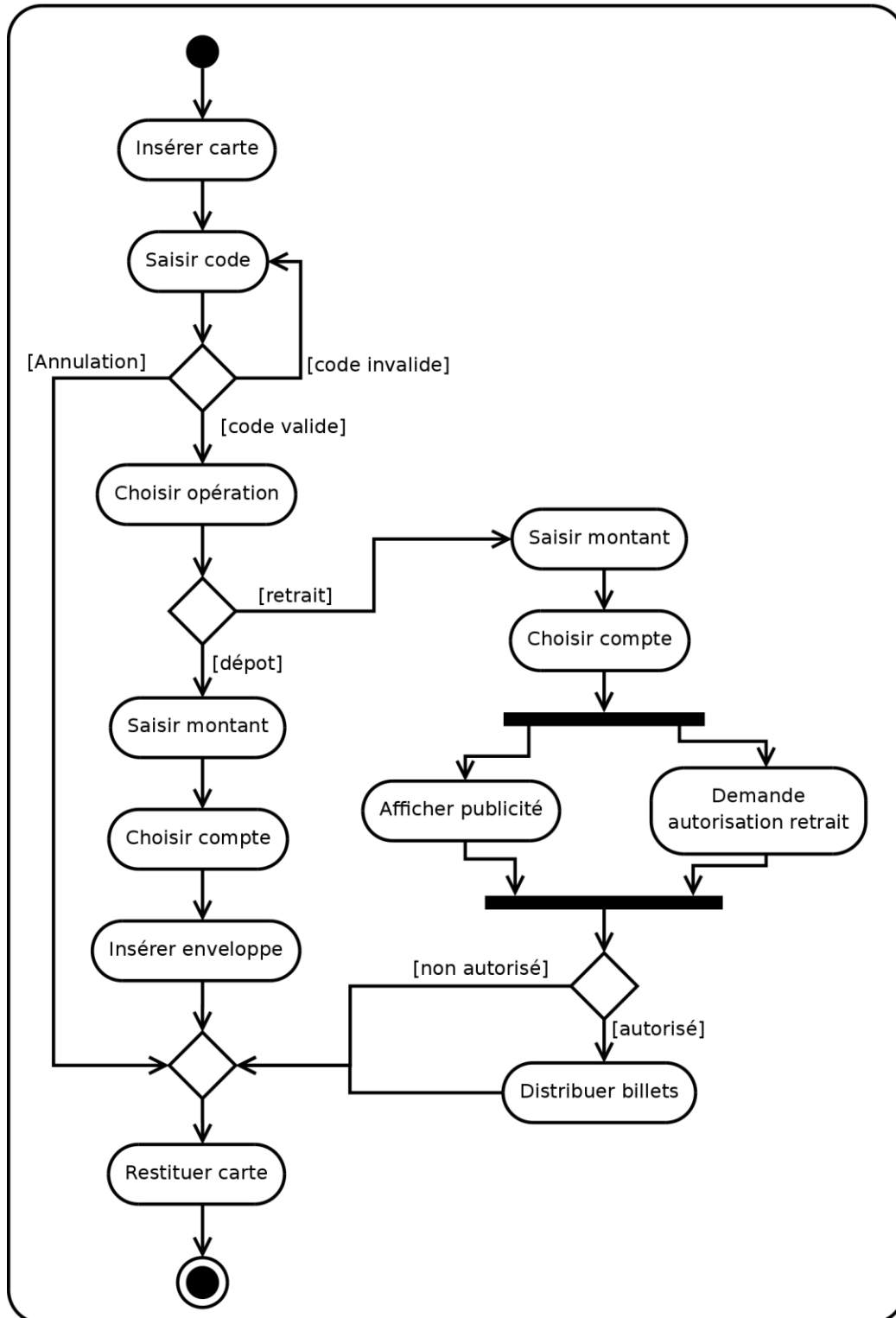
### Groupe d'activités (*activity group*)

Un groupe d'activités est une activité regroupant des nœuds et des arcs. Les nœuds et les arcs peuvent appartenir à plus d'un groupe. Un diagramme d'activités est lui-même un groupe d'activités.

### Nœud d'activité (*activity node*)



Représentation graphique des nœuds d'activité. De la gauche vers la droite, on trouve : le nœud représentant une action, qui est une variété de nœud exécutable, un nœud objet, un nœud de décision ou de fusion, un nœud de bifurcation ou d'union, un nœud initial, un nœud final et un nœud final de flot.

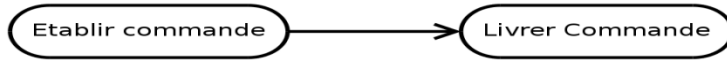


Exemple de diagramme d'activités modélisant le fonctionnement d'une borne bancaire.

Un nœud d'activité est un type d'élément abstrait permettant de représenter les étapes le long du flot d'une activité. Il existe trois familles de nœuds d'activités :

- les nœuds d'exécutions (*executable node* en anglais) ;
- les nœuds objets (*object node* en anglais) ;
- et les nœuds de contrôle (*control node* en anglais).

### Transition



Représentation graphique d'une transition.

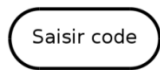
Le passage d'une activité vers une autre est matérialisé par une transition. Graphiquement les transitions sont représentées par des flèches en traits pleins qui connectent les activités entre elles. Elles sont déclenchées dès que l'activité source est terminée et provoquent automatiquement et immédiatement le début de la prochaine activité à déclencher (l'activité cible). Contrairement aux activités, les transitions sont franchies de manière atomique, en principe sans durée perceptible.

Les transitions spécifient l'enchaînement des traitements et définissent le flot de contrôle.

### Nœud exécutable (*executable node*)

Un nœud exécutable est un nœud d'activité qu'on peut exécuter (*i.e.* une activité).

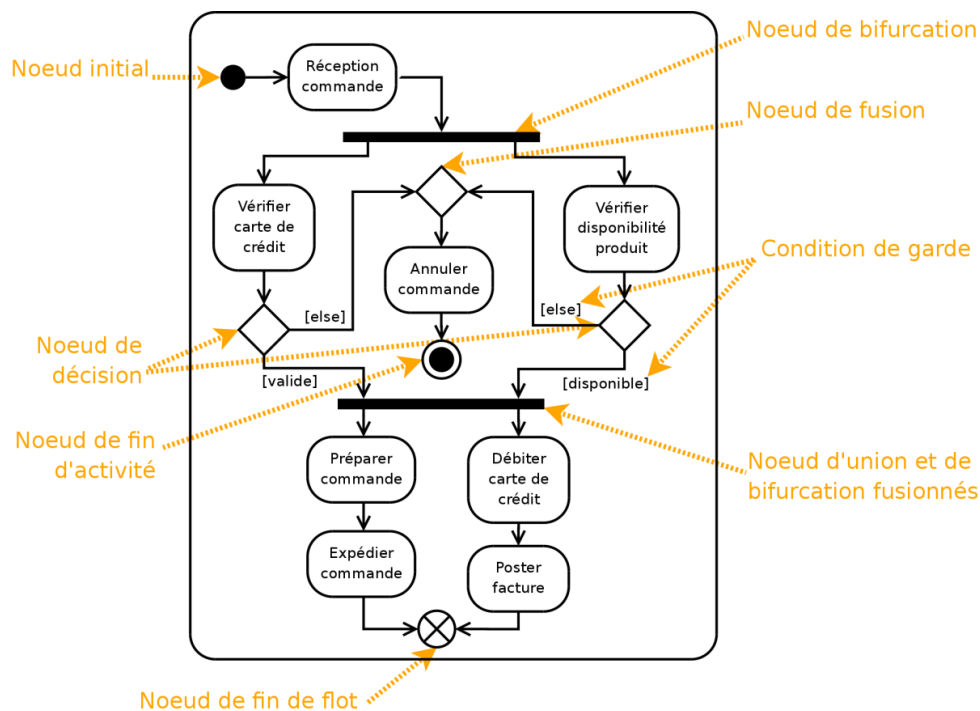
### Nœud d'action



Représentation graphique d'un nœud d'action.

Un nœud d'action est un nœud d'activité exécutable qui constitue l'unité fondamentale de fonctionnalité exécutable dans une activité.

## Nœud de contrôle (control node)



Exemple de diagramme d'activité illustrant l'utilisation de nœuds de contrôle. Ce diagramme décrit la prise en compte d'une commande.

Un nœud de contrôle est un nœud d'activité abstrait utilisé pour coordonner les flots entre les nœuds d'une activité.

Il existe plusieurs types de nœuds de contrôle :

- nœud initial (*initial node* en anglais) ;
- nœud de fin d'activité (*final node* en anglais)
- nœud de fin de flot (*flow final* en anglais) ;
- nœud de décision (*decision node* en anglais) ;
- nœud de fusion (*merge node* en anglais) ;
- nœud de bifurcation (*fork node* en anglais) ;
- nœud d'union (*join node* en anglais).

## Nœud initial

Un nœud initial est un nœud de contrôle à partir duquel le flot débute lorsque l'activité enveloppante est invoquée. Une activité peut avoir plusieurs nœuds initiaux. Un nœud initial possède un arc sortant et pas d'arc entrant.

Graphiquement, un nœud initial est représenté par un petit cercle plein.

### Nœud final

Un nœud final est un nœud de contrôle possédant un ou plusieurs arcs entrants et aucun arc sortant.

### Nœud de fin d'activité

Lorsque l'un des arcs d'un nœud de fin d'activité est activé (*i.e.* lorsqu'un flot d'exécution atteint un nœud de fin d'activité), l'exécution de l'activité enveloppante s'achève et tout nœud ou flot actif au sein de l'activité enveloppante est abandonné. Si l'activité a été invoquée par un appel synchrone, un message (*reply*) contenant les valeurs sortantes est transmis en retour à l'appelant.

Graphiquement, un nœud de fin d'activité est représenté par un cercle vide contenant un petit cercle plein.

### Nœud de fin de flot

Lorsqu'un flot d'exécution atteint un nœud de fin de flot, le flot en question est terminé, mais cette fin de flot n'a aucune incidence sur les autres flots actifs de l'activité enveloppante.

Graphiquement, un nœud de fin de flot est représenté par un cercle vide barré d'un X.

### Nœud de décision et de fusion

#### Nœud de décision (*decision node*)

Un nœud de décision est un nœud de contrôle qui permet de faire un choix entre plusieurs flots sortants. Il possède un arc entrant et plusieurs arcs sortants. Ces derniers sont généralement accompagnés de conditions de garde pour conditionner le choix. Graphiquement, on représente un nœud de décision par un losange.

#### Nœud de fusion (*merge node*)

Un nœud de fusion est un nœud de contrôle qui rassemble plusieurs flots alternatifs entrants en un seul flot sortant. Il n'est pas utilisé pour synchroniser des flots concurrents (c'est le rôle du nœud d'union) mais pour accepter un flot parmi plusieurs.

Graphiquement, on représente un nœud de fusion, comme un nœud de décision, par un losange.

### Nœud de bifurcation et d'union

#### Nœud de bifurcation ou de débranchement (*fork node*)

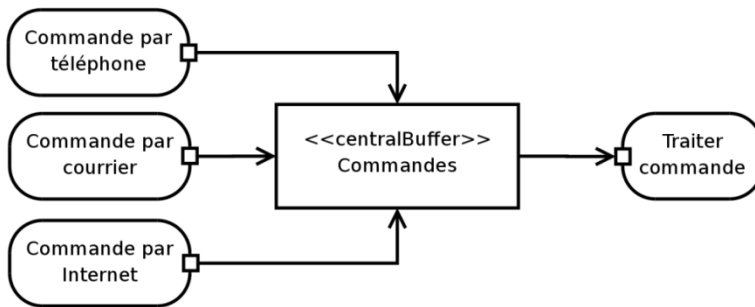
Un nœud de bifurcation, également appelé nœud de débranchement est un nœud de contrôle qui sépare un flot en plusieurs flots concurrents. Un tel nœud possède donc un arc entrant et plusieurs arcs sortants. Graphiquement, on représente un nœud de bifurcation par un trait plein.

## Nœud d'union ou de jointure (*join node*)

Un nœud d'union, également appelé nœud de jointure est un nœud de contrôle qui synchronise des flots multiples. Un tel nœud possède donc plusieurs arcs entrants et un seul arc sortant. Lorsque tous les arcs entrants sont activés, l'arc sortant l'est également.

Graphiquement, on représente un nœud d'union, comme un nœud de bifurcation, par un trait plein.

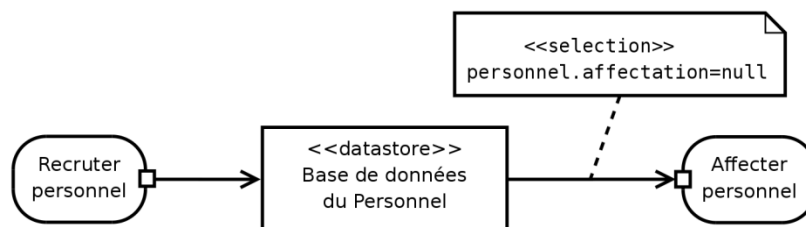
## Nœud tampon central (*central buffer node*)



Exemple d'utilisation d'un nœud tampon central pour centraliser toutes les commandes prises par différents procédés, avant qu'elles soient traitées.

Graphiquement, un nœud tampon central est représenté comme un nœud d'objet stéréotypé «*centralBuffer*».

## Nœud de stockage des données (*data store node*)



Dans cette modélisation, le personnel, après avoir été recruté par l'activité *Recruter personnel*, est stocké de manière persistante dans le nœud de stockage *Base de données du Personnel*. Bien qu'ils restent dans ce nœud, chaque employé qui n'a pas encore reçu d'affectation (étiquette stéréotypée «*selection*» : *personnel.affectation=null*) est disponible pour être utilisé par l'activité *Affecter personnel*.

Un nœud de stockage des données est un nœud tampon central particulier qui assure la persistance des données.

Graphiquement, un nœud tampon central est représenté comme un nœud d'objet détaché stéréotypé «*datastore*».

Partitions

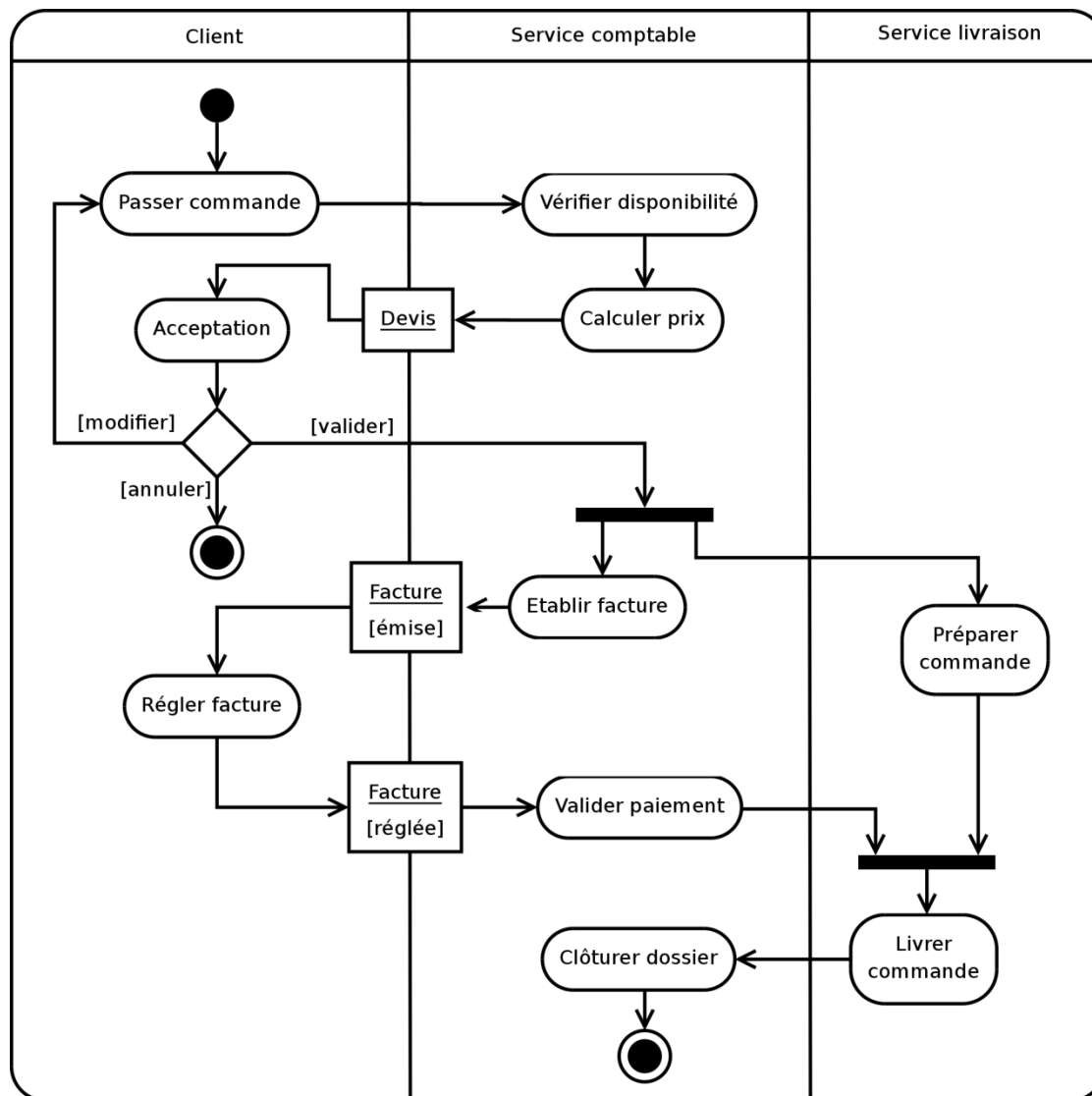


Illustration de l'utilisation de nœuds d'objets et de partitions dans un diagramme d'activités.

Les partitions, souvent appelées couloirs ou lignes d'eau (*swim lane*) du fait de leur notation, permettent d'organiser les nœuds d'activités dans un diagramme d'activités en opérant des regroupements.

Les partitions n'ont pas de signification bien arrêtée, mais correspondent souvent à des unités d'organisation du modèle. On peut, par exemple, les utiliser pour spécifier la classe responsable de la mise en œuvre d'un ensemble tâche. Dans ce cas, la classe en question est responsable de l'implémentation du comportement des nœuds inclus dans ladite partition.

Graphiquement, les partitions sont délimitées par des lignes continues. Il s'agit généralement de lignes verticales.

## **PRÉSENTATION D'UP**

UML n'est qu'un langage de modélisation. Nous n'avons pas aujourd'hui dans la norme, de démarche unifiée pour construire les modèles et conduire un projet mettant en œuvre UML. Cependant les auteurs d'UML, ont décrit, dans un ouvrage [Jacobson2000a] le processus unifié (UP, Unified Process) qui doit être associé à UML. Nous n'allons pas, dans le cadre de cet ouvrage, donner une présentation détaillée d'UP. Cependant il nous a paru intéressant de dégager les idées fondatrices d'UP dans le cadre d'une présentation générale. Nous allons tout d'abord expliciter les principes de la méthode UP. Nous compléterons ensuite cette présentation générale en décrivant l'architecture à deux dimensions d'UP et ses principaux concepts, nous passerons aussi en revue les différentes phases d'UP, et pour finir nous détaillerons les activités d'UP.

### ***LES PRINCIPES D'UP***

Le processus de développement UP, associé à UML, met en œuvre les principes suivants :

- processus guidé par les cas d'utilisation,
- processus itératif et incrémental,
- processus centré sur l'architecture,
- processus orienté par la réduction des risques.

Ces principes sont à la base du processus unifié décrit par les auteurs d'UML.

#### **1 Processus guidé par les cas d'utilisation**

L'orientation forte donnée ici par UP est de montrer que le système à construire se définit d'abord avec les utilisateurs. Les cas d'utilisation permettent d'exprimer les interactions du système avec les utilisateurs, donc de capturer les besoins. Une seconde orientation est de montrer comment les cas d'utilisation constituent un vecteur structurant pour le développement et les tests du système. Ainsi le développement peut se décomposer par cas d'utilisation et la réception du logiciel sera elle aussi articulée par cas d'utilisation.

#### **2 Processus itératif et incrémental**

Ce type de démarche étant relativement connu dans l'approche objet, il paraît naturel qu'UP préconise l'utilisation du principe de développement par itérations successives. Concrètement, la réalisation de maquette et prototype constitue la réponse pratique à ce principe. Le développement progressif, par incrément, est aussi recommandé en s'appuyant sur la décomposition du système en cas d'utilisation. Les avantages du développement itératif se caractérisent comme suit: • les risques sont évalués et traités au fur et à mesure des itérations, • les premières itérations permettent d'avoir un feed-back des utilisateurs, • les tests et l'intégration se font de manière continue, • les avancées sont évaluées au fur et à mesure de l'implémentation.



### 3 Processus centré sur l'architecture

Les auteurs d'UP mettent en avant la préoccupation de l'architecture du système dès le début des travaux d'analyse et de conception. Il est important de définir le plus tôt possible, même à grandes mailles, l'architecture type qui sera retenue pour le développement, l'implémentation et ensuite le déploiement du système. Le vecteur des cas d'utilisation peut aussi être utilisé pour la description de l'architecture.

### 4 Processus orienté par la réduction des risques

L'analyse des risques doit être présentée à tous les stades de développement d'un système. Il est important de bien évaluer les risques des développements afin d'aider à la bonne prise de décision. Du fait de l'application du processus itératif, UP contribue à la diminution des risques au fur et à mesure du déroulement des itérations successives.

## LES CONCEPTS ET LES DEUX DIMENSIONS DU PROCESSUS UP

### 4.3.1 Définition des principaux concepts et schéma d'ensemble

Le processus unifié décrit qui fait quoi, comment et quand les travaux sont réalisés tout au long du cycle de vie du projet. Quatre concepts d'UP répondent à ces questions :

- Rôle (qui ?)
- Activité (comment ?)
- Artefact (quoi ?)
- Workflow (quand ?)

#### Rôle

Un rôle définit le comportement et les responsabilités d'une ressource ou d'un groupe de ressources travaillant en équipe. Le rôle doit être considéré en termes de « casquette » qu'une ressource peut revêtir sur le projet. Une ressource peut jouer plusieurs rôles sur le projet. Par exemple sur un projet, Paul peut être à la fois chef de projet et architecte. Il représente deux rôles au sens d'UP.

#### Activité

Les rôles ont des activités qui définissent le travail qu'ils effectuent. Une activité est une unité de travail qu'une ressource, dans un rôle bien précis, peut effectuer et qui produit un résultat dans le cadre du projet. L'activité a un but clairement établi, généralement exprimée en termes de création ou de mise à jour d'artefacts, comme un modèle, une classe ou un planning. Les ressources sont affectées aux activités selon leurs compétences et leur disponibilité. Par exemple,

les activités « planifier une itération » et « anticiper les risques » sont attribuées au rôle de chef de projet.

### *Phases et itérations du processus (aspect dynamique)*

Le processus unifié, organisé en fonction du temps, est divisé en quatre phases successives.

- Inception (Lancement).
- Élaboration.
- Construction.
- Transition.

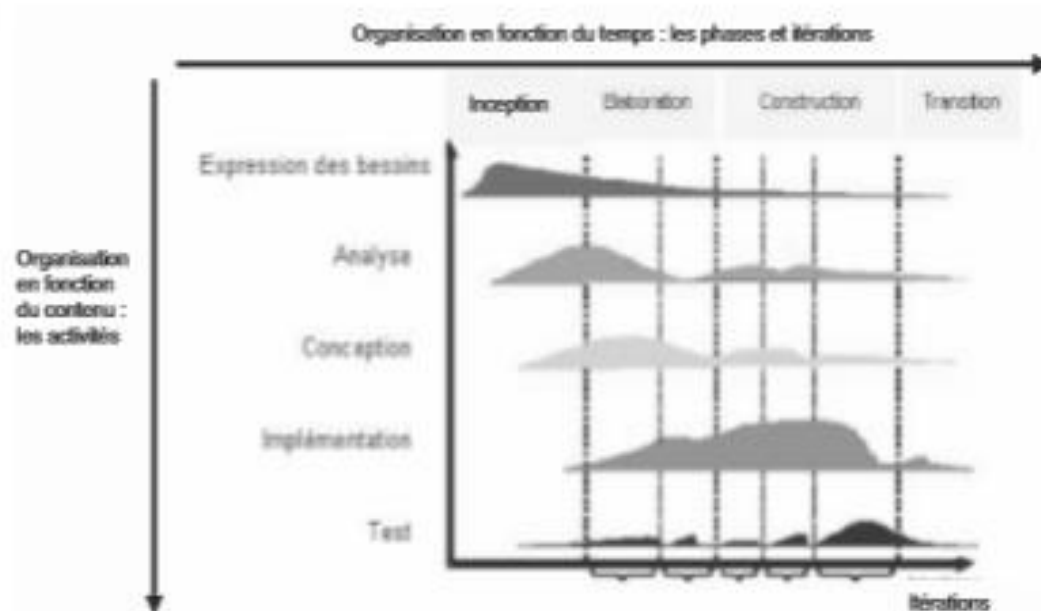


Schéma d'ensemble d'UP

### **Inception (Lancement)**

Cette phase correspond à l'initialisation du projet où l'on mène une étude d'opportunité et de faisabilité du système à construire. Une évaluation des risques est aussi réalisée dès cette phase. En outre, une identification des principaux cas d'utilisation accompagnée d'une description générale est modélisée dans un diagramme de cas d'utilisation afin de définir le périmètre du projet. Il est possible, à ce stade, de faire réaliser des maquettes sur un sous-ensemble des cas d'utilisation identifiés. Ce n'est qu'à l'issue de cette première phase que l'on peut considérer le projet véritablement lancé.

### Élaboration

Cette phase reprend les résultats de la phase d'inception et élargit l'appréciation de la faisabilité sur la quasi-totalité des cas d'utilisation. Ces cas d'utilisation se retrouvent dans le diagramme des cas d'utilisation qui est ainsi complété. Cette phase a aussi pour but d'analyser le domaine technique du système à développer afin d'aboutir à une architecture stable. Ainsi, toutes les exigences non recensées dans les cas d'utilisation, comme par exemple les exigences de performances du système, seront prises en compte dans la conception et l'élaboration de l'architecture. L'évaluation des risques et l'étude de la rentabilité du projet sont aussi précisées. Un planning est réalisé pour les phases suivantes du projet en indiquant le nombre d'itérations à réaliser pour les phases de construction.

### Construction

Cette phase correspond à la production d'une première version du produit. Elle est donc fortement centrée sur les activités de conception, d'implémentation et de test. En effet, les composants et fonctionnalités non implémentés dans la phase précédente le sont ici. Au cours de cette phase, la gestion et le contrôle des ressources ainsi que l'optimisation des coûts représentent les activités essentielles pour aboutir à la réalisation du produit. En parallèle est rédigé le manuel utilisateur de l'application.

### Transition

Après les opérations de test menées dans la phase précédente, il s'agit dans cette phase de livrer le produit pour une exploitation réelle. C'est ainsi que toutes les actions liées au déploiement sont traitées dans cette phase. De plus, des « bêta tests » sont effectués pour valider le nouveau système auprès des utilisateurs.

### Itérations

Une phase peut-être divisée en itérations. Une itération est un circuit complet de développement aboutissant à une livraison (interne ou externe) d'un produit exécutable. Ce produit est un sous-ensemble du produit final en cours de développement, qui croît incrémentalement d'itération en itération pour devenir le système final. Chaque itération au sein d'une phase aboutit à une livraison exécutable du système.

### Activités du processus (aspect statique)

Les activités menées à l'intérieur des quatre phases sont plus classiques, car déjà bien documentées dans les méthodes existantes par ailleurs. Nous nous limiterons donc à ne donner qu'une brève explication de chaque activité.

### Expression des besoins

UP propose d'appréhender l'expression des besoins en se fondant sur une bonne compréhension du domaine concerné pour le système à développer et une modélisation des procédures du système existant. Ainsi, UP distingue deux types de besoins :

- les besoins fonctionnels qui conduisent à l'élaboration des cas d'utilisation,
- les besoins non fonctionnels (techniques) qui aboutissent à la rédaction d'une matrice des exigences.

### Analyse

L'analyse permet une formalisation du système à développer en réponse à l'expression des besoins formulée par les utilisateurs. L'analyse se concrétise par l'élaboration de tous les diagrammes donnant une représentation du système tant statique (diagramme de classe principalement), que dynamique (diagramme des cas d'utilisation, de séquence, d'activité, d'état-transition...).

### Conception

La conception prend en compte les choix d'architecture technique retenus pour le développement et l'exploitation du système. La conception permet d'étendre la représentation des diagrammes effectuée au niveau de l'analyse en y intégrant les aspects techniques plus proches des préoccupations physiques.

### Implémentation

Cette phase correspond à la production du logiciel sous forme de composants, de bibliothèques ou de fichiers. Cette phase reste, comme dans toutes les autres méthodes, la plus lourde en charge par rapport à l'ensemble des autres phases (au moins 40%).

### Test

Les tests permettent de vérifier : • la bonne implémentation de toutes les exigences (fonctionnelles et techniques), • le fonctionnement correct des interactions entre les objets, • la bonne intégration de tous les composants dans le logiciel.

Classiquement, différents niveaux de tests sont réalisés dans cette activité: test unitaire, test d'intégration, test de réception, test de performance et test de non-régression. Après cette présentation d'UP et afin d'éclairer le lecteur sur les principaux processus de développement actuellement utilisés dans l'approche objet, nous donnons ci-après une description générale de RUP. En son temps, la société Rational Software (rachetée par IBM) avait développé une version spécifique d'UP sous le nom de RUP (Rational Unified Process), cette démarche a fait l'objet d'un ouvrage [Kruchten2000]. Dans la présentation

qui suit, nous avons surtout mis l'accent sur les principaux apports de RUP par rapport à UP.

### LES PRINCIPAUX APPORTS DE RUP

RUP (Rational Unified Process) est un processus basé sur une approche disciplinée afin de bien maîtriser l'assignation des tâches et la responsabilisation des différents acteurs participant au cycle de développement du logiciel. RUP a pour objectif principal de faire appliquer les bonnes pratiques de développement aux entreprises, ce qui confère au produit final une meilleure qualité.

RUP se veut être un modèle évolutif qui doit être configuré pour pouvoir être utilisé en intégrant les contraintes, les spécificités et l'historique de l'organisation qui l'adopte.

Nous allons présenter les principaux apports de RUP par rapport à UP en traitant les points suivants :

- les bonnes pratiques,
- les phases du processus,
- les activités du processus.

#### *1 Les bonnes pratiques*

RUP adhère à six bonnes pratiques de développement observées dans l'industrie pour leurs succès. Sur ces six bonnes pratiques, trois sont issues des principes d'UP:

- Développement itératif et incrémental.
- Développement piloté par les cas d'utilisation.
- Forte importance de l'architecture.

Trois autres bonnes pratiques ont été introduites par RUP :

- Modélisation visuelle.
- Vérification continue de la qualité.
- Contrôle des changements du logiciel.

#### **Modélisation visuelle**

RUP préconise l'utilisation d'un langage de modélisation standard comme UML qui permet aux membres de l'équipe de développement de communiquer sans ambiguïté. L'utilisation d'outils de modélisation visuelle est fortement recommandée par RUP. Ceux-ci permettent de modéliser l'architecture et ses composants à l'aide de diagrammes. De

plus, ils facilitent la gestion des modèles de RUP et contribuent à maintenir la cohérence entre les différentes phases du processus: de l'expression des besoins à l'implémentation. En résumé, la modélisation visuelle permet de gérer la complexité des logiciels.

### Vérification continue de la qualité

**RUP** met l'accent sur l'importance d'évaluer continuellement la qualité d'un système du point de vue des fonctionnalités, de la fiabilité et de la performance. Pour cela, RUP vous assiste dans la planification, la conception, l'implémentation et l'exécution des tests adéquats. Ces tests sont réalisés tout au long du processus, dans toutes les activités, en impliquant tous les acteurs, et en utilisant des critères et des mesures objectifs (ex. tests d'intégration continus).

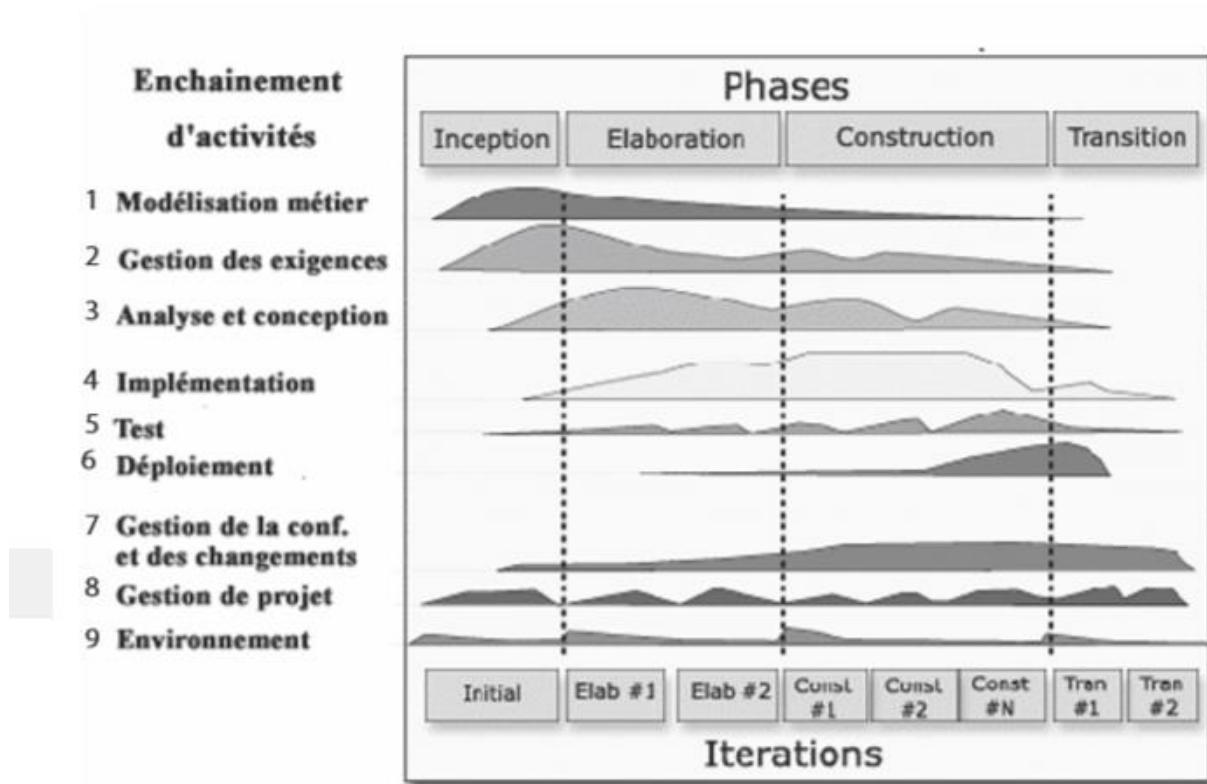
### Contrôle des changements du logiciel

RUP propose une coordination des activités et des livrables des équipes afin de gérer activement les changements du logiciel. Pour cela le processus organise les activités en enchaînement d'activités (workflows). Ces workflows décrivent comment contrôler, suivre et mesurer les changements du logiciel. De plus, ils permettent une meilleure allocation des ressources basée sur les priorités et les risques du projet et facilitent la gestion du travail sur ces changements au travers des itérations. Combinée au développement itératif, cette technique permet de contrôler les changements de sorte qu'il soit possible de découvrir rapidement les éventuels problèmes et d'y réagir.

### Les phases et les activités du processus

Comme UP, RUP est un processus à deux dimensions. Il est modélisé par un schéma articulé suivant deux axes:

- L'axe horizontal représentant le temps et montrant les phases et les itérations du processus,
- L'axe vertical représentant l'aspect statique du processus. Les activités sont représentées sur cet axe, RUP propose neuf activités (quatre de plus que le processus UP).



## DÉMARCHE DE DÉVELOPPEMENT UP7

. La démarche que nous proposons est articulée suivant deux axes : les quatre phases qui correspondent à celles d'UP et sept activités. Ainsi, on peut présenter dès ce stade un premier schéma d'ensemble de la démarche suivant ces deux axes

PHASES ► ACTIVITÉS▼	Lancement	Élaboration	Construction	Transition
1- Modélisation métier				
2- Exigences fonctionnelles				
3- Analyse des cas d'utilisation				
4- Synthèse de l'analyse				
5- Conception				
6- Implémentation				
7- Test				

## Exercices

**Thème : Gestion des étudiants dans une institution d'enseignement supérieur(inscription et paiement frais)**