République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Mohammed Boudiaf d'Oran

Faculté des Mathématiques et Informatique
Département d'Informatique

_____

Polycopié de cours

Matière : Génie logiciel

La modélisation orientée objet Avec UML

Préparé Par :

Dr Guerroudji Meddah Fatiha

Année universitaire 2018/2019

Ce polycopié offre un maximum d'informations sur le langage de modélisation UML dans une optique de mise en pratique. Il s'adresse aux étudiants en Informatique tant en deuxième année Licence qu'en troisième année pour leur permettre de découvrir, étape par étape, les éléments de la modélisation orientée objet par UML à partir d'exemples pédagogiques.

SOMMAIRE

SOMMAIRE

Chapitre 1

Introduction au Génie Logiciel

I. Introduction

Le **Génie Logiciel (GL)** est apparu dans les années 70 pour répondre à la crise du logiciel lorsqu'on a pris conscience que :

- les délais de livraison ne sont pas respectés
- les budgets alloués sont dépassés
- les logiciels ne répondent pas aux besoins de l'utilisateur ou du client
- les logiciels sont difficiles à utiliser, à maintenir, et à faire évoluer

Pour remédier aux problèmes précédents, l'idée était d'appliquer les méthodes classiques d'ingénierie au domaine du logiciel.

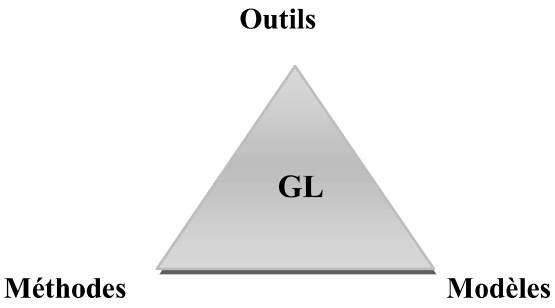
II. Définitions

a) ***Le Génie : (Ingénierie)***

Le **génie** est la **production industrielle** basée sur les **connaissances scientifiques**.

b) ***Le Génie Logiciel***

Le **GL** consiste à appliquer des méthodes, à développer et à utiliser des modèles et des outils dans le but de produire des logiciels de qualité en respectant les contraintes de temps et de coût.



c) Le logiciel

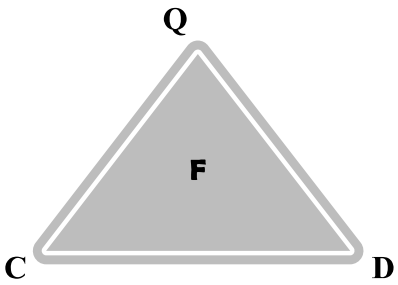
Le logiciel est l'ensemble de programmes, procédés, règles et documentation, relatif au fonctionnement d'un ensemble de traitements de l'information.

III. Objectifs

Le **GL** vise à développer des logiciels qui :

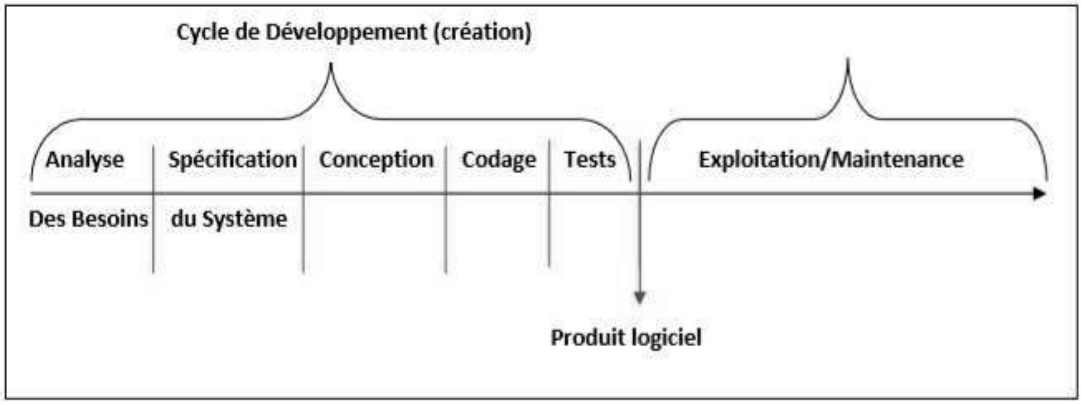
- Répondent aux besoins des clients (F)
- Répondent à la qualité(Q) exigée dans le contrat de service de départ (cahier des charges)
- Répondent aux contraintes de coût (C) prévues dès le départ
- Restent dans les limites de délai (**D**) prévues dès le départ.

-



IV. Cycle de vie d'un logiciel

- Le **cycle de vie d'un logiciel** est un **ensemble d'étapes** permettant de transformer les besoins en traitement automatique de l'information à travers un logiciel opérationnel répondant à ces besoins.
- Le **cycle de vie d'un logiciel** est la description du **processus** couvrant les phases de création (développement) et d'exploitation /maintenance d'un logiciel.



- Cycle de vie d'un logiciel -

V. Composants du cycle de vie d'un logiciel

1. **L'analyse des besoins** : étape visant à établir quelles fonctionnalités le système doit fournir **(besoins fonctionnels)** et les contraintes **(besoins non fonctionnels)** auxquelles il sera soumis en termes de facilité d'utilisation, d'efficacité, de performance, de fiabilité et de portabilité.
2. **La spécification du système** permet de formuler les besoins fonctionnels sous forme de spécifications fonctionnelles décrivant ce que le logiciel doit offrir ainsi que ses différents comportements. Dans cette phase on répond à la question : le logiciel va faire quoi ?
3. **La conception**, à partir des spécifications du système, permet d'élaborer une solution et de répondre à la question : comment réaliser le logiciel ? La conception du logiciel est une activité essentiellement intellectuelle et créative.

4. **Le codage** : lors de cette phase la solution élaborée est traduite dans un langage de programmation.
5. **Les tests** sont l'exécution ou l'évaluation du logiciel ou d'un de ses composants, par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications (validation) ou identifier les différences entre les résultats attendus et les résultats obtenus (vérification).
6. **La maintenance** c'est l'activité de maintenir le logiciel en état opérationnel. Elle débute à la livraison du logiciel et s'achève à la fin de l'exploitation du système. La maintenance se manifeste selon trois aspects :

- **La maintenance corrective ou curative** pour détecter et corriger les erreurs résiduelles dans le cas de non-conformité aux spécifications.
- **La maintenance adaptative :** en cas de modification de l'environnement (matériel, fournitures logicielles, outil, ...)
- **La maintenance évolutive ou perfective** en cas de changement des spécifications fonctionnelles du logiciel.

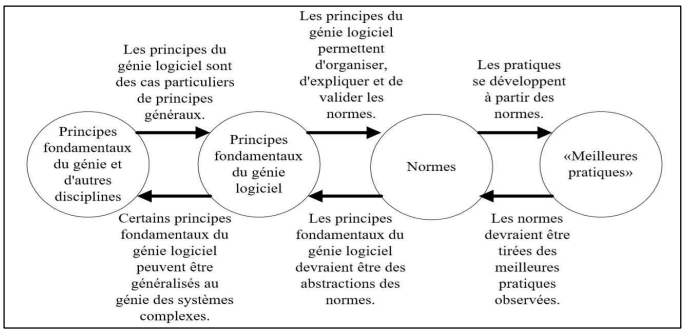
VI. Principes d'ingénierie pour le logiciel

Certains principes d'ingénierie sont indispensables pour construire une industrie du logiciel à savoir :

- la rigueur
- l'abstraction
- la décomposition en sous-problèmes
- la modularité
- la construction incrémentale
- la généricité
- l'anticipation des évolutions
- la documentation
- la standardisation/normalisation

Ces principes permettent de dresser un cadre rigoureux pour :

- guider le développement du logiciel, de sa conception à sa livraison.
- contrôler les coûts, évaluer les risques et respecter les délais.
- établir des critères d'évaluation de la qualité d'un logiciel.



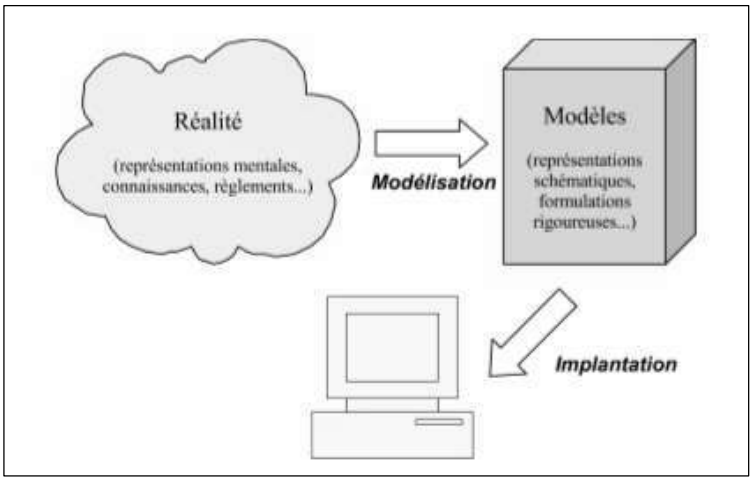
Chapitre 2

La modélisation Orientée Objet

I. Introduction

Pourquoi modéliser ?

- Pour mieux **comprendre un système** (modélisation en physique)
- Pour mieux **construire un système** (modélisation en ingénierie)



Remarques :

➔ Le logiciel est complexe par nature.

➔ Pour gérer cette complexité l'informatisation d'un système nécessite un processus de modélisation dans lequel, plusieurs étapes sont nécessaires pour la compréhension du problème. Ces étapes successives permettent de raffiner le niveau de détails du système à réaliser.

➔ Le processus de modélisation vise à mieux cerner les limites du système à réaliser

II. Concepts de modélisation

- La **modélisation** est la **construction de modèles**.
- Un **modèle** est une ***représentation abstraite de la réalité*** qui exclut certains détails du monde réel (vue subjective).
- Autrement dit, un **modèle** est une ***simplification de la réalité***, faisant abstraction de larges niveaux de détails.

III. Objectifs de la modélisation

- Les modèles aident à *visualiser un système* tel qu'il est ou tel que nous voudrions qu'il soit.

- Les modèles permettent de *préciser la structure ou le comportement d'un système*. Ils permettent de *réduire sa complexité* en éliminant les détails qui n'influencent pas son comportement de manière significative.
- Les modèles reflètent ce que le concepteur croit important pour *la compréhension* et la *prédiction* du système modélisé, les limites du système modélisé dépendent des objectifs du modèle.
- Les modèles fournissent *un canevas* qui guide la construction d'un système ;
- Les modèles permettent de *documente*r les décisions prises.

➔ On en déduit qu'un modèle est donc une abstraction de la réalité permettant de mieux comprendre le système pour le reconstruire.

Remarques :

➔ Un même système peut avoir des modèles selon de très nombreux points de vue
➔ Une seule « vue » du système n'est pas suffisante.
➔ Le choix du modèle à créer est important.
➔ Le modèle d'un système informatique sert :
- ✓ de document d'échange entre clients et développeurs
- ✓ d'outil de conception
- ✓ de référence pour le développement
- ✓ de référence pour la maintenance et l'évolution

IV. Langage de Modélisation

➔ Un langage de modélisation doit définir :
- La sémantique des concepts ;
- Une notation pour la représentation de ces concepts ;
- Des règles de construction et d'utilisation des concepts.

Comme exemples de langages de modélisation à différents niveaux de formalisation nous pouvons citer :

✓ Les langages formels (Z, B,) : le plus souvent mathématiques, au grand pouvoir d'expression et permettant des preuves formelles sur les spécifications ;

✓ Les langages semi-formels (MERISE, UML...) : le plus souvent graphiques, au pouvoir d'expression moindre mais plus faciles d'emploi.

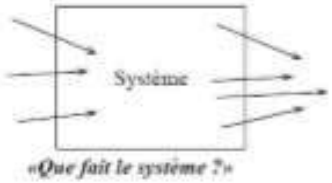
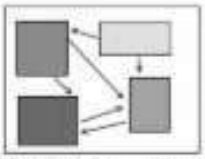
Remarques :

➔ La modélisation à base de graphiques est plus précise qu'une description informelle en langage naturel.
➔ UML divise la visualisation d'un modèle en diagrammes qui correspondent à des vues différentes.

V. La Modélisation Orientée Objet

➔ Les systèmes peuvent être décomposés selon :

- ce qu'ils font (approche fonctionnelle)
- ce qu'ils sont (approche objet)

LA DEMARCHE FONCTIONNELLE	LA DEMARCHE OBJET
Accent mis sur ce que fait le système (ses fonctions)  *«Que fait le système ?»*	Accent mis sur ce qu'est le système (ses composants) : **les objets**  *«De quoi se compose le système ?»*

➔ L'approche OO considère que :

- Un système est un ensemble d'objets connectés et organisés pour atteindre un but spécifique
- Un ***modèle*** est une ***abstraction d'objets*** de la réalité.
- Chaque objet possède une frontière et une identité bien définies qui encapsulent un état et un comportement.

Avantages :

➔ L'approche objet gère plus efficacement la complexité. Ceci est du à:

- la simplicité de la modélisation basée sur 5 concepts de base : objets, messages, classes, héritage, polymorphisme)
- la stabilité de la modélisation par rapport aux entités du monde réel
- la construction itérative facilitée par un faible couplage statique entre les différents composants
- la possibilité de réutiliser des éléments d'un développement à un autre

Chapitre 3

Introduction à La modélisation Orienté Objet

Avec UML

I. Introduction

- Les méthodes objets commencent à émerger dans les années 80.
- Dans celles-ci les éléments structurants un système sont des objets collaborant entre eux.
- Ces **objets** qui sont des **abstractions du monde réel** intègrent une *identité*, un *état* et un *comportement*.
  - ➔ L'**état** représente les **valeurs instantanées de tous les "attributs"** d'un objet. Ceux-ci peuvent prendre des valeurs dans un domaine de définition donné.
  - ➔ Le **comportement** regroupe les "compétences" d'un objet et décrit les **actions** et les **réactions** de cet objet.

Remarque :

- ➔ **L'état d'un objet peut évoluer dans le temps en conséquence de ses comportements passés.**

II. Historique

➔ Les années 90 ont vu une prolifération de méthodes objet (plus d'une cinquantaine de méthodes). Parmi elles, trois étaient considérées comme les plus importantes :

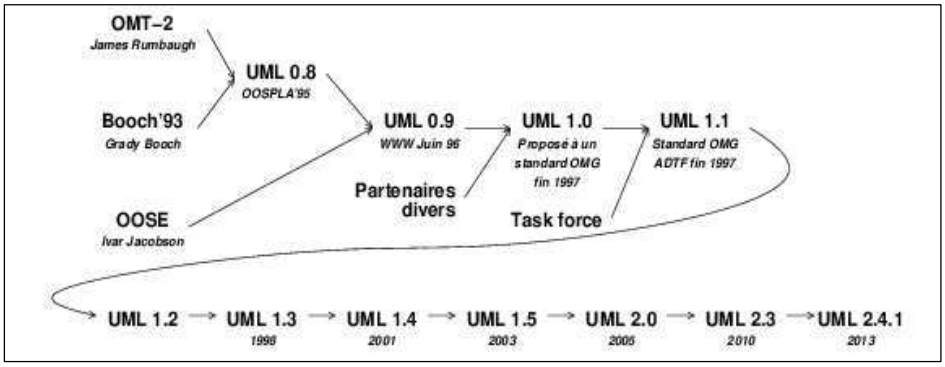
- ✓ **BOOCH** (notions de partitions en sous-systèmes) de Grady Booch,
- ✓ **OMT** (notions de classes et associations) de James Rumbaugh
- ✓ et enfin **OOSE** (cas d'utilisations) d'Ivar Jacobson.

➔ Ces trois auteurs ont ensuite décidé d'unir leurs efforts au sein de la société Rational Software. Ils fixent ainsi des objectifs communs :

- ✓ représenter des **systèmes entiers** par des concepts **Objets**.
- ✓ créer un **langage de modélisation** utilisable à la fois par les humains et par des machines (besoin d'un véritable "langage de modélisation")

➔ Cette unification a donné naissance en 1996 à la version 0.9 d'**U**nified **M**odeling **L**angage (UML). Les principaux acteurs du secteur informatique (DEC, HP, i-Logix, IntelliCorp, IBM, ICON, MCI, Microsoft, Oracle, Rational Software, TI, Unisys.) ont ensuite participé à cet effort, et UML 1.0 a été proposé à l'**O**bject **M**anagement **G**roup (OMG).

➔ Cet organisme international chargé de définir des standards dans le domaine de l'objet normalise UML 1.1 en 1997. Cette norme a depuis continué d'évoluer et nous en sommes aujourd'hui à la norme 2.5



III. Définition d'UML selon l'OMG

UML est un **Langage visuel** dédié à la **spécification**, la **construction** et la **documentation** des artefacts d'un système logiciel

Remarques :

➔ **UML est une norme** de langage de modélisation objet

➔ **UML n'est pas une méthode.** La notation UML est conçue pour servir de langage de modélisation objet indépendamment de la méthode mise en œuvre (Booch, OMT, ...).

➔ **UML est composé d'éléments graphiques,** chacun avec une sémantique clairement définie. Il peut être utilisé dans tous les domaines informatiques.

➔ **UML** permet d'exprimer et d'élaborer des modèles objet, **indépendamment de tout langage de programmation.**

➔ **UML** est **en évolution continue**

IV. Objectifs d'UML

- Faciliter la communication entre les différents acteurs d'un projet informatique
- Faciliter la communication avec la machine
- Documenter un projet informatique de bout en bout
- Spécifier en limitant les ambiguïtés
- Construire (interpréter les diagrammes pour code)

V. Modélisation OO avec UML

- ➔ UML permet de *modéliser un système* selon *différentes vues complémentaires*.
- ➔ UML permet de *définir et de visualiser un modèle*, à l'aide de *diagrammes*.
- ➔ Un *diagramme UML* est une représentation graphique, qui s'intéresse à un aspect précis du modèle ; c'est une *perspective du modèle*, pas "le modèle".
- ➔ Chaque *type de diagramme* UML possède une *structure* (les types des éléments de modélisation qui le composent sont prédéfinis).
- ➔ Un *type de diagramme* UML véhicule une *sémantique précise* (un type de diagramme offre toujours la même vue d'un système).

VI. Les différents types de diagrammes UML

UML 1.3 propose 9 diagrammes. Tandis qu'UML 2 en propose 13.

1. **Diagramme de classes** : définit les classes et les relations entre les classes.
2. **Diagramme d'objets** : définit les instances des classes, ainsi que les liens entre ces instances.
3. **Diagramme de cas d'utilisation** : représente les fonctions du système du point de vue de l'utilisateur.
4. **Diagramme de composants** : permet de décrire les composants (ou modules) d'un système (fichiers sources, fichiers objets, librairies exécutables).
5. **Diagramme de déploiement** : décrit la disposition physique du matériel et la répartition des composants sur le matériel.
6. **Diagramme de séquences** : représente les interactions d'un point de vue chronologique ; d'une part, les interactions entre le système et les acteurs et, d'autre part, les interactions entre les objets à l'intérieur du système.
7. **Diagramme de collaboration** : représente les interactions entre objets d'un point de vue spatial.
8. **Diagramme d'états-transitions** : automates hiérarchiques permettant représenter les comportements d'un acteur, d'un système ou d'un objet d'une certaine classe.
9. **Diagramme d'activités** : permet de modéliser le comportement d'une méthode en représentant un enchaînement d'activités.

En plus des 9 diagrammes précédents, UML 2 propose :

10. **Diagramme de paquetages :** permet de décrire les interactions entre les paquetages (paquetage = structure logique de regroupement et d'organisation des éléments du modèle UML).
11. **Diagramme de structure composite** : décrit sous la forme d'une boîte blanche les relations entre les composants d'une classe.
12. **Diagramme global d'interaction** : permet de décrire les enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences.
13. **Diagramme de temps** : permet de décrire les variations d'une donnée au cours du temps.

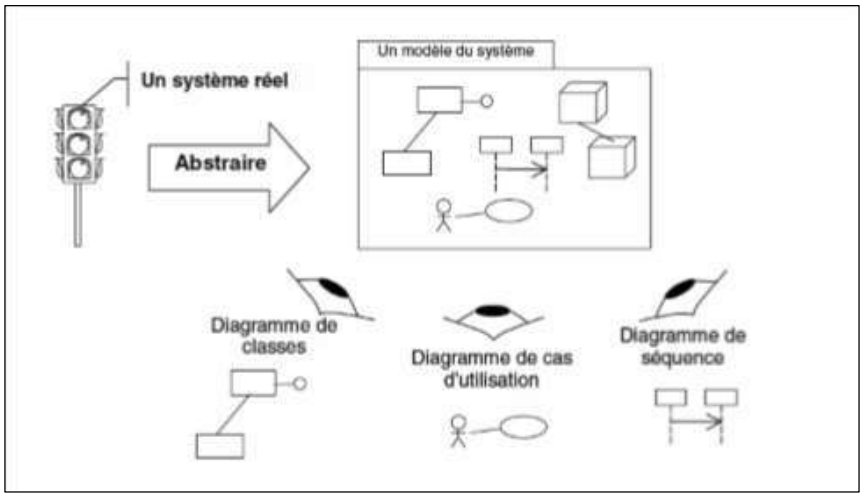
Remarques :

➔ Ces diagrammes sont classés selon **2 types de vues du système** :
- ✓ les **diagrammes de structure** : diagramme de classe, diagramme composite, diagramme de composants, diagramme de déploiement, diagramme d'objets, diagramme de package
- ✓ les **diagrammes de comportement** : diagramme d'activité, diagramme de cas d'utilisation, diagramme d'état-transition ; diagramme de séquence, diagramme de communication, diagramme global d'interaction, diagramme de temps

➔ La modélisation de tous les systèmes ne comporte pas nécessairement ces 13 diagrammes.

➔ Combinés, les différents types de diagrammes UML offrent une vue complète des aspects structurels et comportementaux d'un système.

➔ Un modèle peut être exprimé avec différents niveaux d'abstraction/raffinement



Exemple de diagrammes UML : différents points de vue sur le système

Chapitre 4

Diagrammes UML de Classes et d'Objets

(Vue Statique)

I. Introduction

Avec UML, le modèle structurel ou statique d'un système (où on ne tient pas compte du facteur temporel dans le comportement du système) est décrit à l'aide de trois sortes de diagrammes :

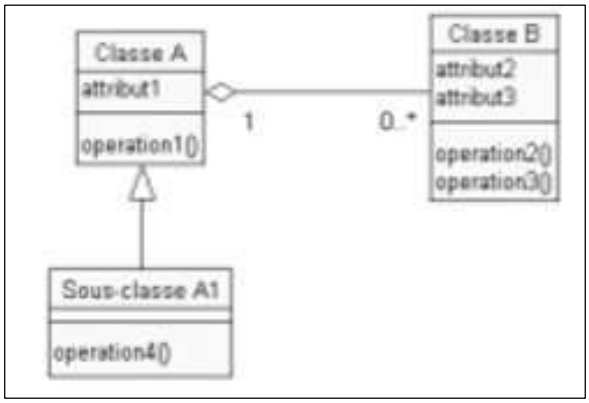
- le diagramme de classes
- le diagramme d'objets
- et le diagramme de packages (paquetages)

II. Diagramme de Classes (DCL)

Un **diagramme de classes** est une vue graphique de la **structure statique** d'un système, exprimée en termes de classes et de relations entre ces classes. Autrement dit, Il représente les classes intervenant dans le système.

Diagramme de classes ⟶ classes + relations

Exemple de diagramme de classes :



Remarques :

- ➔ Le **diagramme de classes** est considéré comme **le plus important** de la modélisation orientée objet. Il est le **seul obligatoire** lors d'une telle modélisation.
- ➔ Il exprime de manière générale la **structure statique** d'un système. Il permet de définir quelles seront les composantes du système final pour permettre de construire le système de manière correcte.
- ➔ Les diagrammes de classes permettent de spécifier la **structure et les liens entre les objets** dont le système est composé.
- ➔ Pour représenter un **contexte précis**, un diagramme de classes peut être instancié en **diagrammes d'objets**

II. 1 Notion de classe

- Une **classe** est une **abstraction** qui représente un ensemble **d'objets de même nature**, c'est-à-dire possédant la même structure statique (attributs) et le même comportement (méthodes).

- Autrement dit, une classe est une **description** d'un ensemble **d'objets similaires**, ayant une sémantique, des caractéristiques et des relations en commun.

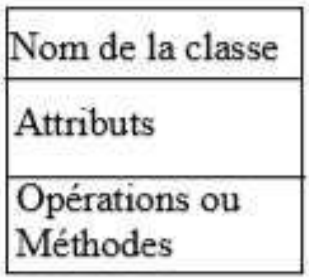
Remarques :

➔ La classe intègre les concepts de type (en tant que « **moule à instances** »).
➔ Un **objet** est une **instance** d'une classe

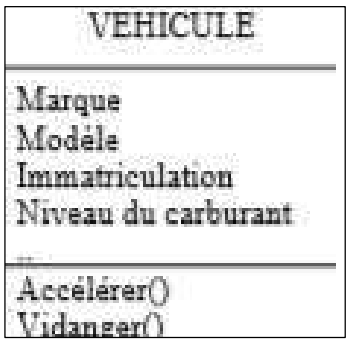
Notation UML :

Une classe est un classeur représenté par un rectangle divisé en 3 compartiments obligatoires :

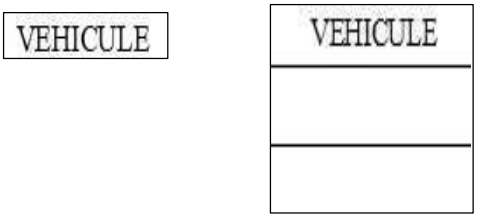
- ✓ le premier indique une chaîne de caractères correspondant au **nom** de la classe.
- ✓ le deuxième compartiment indique les **attributs** de la classe
- ✓ et le troisième ses **opérations**.



Exemple :



Autres représentations simplifiées :



Remarques :

➔ Tout est facultatif sauf le nom de la classe

➔ La visibilité des éléments de la classe (attributs et méthodes) est donnée par:
- ✓ Private (-) : élément accessible uniquement à l'intérieur de la classe
- ✓ Protected (#) : élément accessible à l'intérieur de la classe et de ses sous-classes
- ✓ Publique (+) : élément visible à tout le monde.

➔ Les noms des classes doivent commencer par des majuscules et ceux de tous les autres éléments par des minuscules.

➔ Les mots composés doivent être séparés par des majuscules.

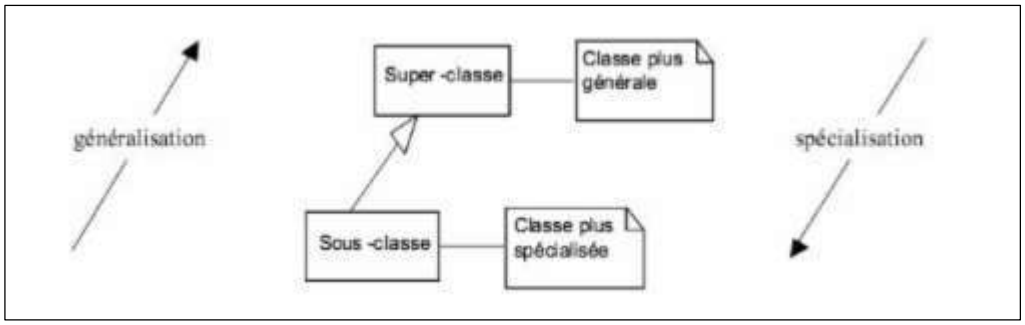
➔ Deux compartiments optionnels peuvent être ajoutés dans le classeur représentant une classe. Il s'agit des compartiments :
- ✓ Responsabilités (op) : pour énumérer l'ensemble des tâches devant être assurées par la classe
- ✓ Exceptions (op) : pour énumérer les situations exceptionnelles devant être gérées par la classe.

II.2 Relations entre classes

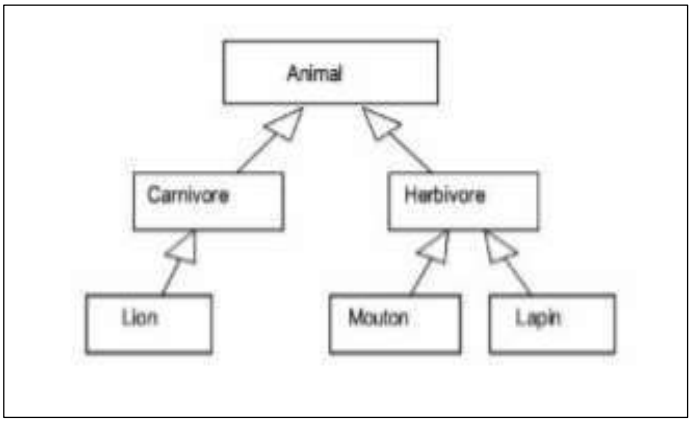
II.2.1 Héritage (Généralisation/Spécialisation) :

- L'*héritage* permet de *factoriser les éléments communs* d'un ensemble de classes dans une classe plus générale appelée super-classe (ou classe parent).
- Les *classes plus spécialisées* sont appelées *sous-classes* (ou classes enfants). Autrement dit, la sous-classe est une sorte de la super classe.

Notation UML :



Exemple :



Remarques :

- Une relation d'**héritage** est une relation de **généralisation/spécialisation** permettant l'abstraction.
- L'**héritage multiple** indique l'existence de **plusieurs super classes** pour une classe enfant.
- La classe enfant possède toutes les propriétés de ses classes parents, mais elle ne peut accéder aux propriétés privées de celle-ci.
- Une classe enfant peut redéfinir (même signature) une ou plusieurs méthodes de la classe parent.
- Toutes les associations de la classe parent s'appliquent aux classes enfants.
- Une instance d'une classe peut être utilisée partout où une instance de sa classe parent est attendue.

II.2.2 Association

Une **association** est une **relation entre** deux **classes** (association binaire) ou plus (association n-aire), qui décrit les **connexions structurelles** entre leurs **instances**.

Remarques

- ➔ L'**association** relie des **classes** au même niveau **hiérarchique**.
- ➔ L'**association** permet de **spécifier le lien** entre les **objets instances**.

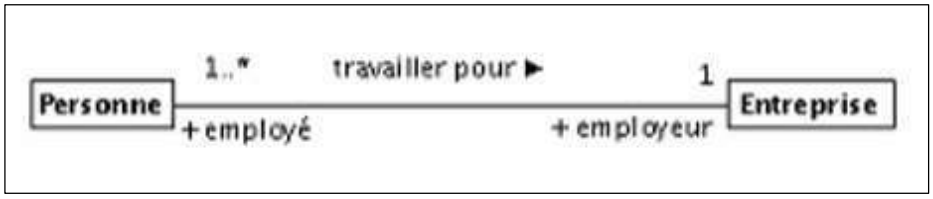
II .2.2.1 Différents Type d'association :

a) **Association binaire**

Avec UML, une association binaire est matérialisée par un trait plein entre les classes associées qui peut être :

- ➔ Orné d'un nom,
- ➔ Avec éventuellement une précision du sens de lecture (< ou >).
- ➔ et 2 terminaisons qui ont chacune :
  - Un rôle : qui exprime le rôle que joue les objets de la classe concernée au niveau de l'association
  - Une multiplicité : qui exprime combien d'objets de la classe peuvent être liés à l'autre classe (minimum, maximum) telle que :
    - ✓ 1 : un et un seul
    - ✓ 0 .. 1 : zéro ou un
    - ✓ M .. N : de M à N (M et N sont des entiers dont on connaît exactement la valeur)
    - ✓ * ou 0 .. * : de zéro à plusieurs
    - ✓ 1 .. * : de 1 à plusieurs

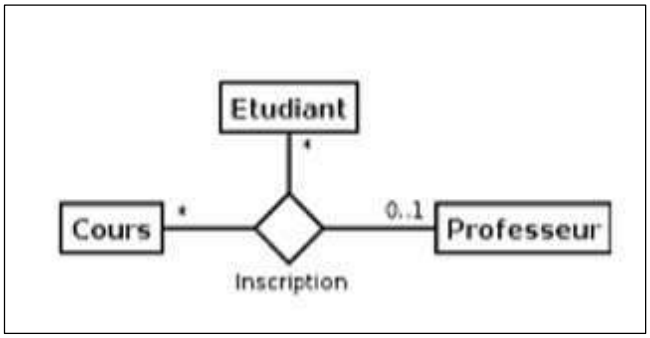
Exemple :



b) **Association n-aire**

Une association n-aire est représentée graphiquement par un grand losange avec un chemin partant vers chaque classe participante. Le nom de l'association, le cas échéant, apparaît à proximité du losange.

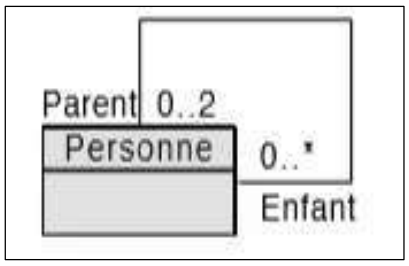
Exemple :



c) Association réflexive :

Si les deux extrémités de l'association pointent vers une même classe, l'association est dite réflexive.

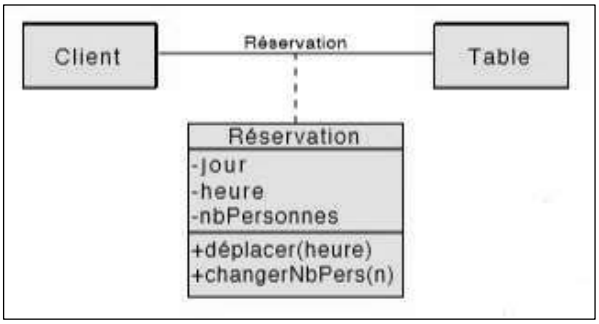
Exemple :



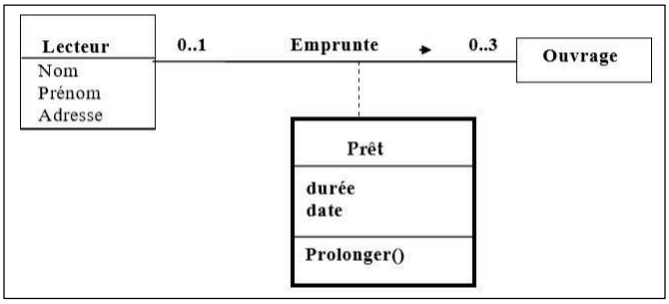
d) Classe-association

Une classe-association possède les propriétés des associations et des classes: elle se connecte à deux ou plusieurs classes et possède également des attributs et des opérations. Une classe-association est caractérisée par un trait discontinu entre la classe et l'association qu'elle représente.

Exemples:



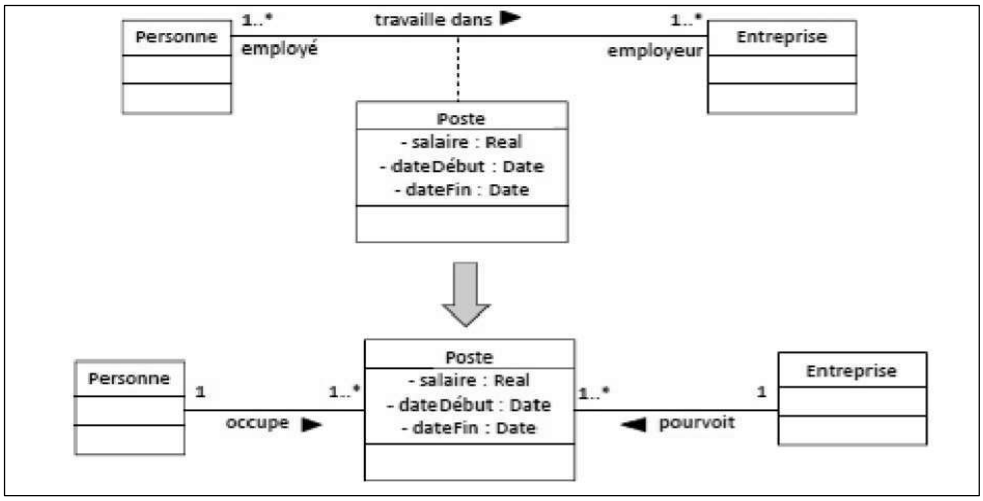
(1)



(2)

Remarque :

Toute classe-association peut être remplacée par une classe intermédiaire qui sert de pivot pour une paire d'associations.



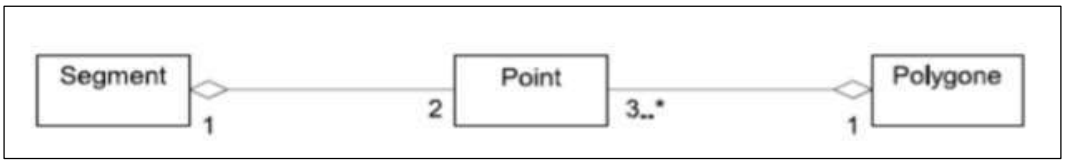
II.2.3 Agrégation

- L'**agrégation** c'est la relation qui relie la **partie au tout**. Une classe fait partie d'une autre classe.
- Une **agrégation** est une association qui représente une **relation d'inclusion structurelle ou comportementale** d'un élément dans un ensemble.

Notation:

Graphiquement, on ajoute un losange vide (◊) du côté de l'agrégat

Exemple:



Remarque:

➔ Une classe peut appartenir à plusieurs agrégats

II.2.4 Composition

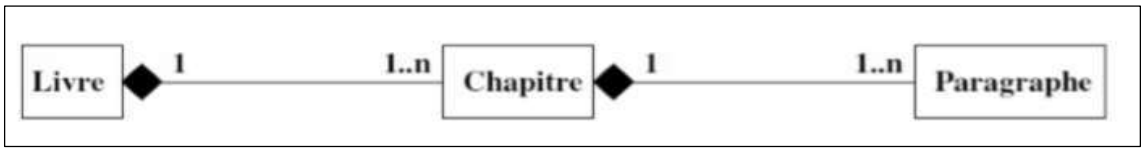
- dite également agrégation forte
- décrit une contenance structurelle (contenance physique) entre instances. Ainsi, la destruction de l'objet composite implique la destruction de ses composants.

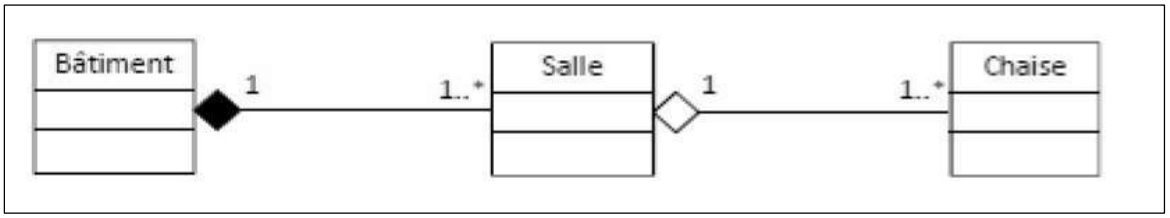
Notation:

Graphiquement, on ajoute un losange plein (♦) du côté de l'agrégat.

Exemples:

Composition





Composition Agrégation

II.3 Élaboration d'un diagramme de classe

Une démarche couramment utilisée pour élaborer un diagramme de classes consiste à :

- Trouver les classes du domaine étudié. Cette étape se fait généralement en collaboration avec un expert du domaine.
- Trouver les relations entre classes.
- Trouver les attributs des classes.
- Organiser et simplifier le modèle en éliminant les classes redondantes et en utilisant l'héritage.
- Itérer et raffiner le modèle.

Remarque :

➔ Un modèle est rarement correct dès sa première construction. La modélisation objet est un processus non pas linéaire mais itératif.

III. Diagramme d'objets (DOB)

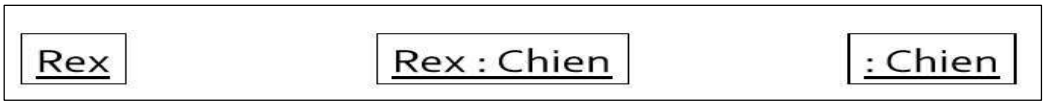
III.1 Notion d'objet

Un **objet** est une **entité aux frontières bien définies**, possédant une identité et encapsulant un état et un comportement. Un objet est une **instance** (ou **occurrence**) d'une classe.

Représentation graphique

Avec UML, un objet est représenté graphiquement de différentes manières :

Exemples:



III.2 Diagramme d'objets

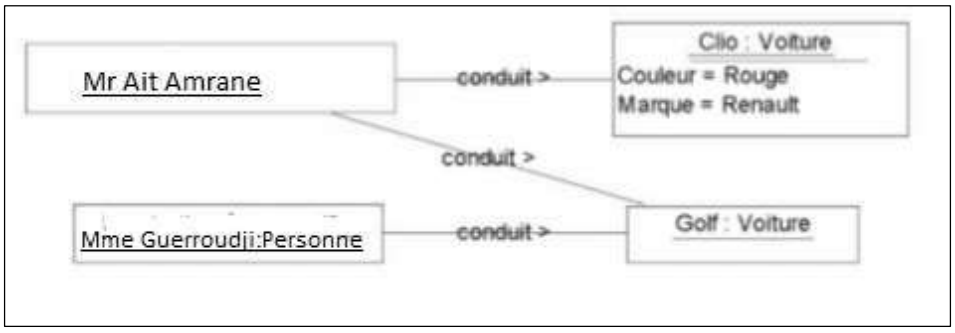
Un **diagramme d'objets** (DOB)représente des **objets** (instances de classes) et **leurs liens** (instances de relations) pour donner une **vue figée de l'état d'un système à un instant donné**.

Intérêts

Un DOB peut être utilisé pour :

- Illustrer le modèle de classes en montrant un exemple qui explique le modèle
- préciser certains aspects du système en mettant en évidence des détails imperceptibles dans le diagramme de classes
- exprimer une exception en modélisant des cas particuliers ou des connaissances non généralisables qui ne sont pas modélisés dans un diagramme de classes

Exemple:



Remarques:

Dans un DOB :

- ➔ les attributs des objets reçoivent des valeurs. Si certaines valeurs ne sont pas renseignées, l'objet est partiellement défini.
- ➔ la relation de spécialisation/ généralisation entre classes n'a pas de transposition au niveau des objets.
- ➔ les multiplicités ne sont pas représentées.

III.3 Relations entre diagrammes de classes et diagrammes d'objets

- Un objet est une instance de classe
- Les relations relient les classes, les liens relient les objets
- Un lien entre deux objets implique obligatoirement une relation entre les classes des deux objets. Un lien est une instance de relation (association, agrégation ou composition)

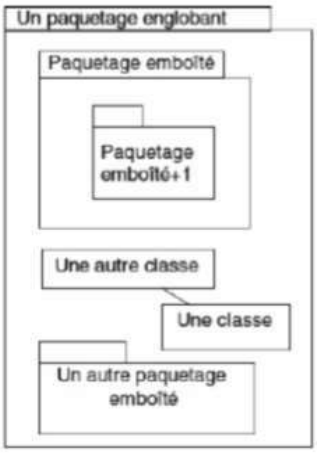
IV. Package (paquetages)

IV.1 Notion de package

- Un package, est un élément de modélisation qui contient d'autres éléments de modélisation.

- Il regroupe les éléments du modèle telles les classes, les cas d'utilisation, les diagrammes, et les paquetages, dans des ensembles fortement cohérents.

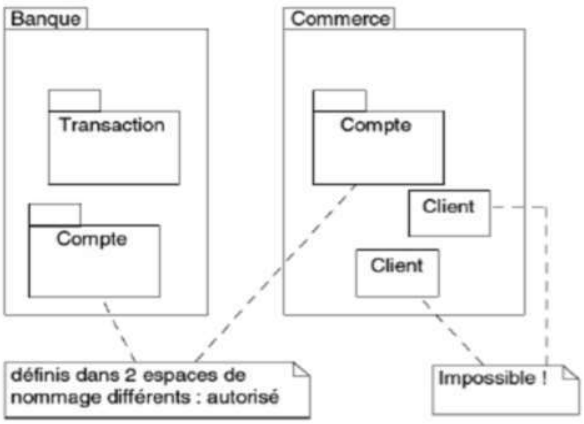
IV.2 Représentation UML



Le package est un mécanisme général pour :

- **organiser les éléments du modèle** (partitionner, hiérarchiser et clarifier les éléments du modèle)
- *et* **les nommer**: un package définit un espace de nom. Deux éléments ne peuvent avoir le même nom dans un package. Deux éléments dans deux paquetages différents sont différents, quelque soit leur nom.

Exemple :



Remarques :

- ➔ On organise les éléments modélisés en packages et sous-packages.
- ➔ Un package contient des éléments y compris d'autres packages : hiérarchie
- ➔ Un package est un élément du modèle de haut niveau.
- ➔ Tout élément n'appartient qu'à un seul package.

Relations entre packages :

- Les relations entre packages découlent des dépendances entre éléments des packages, notamment les classes.

Remarque :

- ➔ Il faut minimiser les dépendances pour maintenir un couplage faible

IV.3 Intérêt du diagramme de packages :

- Organisation globale du modèle mis en place pour :
  - ✓ contrôler la structure du système
  - ✓ comprendre et partager
  - ✓ obtenir une application plus évolutive et facile à maintenir
  - ✓ ne pas se faire déborder par les modifications
  - ✓ viser la généricité et la réutilisabilité des packages
  - ✓ avoir une vue claire des flux de dépendances entre packages pour les minimiser

IV.4 Principes du découpage en packages :

- Lors du découpage en packages, il faut assurer :

  ➔ la cohérence interne du package : relations étroites entre éléments (ex les classes changent pour des raisons similaires, réutilisation commune ....)

  ➔ l'indépendance par rapport aux autres packages.

Chapitre 5

Diagrammes UML de Cas d'Utilisations

(Vue Fonctionnelle)

I. Introduction

Avant de développer un système, il faut savoir précisément à **QUOI** il devra servir, cad à quels besoins il devra répondre. Dans ce contexte :

- la **vue fonctionnelle** cherche à appréhender les **interactions** entre les différents **acteurs/utilisateurs et le système**, sous forme d'objectifs à atteindre, exprimant des **cas d'utilisation**. L'ensemble des fonctionnalités du système est déterminé en examinant les besoins de chaque acteur, exprimés sous forme de famille d'interactions dans les cas d'utilisation.
- le **modèle** qui en découle décrit le **comportement du système**, selon le **point de vue de l'utilisateur**. Il comprend les acteurs, le système et les cas d'utilisation.
- ce modèle est formalisé par un diagramme de cas d'utilisation (Use case).

II. Diagramme de Cas d'Utilisation (DCU)

II.1 Cas d'utilisation

Les cas d'utilisation (use cases) ont été formalisés par Ivar Jacobson. Avant UML, ils n'étaient pas formalisés par les autres méthodes objet telles qu'OMT.

a) Définitions

- Un **cas d'utilisation** représente un **service complet attendu du système**.
- Un **cas d'utilisation** représente une **suite d'interactions entre un acteur et le système** qui produisent un résultat observable intéressant pour un acteur particulier
- Chaque **cas d'utilisation** correspond à une **fonction métier du système**, selon le point de vue d'un de ses acteurs.

Remarques :

➔ Un cas d'utilisation a un début et une fin clairement identifiés

➔ Un cas d'utilisation est décrit par une forme verbale

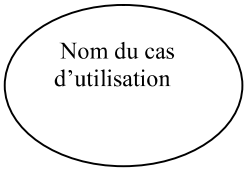
➔ Les cas d'utilisation sont utiles lors de l'élaboration du cahier des charges ou du document de spécifications des besoins du logiciel.

➔ Chaque cas d'utilisation doit avoir un nom qui permet de le distinguer des autres

➔ Quand un cas n'est pas directement relié à un acteur, il est qualifié de cas d'utilisation interne

b) Représentation UML

- Un **cas d'utilisation** est représenté par une **ellipse** et son **nom**.

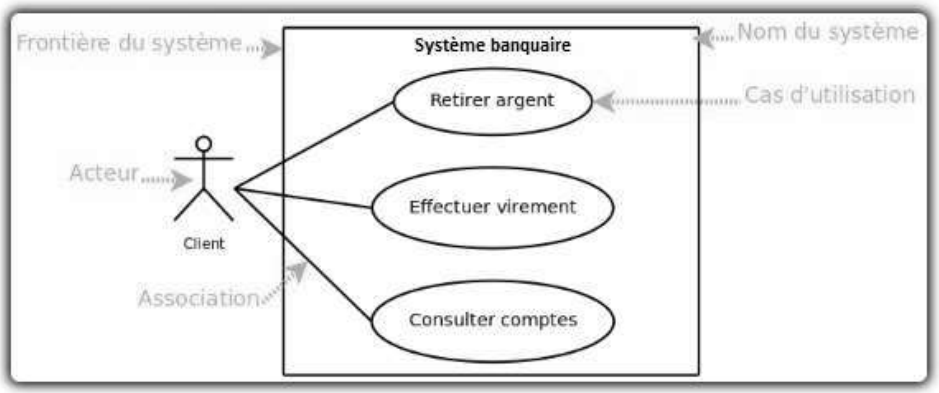


Exemple:

II.2 Diagramme de cas d'utilisation

- Un diagramme de cas d'utilisation définit :
  - ➔ le système
  - ➔ les acteurs qui interagissent avec le système
  - ➔ les cas d'utilisations
  - ➔ les relations entre acteurs et cas d'utilisations

Exemple:



II.3 Les acteurs

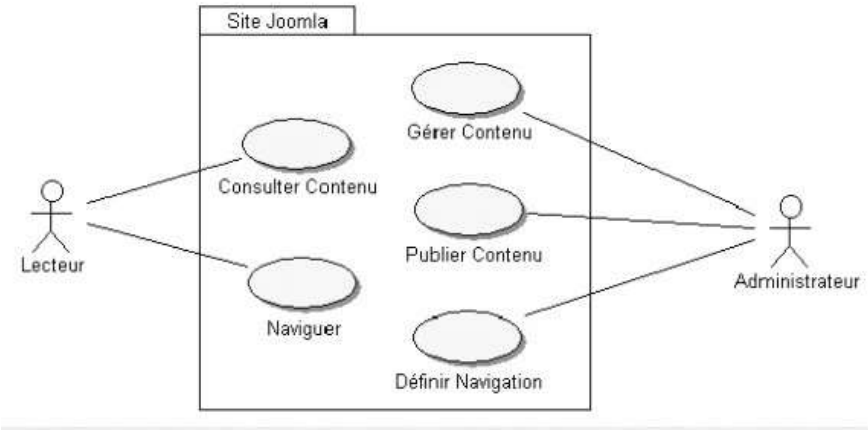
a) Définition:

Un **acteur** représente un **rôle joué par une entité extérieure au système** (humain ou autre système) et qui interagit directement avec le système étudié.

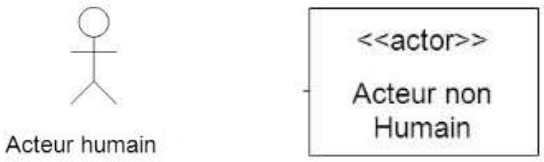
Remarques:

- ➔ Il existe deux catégories d'acteurs : les acteurs principaux et les acteurs secondaires. Un acteur est qualifié de *principal* pour un cas d'utilisation lorsque ce cas rend service à cet acteur. Les autres acteurs sont alors qualifiés de *secondaires*.
- ➔ Un acteur principal obtient un résultat observable du système tandis qu'un acteur secondaire est sollicité pour des informations complémentaires.
- ➔ Un cas d'utilisation a au plus un acteur principal.

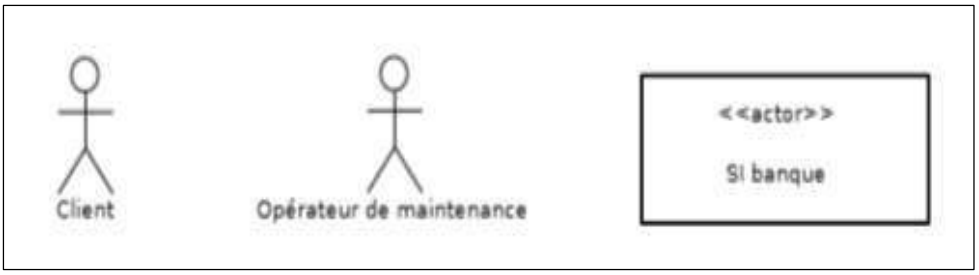
Exemple :



b) **Représentation UML**



Exemple :



II.4 Relations dans les Diagrammes de Cas d'utilisation

a) 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.

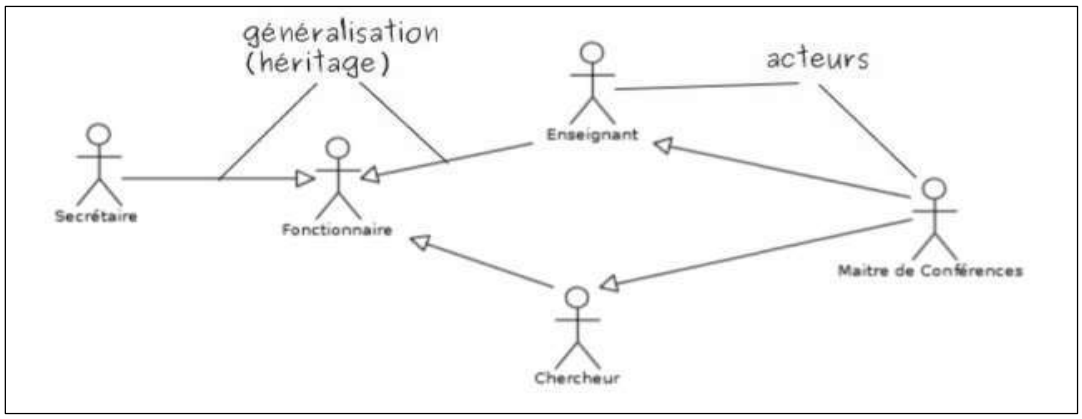
Remarque:

Tous les cas d'utilisation accessibles à A le sont aussi à B, mais l'inverse n'est pas vrai.

b) Représentation UML

Le symbole utilisé pour la généralisation est une flèche avec un trait plein dont la pointe est un triangle fermé désignant le cas le plus général.

Exemple:



c) Relations entre cas d'utilisation

i. ***L'inclusion***

- Un **cas A inclut un cas B** si le comportement décrit par le cas A inclut le comportement du cas B: **le cas A dépend de B**.
- Lorsque A est sollicité, B l'est obligatoirement, comme une partie de A.
- Cette dépendance est symbolisée par le stéréotype ***<<include>>***

Remarques:

➔ Les inclusions permettent essentiellement de factoriser une partie de la description d'un cas d'utilisation qui serait commune à d'autres cas d'utilisation

➔ Les inclusions permettent également de décomposer un cas complexe en sous-cas plus simples

ii. L'extension

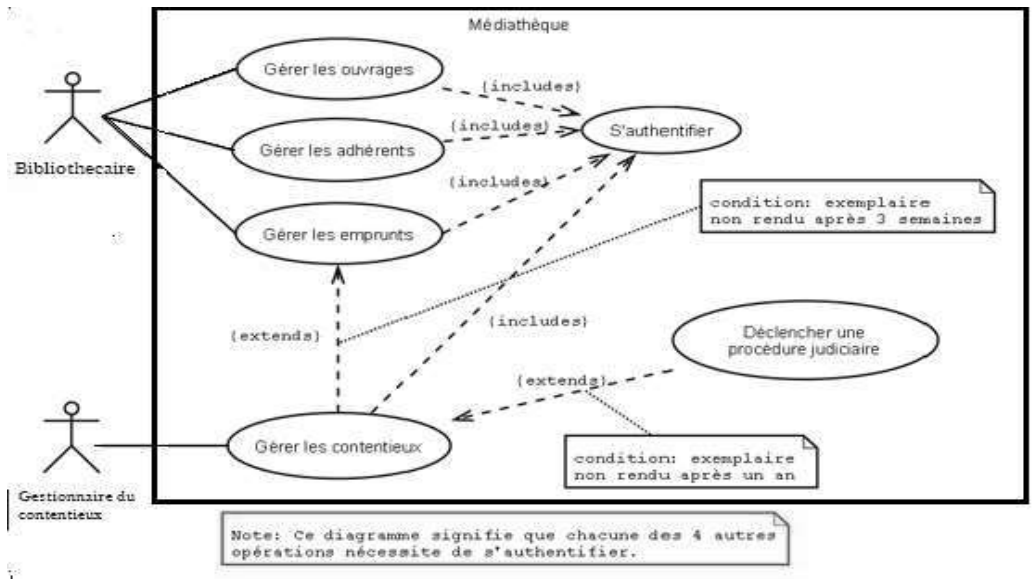
- On dit qu'un **cas d'utilisation A étend un cas d'utilisation B** lorsque le **cas d'utilisation A peut être appelé au cours de l'exécution du cas d'utilisation B**.
- Exécuter B peut éventuellement entraîner l'exécution de A : contrairement à l'inclusion, l'extension est optionnelle.
- Cette dépendance est symbolisée par le stéréotype **<<*extend*>>**

Remarques:

➔ L'extension peut intervenir à un point précis du cas étendu. Ce point s'appelle le point d'extension.

➔ Il porte un nom, qui figure dans un compartiment du cas étendu sous la rubrique ***point d'extension***, et est éventuellement associé à une contrainte indiquant le moment où l'extension intervient.

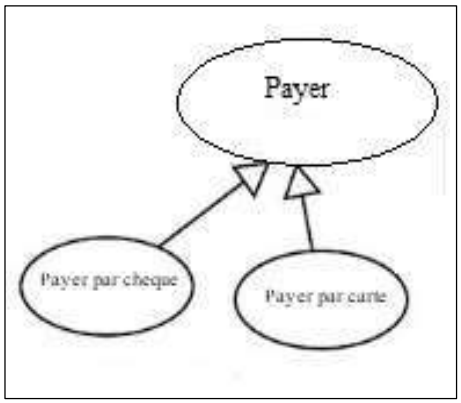
Exemple:



iii. *La généralisation*

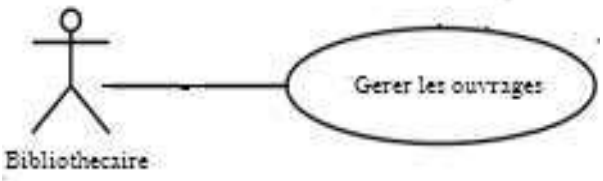
- Un **cas A est une généralisation d'un cas B** si **B est un cas particulier de A**.

Exemple :



d) Relation entre cas d'utilisation et acteurs

- La relation entre cas d'utilisation et acteurs est une relation d'association.
- Elle exprime un chemin de communication entre un acteur et un cas d'utilisation
- Et est représentée par un trait continu.



III. Intérêt d'un Diagramme de cas d'utilisation

- Décrivent les services rendus par le système du point de vue de l'utilisateur
- Technique pour capturer les exigences fonctionnelles d'un système
- Déterminer ses limites
- Déterminer le comportement du système : ce que doit faire le système sans spécifier comment il le fait
- Comprendre l'attente des utilisateurs et les experts du domaine
- Instruments de validation et de tests du système

IV. Description textuelle d'un 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.

1. La première partie permet d'identifier le cas, elle doit contenir les informations qui suivent :

Nom :
- Décrit le nom du cas sous forme verbale (ex. : Réceptionner un colis).

Objectif :
- Une description résumée permettant de comprendre l'intention principale du cas d'utilisation. Cette partie est souvent renseignée au début du projet dans la phase de découverte des cas d'utilisation.

Acteurs principaux :
- Ceux qui vont réaliser le cas d'utilisation (la relation avec le cas d'utilisation est illustrée par le trait liant le cas d'utilisation et l'acteur dans un diagramme de cas d'utilisation).

Acteurs secondaires :
- Ceux qui ne font que recevoir des informations à l'issue de la réalisation du cas d'utilisation.

Dates :
- Les dates de création et de mise à jour de la description courante.

Responsable :
- Les noms des responsables.

Version :
- Le numéro de version.

2. 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. Elle contient toujours une séquence nominale qui décrit de déroulement normal du cas.

À la séquence nominale s'ajoutent fréquemment des séquences alternatives (des embranchements dans la séquence nominale) et des séquences d'exceptions (qui interviennent quand une erreur se produit).

Les préconditions:

- Elles décrivent dans quel état doit être le système (l'application) avant que ce cas d'utilisation puisse être déclenché.

Des scénarios :

- Ces scénarios sont décrits sous la forme d'échanges d'événements entre l'acteur et le système. On distingue le scénario nominal, qui se déroule quand il n'y a pas d'erreur, des scénarios alternatifs qui sont les variantes du scénario nominal et enfin les scénarios d'exception qui décrivent les cas d'erreurs.

Des postconditions :

- Elles décrivent l'état du système à l'issue des différents scénarios.

3. La troisième partie de la description d'un cas d'utilisation est une rubrique optionnelle. Elle contient généralement des spécifications non fonctionnelles (spécifications techniques…). Elle peut éventuellement contenir une description des besoins en termes d'interface graphique.

Exemple:

Partie 1

Nom: Retirer de L'argent au distributeur

Objectif:

- Lorsqu'un client a besoin de liquide il peut en utilisant un distributeur retirer de l'argent de son compte.
- Le client pourra choisir une opération parmi le retrait d'argent et la consultation du solde de son compte. Pour chaque opération, il faudra s'être identifié.
- Un système central extérieur permettra de vérifier le solde des comptes dans le cas d'un retrait.

Acteurs principaux:

- le client

Acteurs secondaires:

- le système bancaire

Dates:

- le 19 mars 2018

Responsable:

- Mme Guerroudji

Version:

- 1.0

Partie 2

Précondition:

Le distributeur contient des billets, il est en attente d'une opération, il n'est ni en panne, ni en maintenance

Début: lorsqu'un client introduit sa carte bancaire dans le distributeur.

Fin: lorsque la carte bancaire et les billets sont sortis.

Postcondition:

Si de l'argent a pu être retirée la somme d'argent sur le compte est égale à la somme d'argent qu'il y avait avant, moins le montant du retrait. Sinon la somme d'argent sur le compte est la même qu'avant.

Scénarios:

Déroulement normal:

(1) le client introduit sa carte bancaire

(2) le système lit la carte et vérifie si la carte est valide

(3) le système demande au client de taper son code

(4) le client tape son code confidentiel

(5) le système vérifie que le code correspond à la carte

(6) le client choisi une opération de retrait

(7) le système demande le montant à retirer

(8) le client saisi le montant du retrait

(9) le système vérifie qu'il y a suffisamment d'argent

(10) si c'est le cas, le système distribue les billets et débite le compte du client

(11) le client prend les billets et retire sa carte.

Variantes:

A. Carte invalide: au cours de l'étape (2) si la carte est jugée invalide, le système affiche un message d'erreur, rejette la carte et le cas d'utilisation se termine.

B. Code erroné: au cours de l'étape (5) si le code n'est pas valide, le système affiche un message d'erreur, rejette la carte et le cas d'utilisation se termine.

C. Montant insuffisant: à l'étape (9) si le montant saisi est supérieur au solde, le système affiche un message d'erreur et passe à l'étape (3)

.....

Partie3

Contraintes non fonctionnelles :

A. Performance : le système doit réagir dans un délai inférieur à4 secondes, quelque soit l'action de l'utilisateur.

B. Résistance aux pannes : si une coupure de courant ou une autre défaillance survient au cours du cas d'utilisation, la transaction sera annulée, l'argent ne sera pas distribué. Le système doit pouvoir redémarrer automatiquement dans un état cohérent et sans intervention humaine.

C. Résistance à la charge : le système doit pouvoir gérer plus de 1000 retraits d'argent par jour

...

Chapitre 6

Diagrammes UML du modèle dynamique

(Vue dynamique)

I. Introduction

- Le modèle dynamique montre le comportement du système, les interactions des objets et leur évolution dans le temps.
- Le modèle dynamique est représenté par différents diagrammes. Dans ce modèle l'accent est mis sur la chronologie pour répondre aux questions suivantes :
  - ✓ Comment interagissent les objets pour atteindre un but ?
  - ✓ Comment évoluent les états des objets ?
- On utilise pour répondre à la première question des diagrammes d'interactions tels le diagramme de séquence et le diagramme de communication (collaboration) et pour répondre à le seconde question les diagrammes d'états-transitions et les diagrammes d'activités

II. Diagrammes d'interaction

- Représentent les interactions avec le système et au sein du système.
- Les **interactions** se traduisent par des **envois de messages** entre les acteurs et le système, entre les objets du système de façon chronologique.

II.1 Diagramme de séquence

- Le **diagramme de séquence** fait partie des **diagrammes d'interaction**. Il permet de décrire les interactions entre objets et acteurs en insistant sur l'ordonnancement temporel des messages.

Remarques:

➔ Les diagrammes de séquences peuvent servir à illustrer un cas d'utilisation.

➔ L'ordre d'envoi d'un message est déterminé par sa position sur l'axe vertical du diagramme ; le temps s'écoule "de haut en bas" de cet axe.

➔ La disposition des objets sur l'axe horizontal n'a pas de conséquence pour la sémantique du diagramme.

II.1.1 Éléments du diagramme de séquence

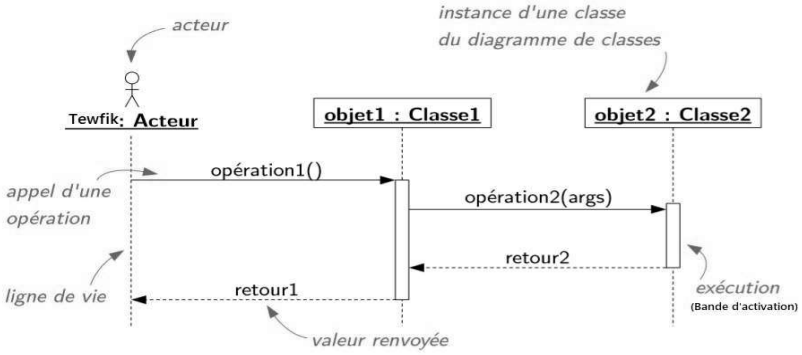
a) **Les objets**

- Ce sont des entités appartenant au système (instance d'une classe), ou se trouvant à ses limites (les acteurs), ou le système lui-même.
- Ils représentent soit des concepts abstraits, ou des acteurs (documentation des cas d'utilisations), soit des objets d'implantation (interactions informatiques)

Remarques:

- ➔ Les objets sont identifiés par l'intermédiaire des diagrammes de cas d'utilisation ou des diagrammes de classes.
- ➔ La ligne de vie des objets représente la période de temps durant laquelle l'objet existe.
- ➔ La période d'activation correspond à un intervalle de temps durant lequel l'objet effectue une action, soit directement, soit par l'intermédiaire d'un autre objet qui lui sert de sous-traitant (appel de procédure).
- ➔ La vie de l'objet commence par l'objet lui-même et se termine à la fin de la ligne de vie, éventuellement marquée par une croix indiquant la destruction de l'objet.
- ➔ Axe vertical : représentation implicite du temps, non gradué
- ➔ Emplacement de l'acteur : Si l'acteur existe dès le début du scénario, en haut ; Sinon, à l'extrémité du message donnant lieu à sa création
- ➔ Fin de l'acteur : Aucune si l'acteur n'est pas détruit ; Sinon, une croix là ou l'objet est détruit
- ➔ Durant sa vie, un acteur peut être actif ou inactif : Un acteur actif effectue une opération ou attend le retour d'un envoi de message en mode synchrone (ex : appel de fonction)

Représentation UML



b) Types de messages

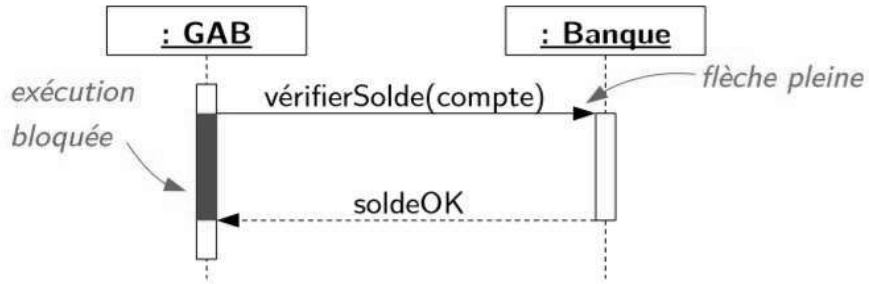
- Les principales informations contenues dans un diagramme de séquence sont les messages échangés entre les lignes de vie.
- Un message définit une communication particulière entre des lignes de vie (objets ou acteurs).
- Ils sont représentés par des flèches
- Ils sont présentés du haut vers le bas le long des lignes de vie, dans un ordre chronologique

- La réception des messages provoque une période d'activité (rectangle vertical sur la ligne de vie) marquant le traitement du message (spécification d'exécution dans le cas d'un appel de méthode).
- Plusieurs types de messages existent, dont les plus courants : l'envoi d'un signal ; l'invocation d'une opération (appel de méthode) ; la création ou la destruction d'un objet.
- On distingue également :

i. Message synchrone

- L'émetteur envoi un message
- Il suspend son exécution durant l'attente de la réponse

Exemple :



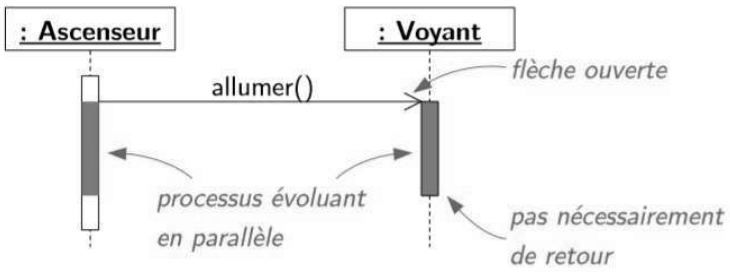
Remarque :

➔ Le message synchrone correspond à un appel de procédure ou flot de contrôle imbriqué. La séquence imbriquée est entièrement faite avant que l'appelant ne continue : l'appelant est bloqué jusqu'à ce que l'appelé termine.

ii. Message asynchrone

- L'émetteur envoi un message
- Il continue son exécution sans attendre la réponse

Exemple :



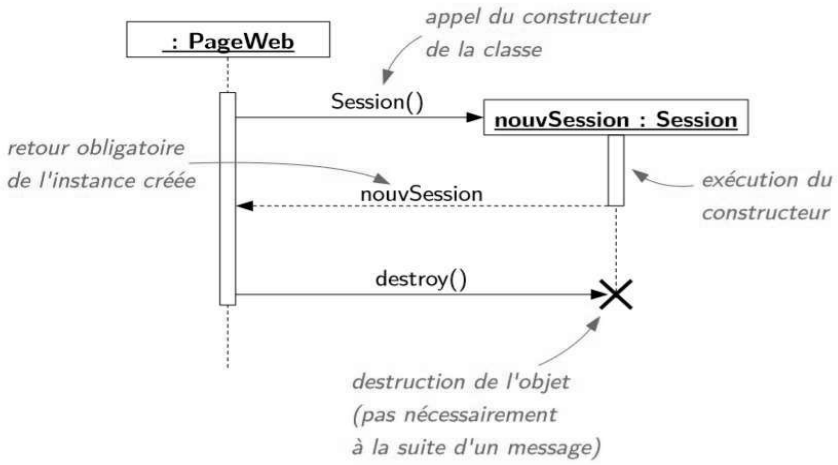
Remarque:

- ➔ Le message asynchrone correspond à un message "à plat", non imbriqué.
- ➔ L'appelant n'est pas bloqué jusqu'à ce que l'appelé termine.

iii. **Messages de création et de destruction d'objet**

- ➔ **Création** :
  - Message envoyé à une ligne de vie, pointant sur la tête d'une ligne de vie
- ➔ **Destruction** :
  - Message stéréotype **<< destroy >>** précédant une croix sur la ligne de vie

Exemple :



Remarques :

- ➔ La création se symbolise par un message d'appel du constructeur
- ➔ La syntaxe des étiquettes de message est la suivante :

[Garde] Iteration resultat : = nom_message(arguments)

Avec :

- **[Garde]** est la condition d'envoi du message (facultatif)
- **Iteration** décrit une éventuelle itération sur l'envoi du message (facultatif). On distingue deux types d'itérations :
  - ✓ **Itération séquentielle** : envoi séquentiel de plusieurs instances du même message, dénoté par ∗ **[clause d'itération]**

✓ **Itération parallèle** : envoi en parallèle de plusieurs instances du même message, dénoté par ∗ **II [clause d'itération]**

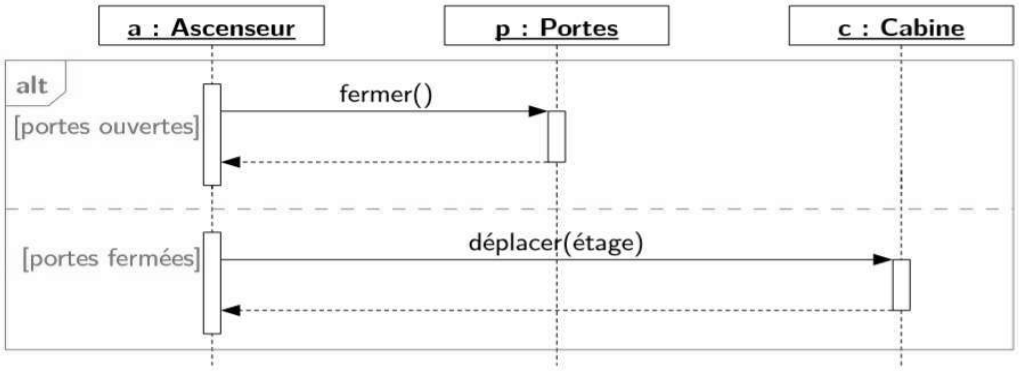
- **resultat :=** est l'affectation d'une valeur de retour (facultatif)
- **nom_message** est le nom du message
- **Arguments** sont les arguments du message (facultatif)

c) Fragments d'alternatives

i. Fragment alt : opérateur conditionnel

- Les différentes alternatives sont spécifiées dans des zones délimitées par des pointillés.
- Les conditions sont spécifiées entre crochets dans chaque zone.

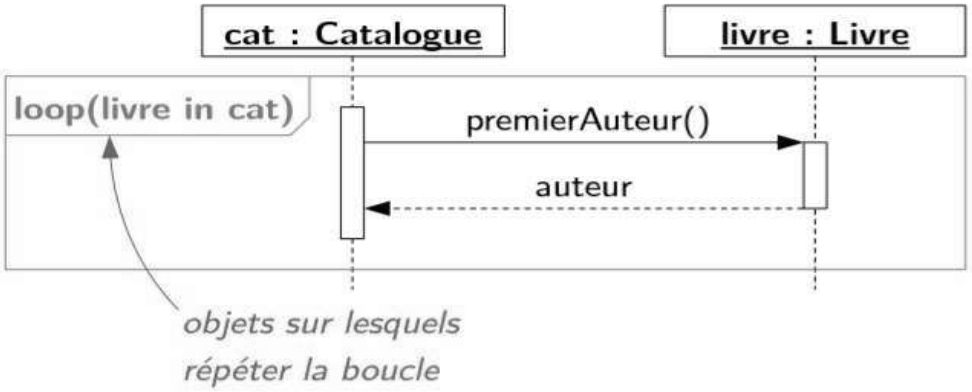
Exemple :



ii. Fragment loop : opérateur d'itération

- Le fragment loop permet de répéter ce qui se trouve en son sein.
- On peut spécifier entre crochets à quelle condition continuer.

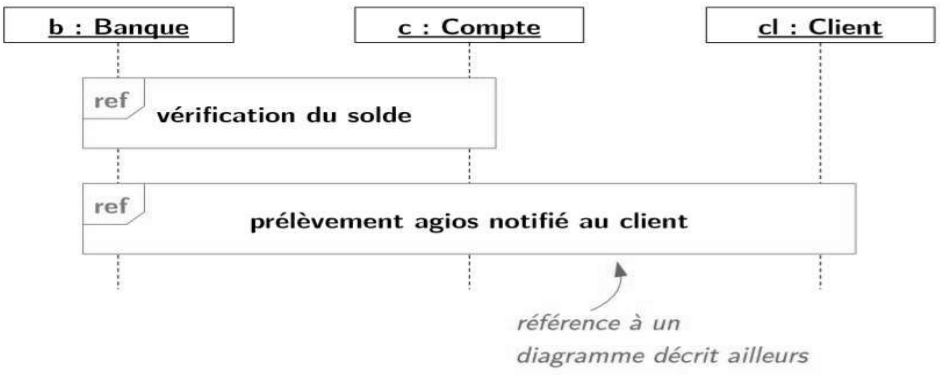
Exemple :



iii. **Fragment ref: réutilisation de séquences**

- Un fragment ref permet d'indiquer la réutilisation d'un diagramme de séquences défini ailleurs.

Exemple :

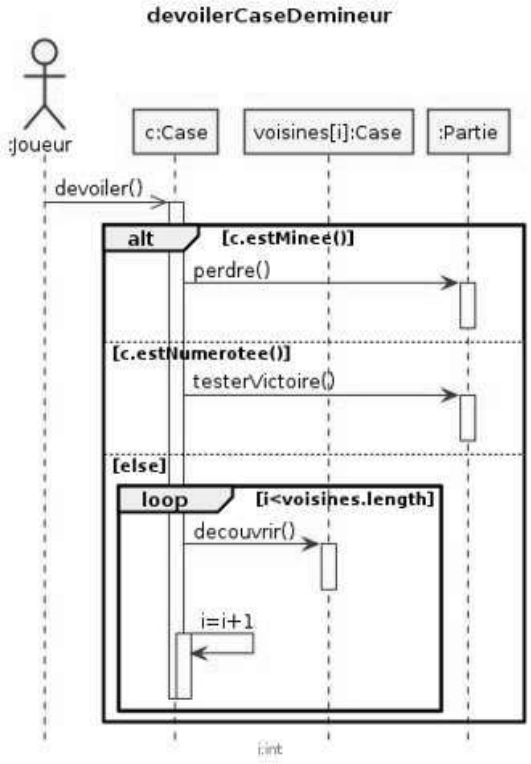


Remarques

➔ Les fragments peuvent s'imbriquer les uns dans les autres

➔ On peut émettre des messages réflexifs et dans ce cas, on définit une activité "dans" l'activité

Exemple :



II.1.2 Intérêt du diagramme de séquence

- Le diagramme de séquence est utilisé pour modéliser des flux de contrôle ordonnés selon le temps. Il ne décrit toutefois pas le contexte
- Il sert à :

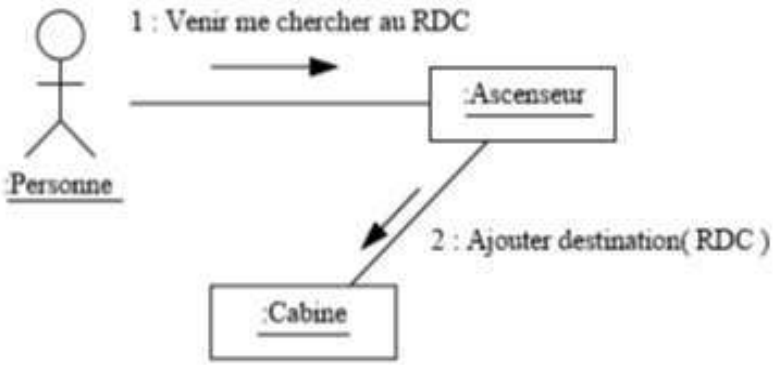
  ✓ la documentation des cas d'utilisation, en représentant les interactions entre les acteurs et le système. Les étiquettes des messages sont des événements qui se produisent dans le système.

  ✓ la représentation des interactions informatiques et la répartition des flots de contrôle. Les messages échangés unifient les différentes formes de communication entre les objets (appels de procédure, émission de signaux, etc.).

II.2 Diagramme de communication

- Appelé aussi diagramme de collaboration (UML1.X),
- Montre des interactions entre objets,
- S'intéresse à la structure de collaboration entre objets (séquencement, itération, concurrence, etc.) à travers la représentation chronologique d'envois de messages, mais le temps n'est pas représenté implicitement.
- Représente le contexte d'une interaction car on peut y précise les états des objets qui interagissent,
- Fournit des outils pour déterminer les interfaces des objets : la description du message donne la signature de l'opération pour l'objet récepteur.

Exemple :



Remarques :

- ➔ La communication est vue comme une interaction qui se déroule dans un contexte
- ➔ La structure spatiale est décrite par les liens entre les objets
- ➔ La dimension temporelle est illustrée par les numérotations des messages
- ➔ Ce diagramme est équivalent au diagramme de séquence.

II.2.1 Éléments du diagramme de communication

Les diagrammes de communication comportent trois composants :

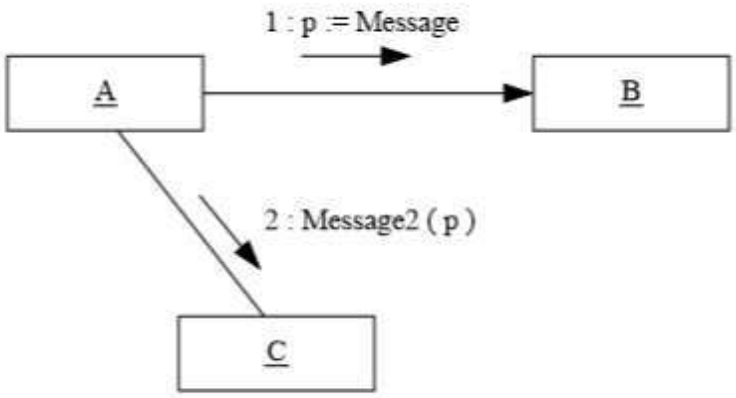
- Les objets (instances de classe) et les acteurs
- Les liens entre deux objets (comme dans le diagramme de classe) ou entre un objet et un acteur
- Les interactions sous forme d'une suite de messages

Remarque :

➔ Il est possible de spécifier de manière très précise l'ordre et les conditions d'envoi des messages. Pour chaque message, il est possible d'indiquer :

[synchronisation]['['garde']'][séquence][itération][résultat:=] nommessage['('arguments')']

Exemple:



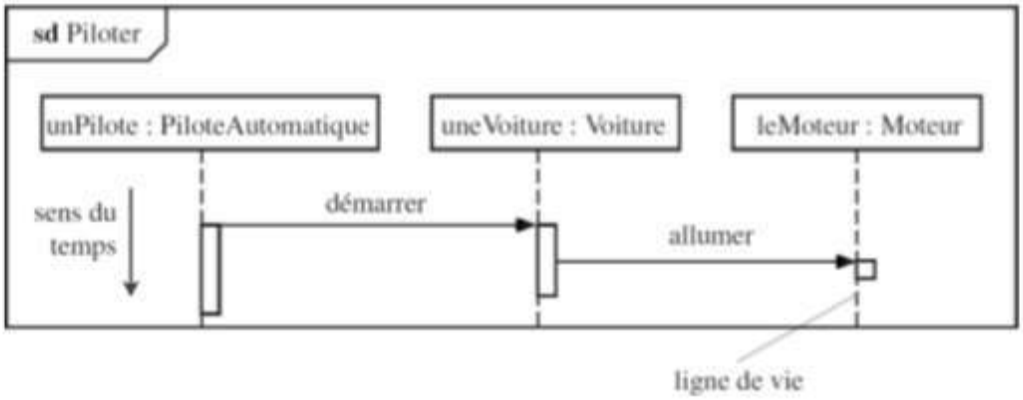
II.2.2 Intérêt du diagramme de communication

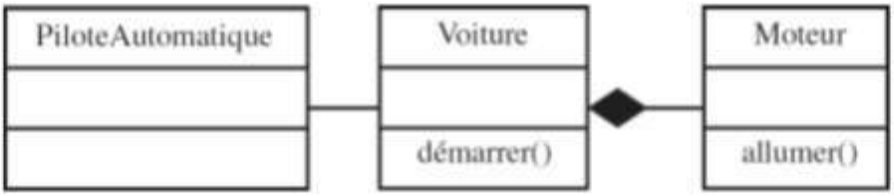
- Le diagramme de communication représente les éléments réalisant un objectif particulier (une fonctionnalité du système).
- Il peut être utilisé pour illustrer l'exécution d'une opération ou d'un cas d'utilisation, ou simplement un scénario d'interactions dans le système.
- Il permet de concevoir un exemple d'interactions entre objets.
- Aide à valider le diagramme de classes en mettant en évidence la nécessité de chaque association comme moyen de transmission des messages,

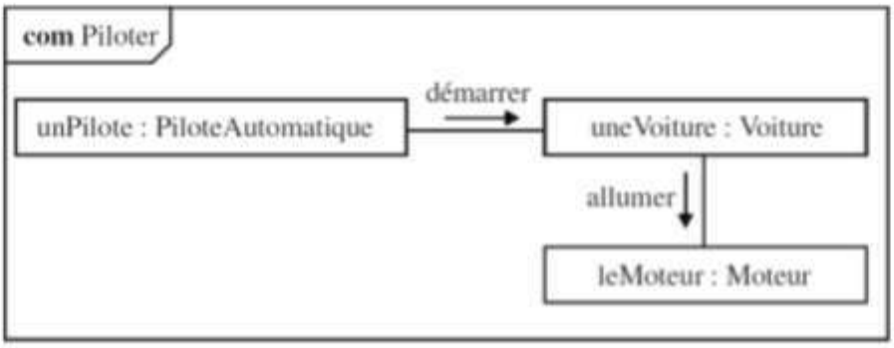
II.3 Exemple de diagramme de séquence et de communication

- Le diagramme de séquence et de communication sont deux vues du même phénomène
- Il existe des outils de passage de l'un à l'autre

Exemple:







Remarque :

➔ Les diagrammes d'interactions décrivent la réalisation des cas d'utilisation (description de scénarios particuliers) sur le système décrit par le diagramme de classes.

III. Diagrammes d'états-transitions

- Décrit le comportement dynamique des objets d'une classe au moyen d'un automate d'états associé à la classe.
- Le comportement est modélisé par un graphe :
  - ➔ Les nœuds représentent les états possibles des objets
  - ➔ Les arcs représentent les transitions d'état à état.
- Le comportement est décrit par la séquence d'états que subira l'objet au cours de son cycle de vie et les transitions entre ses états.

Remarques :

➔ Les objets changent d'état en réponse à des événements extérieurs donnant lieu à des transitions entre états.

➔ Pendant qu'il se trouve dans un état, un objet peut se contenter d'attendre un signal provenant d'autres objets. Il est alors inactif. Il peut également être actif et réaliser une activité. Une activité est l'exécution d'une série de méthodes et d'interactions avec d'autres objets.

➔ Le modèle dynamique comprend plusieurs diagrammes d'états. Chaque diagramme d'états ne concerne qu'une seule entité (classe)

III.1 Éléments du diagramme

a) **État :**

- Abstraction d'un moment de la vie d'un objet pendant lequel il satisfait un ensemble de conditions
- Une situation stable qui possède une certaine durée pendant laquelle un objet exécute une activité ou attend un événement.

b) Transition :

- Une transition entre deux états indique qu'une instance peut changer d'état et exécuter certaines activités, si un événement déclencheur se produit et que les conditions de garde sont vérifiées. C'est la réaction de l'objet sous l'effet d'une occurrence d'événement.
- Sa syntaxe est la suivante :

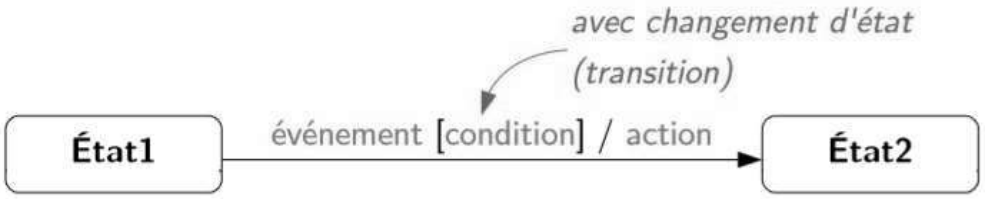
nomEvenement(params) [garde] / activité

➔ La garde désigne une condition qui doit être remplie pour pouvoir déclencher la transition,

➔ L'activité désigne des actions à effectuer lorsque l'événement se produit, c'est-à-dire si la condition est vérifiée

III.2 Représentation UML

- Les états sont représentés par des rectangles aux coins arrondis
- Les transitions sont représentées par des arcs orientés étiquetés liant les états entre eux.



Remarques :

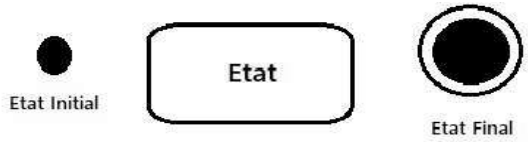
- ➔ Chaque objet possède à un instant donné un état particulier
- ➔ Chaque état est identifié par un nom
- ➔ Un état est stable et durable
- ➔ L'événement est déclencheur de la transition d'état à état. Un objet, placé dans un état donné, attend l'occurrence d'un événement pour passer dans un autre état

III.3 Les états d'un objet

- L'ensemble des états du cycle de vie d'un objet contient un état initial. Celui-ci correspond à l'état de l'objet juste après sa création (défini par l'un des constructeurs de l'objet).
- Il peut également contenir un ou plusieurs états finaux. Un état final correspond à une étape où l'objet n'est plus nécessaire dans le système.
- Tous les objets n'ont pas d'état final. C'est notamment le cas des objets permanents dans le système.

Représentation UML

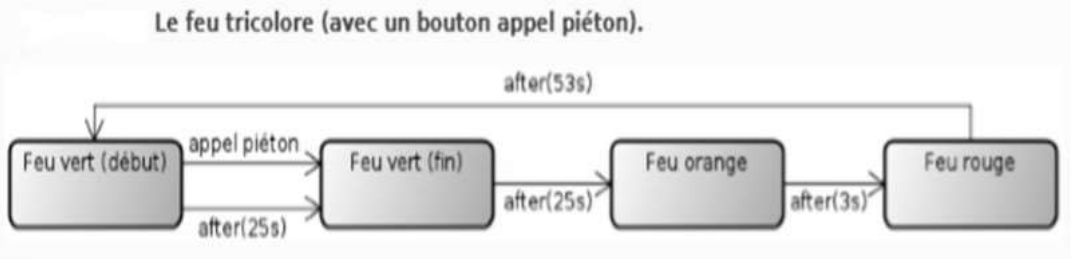
- ➔ L'état initial est représenté par un point noir
- ➔ L'état final se représente par un point entouré d'un cercle.



a) **Types d'événements :**

- Signal : réception d'un message asynchrone
- Appel d'une opération (synchrone) : liée aux cas d'utilisation, opération du diagramme de classes...
- Satisfaction d'une condition booléenne : when(cond), évaluée continuellement jusqu'à ce qu'elle soit vraie
- Temps
  - Date relative : when (date = date)
  - Date absolue : after (durée)

Exemple:

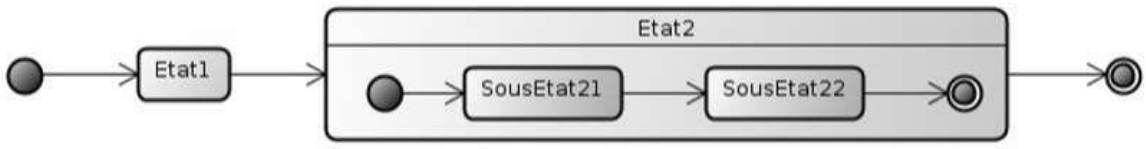


b) Types d'états

1. **État composite :**

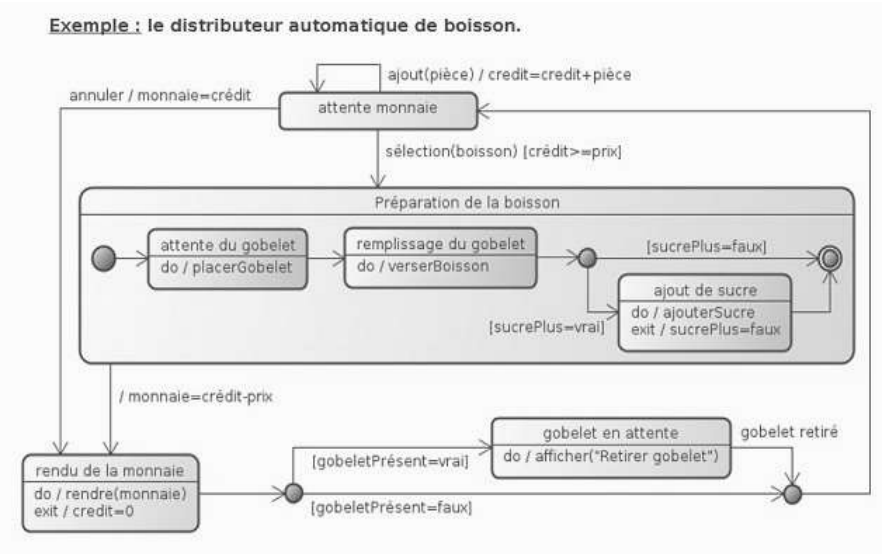
- Certains états sont complexes et correspondent à la réalisation de plusieurs activités (séquentielles ou simultanées) qui ne pourront pas être définis par des transitions internes. Ils sont alors décomposés en sous-états. On parle d'état composite.
- L'état composite est un état regroupant un ensemble d'états. Il permet alors de :
  - ✓ Hiérarchiser les états
  - ✓ Structurer les comportements complexes
  - ✓ Factoriser les actions

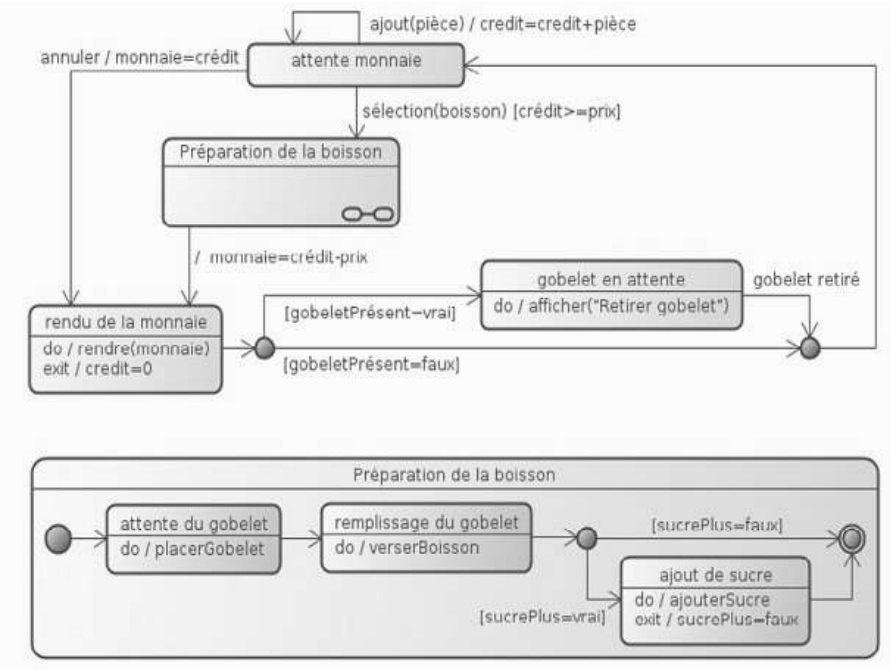
Représentation UML



Afin d'éviter de surcharger le diagramme d'états-transition, il est possible de placer le symbole (o-o) à l'intérieur de l'état composite et de spécifier ensuite son contenu ailleurs dans une autre page.

Exemple :

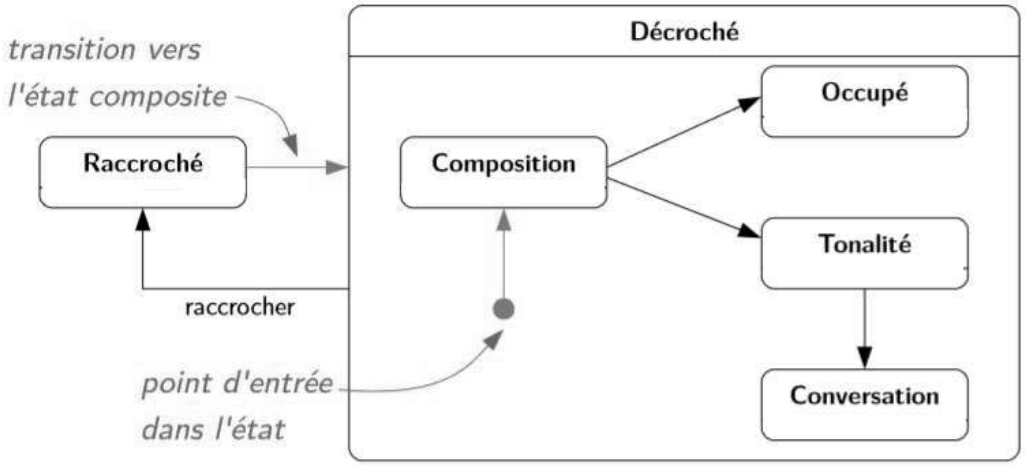




Remarques:

- ➔ Dès qu'un état composite est actif, il active son sous état initial.
- ➔ Les transitions peuvent avoir pour cible la frontière d'un état composite. Elles sont alors équivalentes à une transition ayant pour cible l'état initial de l'état composite.
- ➔ Une transition ayant pour source la frontière d'un état composite est équivalente à une transition qui s'applique à tout sous-état de l'état composite source.
- ➔ Cette relation est transitive et peut traverser plusieurs niveaux d'imbrication.

Exemple :

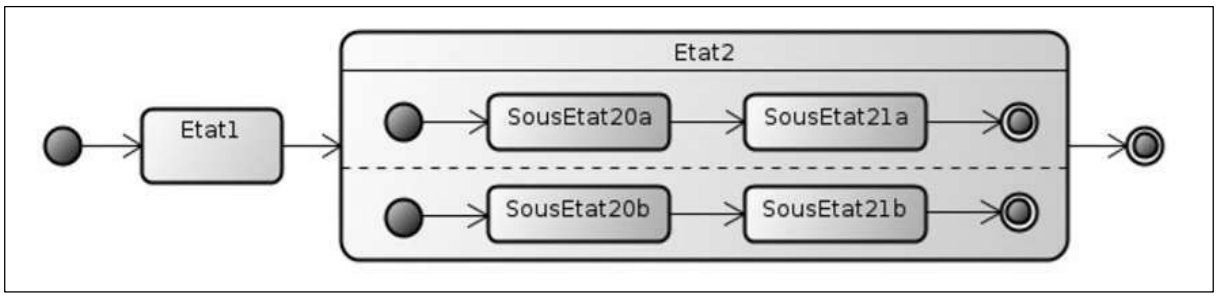


2. **État composite concurrent ou État orthogonal**

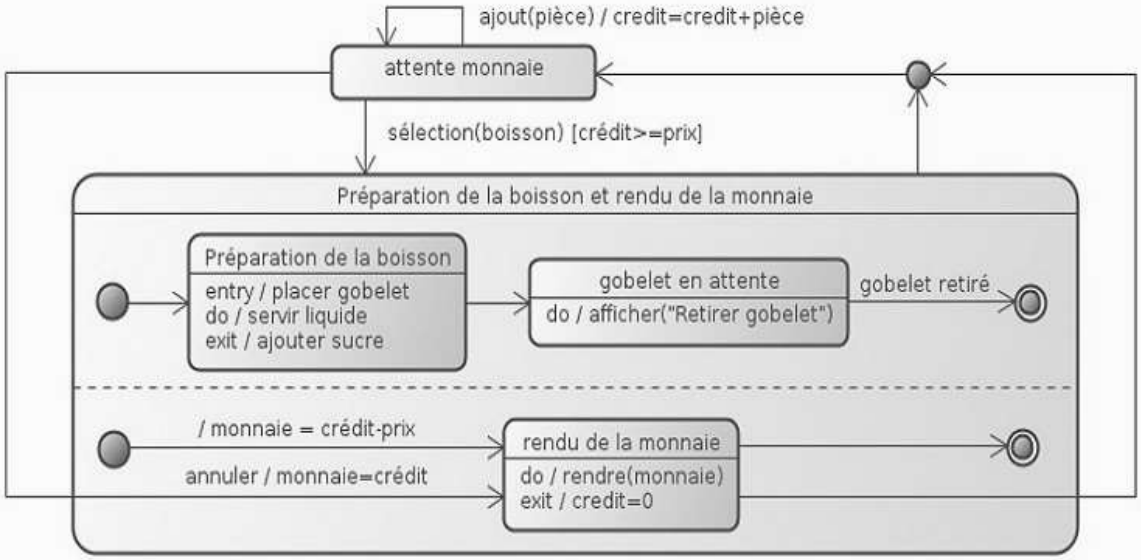
Les diagrammes d'états-transitions permettent de décrire efficacement les mécanismes concurrents grâce à l'utilisation d'états composites concurrents dits états orthogonaux.

- Un état orthogonal est un état dans lequel plusieurs états sont actifs simultanément (concurrence/parallélisme)
- Il possède plusieurs régions (séparées par des pointillés horizontaux) qui évoluent simultanément et en parallèle.
- Chaque région représente un flot d'exécution, elle peut posséder un état initial et un état final.
- Une transition qui atteint la bordure d'un état composite orthogonal est équivalente à une transition qui atteint les états initiaux de toutes ses régions concurrentes.
- Toutes les régions concurrentes d'un état composite orthogonal doivent attendre leur état final pour que l'état composite soit considéré comme terminé.

Représentation UML



Exemple : *Dans l'exemple du distributeur automatique de boisson que nous avons vu précédemment, si nous considérons que nous pouvons rendre la monnaie en même que se prépare la boisson, nous obtenons le diagramme d'états-transitions suivant :*

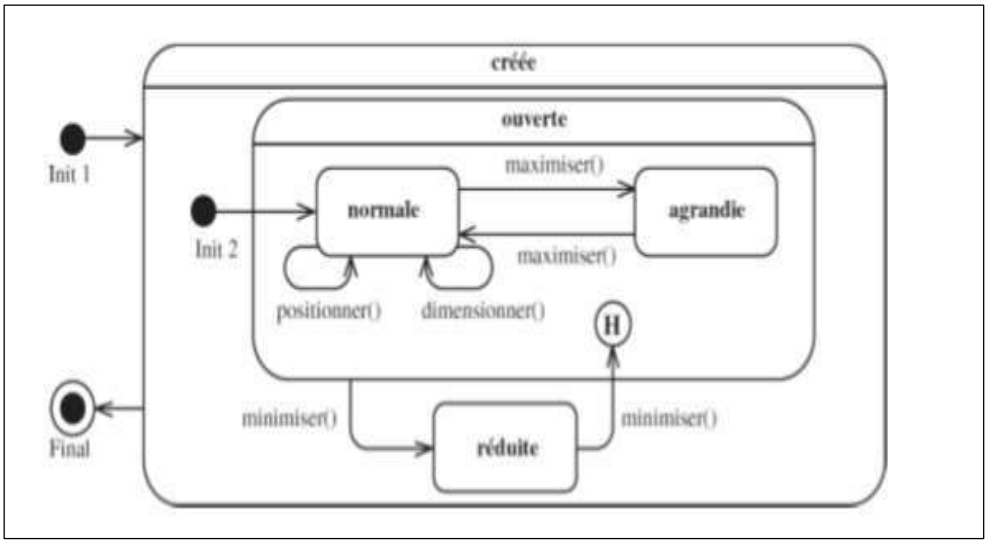


3. État Historique

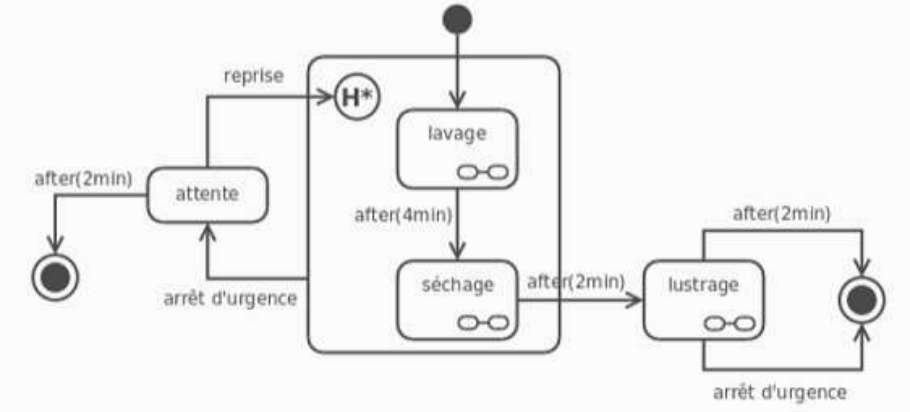
Si nous désirons qu'un état composite reprenne sont activité à l'endroit ou il s'était interrompu lors de sa précédente activation, il faut lui définir un nouveau sous état d'entrée appelé état historique et désigné par H ou H* :

- H pour reprendre au début du sous-état du plus haut niveau dans lequel nous nous étions arrêtés.
- H* pour reprendre au début du sous état dans lequel nous nous étions arrêtés, quelque soit son niveau d'imbrication (historique profond).

Exemple 1 : une fenêtre à l'écran



Exemple 2: système de lavage automatique de voiture

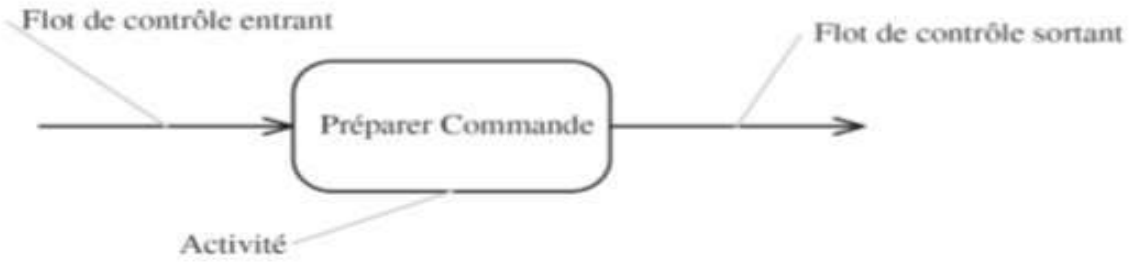


IV. Diagramme d'activité

- Ce diagramme met l'accent sur les activités, leurs relations et leurs impacts sur les objets
- Un diagramme d'activités (activités et transitions) est une variante du diagramme d'états-transitions (états et transitions). Les deux types de diagrammes permettent d'avoir deux vues différentes sur des automates donnés.
- Un diagramme d'activités visualise un graphe d'activités qui modélise le comportement interne d'un processus impliquant un ou plusieurs classificateurs (classes / cas d'utilisation / paquetages /...).

- Un diagramme d'activités représente l'état d'exécution d'un mécanisme, sous la forme d'un déroulement d'étapes regroupées séquentiellement dans des branches parallèles de flots de contrôle.

Exemple:



IV.1 Éléments du diagramme d'activité

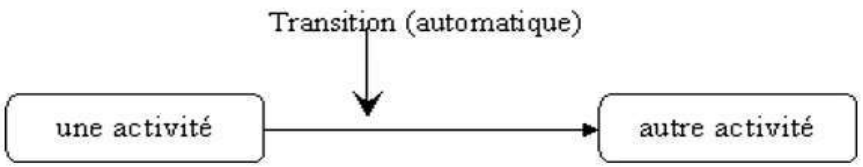
a) **Activité**

- Les activités décrivent un traitement.
- Le flot de contrôle reste dans l'activité jusqu'à ce que les traitements soient terminés.
- On peut définir des variables locales à une activité et manipuler les variables accessibles depuis le contexte de l'activité (classe contenante en particulier).
- Les activités peuvent être imbriquées hiérarchiquement : on parle alors d'activités composites.

b) Transition

- **L**es transitions sont représentées par des flèches pleines qui connectent les activités entre elles.
- Elles sont déclenchées dès que l'activité source est terminée.
- Elles provoquent automatiquement le début immédiat 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.

Représentation UML

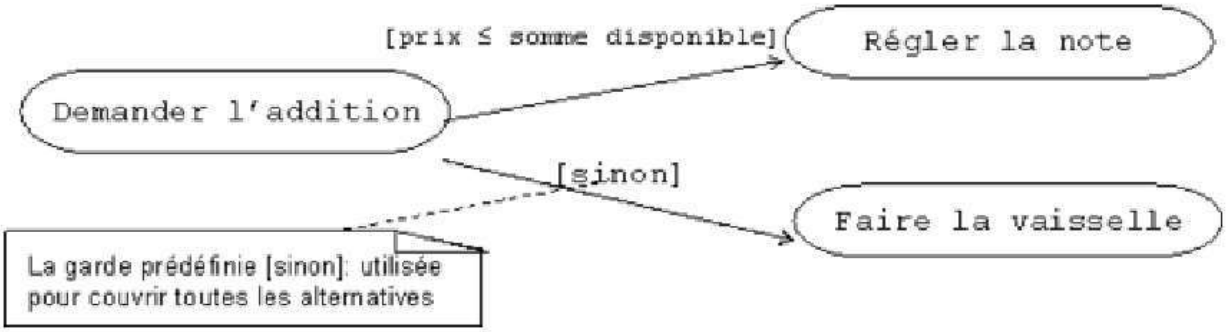


Remarques:

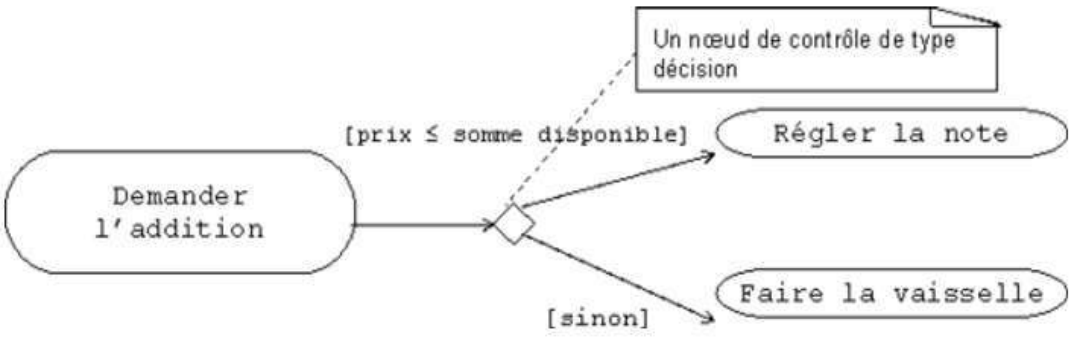
- ➔ Une activité définit un comportement décrit par un séquencement organisé d'unités dont les éléments simples sont les actions.
- ➔ Le flot d'exécution est modélisé par des nœuds reliés par des arcs (transitions). Le flot de contrôle reste dans l'activité jusqu'à ce que les traitements soient terminés.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

c) **Types de transitions :**

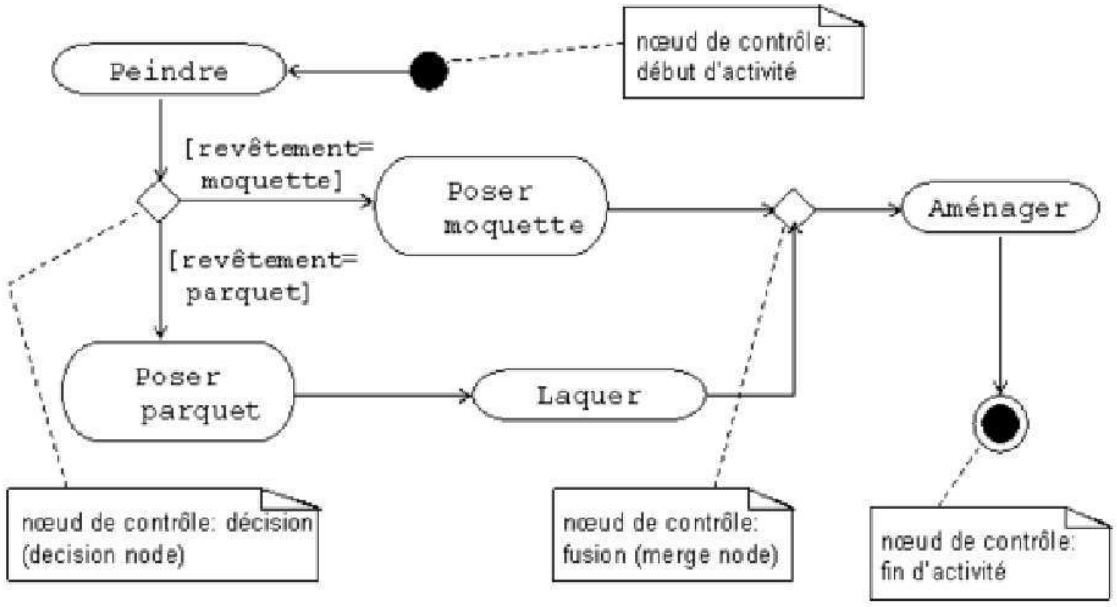
- ➔ Il est possible de représenter des transitions conditionnelles en utilisant des gardes (appelées aussi décisions) qui doivent être mutuellement exclusives.



- ➔ Une décision peut aussi utiliser des transitions composites et créer un point de jonction. Le point de jonction (décision) : matérialisé par un losange ayant en entrée une seule transition et en sortie plusieurs transitions



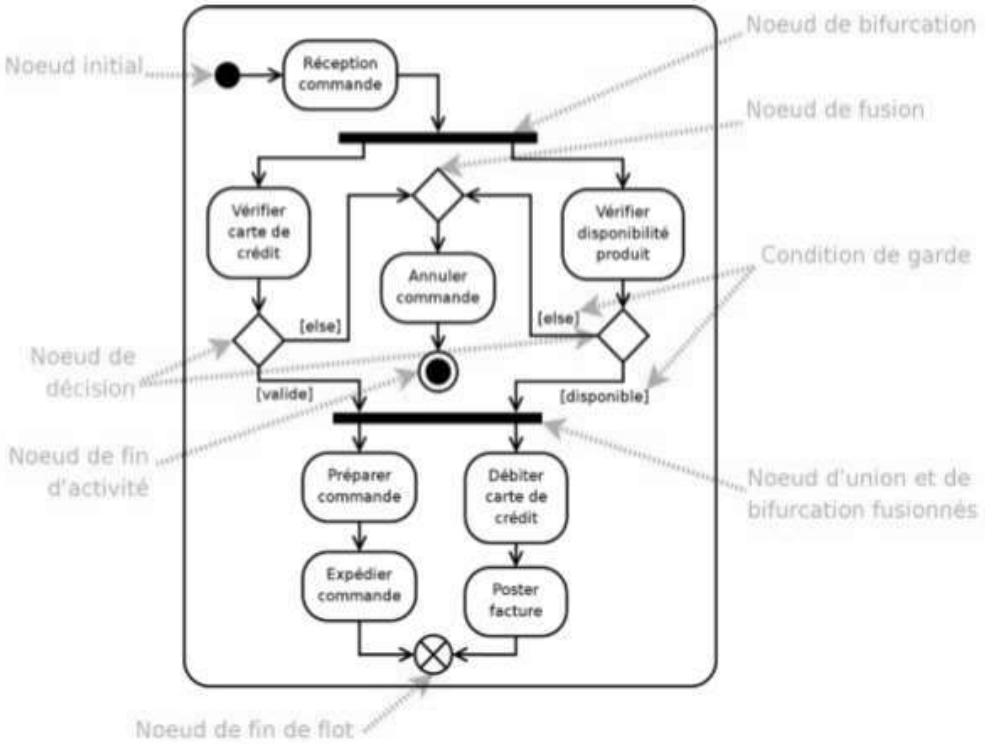
➔ À un point de jonction peuvent se rejoindre des chemins séparés par de précédentes décisions. Ce point reçoit plusieurs transitions gardées ou non en entrée (flots entrant) mais il permet une seule transition en sortie (flot sortant). Il n'est pas utilisé pour synchroniser des flots concurrents mais pour accepter un parmi plusieurs



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

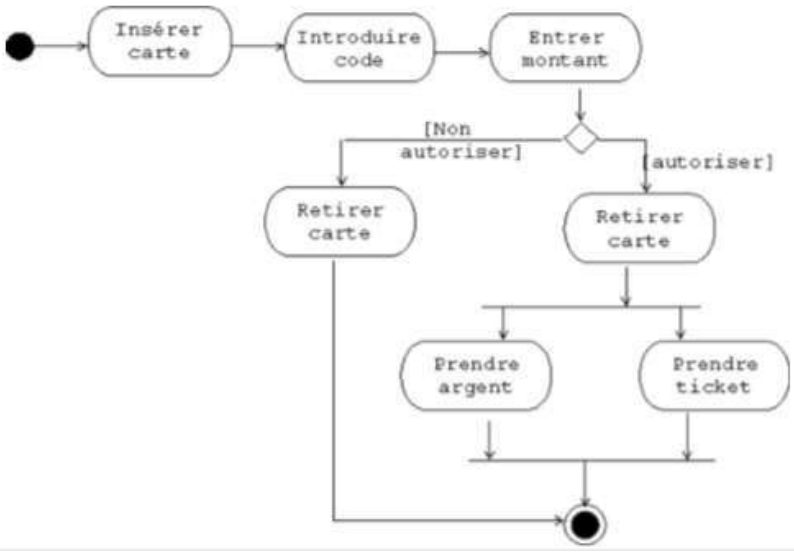
Exemple :



Remarque:

➔ Dans un diagramme d'activités, la même action peut être exécutée plusieurs fois.

Exemple :



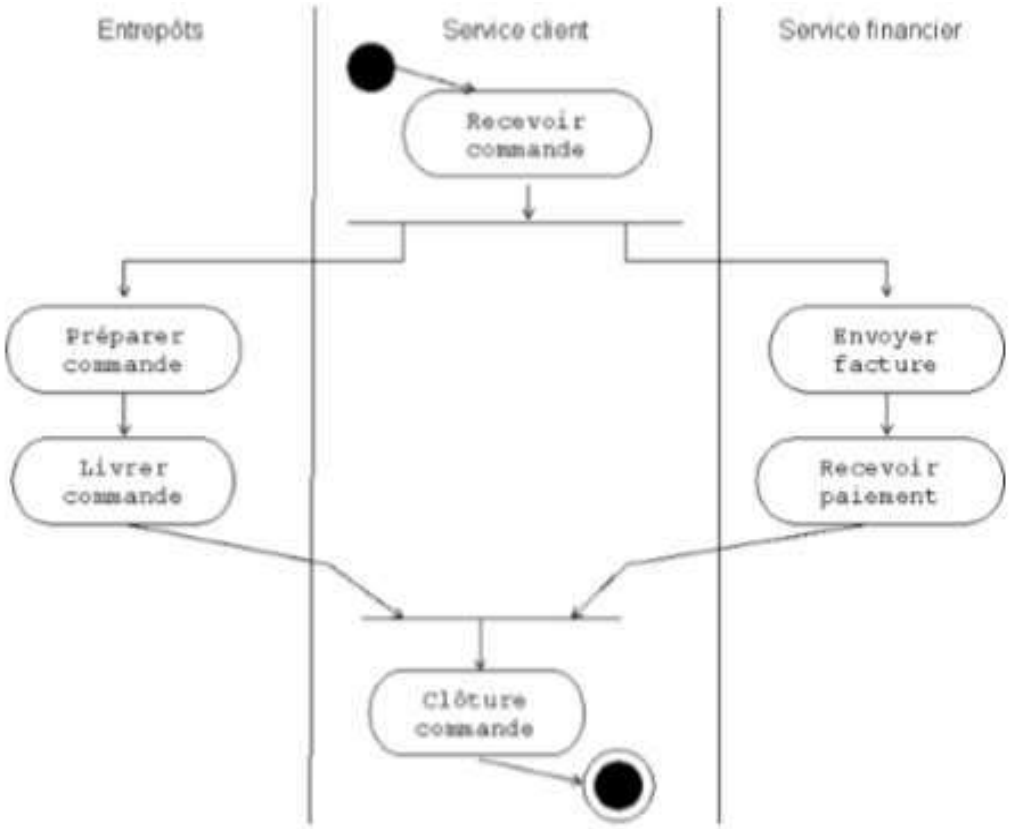
d) **Couloirs d'activités**

- Les diagrammes d'activités indiquent ce qui se passe sans préciser qui fait quoi (en termes de programmation, ils ne précisent pas quelle classe est responsable et en termes de processus métier, ils ne précisent pas quelle partie de l'organisation exécute chaque action).

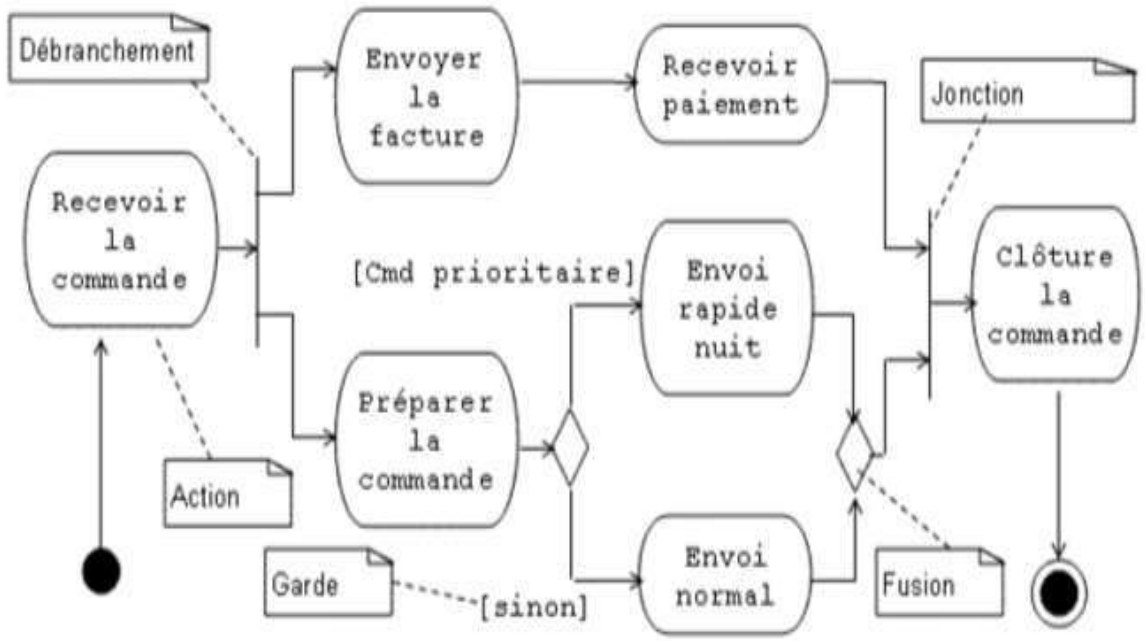
- Il est possible de diviser un diagramme d'activités en partitions ou couloirs d'activités (travées, swimlanes).

- Chaque partition montre quelles actions sont exécutées par une classe ou une unité organisationnelle.

Exemple :



EXEMPLE (récapitulatif) :



Remarques:

- ➔ Les diagrammes UML du modèle dynamique sont complémentaires.
- ➔ Le modèle dynamique identifie les différents événements venant du monde externe et montre l'enchaînement dans le système que provoquent ces événements.

Références bibliographiques

1. L. Audibert L. « UML 2 : de l'apprentissage à la pratique » Ellipses, octobre 2014
2. M. Blaha et J. Rumbaugh. « Modélisation et conception orientées objet avec UML 2 ». 2ème édition. Pearson Education, 2005.
3. G. Booh, J. Rumbaugh, I. Jacobson, « The Unified Modeling Laguage (UML) Reference Guide », Addison-Wesley, 1999.
4. G. Booh, J. Rumbaugh, I. Jacobson « The Unified Modeling Laguage (UML) User Guide », Addison-Wesley, 1999.
5. G. Booh et al., « Object-Oriented Analysis ad Design, with applications », Addison- Wesley, 2007.
6. B. Bruegge and Allen H. Dutoit, « Object-Oriented Software Engineering – using UML, Patterns and Java ». Third Edition, Pearson, 2010
7. S. Lawrence Pfleeger and Joanne M. Atlee. « Software Engineering » Fourth Edition, Pearson, 2010.
8. P.A. Muller. « Modélisation objet avec UML ». Éditions Eyrolles, 2003.
9. P. Roques « UML 2.5 par la pratique Etude de cas et exercices corrigés » - Eyrolles - avril 2018
10. http://uml.org/