RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ EL BACHIR EL IBRAHIMI DE BORDJ BOU ARRÉRIDJ

FACULTÉ DES MATHEMATIQUES ET INFORMATIQUE

DÉPARTEMENT D'INFORMATIQUE





POLYCOPIÉ DE COURS

MODÉLISATION OBJET AVEC UML (MOU)

Destiné aux étudiants de :

Master 1 – Option : Ingénierie de l'informatique décisionnelle

Réalisé par :

Dr. Regouid Meryem

meryem.regouid@univ-bba.dz

mregouid@yahoo.com

Année Universitaire : 2025 / 2026

Table des matières

Plan du cours MOU 7
1. Informations sur le cours 7
2. Présentation du cours 7
3. Contenu 8
4. Pré-requis 9
5. Visées d'apprentissage 9
6. Modalités d'évaluation des apprentissages 10
7. Activités d'enseignement-apprentissage 11
8. Alignement pédagogique 11
9. Modalités de fonctionnement 12
10. Ressources d'aide 13
Chapitre 01 : L'approche Orientée Objet 14
1. Introduction 14
2. Objectifs du Chapitre 01 14
3. Les classes et les objets 14
3.1 Notion d'Objet (L'entité concrète) 14
3.2 Notion de Classe (Le moule) 16
4. Propriétés de l'approche objet 17
4.1 L'Abstraction 17
4.2 L'Encapsulation 17
4.3 L'Héritage 18
4.4 Le Polymorphisme 19
5. Les méthodes orientées objets 19
5.1 Évolution des méthodes 19
5.2 Comparaison : Procédurale vs OO 20
5.3 Avantages de l'approche orientée objets 20
6. L'approche objet dans les Systèmes d'Information (SI) 21
6.1 La Réduction du "Fossé Sémantique" 21
6.2 La Stabilité face au Changement (Critère décisif) 21
6.3 Gestion de la Complexité et Modularité 21

Chapitre 02 : Notations UML 22
1. Introduction 22
2. Les objectifs de chapitre 02 22
3. Cycle de vie d'un logiciel 22
4. CONCEPTION 23
5. La modélisation 23
6. Un modèle 23
7. Pourquoi modéliser ? 23
8. L'apparition de l'UML 24
9. Diagrammes UML 24
Chapitre 03 : Les diagrammes UML structurels 26
1. Introduction 26
2. Les objectifs de chapitre 03 26
3. Diagramme de classe 26
3.1. Définitions 26
3.2. Les relations entre les classes 28
3.3. Dépendance 35
3.4. Agrégation 36
3.5. Composition 36
3.6. Généralisation/Spécialisation 36
3.7. Héritage Multiple 37
3.8. La classe abstraite 37
3.9. Classe Interface 38
4. Diagramme d'objet 39
4.1. Pourquoi les diagrammes d'objets ? 39
4.2. Elément des diagrammes d'objets 39
4.3. Exemples d'un diagramme d'objets 40
4.4. Association réflexive 41
5. Diagramme de paquetage 42
5.1. Pourquoi les diagrammes de paquetage? 42
5.2. Représentation de paquetage 42
5.3. Fondamentaux du diagramme de package 43
5.4. Dépendance entre paquetages 43

6. Diagramme de composants 46
6.1. Pourquoi les diagrammes de composants 46
6.2. Un Composant 46
6.3. Modélisation des composants 47
7. Diagramme de déploiement 51
7.1. Les éléments du diagramme de déploiement 51
7.2. Exemple d'un diagramme de déploiement 54
8. Diagramme de structure composite 55
8.1. Avantages de diagramme de structure composite (DSC) 55
8.2. Composants de base d'un DSC 55
8.3. Les différences entre un DC et un DSC 58
9. Diagramme de profil 59
9.1. Éléments d'un diagramme de profil 60
9.2. Représentation d'un diagramme de profil 63
Chapitre 04: Diagrammes UML comportementaux 64
1. Introduction 64
2. Les objectifs de chapitre 04 64
3. Diagramme des cas d'utilisation 64
3.1. Les Composants d'un (DCU) 64
3.2. Paquets des cas d'utilisation 68
3.3. Description textuelle d'un cas d'utilisation 68
4. Diagramme de séquence 71
4.1. Avantages des diagrammes de séquence 71
4.2. Les Composants d'un (DS) 71
4.3. Création et destruction d'un objet 75
4.4. Contrainte temporelle 76
4.5. Les fragments dans un DS 76
4.6. Messages Perdus / Trouvés 80
5. Diagramme d'activités DA 81
5.1. Avantages des diagrammes d'activités 81
5.2. Représentation d'un diagramme d'activités 81
5.3. Les Composants d'un (DA) 82
5.4. Nœuds d'activité 83

5.5. Pin d'entrée ou de sortie 86
5.6. Nœud tampon central 87
5.7. Nœud de stockage des données 87
5.8. Les partitions ou couloirs d'activités 87
6. Diagramme d'états-transitions DET 89
6.1. Avantages d'un DET 89
6.2. Représentation d'un diagramme d'ET 89
6.3. Les Composants d'un (DET) 89
6.4. États composites 93
7. Diagramme de communication (collaboration) 95
7.1. Avantages d'un DC 95
7.2. Représentation d'un DC 95
7.3. Les Composants d'un (DC) 96
7.4. Les contraintes associées 98
7.5. Exemple d'un DS et équivalent au DC 98
8. Diagramme d'interaction globale 99
8.1. Avantages d'un DIG 99
8.2. Représentation d'un DIG 99
8.3. Les Composants d'un (DIG) 99
9. Diagramme de temps 101
9.1. Avantages d'un DT 101
9.2. Représentation d'un DT 101
9.3. Les Composants d'un (DT) 101
9.4. Contrainte de durée 102
9.5. Événement de destruction 103
9.6. Exemple d'un DT 104
Chapitre 05: Étude de cas SI 105
1. Introduction : 105
2. Les objectifs de chapitre 05 105
3. Outil de modélisation utilisé 105
4. Présentation générale du système de gestion de pharmacie en ligne 105
4.1. Problématique 105
4.2. Acteurs et fonctionnalités principales 106

4.3. Objectifs du système 107
4.4. Les Digramme UML 107
Chapitre 06: Introduction à AUML 123
1. Introduction 123
2. Les objectifs de chapitre 06 123
3. Exemple concret 123
4. Définition 124
5. Caractéristiques principales d'un agent logiciel 124
6. AUML comme extension d'UML 125
7. Du Diagramme de Classes au Diagramme de Classes d'Agent 125
8. Exemple d'une représentation de capacités 126
9. Intérêt de représentation de capacités 127
10. Protocoles d'interaction 127
11. Caractéristiques d'un protocole d'interaction 127
12. Structures de contrôle dans les protocoles 128
13. Messages et actes de communication 128
14. Avantages et limites d'AUML 128
15. Applications 129
16. Exemple 129
Bibliographie 131
Annexe A : Séries de Travaux Dirigés (TD) 132
Annexe B : Séries de Travaux Pratiques (TP) 146

Plan du cours MOU

1. Informations sur le cours

- Faculté: Mathématiques et Informatique
- Département: Informatique
- Spécialité : Ingénierie de l'informatique décisionnelle
- Public cible : 1ère année Master
- Intitulé du cours : Modélisation Objet avec UML (MOU)
- Unité d'enseignement : UEF 01
- Crédits: 06
- Coefficient: 03
- Durée : 15 semaines. (67,5 heures : 22.5h cours, 22.5h TD, 22.5h TP)
- Horaire: Dimanche: 09h30-15h30. Lundi : 09h30 – 14h
- Salle: salle 18 (département de l'économie) Salle TD : 19, Salle TP : L4
- **Enseignant :**

  Cours, TD et TP: Dr. REGOUID Meryem
- Contact :

  par e-mail au mregouid@yahoo.com
- **Disponibilité :**

  Au département : Dimanche, lundi, Mardi de 9h00 -14h00

2. Présentation du cours

Le module de Modélisation Objet avec UML constitue une étape fondamentale dans la formation de l'architecte logiciel et de l'ingénieur en systèmes d'information. Face à la complexité croissante des logiciels modernes, le codage direct est devenu obsolète et risqué. Ce cours introduit UML (Unified Modeling Language), non pas comme une simple notation graphique, mais comme le standard industriel international permettant de visualiser, spécifier, construire et documenter les éléments d'un système. Il offre un langage commun universel qui permet de combler le fossé sémantique entre les experts métier (maîtrise d'ouvrage) et les développeurs techniques (maîtrise d'œuvre).

L'objectif pédagogique de ce module est de doter les étudiants d'une capacité d'abstraction nécessaire pour concevoir des systèmes robustes et évolutifs. Le programme est structuré pour accompagner l'étudiant du concept à la pratique : nous débuterons par les fondements du paradigme Orienté Objet (Classes, Encapsulation, Héritage), avant de maîtriser les vues statiques (structure des données) et

dynamiques (comportement et interactions) du système. Enfin, le cours s'ouvrira sur des extensions avancées telles que AUML (pour les systèmes multi-agents), préparant ainsi les futurs ingénieurs aux défis des architectures décisionnelles et distribuées. Une description de MOU est illustrée dans la figure 1.

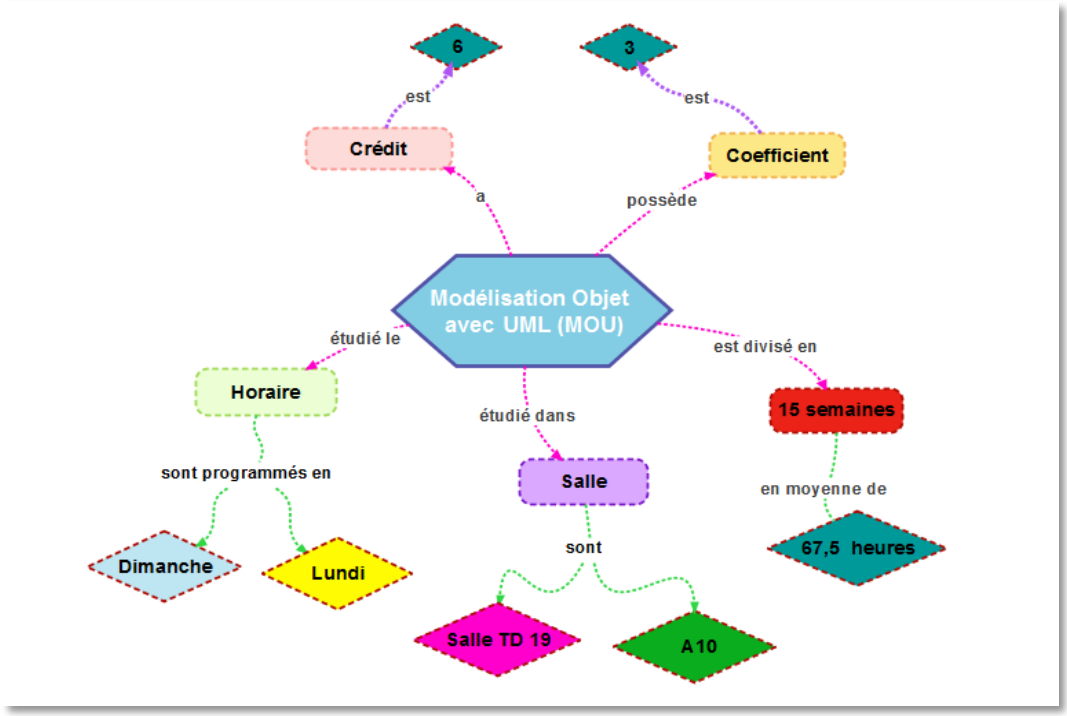


Figure 1 : Carte mentale du cours MOU

3. Contenu

Le cours est divisé en six chapitres. Des séquences pédagogiques sont proposées pour chaque chapitre afin de faciliter l'assimilation des notions abordées. La mise en application de ces concepts est renforcée à travers des activités d'apprentissage prévues pour chaque unité. Ci-dessous sont présentés les chapitres de ce cours :

Chapitre 01: L'Approche Orientée Objet

Ce chapitre pose les fondements du paradigme objet en définissant les concepts clés comme la classe, l'encapsulation et l'héritage. Il met en évidence les avantages de cette approche modulaire et stable par rapport aux méthodes fonctionnelles

Chapitre 02: Notations UML

Ce chapitre introduit UML comme le langage standard industriel de modélisation visuelle pour spécifier et documenter les systèmes d'information. Il présente l'organisation générale des vues et des diagrammes sans imposer de méthode de

Chapitre 03: Les diagrammes UML structurels

Consacré à la vue statique, ce chapitre détaille la modélisation de l'architecture des données et des relations permanentes entre les entités. Il approfondit notamment le diagramme de classes, véritable pilier structurel de toute conception logicielle.

Chapitre 04: Les diagrammes UML comportementaux

Ce chapitre se concentre sur l'aspect évolutif du système pour décrire les flux de travail et les interactions temporelles entre les objets. Il couvre les diagrammes essentiels tels que les Cas d'Utilisation, la Séquence et l'Activité pour capturer la logique métier.

Chapitre 05: Etude de cas SI

Cette section pratique vise à synthétiser l'ensemble des connaissances acquises à travers la résolution complète d'un problème métier concret. L'objectif est d'apprendre à traduire un cahier des charges textuel en une solution de

Chapitre 06: Introduction à AUML

Ce dernier chapitre explore l'extension AUML dédiée à la modélisation des Systèmes Multi-Agents (SMA) et de leurs interactions complexes. Il marque la transition conceptuelle de l'objet passif vers l'agent autonome, nécessaire pour les systèmes

4. Pré-requis

Afin que l'étudiant puisse assimiler ce cours autant que possible, il doit être conscient de ce qui suit :

1. L'algorithmique et Structures de Données : Compréhension des types abstraits, des boucles et de la logique conditionnelle.
2. Les fondements de la Programmation : Connaissance d'un langage de programmation, idéalement avec une sensibilisation à l'approche Orientée Objet (Java, C++, C# ou Python).

5. Visées d'apprentissage

Être capable de lire et d'utiliser la notation UML, et être capable de produire des modèles d'analyse ou de conception qui respectent les principes de l'objet et améliorent la qualité des logiciels.

Le cours MOU vise à :

- ✓ **En termes de connaissances** :
  - Comprendre les fondements du paradigme Orienté Objet (Classe, Héritage, Polymorphisme) par opposition à l'approche fonctionnelle.

- Identifier les différents diagrammes de la norme UML et comprendre leur rôle spécifique (Vue statique vs Vue dynamique).

✓ **En termes de savoir-faire** :

- Traduire un cahier des charges textuel (besoins métier) en une solution formelle modélisée.
- Concevoir les diagrammes structurels (Classes) et comportementaux (Cas d'Utilisation, Séquence, Activité) de manière cohérente.

✓ **En termes de savoir-être** :

- Développer votre capacité d'abstraction et de synthèse pour isoler l'essentiel d'un problème complexe.
- Adopter une démarche de conception rigoureuse avant de passer à l'étape de programmation, afin de garantir la qualité et la maintenabilité du logiciel.

6. Modalités d'évaluation des apprentissages

L'évaluation finale sera faite par un examen final plus un contrôle ou une évaluation continue (Figure 2) :

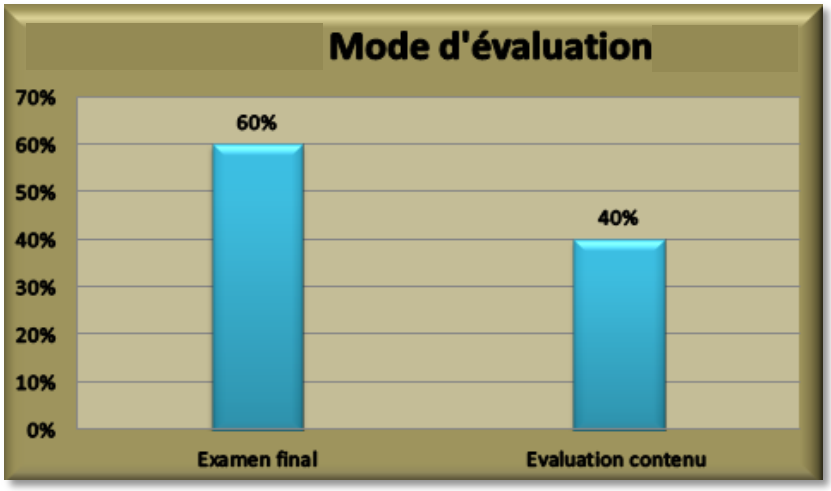


Figure 2 : Mode dévaluation

i. **Examen final:** est un examen sur table. Il représente 60 % du point final. cet examen inclura tout ce qui a été abordé dans ce cours pendant le premier semestre.

ii. **Evaluation contenu** : c'est une évaluation régulière tout au long du semestre. Elle représente 40% de la note finale de l'étudiant. Cette évaluation permet aux étudiants d'acquérir des points via leurs participations lors des séances des TDs et TPs, leurs présences et aussi leurs travaux individuels ou collectifs (voir figure 3).

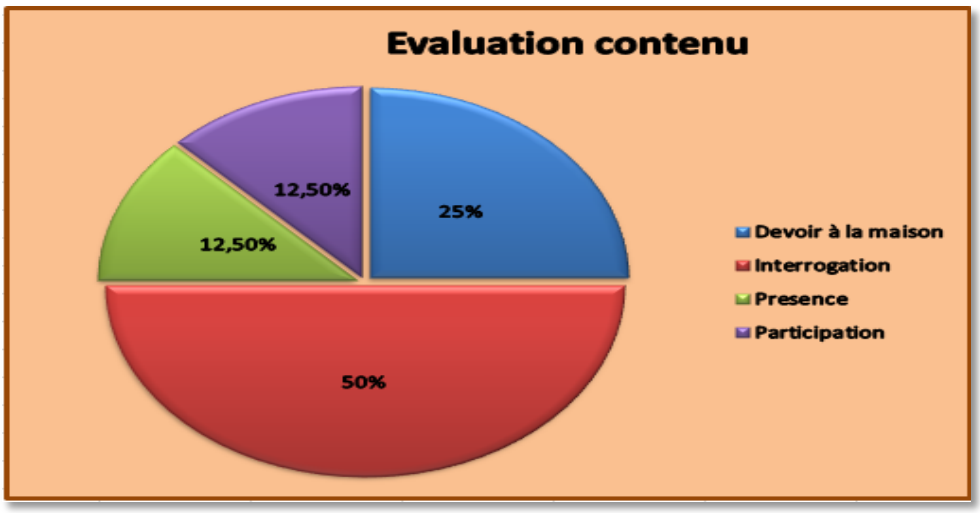


Figure 3 : Carte mentale du cours MOU

Une note finale de 10 ou plus est requise pour garantir la réussite de ce cours

7. Activités d'enseignement-apprentissage

Pour que l'étudiant assimile tous les concepts abordés pendant le cours MOU et soit capable de réaliser toutes les activités d'apprentissage demandées, l'étudiant est invité de :

- Assister régulièrement au cours ce qui lui permet de ne manquer aucun des concepts transmis.
- Prendre des notes qui vous serviront plus tard à comprendre et à réaliser les activités d'apprentissage proposées.
- Participer aux discussions tout au long de la séance afin de développer des échanges entre eux, soit par poser des questions sur la séquence pédagogique en cours, ou bien de répondre aux questions posées afin d'échanger leurs points de vue et mobiliser leurs connaissances.
- Réaliser les devoirs à faire à la maison individuellement dans le but de développer leur autonomie et leur réflexion.
- Préparer les TD et TP demandés et n'hésitez pas de participer en classe à la correction des exercices proposés.

8. Alignement pédagogique

Les compétences cibles reposent sur les trois piliers : savoir ; savoir-faire et savoir être. Afin d'atteindre tous ces piliers, des méthodes diffèrent sont proposées. Aussi, des évaluations continu et final sont effectuées pour tester la compréhension de l'étudiant et leur niveau d'assimilation à ces concepts.

- Le premier pilier « les savoir » seront atteints au niveau de cours. Cette méthode transmissive est très actif et plus efficace pour diffuser des connaissances et des concepts théoriques. La simple connaissance de ces notions ne suffit pas, il faut aussi comprendre et mémoriser ces concepts. Pour tester la compréhension de l'étudiant, des questions théoriques ainsi que des quiz en ligne à la fin de chaque chapitre sont présentés.
- Le savoir-faire consiste en l'application des concepts de l'approche objet et de la notation UML pour concevoir des architectures logicielles cohérentes. Différents exercices de modélisation et études de cas sont mis à votre disposition en TD afin de s'entraîner à analyser des besoins métier et à élaborer manuellement les diagrammes correspondants. Par la suite, l'étudiant concrétisera ces concepts théoriques via l'écriture de programmes en langage Java, ce qui lui permettra de saisir le lien entre le modèle abstrait et le code exécutable. Enfin, il passera à un stade professionnel en transférant les modèles conçus sur papier vers l'outil de modélisation Visual Paradigm, maîtrisant ainsi l'ingénierie logicielle assistée par ordinateur.
- Le savoir être permet à l'étudiant de développer son esprit d'analyse et sa capacité d'abstraction. Au-delà de la technique, ce cours vise à acquérir une rigueur conceptuelle indispensable pour structurer des projets complexes. Cette aptitude à modéliser la pensée sera un atout majeur pour ses futures collaborations professionnelles et sa capacité à communiquer avec les différentes parties prenantes d'un projet.

9. Modalités de fonctionnement

Le cours MOU est organisé en trois types de séances :

- **Cours** : c'est des séances théoriques afin de vous transmettre l'ensemble des savoirs permettant de cerner rapidement les définitions et les concepts de bases et les méthodes de travail à suivre.
- **TD** : ce type de séance consiste aux des travaux dirigés, présents après chaque unité d'apprentissage (chapitre), afin que vous puissiez mobiliser les savoirs dans la résolution des exercices et des problèmes proposés.
- **TP** : Ces séances visent à mobiliser et concrétiser les compétences acquises en programmation et en conception. Le travail s'articule autour de deux axes : l'implémentation des concepts orientés objet en langage Java pour valider la logique technique, et l'utilisation d'un atelier de génie logiciel (Visual Paradigm) pour formaliser proprement les diagrammes et solutions élaborés lors des séances de TD.

Le déroulement du cours est assuré en hybride, en présentiel (en classe) et à distance via la plateforme d'enseignement à distance qui va vous permettre, le cas échéant, de revoir ou d'approfondir les concepts vus en présentiel et de surmonter les difficultés rencontrées. C'est une condition pour pouvoir contribuer positivement à la réalisation des activités collectives et de réaliser les activités à distance dans les limites des échéances communiquées.

10. Ressources d'aide

Mise à disposition des ressources d'aide utile à l'étudiant sur la plateforme d'enseignement :

- Environnement de développement Java : Les liens officiels pour télécharger et installer le kit de développement (JDK) et l'IDE sur votre ordinateur personnel (Windows, Mac ou Linux).
- Tutoriels d'installation : Une sélection de vidéos explicatives guidant pas à pas les étapes de configuration de votre environnement de travail.
- Documentation Visual Paradigm : Le manuel d'utilisation et les guides de référence du logiciel. Cette ressource est indispensable pour la réalisation des travaux pratiques de modélisation.

Chapitre 01 : L'approche Orientée Objet

1. Introduction

La conception d'un système d'information est une étape critique. Historiquement, l'informatique s'est d'abord focalisée sur les algorithmes (le "comment faire"), mais avec l'augmentation exponentielle de la complexité des logiciels et des volumes de données, cette approche a montré ses limites.

Aujourd'hui, pour construire des systèmes robustes, évolutifs et capables de gérer des règles métier complexes, nous avons besoin d'un changement de paradigme. Nous passons d'une vision centrée sur les traitements (approche fonctionnelle) à une vision centrée sur les données et leurs interactions : c'est l'Approche Orientée Objet (AOO).

2. Objectifs du Chapitre 01

À la fin de ce chapitre, l'étudiant sera capable de :

I. Définir les concepts fondamentaux de l'approche objet, notamment les notions de classes et d'objets.
II. Expliquer les principales propriétés de l'approche objet (encapsulation, abstraction, héritage, polymorphisme.
III. Appliquer les concepts de classes et d'objets pour modéliser une situation simple.
IV. Analyser le rôle et l'apport des méthodes orientées objet dans le développement des systèmes informatiques.
V. Évaluer l'importance de l'approche objet dans la conception et l'évolution des systèmes d'information.

3. Les classes et les objets

Pour comprendre l'approche objet, il faut d'abord définir ses deux piliers fondamentaux : l'objet et la classe.

3.1 Notion d'Objet (L'entité concrète)

Dans l'approche Orientée Objet, le logiciel est considéré comme une collection d'objets autonomes qui collaborent. Les objets sont des entités autonomes qui collaborent afin de réaliser un projet global.

Un objet représente une entité du monde réel (ex: une voiture précise, un étudiant précis). Tout objet est caractérisé par un triplet fondamental (Identité, Attributs, méthodes (Comportement)) :

a. ***L'Identité (OID) :*** Concept technique. Chaque objet possède une identité unique qui le distingue des autres (souvent son adresse mémoire), indépendamment de ses valeurs.
b. ***Les Attributs (L'État) :*** Ils s'agissent des données caractérisant l'objet à un instant *t*. *Exemple (Objet Personne) :* Mohamed, 25 ans.
c. ***Le Comportement (Les Méthodes)*** : L'ensemble des opérations que l'objet peut effectuer. Ce sont les seules opérations capables de modifier l'état de l'objet. *Exemple (Objet Voiture) :* Démarrer(), Accélérer().

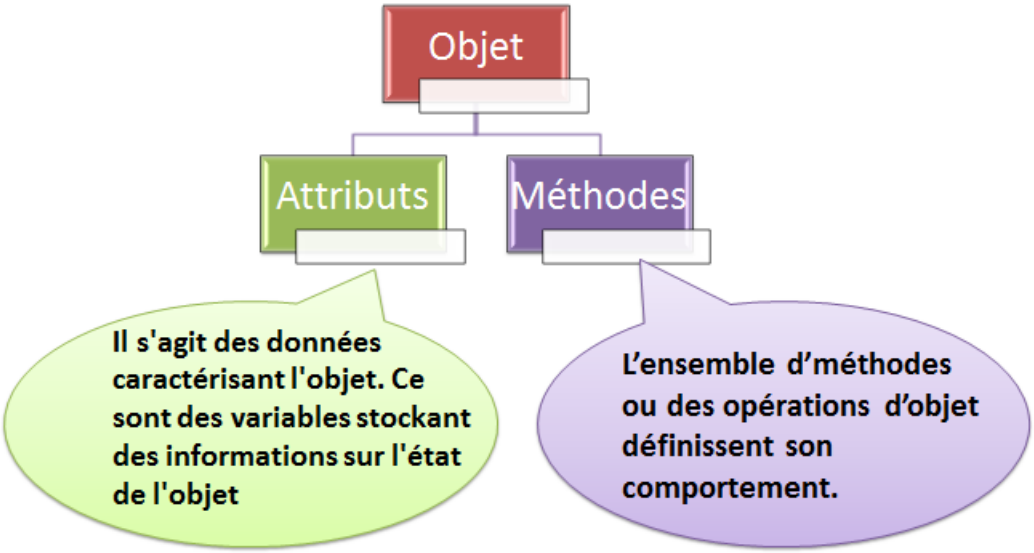


Figure 1: un objet.

Exemple 01:

Quelle sont les attributs et les méthodes de cet objet?

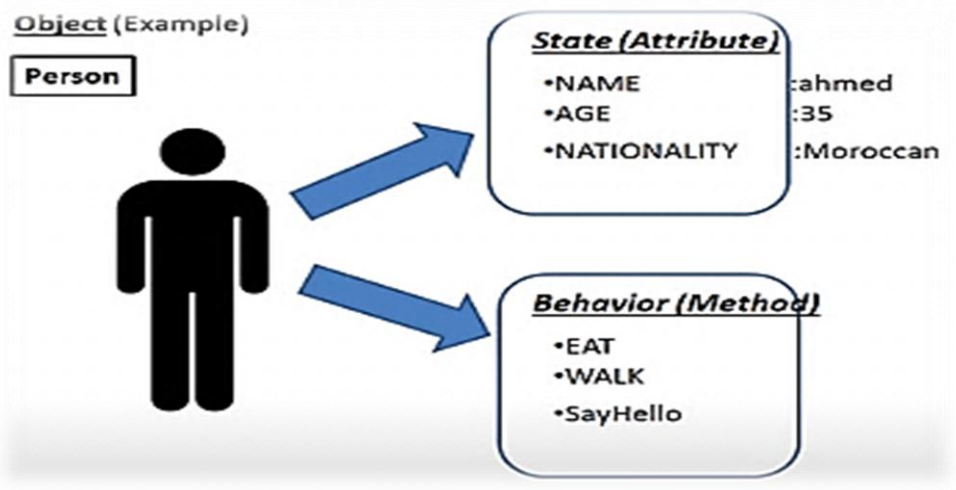


Figure 2: Exemple (Objet Personne).

Exemple 02:

Quelle sont les attributs et les méthodes de cet objet?

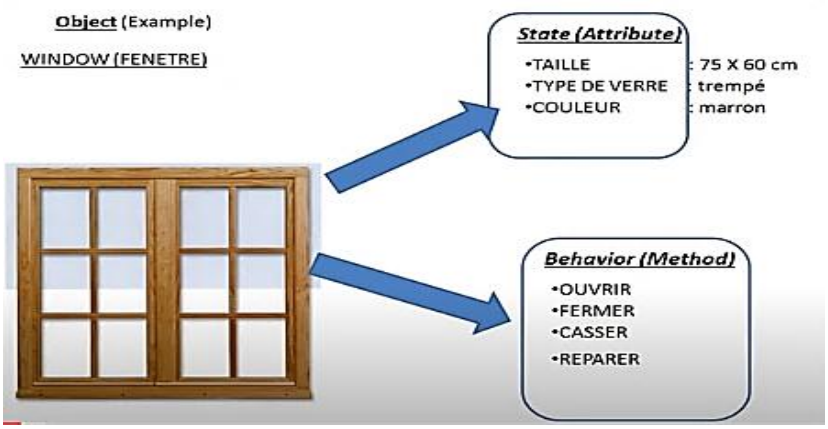


Figure 3 : Exemple (Objet Fenêtre).



Prenez une minute dès maintenant pour observer les objets du monde réel qui font partie de votre vie. Pour chaque objet, posez-vous deux questions :
« Dans quels états cet objet peut-il se trouver ? » et
"Quel comportement possible cet objet peut-il effectuer ?

3.2 Notion de Classe (Le moule)

Une classe regroupe l'ensemble des objets ayant la même structure et le même comportement.

- La classe représente l'abstraction de ses objets (le plan de construction).
- L'objet est une instance d'une classe.
- La création d'un objet se fait via une méthode spéciale : le constructeur.

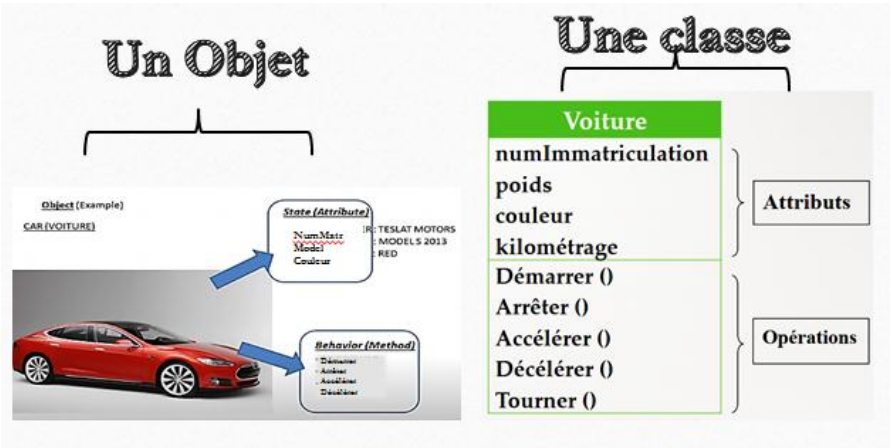


Figure 4 : Exemple (classe Voiture)

Important **:** Un objet est une instance d'une et une seule classe.

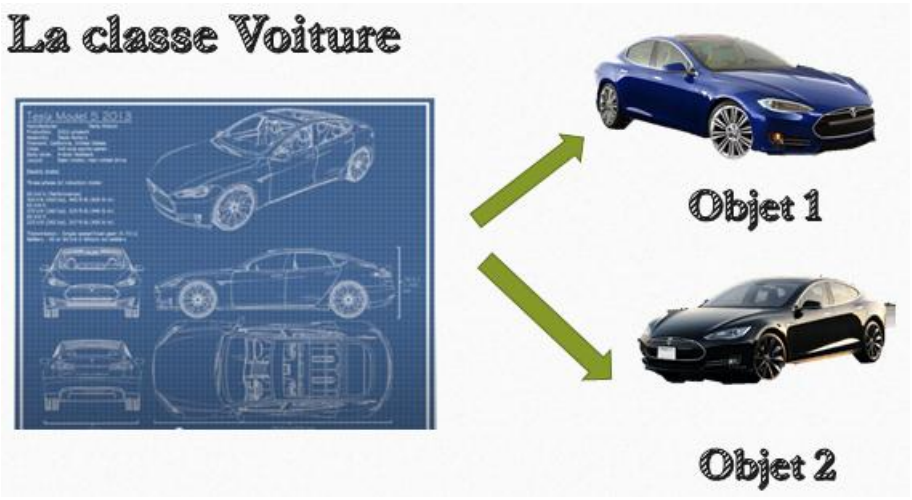


Figure 5 : Deux objets de la classe : Voiture.

4. Propriétés de l'approche objet

L'approche objet repose sur quatre concepts clés (piliers) :

- ✓ L'abstraction
- ✓ L'encapsulation
- ✓ L'héritage
- ✓ Le polymorphisme

4.1 L'Abstraction

L'abstraction est l'un des concepts clés dans les langages de programmation orientée objet (POO). Son objectif principal est de gérer la complexité en masquant les détails inutiles à l'utilisateur. Cela permet à l'utilisateur d'implémenter une logique plus complexe sans comprendre ni même penser à toute la complexité cachée. (ex: ignorer la couleur des yeux d'un client dans un SI bancaire).

Ces abstractions facilitent beaucoup la gestion de la complexité en les divisant en petites parties. Dans le meilleur des cas, vous pouvez les utiliser sans comprendre comment ils fournissent les fonctionnalités. Et cela vous aide à diviser la complexité de votre logiciel en parties contrôlable.

4.2 L'Encapsulation

L'encapsulation est le mécanisme qui rassemble les données et les méthodes au sein d'une même structure (la classe) et qui en restreint l'accès.

- Principe de la "Boîte Noire" : On masque l'implémentation interne.
- Visibilité : On protège les attributs (souvent privés) et on y accède via des méthodes publiques (getters/setters). Cela garantit l'intégrité des données.

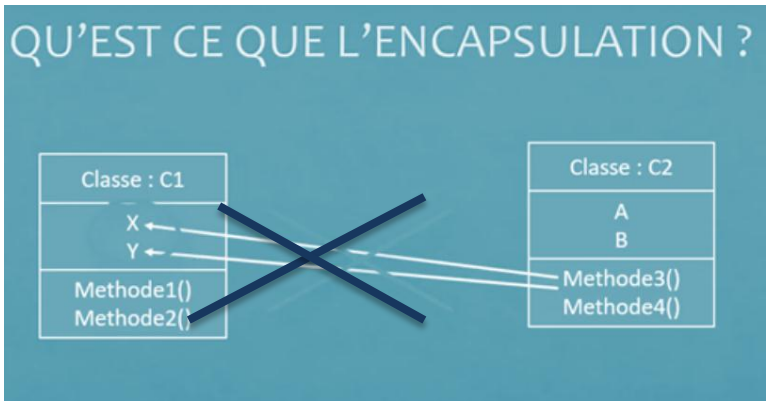


Figure 6 : L'Encapsulation.

⇨ **Droits d'accès dans l'encapsulation**

1. Public : accessible partout, à l'intérieur et à l'extérieur de la classe.
2. Private (privé) : accessible seulement à l'intérieur de la classe.
3. Protected (protégé) : accessible dans la classe et ses sous-classes seulement.

4.3 L'Héritage

C'est un mécanisme permettant de définir une nouvelle classe (fille) à partir d'une classe existante (mère). La classe fille hérite des attributs et méthodes de la mère.

- Favorise la réutilisation du code.
- Établit une relation "EST-UN" (Une Moto EST UN Véhicule).

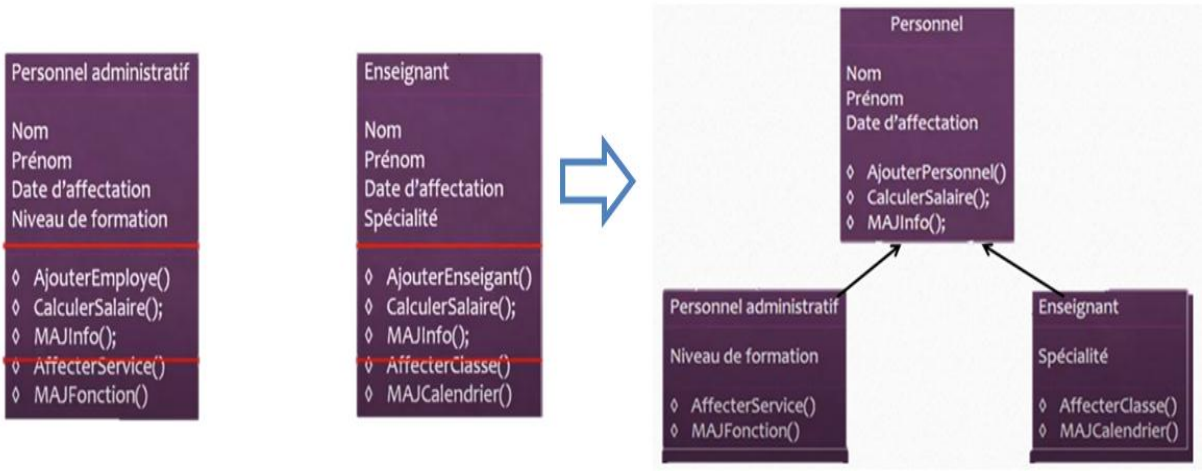


Figure 7 : Exemple d'héritage.

```java
public class enseignant extends personnel{
    private String specialite;

    public enseignant(String NomE, String PrenomE,
    String dateAffectationE, String specialiteE) {
        super(NomE, PrenomE, dateAffectationE);
        specialite = specialiteE;
    }
```

Figure 8 : L'héritage (code java).

4.4 Le Polymorphisme

C'est la capacité d'objets de classes différentes de répondre au même message (méthode) mais d'une manière spécifique.

- *Exemple :* La méthode CalculerTaxe() sera différente pour un produit local et un produit importé, mais le système les appellera de la même façon.

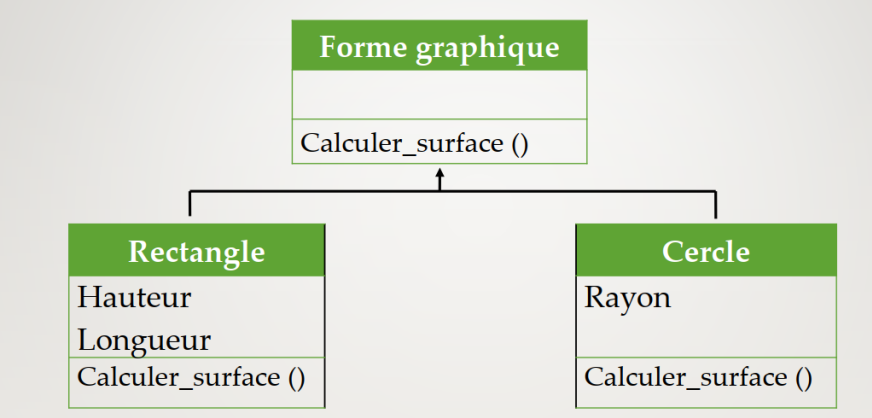


Figure 9 : Le Polymorphisme

5. Les méthodes orientées objets

Il est important de situer l'approche objet par rapport aux méthodes classiques.

5.1 Évolution des méthodes

1. Méthodes Fonctionnelles (1ère génération) : Décomposition de l'application en sous-applications. Approche Descendante (Top-Down).
2. Méthodes Systémiques (2nde génération - ex: Merise) : Séparation stricte des Données et des Traitements. Approche Descendante.

3. Méthodes Orientées Objets : Fusion des données et traitements. Approche Ascendante (Bottom-Up) : on construit les briques (objets) puis on les assemble.

5.2 Comparaison : Procédurale vs OO

- **Approche Procédurale** : Dissocie les données des fonctions. Si la structure des données change, tout le code doit être révisé.
- **Approche OO** : Les données sont protégées dans l'objet. Si la structure interne change, cela n'impacte pas le reste du système (grâce à l'encapsulation).

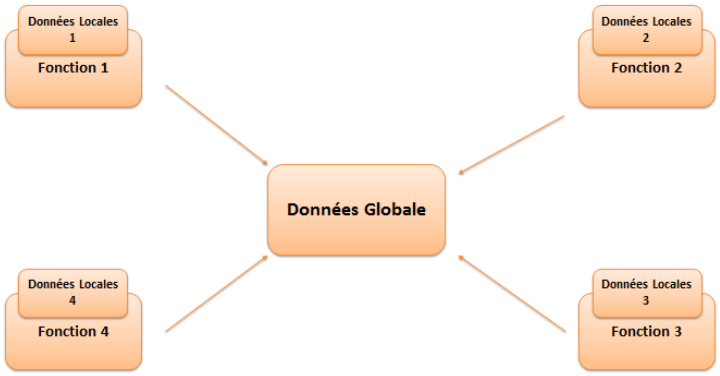


Figure 10 : Approche Procédurale.

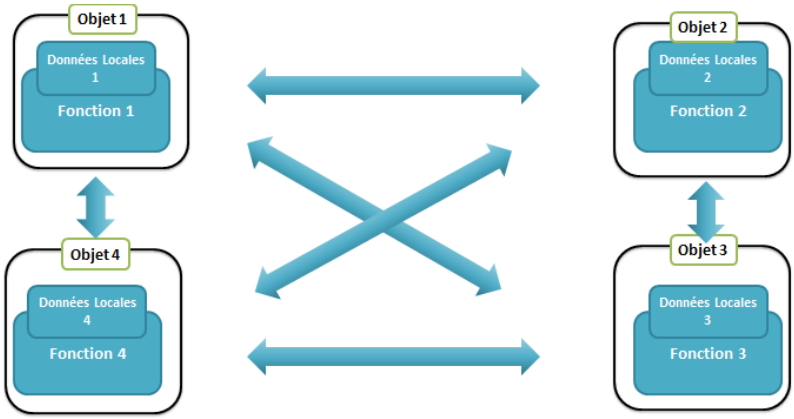


Figure 11: Approche OO.

5.3 Avantages de l'approche orientée objets

- ✓ Reflète plus finement les objets du monde réel (Basés sur le monde réel)
- ✓ Facile à maintenir
- ✓ Réutilisation des composantes
- ✓ Simplifier le passage conceptuel/physique
- ✓ Conduit à des modèles plus stables
- ✓ Structure indépendante des fonctions
- ✓ Encapsule la complexité

6. L'approche objet dans les Systèmes d'Information (SI)

L'intégration de l'approche objet a transformé la manière de concevoir, de déployer et de maintenir les Systèmes d'Information modernes. Elle répond aux limites des approches systémiques classiques (comme Merise) face à la complexité croissante des besoins métiers.

6.1 La Réduction du "Fossé Sémantique"

Le principal défi d'un SI est de traduire un besoin métier (exprimé en langage naturel par des experts fonctionnels) en langage informatique (exprimé en code par des développeurs).

- Approche Classique : Il y a une rupture. On transforme des objets réels (Clients, Commandes) en tableaux de données d'un côté et en procédures de l'autre. Le modèle informatique ne ressemble pas au modèle réel.
- Approche Objet : Elle assure une continuité. Une classe Facture dans le code correspond exactement au concept de "Facture" dans le métier.
    - *Conséquence :* Le dialogue entre la maîtrise d'ouvrage (le métier) et la maîtrise d'œuvre (l'IT) est simplifié, car ils parlent le même langage.

6.2 La Stabilité face au Changement (Critère décisif)

Un Système d'Information doit durer dans le temps (10, 15 ans ou plus), mais les règles de gestion changent tout le temps (nouvelles lois, nouvelles stratégies marketing).

- L'instabilité des traitements : Les *processus* changent souvent. (Ex : "La méthode de calcul de la TVA change"). Si le SI est basé sur les fonctions (approche fonctionnelle), chaque changement de règle risque de casser l'architecture.
- La stabilité des objets : Les *entités* changent peu. (Ex : Un "Client" restera toujours un Client, avec un nom et une adresse, quelle que soit la loi fiscale).
    - *Conclusion :* En bâtissant le SI sur des objets (structure stable) plutôt que sur des fonctions (structure instable), on garantit la pérennité du système.

6.3 Gestion de la Complexité et Modularité

Les SI modernes sont gigantesques (ERP, CRM, SCM interconnectés). Il est impossible de les concevoir comme un bloc monolithique.

1. Composants Réutilisables : L'objet permet de créer des "briques" logicielles autonomes (Composants). Une fois la classe GestionPaiement validée, elle peut être réutilisée par le module "Vente", le module "Comptabilité" et le module "RH" sans être réécrite.
2. Maintenance facilitée : Grâce à l'encapsulation, si un bug survient dans la gestion des stocks, on sait exactement dans quel objet chercher. On peut corriger l'intérieur de l'objet Stock sans paralyser le reste du SI.

Chapitre 02 : Notations UML

1. Introduction

Ce chapitre introduit UML (Unified Modeling Language), le standard industriel de l'OMG pour la modélisation logicielle. Nous y aborderons son historique, sa nature en tant que langage graphique (et non comme méthode), ainsi que l'architecture globale de ses diagrammes classés en vues statiques et dynamiques.

2. Les objectifs de chapitre 02

A la fin de chapitre, vous serez capable de :

I. Se familiariser avec les forces de la COO en les comparants avec les Approches procédurales.
II. Identifier la notation UML à travers une discussion sur l'historique de la modélisation
III. Classer les diagrammes UML en se basant sur la vue de système
IV. Découvrir la différence entre les digrammes structurels et les diagrammes comportementaux en comprenant le principe de chacun d'eux.

3. Cycle de vie d'un logiciel

- Expression des besoins
  - Définir le cahier des charges
- Spécification
  - Ce que le système doit être et comment il peut être utilisé
- Analyse
  - Éléments intervenant dans le SI, leurs structures et relations
  - A définir sur 3 axes
    - Savoir-faire de l'objet => axe fonctionnel
    - Structure de l'objet => axe statique
    - Cycle de vie de l'objet => axe dynamique
- Conception
  - Apport de solutions techniques : architecture, performance et optimisation
  - Définition des structures et des algorithmes
- Implémentation
  - Réalisation et programmation
- Tests et Vérification
  - Contrôles de qualité
- Vérification de la correspondance avec le cahier des charges

- Maintenance et Evolution
  - Maintenance corrective : traiter les erreurs (bugs)
  - Maintenance évolutive : intégration de nouveaux changements



Figure 1 : Cycle de vie d'un logiciel.

4. CONCEPTION

- Phase essentielle qui consistant à définir précisément chaque sous-ensemble du logiciel.
- Permet de représenter les différentes fonctions du système.
- Construction des programmes qui permet l'exécution de ces fonctions.
- Une « bonne » conception se définit en termes de la satisfaction des besoins et des spécifications.
- Une bonne Conception participe largement à la production d'un logiciel qui répond aux facteurs de qualité.

5. La modélisation

La programmation d'une application nécessite une organisation des idées, une documentation et en fin la réalisation. Cela peut être accompli en définissant les modules et étapes de la réalisation. C'est cette démarche antérieure à l'écriture que l'on appelle modélisation ; son produit est un modèle.

6. Un modèle

Un modèle est une simplification de la réalité. C'est une représentation abstraite et simplifiée d'une entité du monde réel en vue de le décrire, de l'expliquer ou de le prévoir.

En informatique, la modélisation consiste à :

- décrire un problème (Analyse)
- décrire la solution de ce problème (conception)

7. Pourquoi modéliser ?

- Aider à visualiser le système afin de mieux comprendre son fonctionnement ou comment il devrait l'être.
- Communiquer avec le client et valider le modèle à établie.

- Spécifier les structures de données et le comportement de système.
- Pérenniser un savoir-faire (des projets peuvent durer des années)
- Répartir les tâches et d'automatiser certaines d'entre elles.
- Fournir un plan pour la construction effective
- Réduire la complexité et assurer sa cohérence
- Permettre de documenter les choix
- Réduire les couts de développement
- Réduire les délais de mise en production

Remarque

- Les meilleurs modèles sont liés à la réalité.
- Un seul modèle n'est jamais suffisant

8. L'apparition de l'UML

Plusieurs méthodes de modélisation sont apparues entre 1990 et 1995, mais aucune ne parvient à s'imposer. En 1994, le consensus se fait autour de trois méthodes : OMT (Object Modeling Technique) de James Rumbaugh et OOD (Object Oriented Design) de Grady Booch, OOSE (Object Oriented Software Engineering) de Ivar Jacobson pour créer un langage de modélisation unifié (UML : Unified Modeling Language (Langage de Modélisation Unifié)).

En 1997, la norme UML est définie comme standard de modélisation des systèmes d'information objet par l'OMG (Object Management Group).

UML est un langage de modélisation orientée objet et non une méthode, car il est indépendant du procédé et de la méthode de développement utilisés.

9. Diagrammes UML

Un diagramme est une représentation visuelle de l'ensemble des éléments qui constituent le système. Un diagramme permet de fournir une vue partielle du système. Pour obtenir une vue globale du système, différents diagrammes doivent être réunis où chaqun possède sa propre vue (spécificité) du système.

La 2éme version d'UML propose 14 diagrammes pour modéliser un système. Selon la vue que l'on veut décrire, structurels ou comportementaux

- ***Diagrammes structurels ou diagrammes statiques (UML Structure)*** : la vue structurelle regroupe six digrammes qui permettent de représenter l'aspect statique d'un système (classes, objets, composants…).
- ***Diagrammes comportementaux ou diagrammes dynamiques (UML Behavior)*** : pour représenter la vue dynamique d'un système, huit diagrammes sont proposés. Le tableau ci-dessous présente ces diagrammes :

Tableau 01 : Les diagrammes UML

Diagrammes structurels	Diagrammes dynamiques
Diagramme de classe	Diagramme des cas d'utilisation
Diagramme d'objet	Diagramme d'états-transitions
Diagramme de composant	Diagramme d'activités
Diagramme de	Diagramme de séquence
déploiement	Diagramme de communication
Diagramme de paquetage	(anciennement appelé collaboration)
Diagramme de structure composite	Diagramme d'interaction global
Diagramme de profils	Diagramme de temps

Chapitre 03 : Les diagrammes UML structurels

1. Introduction

UML offre un ensemble de diagrammes (14 diagrammes) pour la modélisation du système selon différents vues. On peut regrouper ces diagrammes en deux grands groupes : diagrammes structurels ou diagrammes statiques et diagrammes comportementaux ou diagrammes dynamiques.

Les diagrammes structurels permettent de modéliser la structure statique du système à développer. Six diagrammes sont utilisés pour la modélisation de cet aspect de système, chacun offrant une vue différente de ses objets et de leurs relations.

2. Les objectifs de chapitre 03

À la fin de ce chapitre, l'étudiant sera capable de :

Identifier et nommer les diagrammes UML dédiés à la vue statique.

Expliquer les concepts fondamentaux de la modélisation structurelle.

Traduire des règles de gestion métier en éléments graphiques (classes, attributs, opérations).

Concevoir un Diagramme de Classes complet et cohérent à partir d'un cahier des charges.

3. Diagramme de classe

Dans les méthodes d'analyse et conception orientées objet, le diagramme de classes est considéré comme le diagramme le plus important. C'est le point central de tout développement orienté objet.

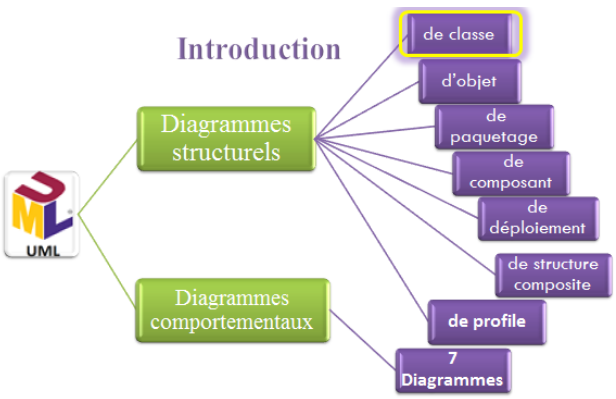


Figure 1 : Diagramme de classe.

3.1. Définitions

Classe :

- La classe est une définition abstraite qui permet de définir la structure et le comportement d'un ensemble d'objets de même nature.

- On peut dire que la classe est le type de l'objet
- Représenter par un rectangle divisé en trois compartiments principaux :

NomClasse
Attributs
Les opérations

Figure 2 : Une classe.

Attribut

- une propriété structurelle d'une classe
- Un attribut peut être un attribut de classe. Sa valeur est partagée par toutes les instances de cette classe. L'attribut de classe doit être souligné.
- Un attribut peut être dérivé (calculé) qui est calculé à partir d'autres attributs. L'attribut dérivé est procédé par (/).
- La syntaxe d'un attribut est :
  [Visibilité] Nom [multiplicité] [type] [=valeur initiale {propriété}]

  - Visibilité : les droits d'accès à cet attribut (+ : public) (# proteceted) (- private) (~ paquetage)
  - Nom d'attribut : représente un nom unique dans sa classe.
  - Multiplicité : la cardinalité (un attribut peut avoir des valeurs multiples)
  - Type : Uml définit ses propres types (Integer, real, String…). Les attributs peuvent être des types références (objet..).
  - valeur initiale : l'attribut peut contenir une valeur par défaut
  - {propriété} : valeurs marquées facultatives

Opération

- Une fonction (méthode) appartient à une classe.
- L'opération décrire le comportement d'un objet.
- L'opération de type classe accéder seulement aux attributs classe. Elle doit être aussi soulignée.
- La syntaxe d'une opération est :

[Visibilité] Nom (paramètres) :type_résultat

  - Visibilité : les droits d'accès à cette opération (+ : public) (# proteceted) (- private) (~ paquetage)
    - Nom : généralement est nommé par un verbe décrivant l'action à accomplir
    - Paramètre : l'ensemble de paramètre de cette opération (Genre : nom : type)
    - type_résultat : le type des valeurs à retourner par cette opération

Etudiant
Id : Integer {id} Nom : String Prénom [3] : String Age : Integer
+getNom() :string +participerCours() +changerGroupe()

Figure 3 : Une classe Etudiant.

3.2. Les relations entre les classes

Il existe sept types de relations de classe, à savoir l'association, association dirigée, dépendance, agrégation, composition, généralisation, réalisation.

3.2.1 L'association

- Une connexion sémantique bidirectionnelle (ou bien unidirectionnelle) entre classes.
- Une association peut être identifiée par son nom.
- Le sens de lecture peut être précisé par une flèche < >
- Une association est représentée par lien qui est considéré comme un « tube à message »
- Un lien stable est une instance d'une association
- Un rôle peut être ajouté de part et d'autre de l'association pour améliorer la lisibilité.

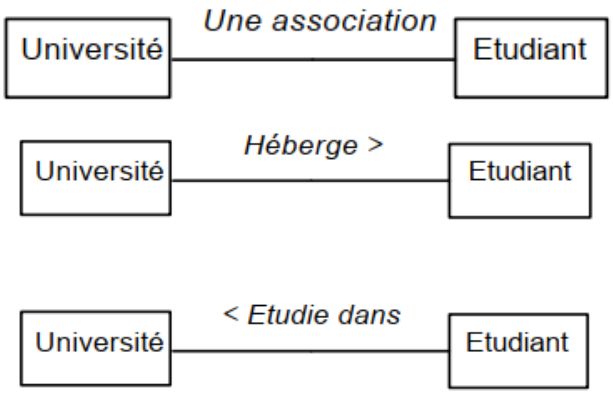


Figure 4: Exemple 1 d'une association.

3.2.2 Représentation de l'association

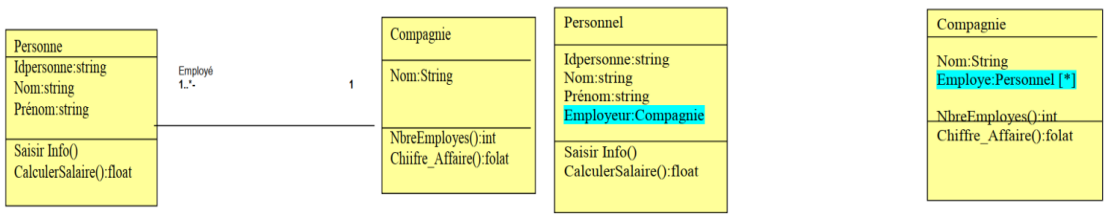


Figure 5 : *Représentation de l'association.*

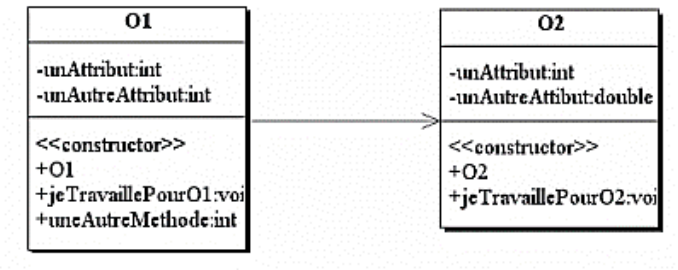


Figure 6 : Exemple 2 d'une association.

```
class O1 {
private int unAttribut;
private int unAutreAttribut ;
private O2 lienO2; // réalise l'association avec la classe O2
public O1(O2 lienO2) /* Le constructeur prévoit de recevoir un référent vers un objet de
classe 02 */{
this.lienO2 = lienO2;
}
public void jeTravaillePourO1() {
lienO2.jeTravaillePourO2() ; /* Voici l'envoi de message */
}
public int uneAutreMethode(int a){
return a;
}
}
class O2 {
private int unAttribut ;
private double unAutreAttibut ;
public O2() {}
public void jeTravaillePourO2() {
System.out.println("Je suis au service de toutes les classes ") ;
}
}

public class UML 2{
public static void main(String[] args){
O2 unObjet2 = new O2();
O1 unObjet1 = new O1(unObjet2) ;
/* on passe dans le constructeur de l'objet O1 le référent de l'objet O2 */
unObjet1.jeTravaillePourO1();
/* un premier message envoyé par le main à l'objet de classe O1 en déclenchera un autre
vers un objet de classe O2 */
}
}
```

Résultat

Je suis au service de toutes les classes.

3.2.3 La cardinalité

- On peut dire aussi la multiplicité
- c'est le nombre minimum et maximum de participation d'une instance d'une classe à l'association

1 / 1..1	Un et un seul (min = max=1).
0..1	zéro ou un (min =0 et max=1).
m / m.m	m (min=max=m≠0)
1,n,m	Un ou n ou m / m,n sont des entiers positifs
m..n	De m à n (min=m, max=n'existe pas / m,n sont des entiers positifs)
* / 0..*	De zéro à plusieurs (min=0, max=n)
1..*	De un à plusieurs (min=1, max=n)

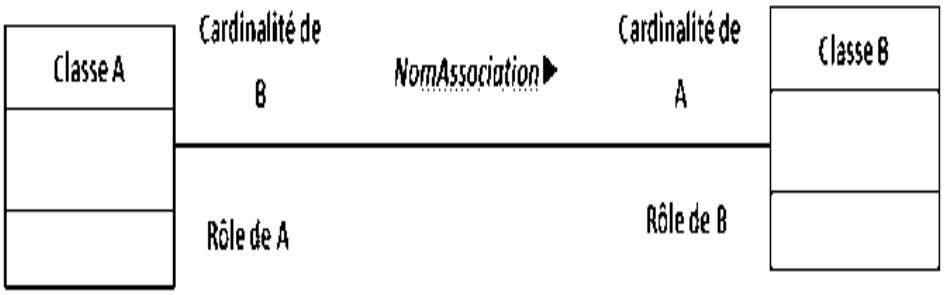
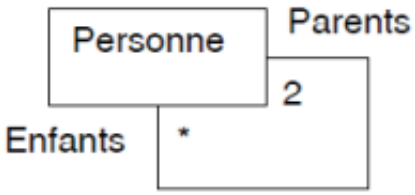


Figure 7 : La cardinalité.

3.2.4 Association réflexive

- une association réflexive consiste à lier des objets de la même classe



- Figure 8 : Exemple d'une association réflexive.

3.2.5 Association N-aires

- Une relation entre plus de deux classes.
- Ce type de relation est utilisé lorsqu'on veut éviter de définir une classe supplémentaire

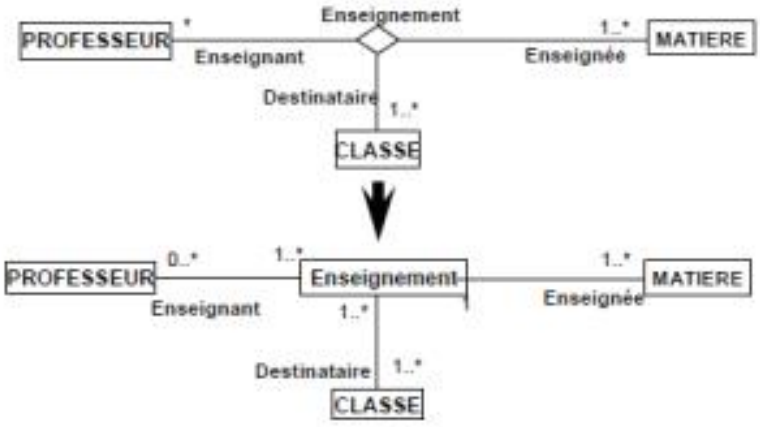


Figure 9 : Association N-aires

3.2.6 Classe Association

- Lorsque l'association contient de l'information
- Une classe association est une classe contient des attributs qui appartiennent à la relation elle-même et non à l'un des classes.

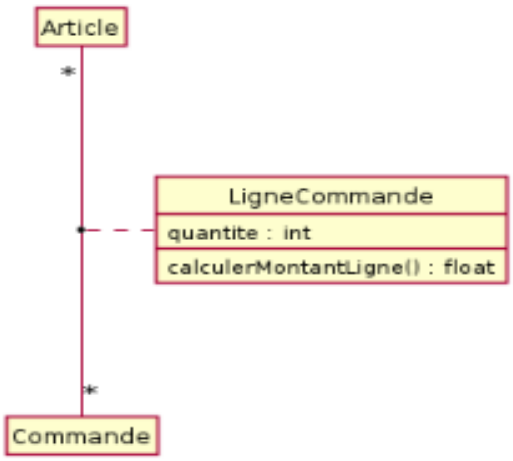


Figure 10 : Exemple d'une classe association.

3.2.7Navigabilité

- Le moyen d'accéder aux attributs et opération d'autres classes
- Une association peut fonctionne de manière unidirectionnelle ou bidirectionnelle
- La navigabilité est représentée graphiquement par une flèche de la terminaison navigable
- L'empêchement de la navigabilité est représenté par une croix du côté de la terminaison non navigable.
- Par défaut, une association est navigable des deux côtés (bidirectionnels).

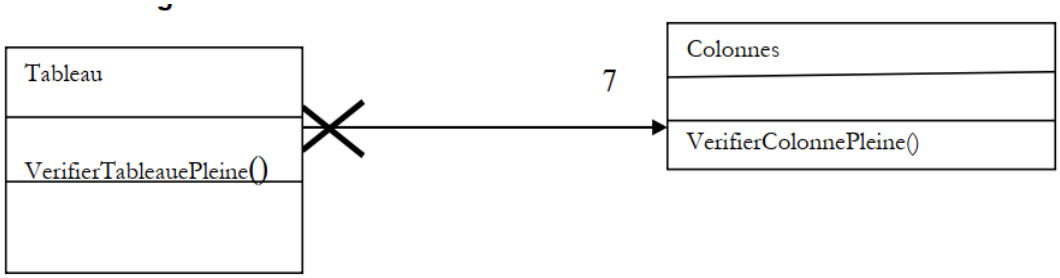


Figure 11 : Exemple d'empêchement de la navigabilité d'une association.

3.2.8 Qualificateurs

- Restriction de la cardinalité
- Identification des sous-ensembles d'instances associées lors des navigations dans les associations.
- La sélection d'une paire d'objets spécifique dans l'ensemble d'objets associés.

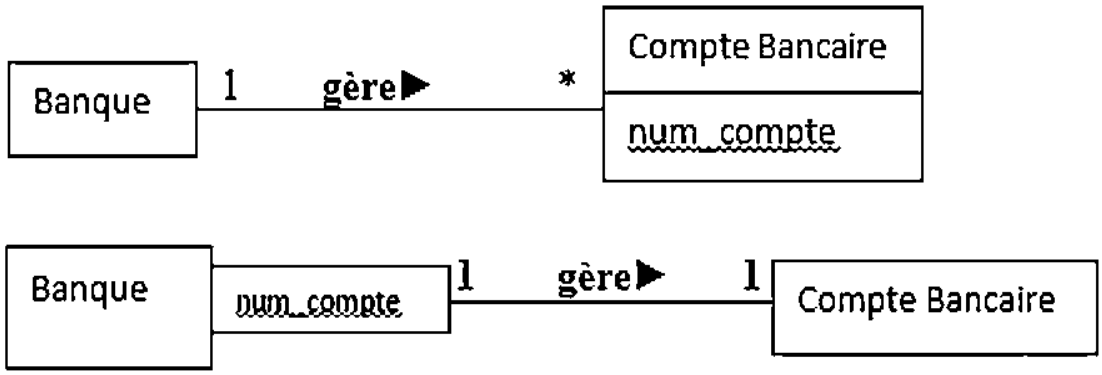


Figure 12 : Exemple de la sélection d'une paire d'objets spécifique.

3.2.9 Contraintes d'association

- Une contrainte d'association est une règle de gestion attachée à un élément du modèle.
- Une contrainte doit être exprimée entre accolades {…}
- Les contraintes peuvent être exprimé par :
  - Une Langues naturelles telles que l'anglais
  - Langages de programmation tels que Java™
  - Notations mathématiques
  - Langage OCL (Object Constraint Language)
- Des contraintes prédéfinies l'extrémité d'une association : {ordered}, {unique},
- Des contraintes prédéfinies entre deux associations : {subset}, {xor}, ...

OCL (Object Constraint Language)

C'est avec OCL (Object Constraint Language) qu'UML formalise l'expression des contraintes. Il s'agit donc d'un langage formel d'expression de contraintes bien adapté aux diagrammes d'UML, et en particulier au diagramme de classes.

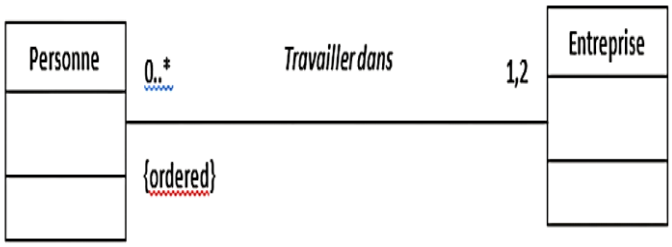


Figure 13 : Exemple 1 d'une contrainte d'association.

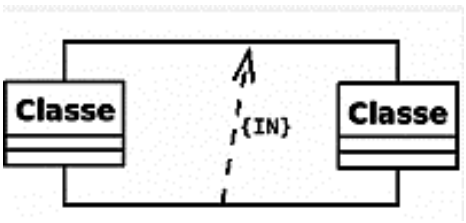


Figure 14 : Exemple 2 d'une contrainte d'association.

Dans la figure 14, Si l'association incluse est instanciée, l'autre doit l'être aussi, la contrainte d'inclusion a un sens, représenté par une flèche.

Tableau 01 : Évaluation des contraintes logiques entre deux classes.

Classe1	Classe2	IN(C1→C2)	IN(C2→C1)	AND ou S	X	OR ou T	XOR ou XT
0	0	0	0	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	1	0	1	1	1
1	1	1	1	1	0	1	0

Simultanéité {S}, également notée {=} ou {AND}

- Si une association est instanciée, l'autre doit l'être aussi.

Totalité {T}, également notée {OR}

- Au moins une des deux associations doit être instanciée.

Exclusion {X}

- Les deux associations ne peuvent être instanciées en même temps.

Partition {XT}, également notée {+} ou {XOR} ou {P}

- Exactement une des deux associations doit être instanciée.

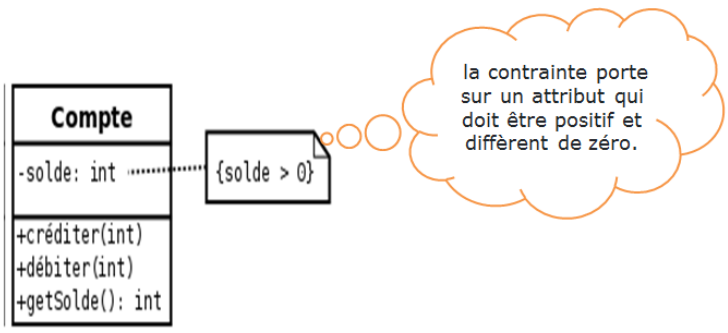


Figure 15 : Exemple 3 d'une contrainte d'association.

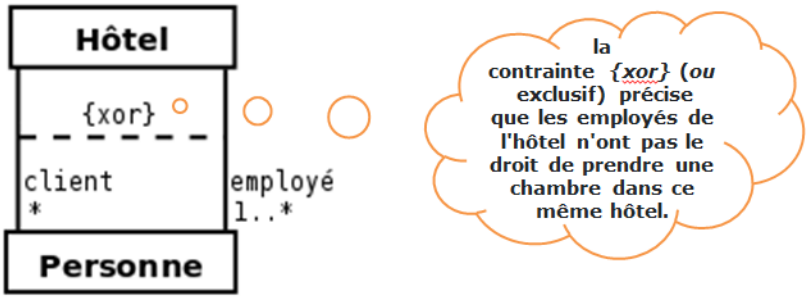


Figure 16 : Exemple 4 d'une contrainte d'association.

3.2.10 Propriétés de mise à jour de liens

- {variable} : instance modifiable (par défaut).
- {frozen} : instance non modifiable fixée lors de la création de l'objet.
- {addOnly} : instances ajoutables mais non retirables (impossible de supprimer un élément).

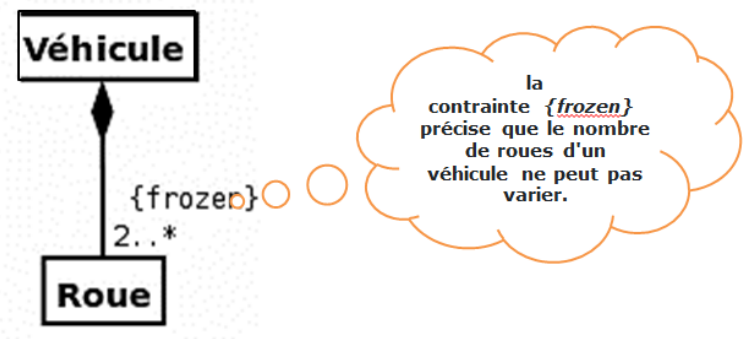


Figure 17 : Exemple d'utilisation d'une propriété « frozen ».

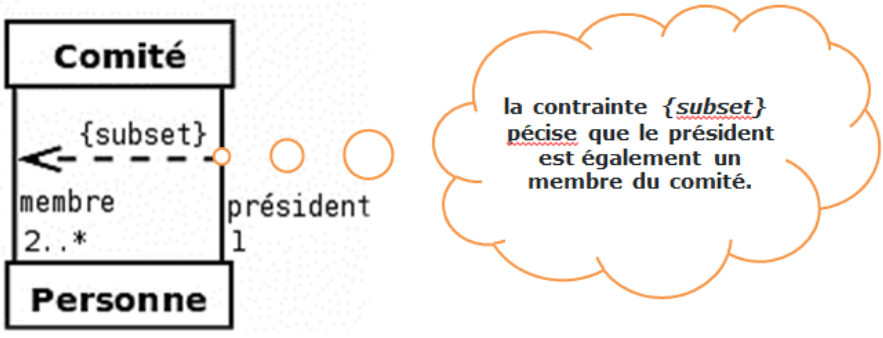


Figure 18 : Exemple d'utilisation d'une propriété « subset ».

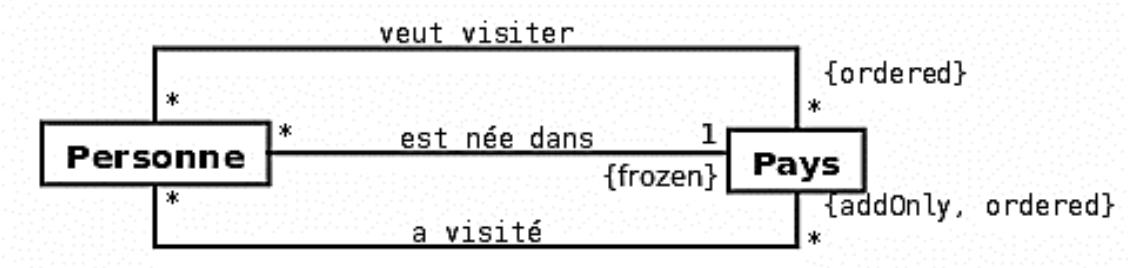


Figure 19 : Exemple d'utilisation des différentes propriétés.

Le diagramme présenté dans la figure 19 exprime que : une personne est née dans un pays, et que cette association ne peut être modifiée ; une personne a visité un certain nombre de pays, dans un ordre donné, et que le nombre de pays visités ne peut que croître ; une personne aimerait encore visiter toute une liste de pays, et que cette liste est ordonnée (probablement par ordre de préférence).

3.3. Dépendance

- L'existence d'un lien sémantique entre deux classes.
- le lien de dépendance entre classes est moins fréquemment rencontré que le lien d'association qui est plus effectif.
- Lorsque les éléments de la classe 1 sont importants pour construire la classe 2, on dit que la classe 2 est en dépendance de la classe 1.
- Le trait d'association se transforme alors en un trait en pointillés



Figure 20 : la relation de dépendance.

```
class O1 {
private int unAttribut;
private int unAutreAttribut ;
public void jeTravaillePourO1() {
        O2 lienO2 = new O2() ; /* création d'un objet local O2 */

        lienO2.jeTravaillePourO2() ; /* Voici l'envoi de message */

        }       // l'objet lienO2 disparaît
public int uneAutreMethode(int a){
return a; } }
```

3.4. Agrégation

- L'association est transformé en agrégation si la classe 2 de fait partie de la classe 1.
- Cette relation peut être expliquer en utilisant les verbes suivants: contient, est construit à partir de, est former de,,,etc

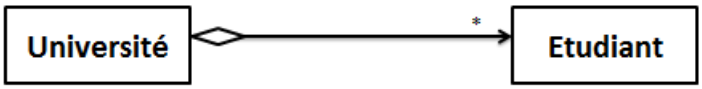


Figure 21 : la relation d'agrégation.

3.5. Composition

- L'association est transformée en Composition si la classe 2 ne fait partie que de la classe 1.
- La composition est une agrégation forte
- Le cycle de vie du composant est lié à celui du composite



Figure 22 : la relation de composition.

3.6. Généralisation/Spécialisation

Héritage:

- ✓ Une relation entre une classe générale qui est nommée « SuperClasse » avec une classe plus spécifique (sous-classe)

Généralisation:

- ✓ Regrouper les attributs et les opérations communes des classes existantes dans une nouvelle classe, (démarche ascendante)

Spécialisation:

- ✓ Regrouper les attributs et les opérations communes des classes existantes dans une nouvelle classe, (démarche descendante)

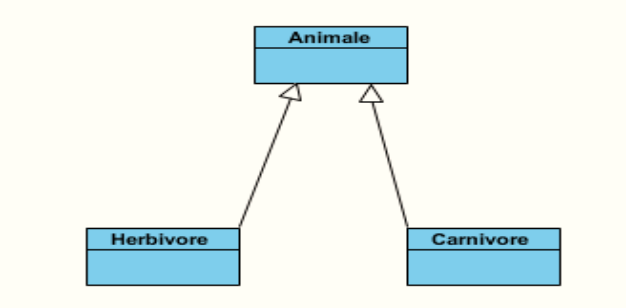


Figure 23 : Exemple d'héritage.

3.7. Héritage Multiple

Une classe peut avoir plusieurs parents (Super-Classes)

```
public class CliantSpecial extends Client, Salarie { ... }
```

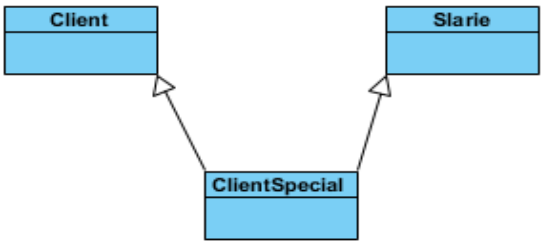


Figure 24 : Exemple d'héritage multiple.

3.8. La classe abstraite

- Contient au moins une méthode abstraite
- Une méthode abstraite ne contient pas une implémentation. Ce dernier es laissé sous-classes.
- Son objectif est de structurer le système,
- Se note en *Italique*,
- Ne peut pas être instanciée. Son constructeur peut être appelé par la classe dérivée.
- Généralement dans une relation d'héritage, la classe mère (superclasse) n'y a pas vraiment de sens d'être instanciée. Dans ce cas, on peut considérer comme une classe *abstraite*.

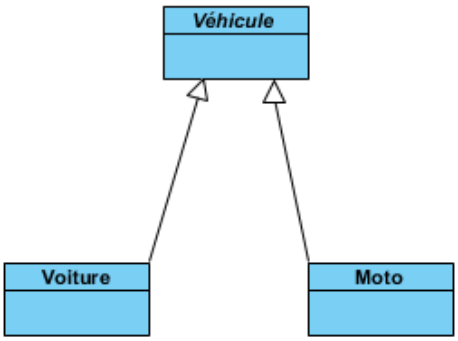


Figure 25 : Exemple d'une classe abstraite.

La classe *Vehicule* permet de faire apparaître un type à travers lequel il sera possible de manipuler des instances de *Voiture* ou de *Moto*. Il y a peu d'intérêt dans ce contexte à créer une instance de *Vehicule*.

3.9. Classe Interface

- Contient des signatures des méthodes, ce qui implique que les méthodes n'ont pas de corps.
- L'interface ne peut rien faire. C'est juste un modèle.
- En Java, plusieurs interfaces peuvent être implémentées

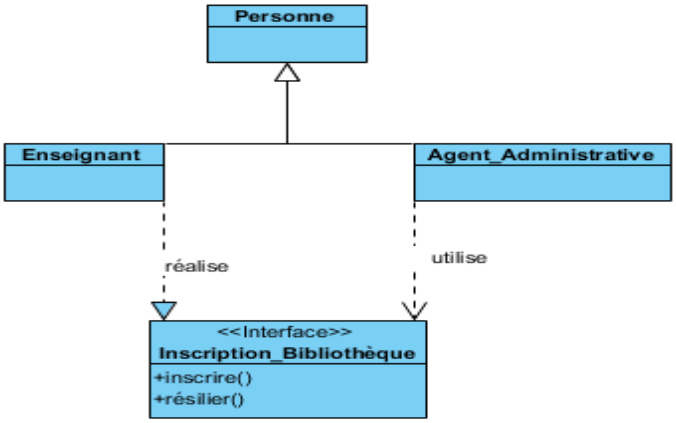


Figure 26 : Exemple d'une interface.

Tableau 02 : Comparaison entre une classe abstraite et une interface.

Interface	Classe abstraite
Ne contient pas des attributs	Contient des attributs
Ne contient pas des constructeurs	contient des constructeurs
Ne contient que des signatures (abstraites)	Contient des méthodes abstraites et des méthodes implémentées
La visibilité est par défaut public, et ne peut pas être modifié	La visibilité peut être (+ , _ , # , ~)
Les méthodes ne peuvent pas être statiques	Les méthodes implémentées peuvent être statique

4. Diagramme d'objet

- Tandis que un diagramme de classes montre une abstraction de la réalité, Un diagramme d'objets représente plutôt un cas particulier, une situation concrète à un instant donné ;
- Chaque objet représente une instance de la classe
- Les relations dans un diagramme de classes et représenté par des liens dans le diagramme d'objets.

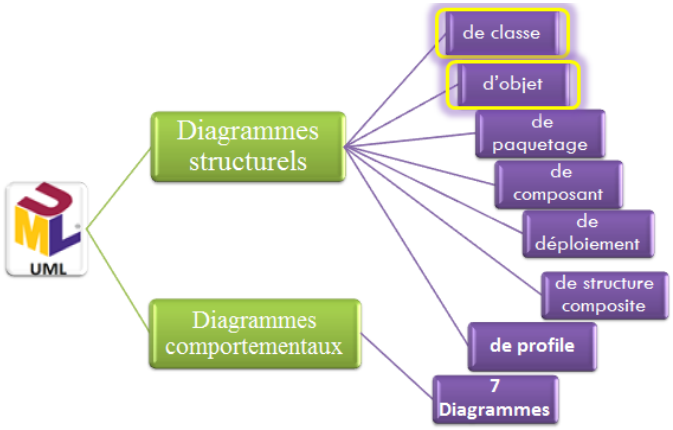


Figure 27 : Diagramme d'objets.

4.1. Pourquoi les diagrammes d'objets ?

- ✓ Permettre de vérifier si le diagramme de classe est correctement construit ou non,
- ✓ Comprendre les parties complexes d'un diagramme de classes
- ✓ Gérer les cas particulier des systèmes
- ✓ Lors de l'étape d'analyse d'un projet, il est possible de concevoir un diagramme de classes afin de représenter la configuration du système, suivi de la création d'un ensemble de diagrammes d'objets servant de scénarios de test visant à confirmer l'intégralité et l'exactitude du diagramme de classes.
- ✓ Avant d'établir un diagramme de classes, il est judicieux de créer un diagramme d'objets afin de comprendre des informations spécifiques sur des éléments de modèles particuliers, ainsi que leurs relations, ou pour illustrer des exemples concrets des attributs requis.

4.2. Elément des diagrammes d'objets

L'état d'un objet est déterminé par les valeurs de ses attributs :

- Tous les objets du diagramme de classes ne doivent pas obligatoirement figurer sur le diagramme d'objets
- Un lien se représente comme une relation, mais, s'il y a un nom, il est souligné. Naturellement, on ne représente pas les multiplicités.

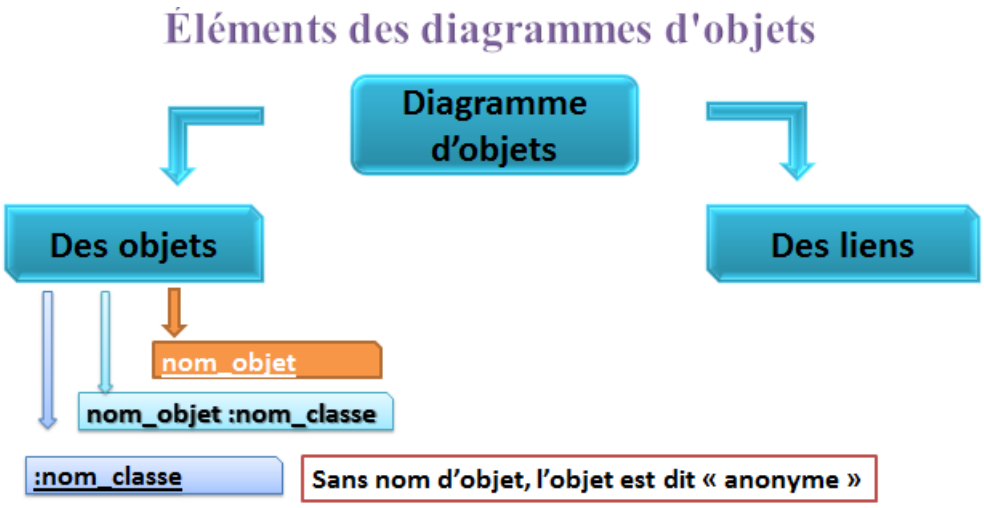


Figure 28 : Elément des diagrammes d'objets.

ATTENTION :

La multiplicité des extrémités des liens est toujours = 1

- Les rôles des associations peuvent être représentés explicitement :

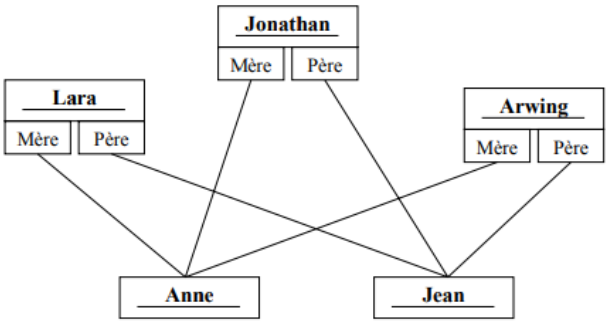


Figure 29 : Exemple des rôles dans un diagramme d'objet.

4.3. Exemples d'un diagramme d'objets

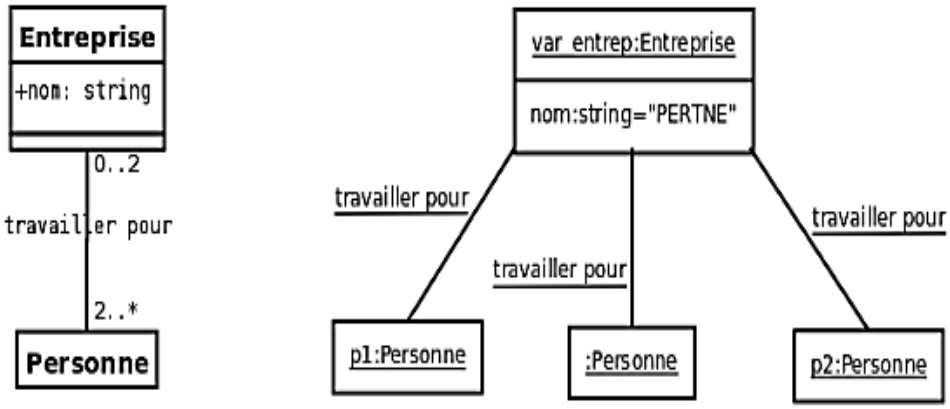


Figure 30 : Exemple 01 d'un diagramme d'objets.

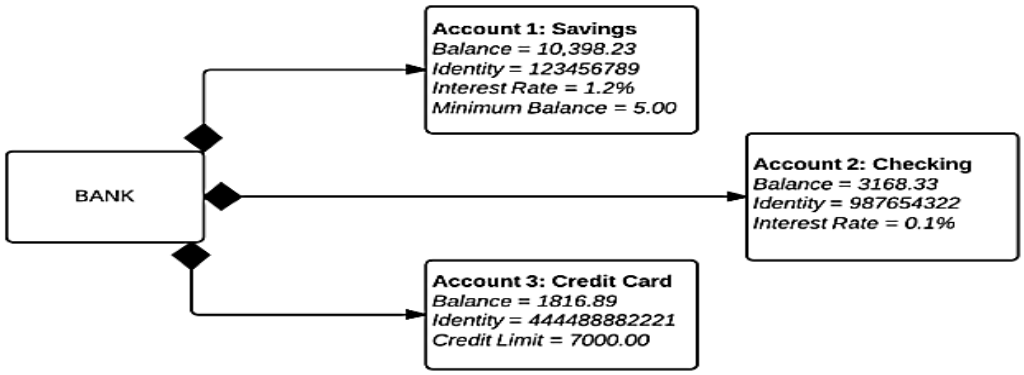


Figure 31: Exemple 02 d'un diagramme d'objets.

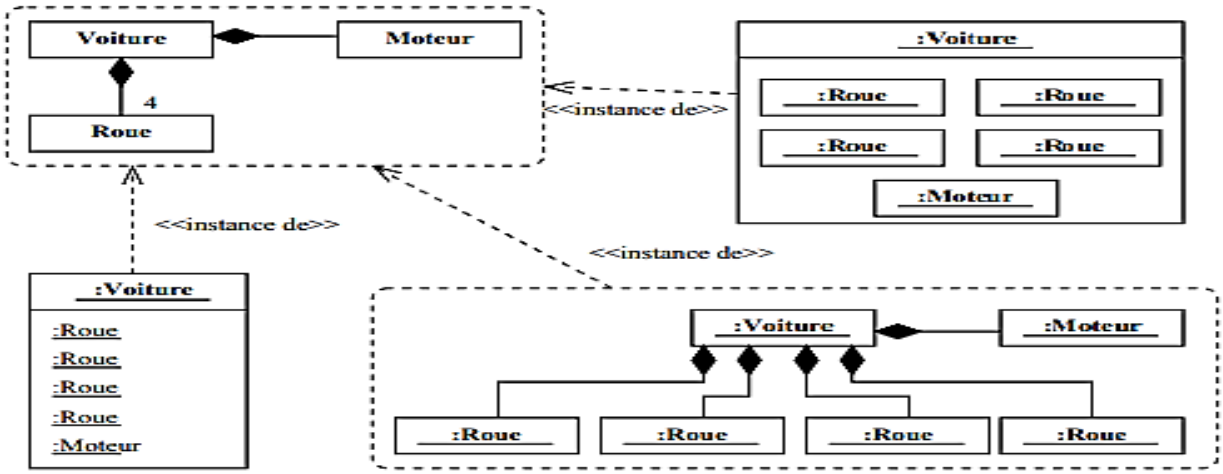


Figure 32 : Exemple 03 d'un diagramme d'objets.

4.4. Association réflexive

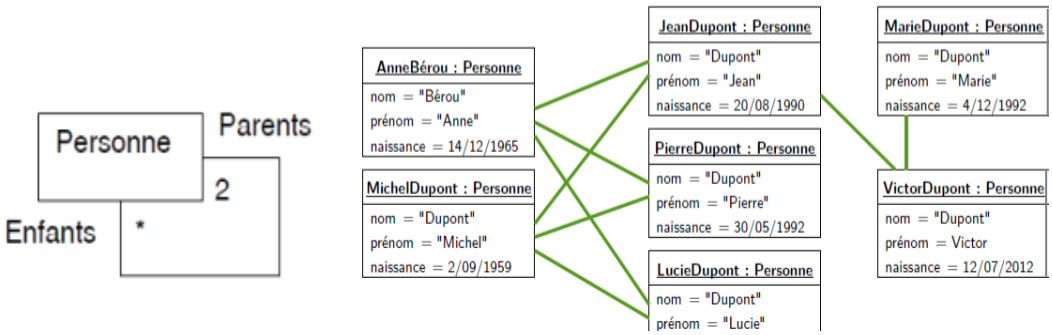


Figure 33 : Exemple d'une association réflexive.

5. Diagramme de paquetage

- Organisation des différents éléments UML modélisés en regroupant ces derniers dans des paquetages.
- Les paquetages sont l'équivalent des répertoires qui organisent les systèmes de fichiers.
- Vise à décomposer le logiciel en sous-systèmes cohérents
- Un paquetage peut contenir des sous- paquetages
- Un paquetage peut importer des éléments d'un autre paquetage.

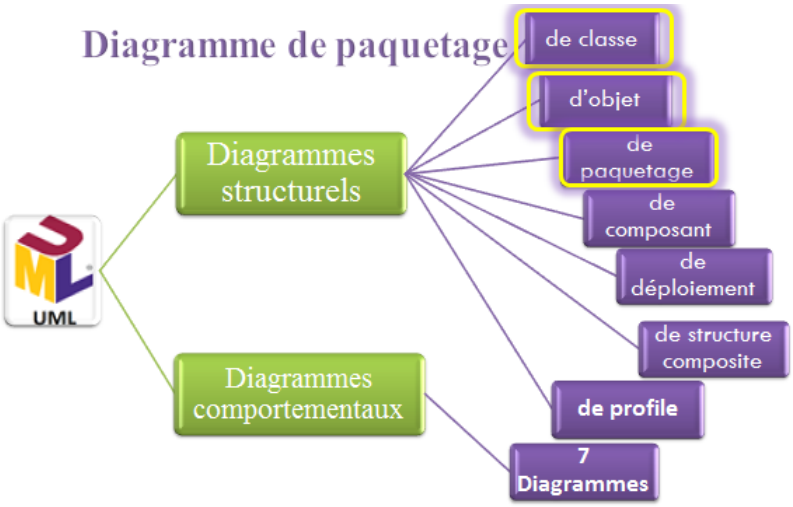


Figure 34: Diagramme de paquetage.

5.1. Pourquoi les diagrammes de paquetage?

- Un aperçu visuel de l'architecture des différents éléments UML au sein d'un système donné.
- Faciliter la maintenance du projet.
- Assurer l'évolutivité et favoriser la réutilisation
- Ces diagrammes permettent de simplifier les diagrammes de classes complexes sous une forme visuelle bien ordonnée.

5.2. Représentation de paquetage

- Représentation globale
- Représentation détaillée
- Représentation éclatée

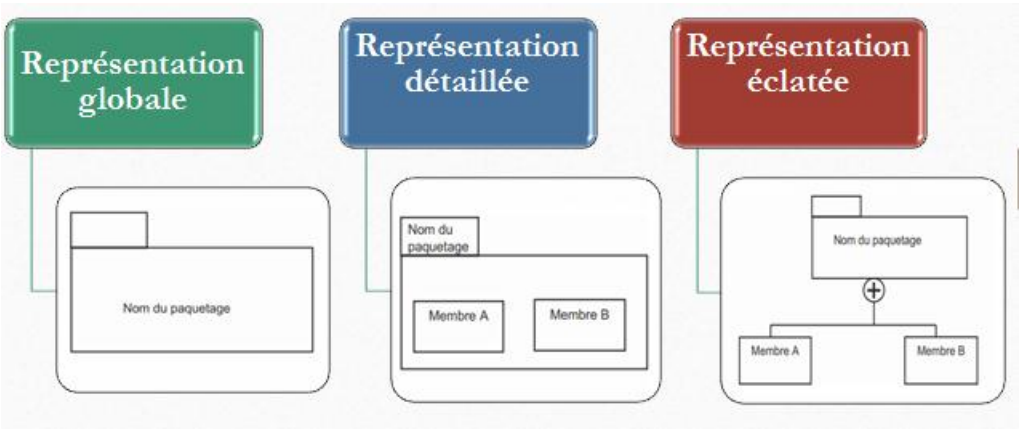


Figure 35: Représentation de paquetage.

5.3. Fondamentaux du diagramme de package

- Le diagramme de package reflète la hiérarchie des packages imbriqués.
- Le nom du package doit être unique pour un système donné, mais les classes à l'intérieur de différents packages peuvent avoir des noms identiques.
- Les packages peuvent contenir des diagrammes entiers, seulement le nom des composants, ou même ne contenir aucun composant.
- Le nom qualifié complet d'un package suit la syntaxe suivante.

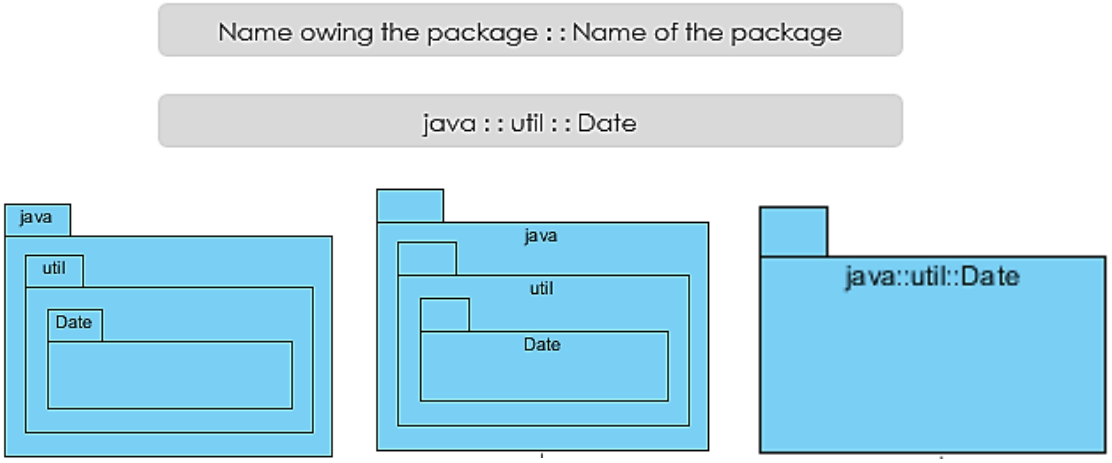


Figure 36: La hiérarchie des packages

5.4. Dépendance entre paquetages

Un paquetage A possède une relation de dépendance avec un paquetage B, si il existe au moins une classe du paquetage A utilise les services offerts par une classe du paquetage B.

Les packages encapsulent ces éléments où chacun de celle-ci peut être qualifiée par un niveau de visibilité qui est soit public soit privé.

- « import » : Importer l'espace de nommage d'un autre paquetage et Accéder à tous les noms des membres de ce paquetage importé
- « access » : Avoir accès à l'espace de nommage d'un paquetage cible. Ce dernier ne peut être transmis
- « use» : Il existe au moins un élément de A qui a besoin d'un élément de B
- « merge» : Le contenu du package A fusionne avec celui de B (B est le package père)

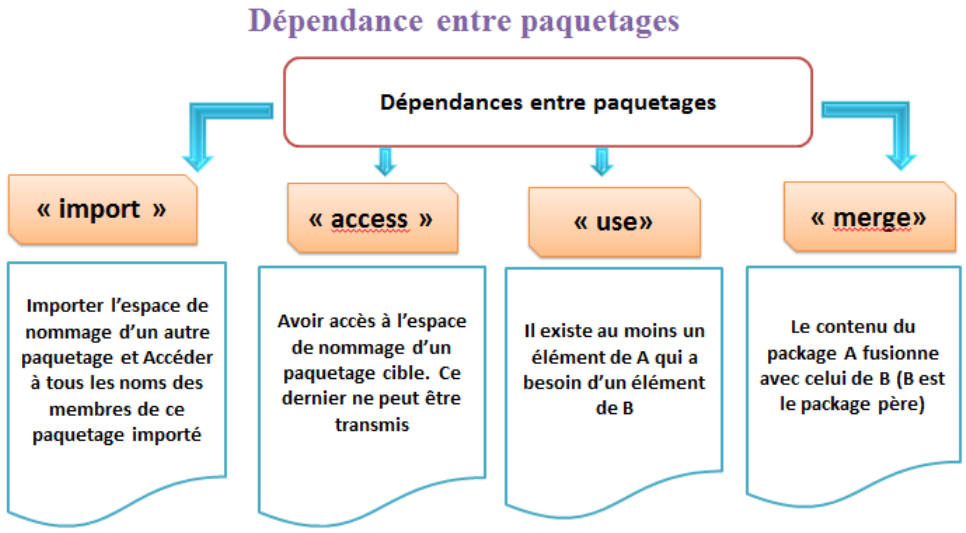


Figure 37: Dépendance entre paquetages.

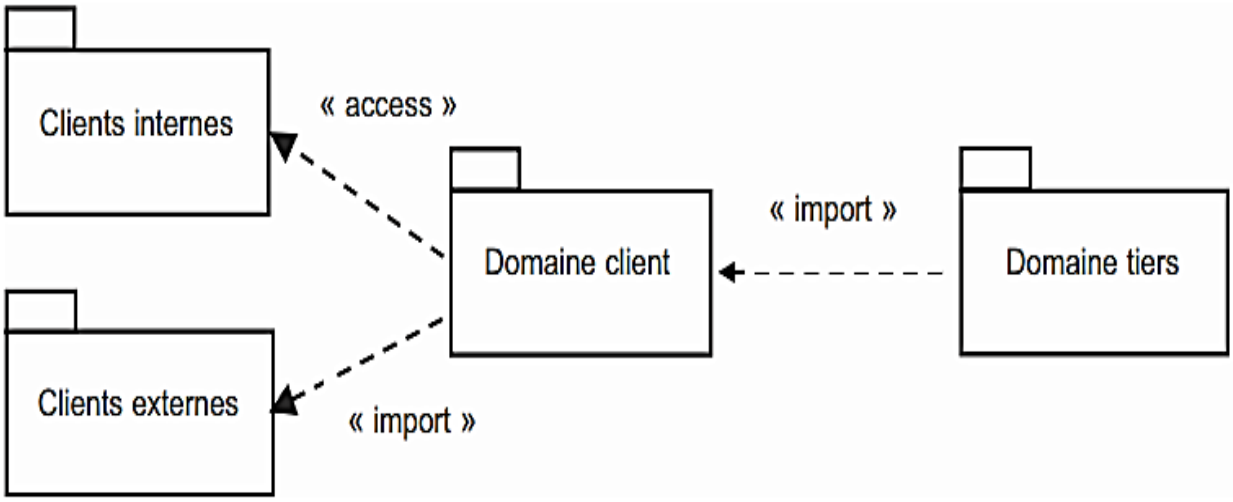


Figure 38: Exemple 01 d'un diagramme de paquetages.

Quelle est la différence entre accès et import dans le diagramme de package ?

1. Accès (Access)

- Le package client peut utiliser les éléments publics d'un autre package.
- Les éléments ne sont pas ajoutés à l'espace de noms du package client.

- Il faut donc référencer explicitement le package fournisseur pour utiliser ses éléments.
- Cela exprime une dépendance faible.

2. Import (Package Import)

- Le package client importe les éléments publics du package fournisseur.
- Les éléments importés deviennent directement visibles dans le package client.
- On peut les utiliser sans préfixer le nom du package fournisseur.
- Cela exprime une dépendance plus forte.

Tableau 03 : Résumé comparatif

Relation	Visibilité des éléments	Espace de noms	Type de dépendance
Access	Avec préfixe	Séparé	Faible
Import	Directe	Partagé	Forte
Use	Temporaire	Séparé	Très faible
Merge	Directe	Fusionné	Très forte

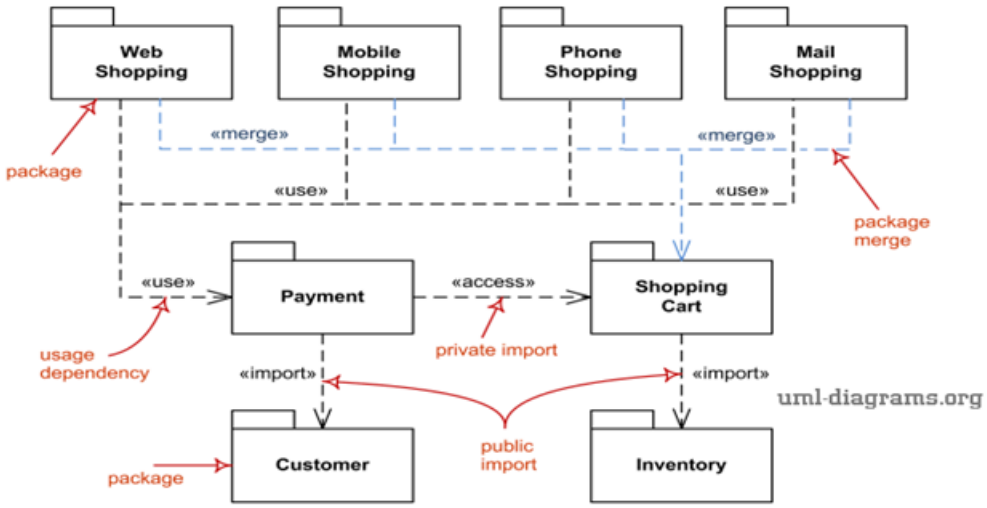


Figure 39: Exemple 02 d'un diagramme de paquetages.

6. Diagramme de composants

- ✓ Représenter les composants logiciels d'un système ainsi que les liens existant entre ces composants
- ✓ Sont des vues statiques de l'implémentation des systèmes qui montrent les choix de réalisation
- ✓ Ils sont utilisés généralement pour les systèmes complexes

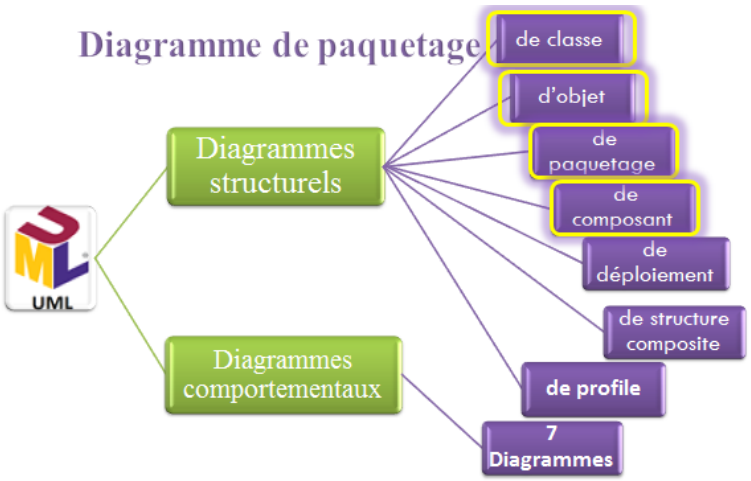


Figure 40: Diagramme de composants.

6.1. Pourquoi les diagrammes de composants

- Illustrer la relation entre les différents composants d'un système.
- Imaginer la structure physique du système
- Mettre en évidence le comportement de chaque service vis-à-vis de l'interface

6.2. Un Composant

- ➢ Elément physique qui représente une partie implémentée du système.
- ➢ Un module de classes qui représentent des systèmes ou des sous-systèmes indépendants ayant la capacité de s'interfacer avec le reste du système.
- ➢ Un composant doit fournir un service bien précis
- ➢ Il peut réaliser un ensemble d'interfaces qui définissent alors le comportement offert à d'autres composants.
- ➢ Un composant peut être :
  - Un code (source, binaire, exécutable)
  - Un fichier de commande
  - Un fichier de données
  - Une table….etc
  - un regroupement logique (composant vente par exemple)
- ➢ Un composant 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.

Un port est un point de connexion entre un composant et une interface ou bien avec son environnement.

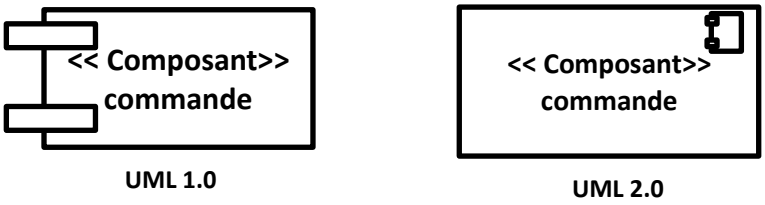


Figure 41: Notation d'un composant en UML.

6.3. Modélisation des composants

6.3.1. Boîte noire

C'est une vue externe du composant qui présente ses interfaces fournies et requises sans entrer dans le détail de l'implémentation du composant.

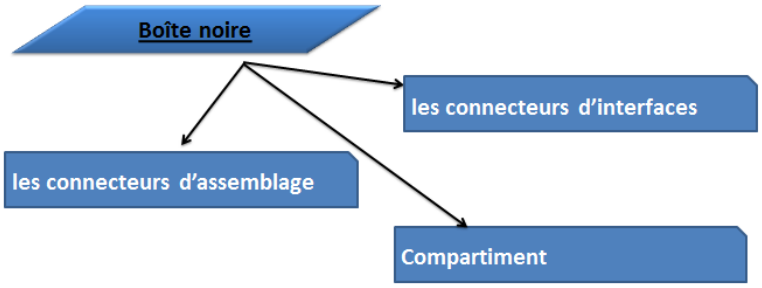


Figure 42: Les techniques de modélisation de la boite noire.

Les figures 43, 44 et 45 représentent les méthodes utilisées sous la forme d'une boîte noire, à savoir respectivement les connecteurs d'assemblage, les connecteurs d'interface et les compartiments.

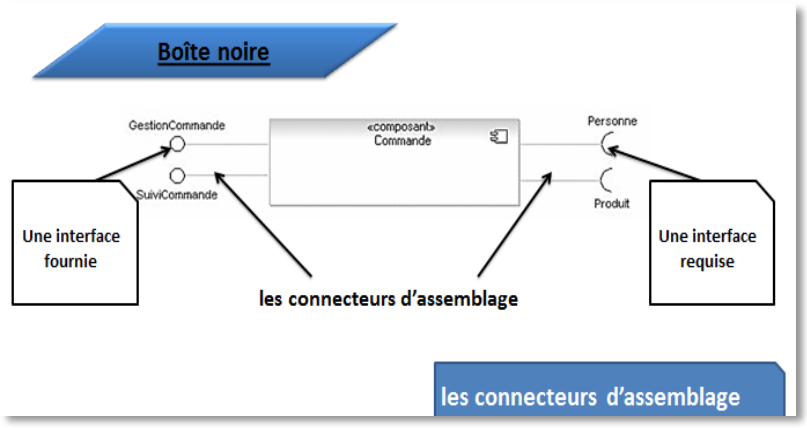


Figure 43: Les connecteurs d'assemblage.

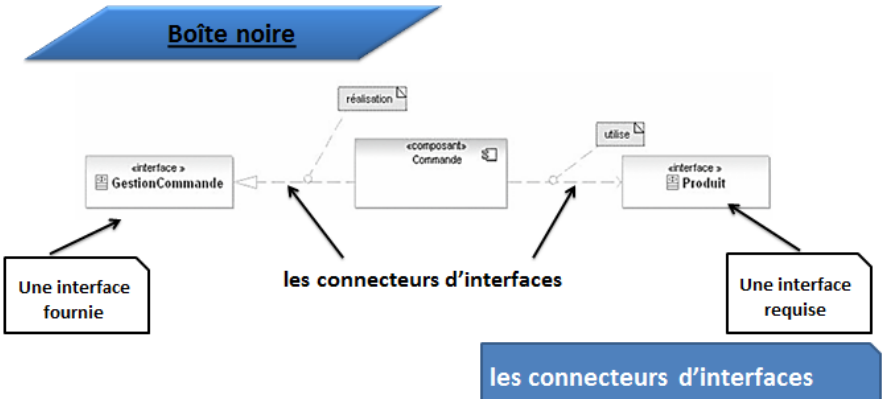


Figure 44: Les connecteurs d'interface.

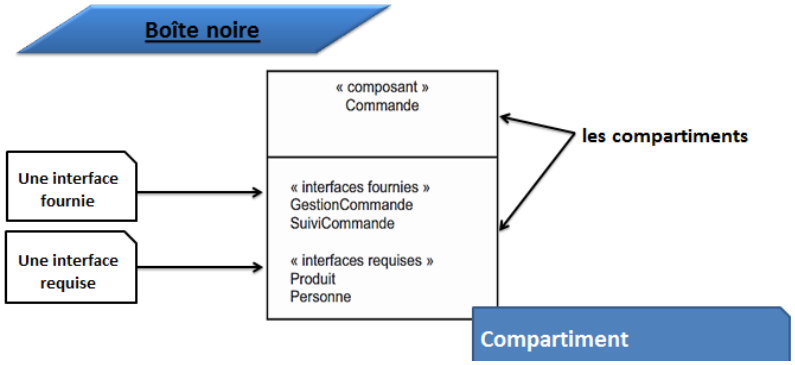


Figure 45: Les compartiments.

6.3.2. Boîte blanche

C'est une vue interne du composant qui décrit son implémentation à l'aide de classificateurs (classes, autres composants)

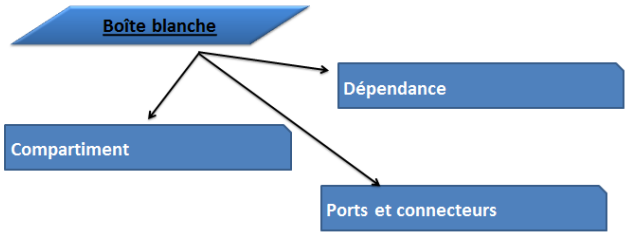


Figure 46: Les techniques de modélisation de la boite blanche.

Les figures 47, 48 et 49 illustrent les différentes méthodes adoptées selon une approche de type boîte blanche, correspondant respectivement aux compartiments, aux dépendances et aux ports et connecteurs.

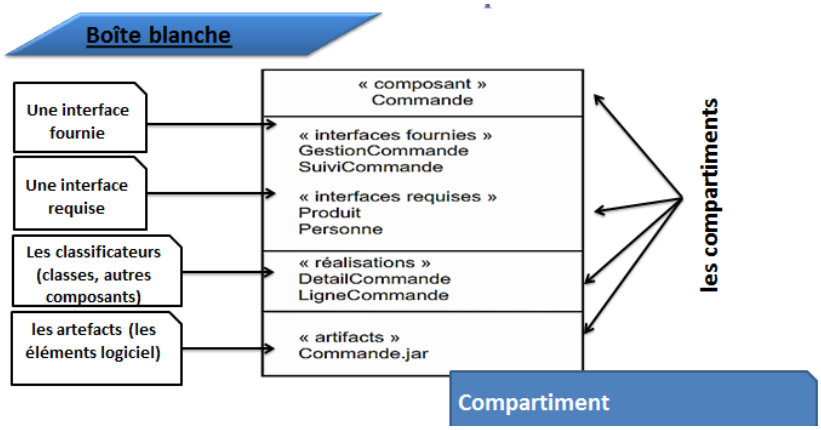


Figure 47: Les compartiments.

La relation de dépendance est utilisée dans les diagrammes de composants pour indiquer qu'un élément de l'implémentation d'un composant fait appel aux services offerts par les éléments d'implémentation d'un autre composant

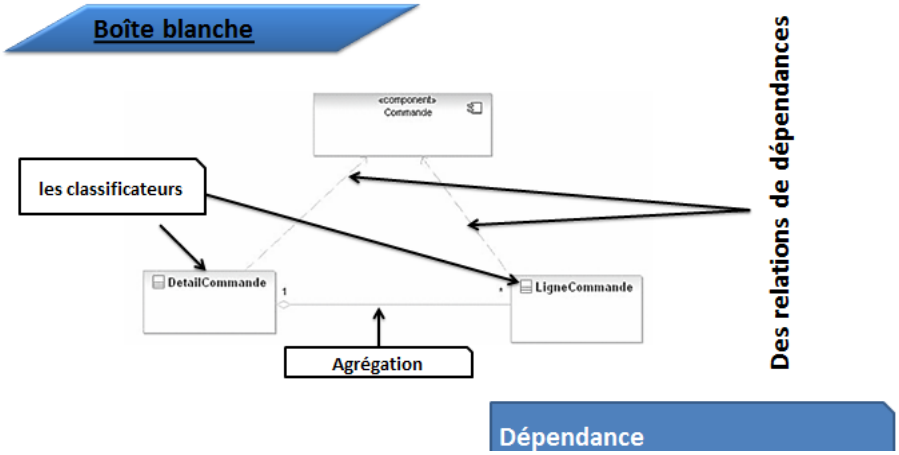


Figure 48: Les dépendances.

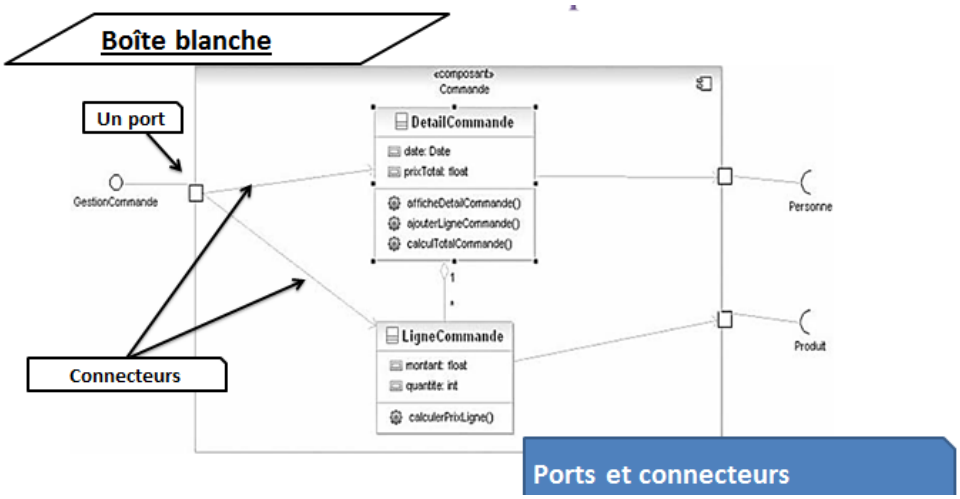


Figure 49: Les ports et connecteurs.

Un port est une interface d'un composant qui expose certaines fonctionnalités ou services que le composant peut fournir ou consommer.

La représentation de ports et de connecteurs dans ces diagrammes aide à visualiser ces interactions et à comprendre comment les fonctionnalités sont distribuées entre les différents composants du système.

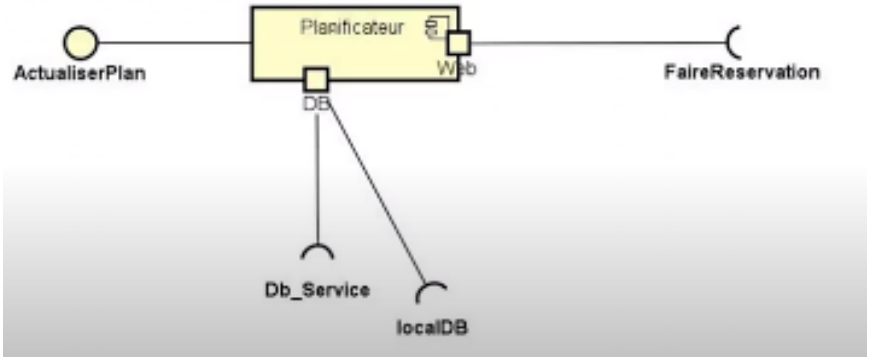


Figure 50: Exemple d'un diagramme de composants.

La figure 51 présente un exemple d'un composant complexe contenant des sous-composants. Le composant délégateur (Programmeur) délègue certaines de ses fonctionnalités aux ports du composant délégué (Réservation) en utilisant la relation de délégation, indiquant ainsi qu'il ne gère pas directement ces fonctionnalités, mais les transmet au composant délégué.

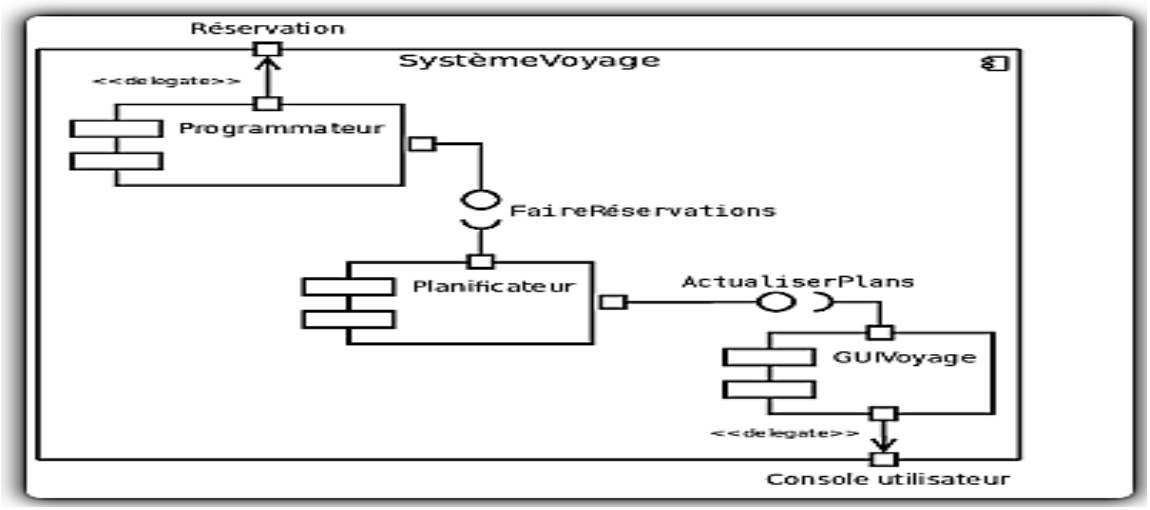


Figure 51: Exemple d'un composant complexe contenant des sous-composants.

7. Diagramme de déploiement

- Représente la structure du système lors de son exécution.
- Décrit l'architecture physique des composants matériels du système.
- Chaque ressource étant matérialisée par un nœud
- Précise comment les composants sont répartis Sur les nœuds et quelles sont les connexions entre les composants ou les nœuds.
- Graphe de nœuds connectés par des associations qui représentent la communication.

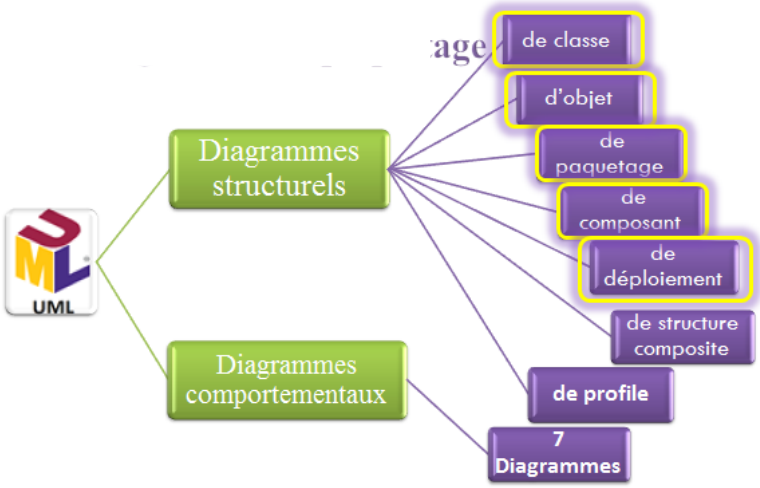


Figure 52: Le diagramme de déploiement.

7.1. Les éléments du diagramme de déploiement

- Les nœuds
- Les artefacts
- Les chemins de communications

7.1.1. Les nœuds

- Elément physique qui représente dans la plupart de cas, une ressource matérielle
- Les nœuds peuvent être interconnectés pour former un réseau d'éléments physiques
- Un nœud possède des attributs (quantité de mémoire, vitesse de processeur, marque, type…) que nous pouvons spécifier à l'intérieur du parallélépipède (un cube)
- Plusieurs stéréotypes peuvent être utilisés pour préciser la nature de l'élément,

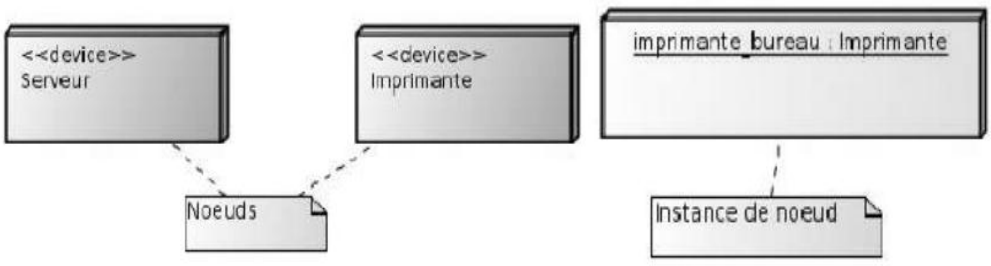


Figure 53: Représentation des nœuds.

Pour montrer qu'un composant est affecté sur un nœud, il faut :

- Soit placer le composant dans le nœud.
- Soit en le reliant à l'aide d'une relation de dépendance (flèche en pointillées) stéréotypée *«support»* orientée du composant vers le nœud

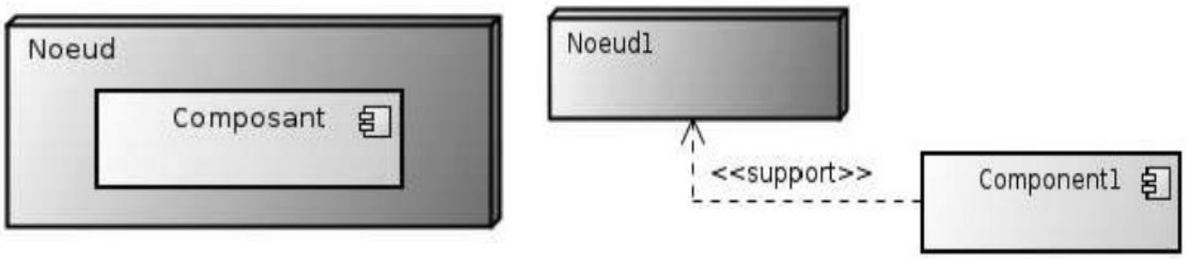


Figure 54: Représentation des composants.

7.1.2. Les artefacts

Un artefact correspond à un élément concret existant dans le monde réel (document, exécutable, fichier, tables de bases de données, script…).

Il se représente comme un classeur par un rectangle contenant le mot-clef « artefact » suivi du nom de l'artefact.

Ce sont les artefacts qui sont déployés sur les nœuds et non pas les composants

- ✓ On appelle manifestation la relation entre un élément de modèle et l'artefact qui l'implémente.
- ✓ Graphiquement, une manifestation se représente par une relation de dépendance stéréotypée « manifest ».

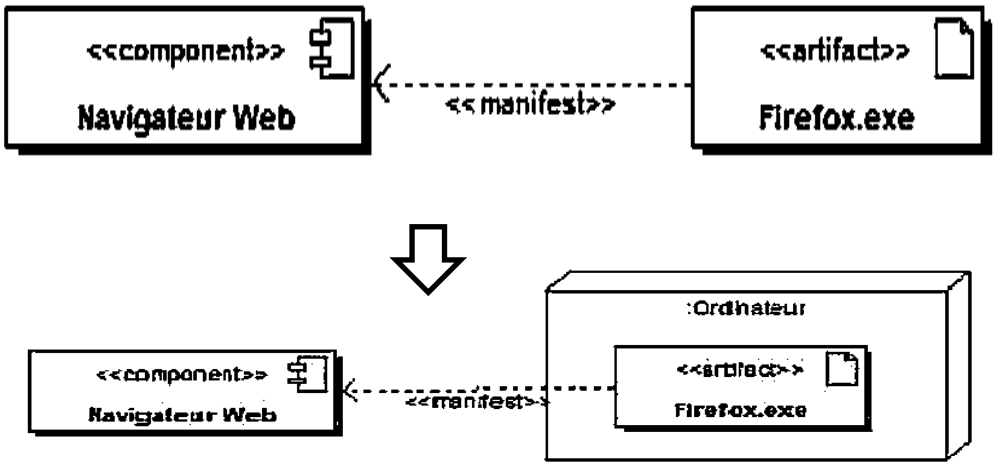


Figure 55: Représentation des artefacts avec les composants.

Il y a deux méthodes pour représenter un artéfact dans un nœud :

- Représentation inclusive
- Représentation avec un lien de dépendance typé « deploy »

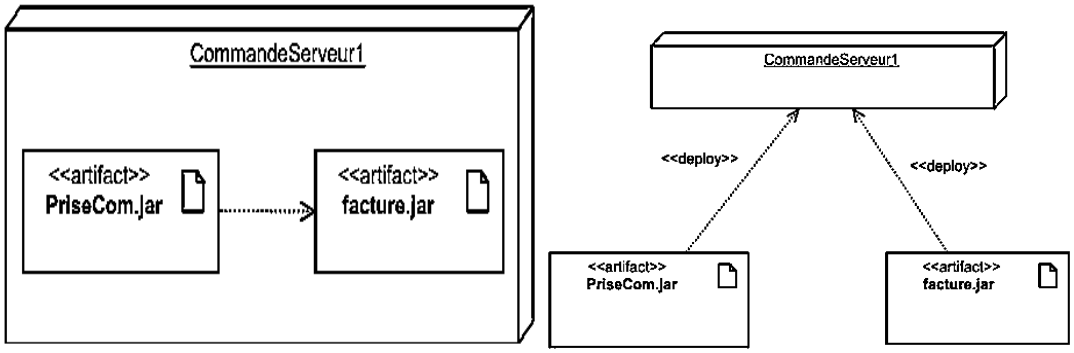


Figure 56: Représentation des artefacts avec les nœuds.

Important

Seuls des artefacts doivent être déployés sur des nœuds. Un composant doit donc être manifesté par un artefact qui, lui-même, peut être déployé sur un nœud.

7.1.3. Les chemins de communications

Un lien qui permet de modéliser de façon simpliste la communication entre 2 nœuds (liaison Ethernet, USB, série…). Il est possible de faire figurer sur ce lien :

- les cardinalités ;
- des contraintes entre accolades
- le type de réseau en l'indiquant comme un stéréotype…

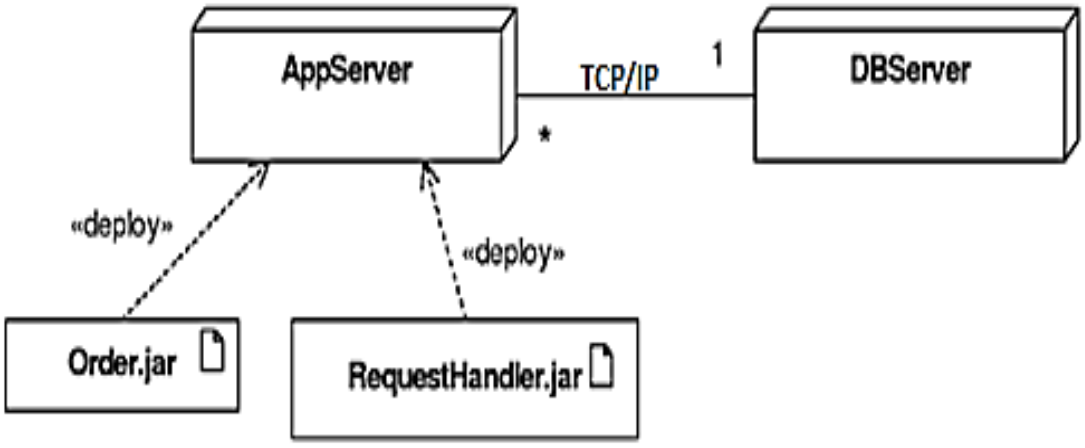


Figure 57: Exemple 01 d'un diagramme de déploiement.

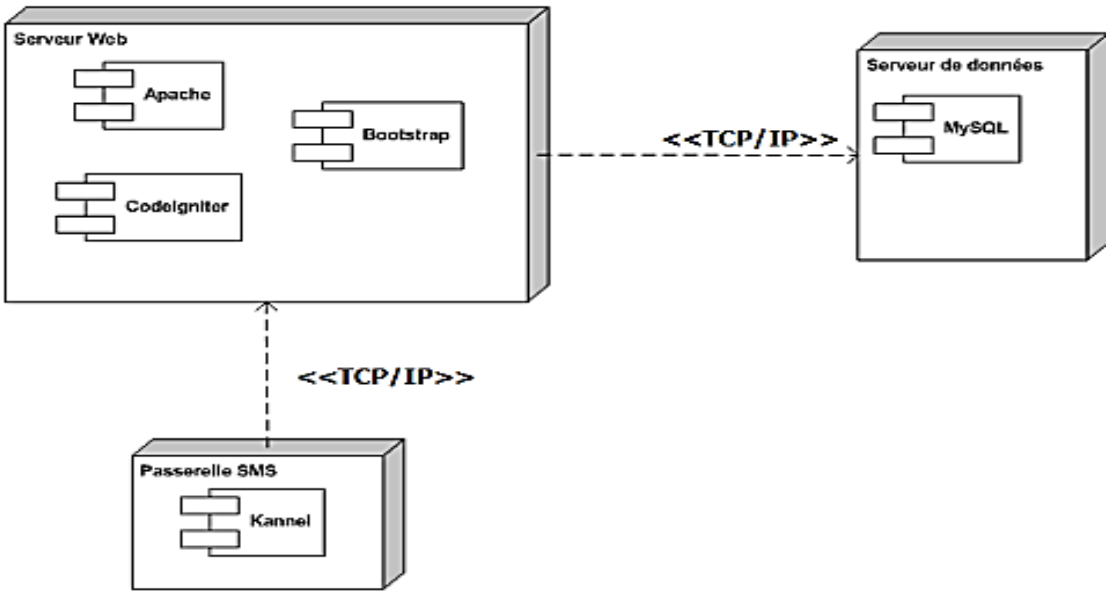


Figure 58: Exemple 02 d'un diagramme de déploiement.

7.2. Exemple d'un diagramme de déploiement

- Une architecture matérielle est constituée d'un Pc, de trois serveurs et un périphérique.
- Plusieurs Pc portable communiquent avec un serveur "AppServer" en utilisant le protocole http. Deux bases de donnée qui sont fourni par le serveur "BD_Server" sont utilisées par "AppServer" par une liaison TCP/IP.
- AppServer communique avec PrinterServer via Intranet
- PrinterServer peut utiliser dix imprimantes

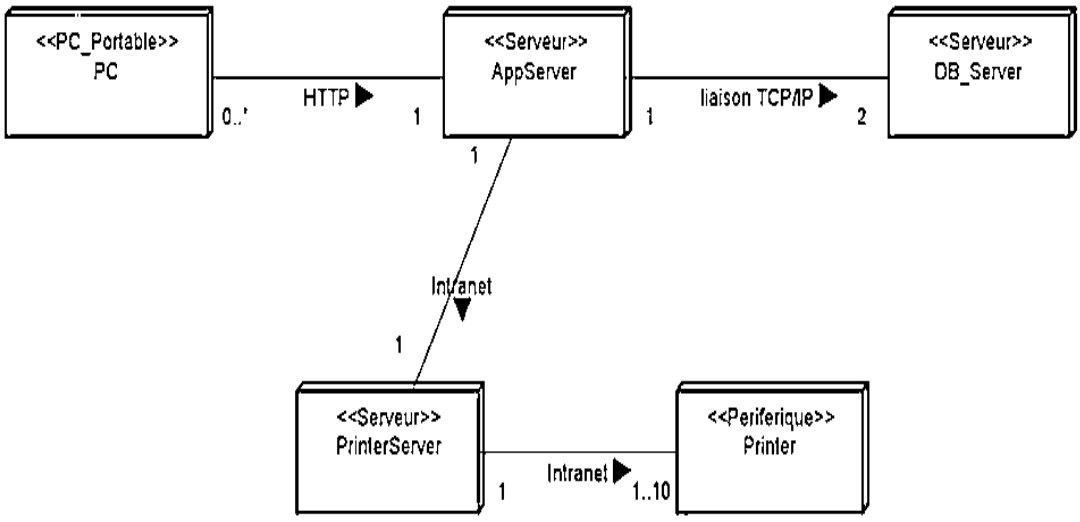


Figure 59: Exemple 03 d'un diagramme de déploiement.

8. Diagramme de structure composite

- Même rôle qu'un diagramme de classes.
- Approfondir la description de la structure interne de plusieurs classes.
- Représentation de la collaboration d'instances de classes via des liens de communication.
- Fournit une vue d'ensemble logique de l'ensemble ou d'une partie d'un système logiciel.
- Permet d'afficher: La structure interne d'un classificateur. Les interactions avec l'environnement

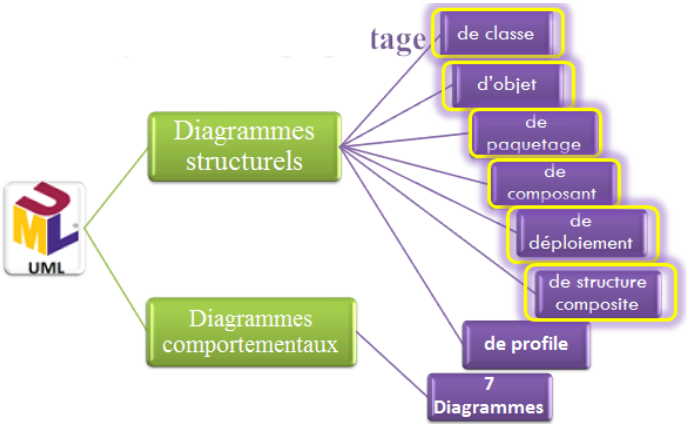


Figure 60: Diagramme de structure composite.

8.1. Avantages de diagramme de structure composite (DSC)

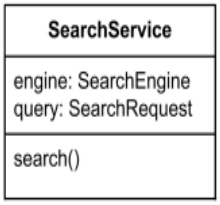
- Permet aux utilisateurs de déterminer précisément ce qui est contenu dans un objet et le comportement produit
- la décomposition des fonctionnalités peut fournir des informations précieuses sur la manière dont les structures sont interconnectées
- Aider les utilisateurs à comprendre l'état actuel de leur système.
- détailler les architectures d'exécution et les modèles d'utilisation que les autres diagrammes statiques ne peuvent pas représenter.
- Fournir aux utilisateurs des informations pour optimiser et dépanner leur système.

8.2. Composants de base d'un DSC

- Classificateur
- Ports
- Parties
- Les connecteurs

8.2.1. Classificateur

Un élément de modèle (souvent une classe abstraite) qui décrit des caractéristiques structurelles et comportementales.



Class SearchService - analysis level details

Figure 61: Un classificateur.

8.2.2. Ports

Un point d'interaction entre le classificateur et son environnement ou bien entre le classificateur et ses parties internes.

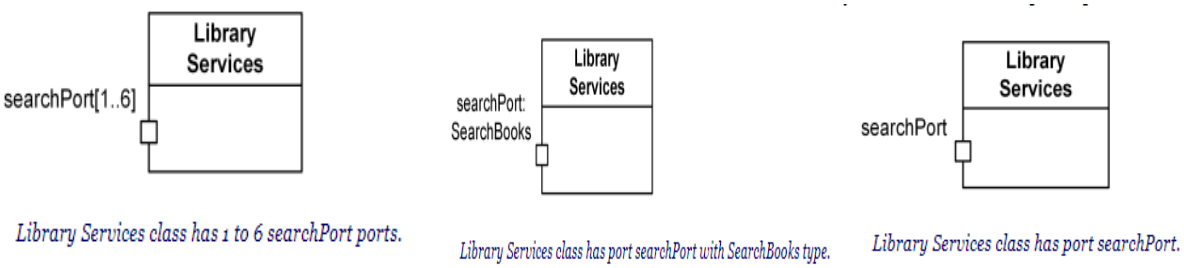


Figure 62: Un port.

8.2.3. Parties

Une zone bien délimitée à l'intérieur d'une classe ou d'un composant.

Les parties peuvent être connectées aux autres parties ou ports, soit directement, soit via un port situé à l'extérieur de la partie.

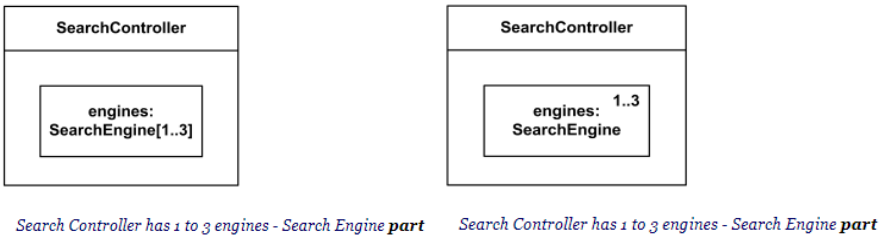


Figure 63: Une partie.

8.2.4. Les connecteurs

Les connecteurs relient plusieurs entités, leur permettant d'interagir entre elles lors de l'exécution. Un connecteur est représenté par une ligne reliant une combinaison de parties, des ports ou des classificateurs structurés.

- ❑ Le connecteur reliant des composants peut être soit un connecteur de délégation (delegation connector) soit un connecteur d'assemblage (assembly connector).

Le connecteur d'assemblage

- Un connecteur d'assemblage est un connecteur entre deux parties ou entre les ports de ces parties qui définit que l'une ou plusieurs parties fournissent les services que d'autres parties utilisent.

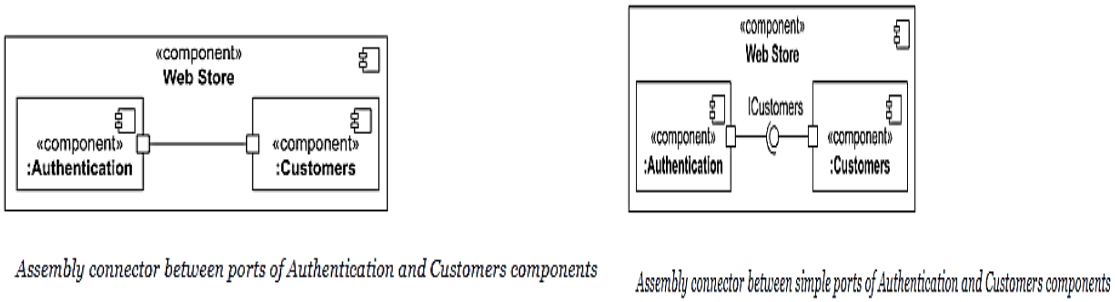


Figure 64: Un connecteur d'assemblage.

Le connecteur de délégation

- Un connecteur de délégation est un connecteur qui relie le contrat externe d'un composant à la réalisation de ce comportement. Il représente la transmission des événements (demandes d'opérations et événements)

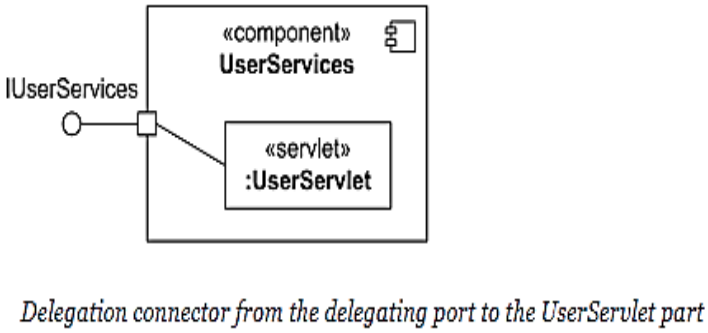


Figure 65: Un connecteur de délégation.

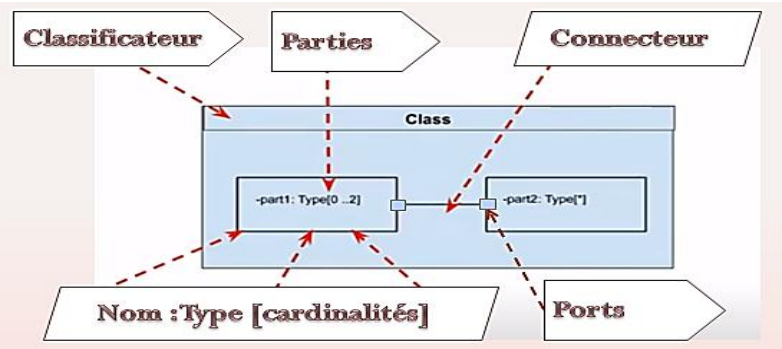


Figure 66: Les composants de base d'un DSC.

Important

- La partie n'est pas nécessairement une classe, La partie c'est tout élément utilisé pour constituer la classe contenant

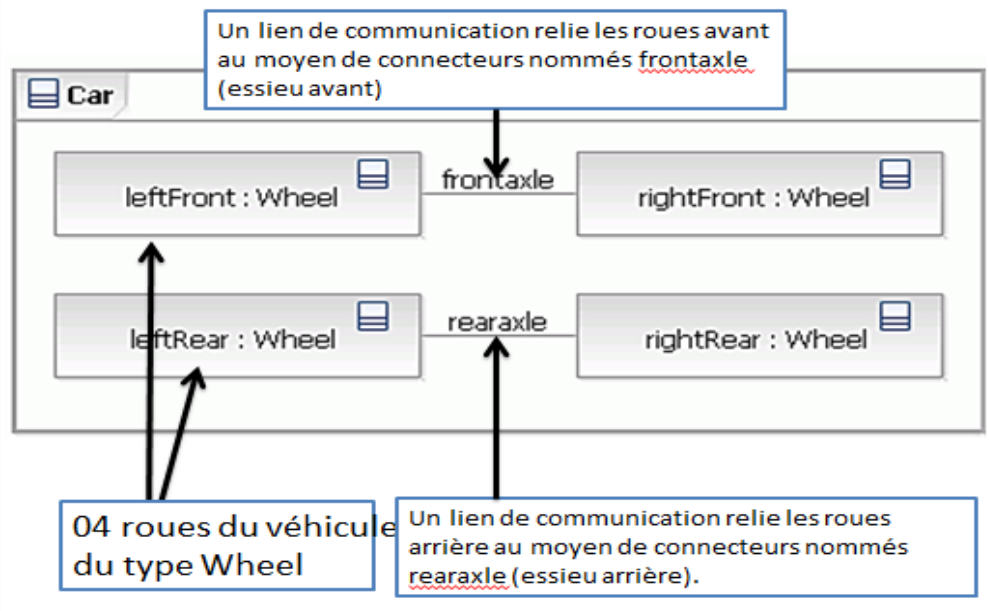


Figure 67: Exemple d'un DSC.

8.3. Les différences entre un DC et un DSC

Le diagramme de composants (DC) met l'accent sur la représentation des composants logiciels, leurs interfaces fournies et requises, ainsi que leurs dépendances. Il offre une vue externe des composants et de leurs interactions au sein de l'architecture globale du système.

En revanche, le diagramme de structure composite (DSC) se concentre sur la structure interne d'un composant, en montrant comment celui-ci est construit à partir de parties internes, de ports, de connecteurs et d'interfaces. Il permet de détailler l'organisation interne d'un composant complexe.

Ainsi, le DC est principalement utilisé pour modéliser l'architecture logicielle d'un système, tandis que le DSC est utilisé pour décrire en détail la composition interne d'un composant lorsque celui-ci nécessite une modélisation plus approfondie.

9. Diagramme de profil

- Il s'agit plus d'un langage que d'un diagramme
- Servir à définit les autres diagrammes

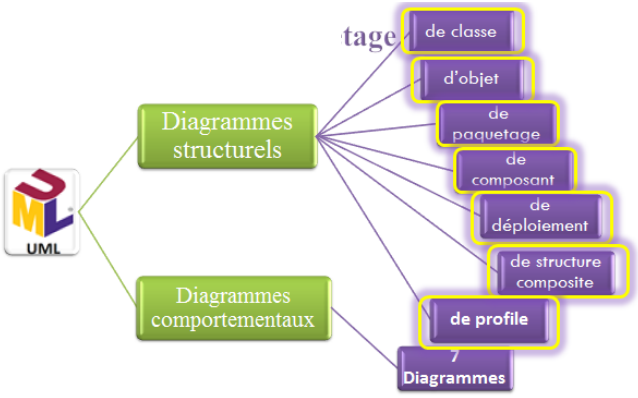


Figure 68: Diagramme de profil.

- Permet de créer de nouvelles propriétés et sémantiques pour les diagrammes UML en définissant:
  - des stéréotypes personnalisés
  - des valeurs balisées
  - des contraintes.

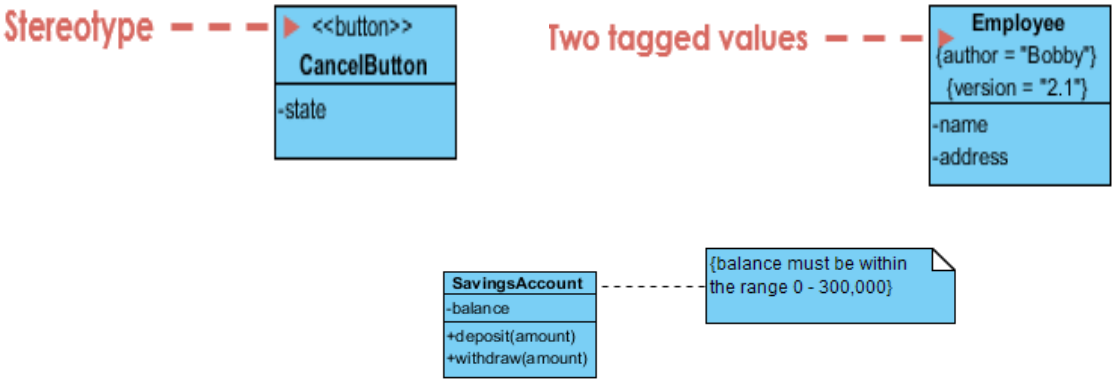


Figure 69: Des stéréotypes personnalisés, des valeurs balisées et des contraintes.

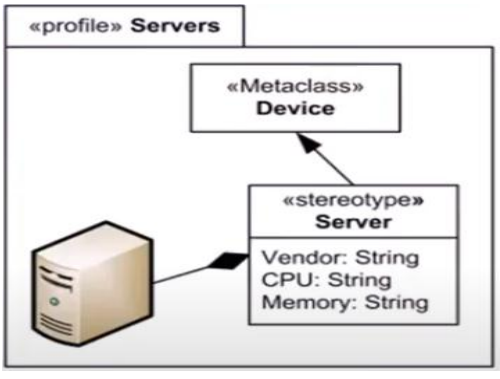


Figure 70: Exemple d'un diagramme de profil.

9.1. Éléments d'un diagramme de profil

9.1.1. *Profile (Packages)*

- Un profil est un paquet (package) de profil. Il est un moyen d'adapter un métamodèle de référence (comme UML).
- Il s'agit essentiellement d'une extension légère de la norme UML.
- Un profil introduit des contraintes sur la modélisation en utilisant des méta-classes spécifiques.
- La principale extension est le "stéréotype", qui étend une métaclasse existante pour introduire des caractéristiques spécifiques.

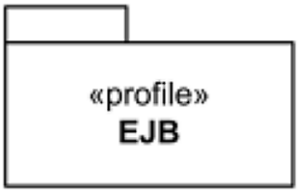


Figure 71: Un profil.

9.1.2. *Metaclass*

- Une classe de profil et un élément pouvant être inclus dans un paquetage qui peut être étendu par un ou plusieurs stéréotypes.
- Une métaclasse peut être représentée avec le stéréotype optionnel «Métaclasse» indiqué au-dessus ou avant son nom
- Une métaclasse peut être étendue par un ou plusieurs stéréotypes en utilisant un type spécial d'association appelée «extension».

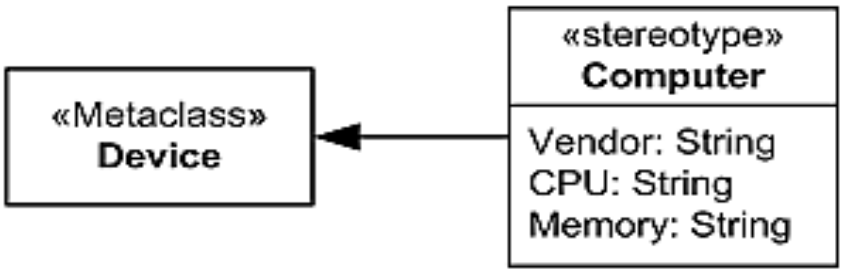


Figure 72: Une Metaclass.

9.1.3. *Stereotype*

- Un stéréotype est une classe de profil qui permet d'étendre (extend) une métaclasse existante,
- Les stéréotypes ne peuvent pas être étendus par d'autres stéréotypes.
- Ils sont représentés de la même manière qu'une classe, avec le mot-clé «stéréotype» avant ou au-dessus de leur nom, et leur nom ne doit pas entrer en conflit avec les mots-clés des éléments de modèle étendus.

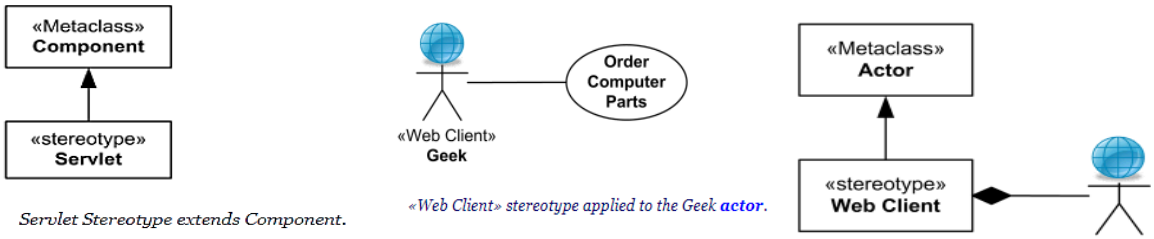


Figure 73: Un stéréotype.

⇨ Un stéréotype ne peut généraliser ou spécialiser qu'un autre stéréotype.

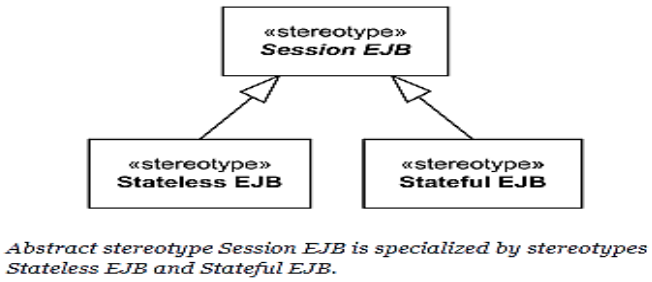


Figure 74: Un stéréotype généralise d'un autre stéréotype.

⇨ Lorsqu'un stéréotype est appliqué à un élément du modèle, les valeurs de ses propriétés peuvent être désignées sous le terme de " valeurs balisées" ou "tagged values" en anglais.

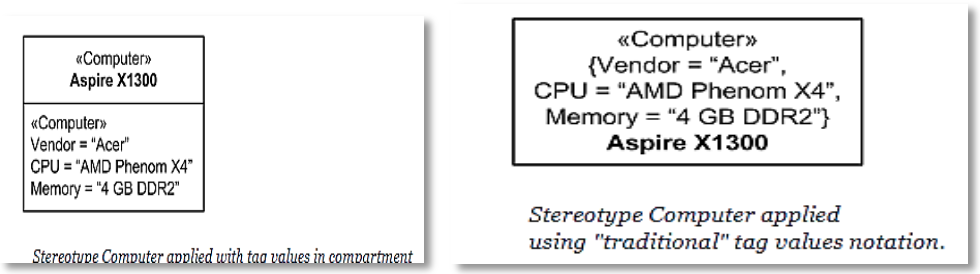


Figure 75: Des stéréotype avec des valeurs balisées.

9.1.4. Extension

⇨ Une extension est une relation d'association utilisée pour indiquer que les propriétés d'une métaclasse sont étendues au moyen d'un stéréotype, et elle offre la possibilité d'ajouter de manière flexible des stéréotypes à des classes et de les retirer ultérieurement si nécessaire.

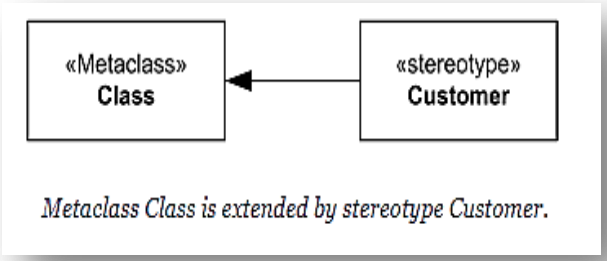


Figure 76: La relation d'extension.

9.1.5. Reference

⇨ Une référence est une relation d'importation représentée par l'élément d'importation de métaclasses appelé "metaclassReference" et l'importation de paquetage appelée "metamodelReference".

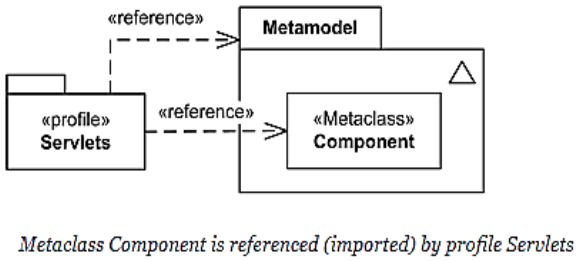


Figure 77: La relation de référence.

9.1.6. Profile application

⇨ L'application de profil est une relation dirigée utilisée pour indiquer quels profils ont été appliqués à un paquetage.

⇨ Applying a profile means that it is allowed, but not necessarily required, to apply the stereotypes that are defined as part of the profile.

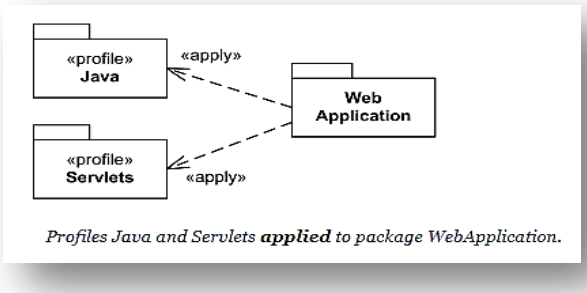


Figure 78: L'application de profil.

9.2. Représentation d'un diagramme de profil

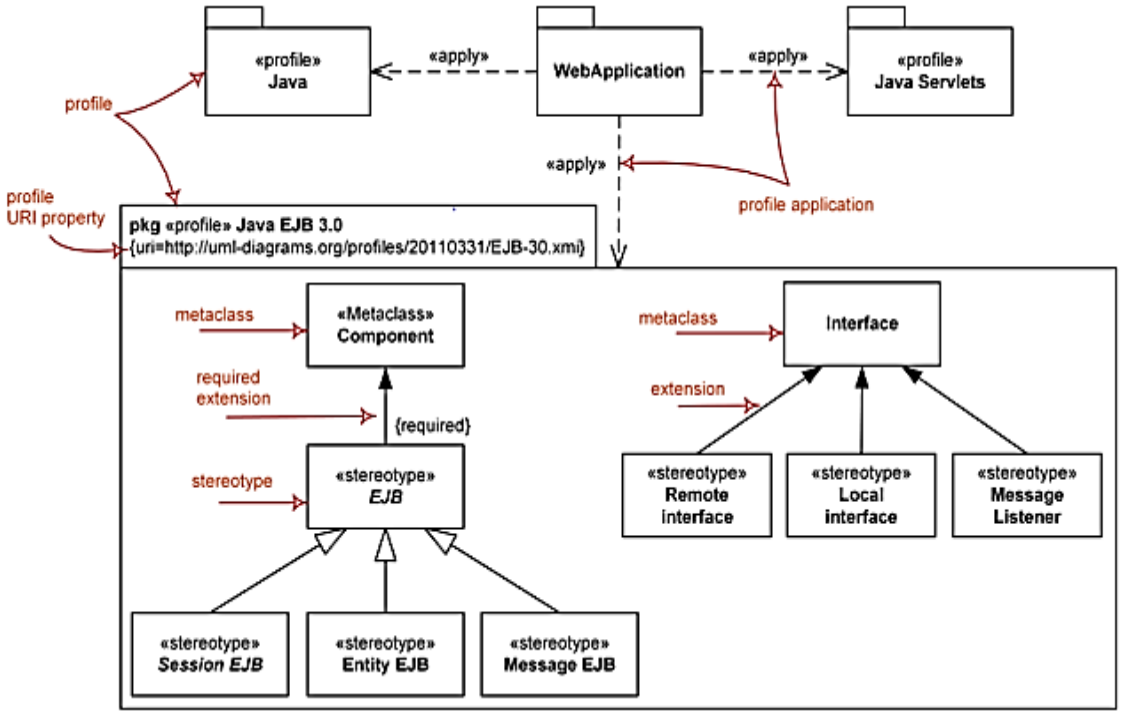


Figure 79: Exemple d'un diagramme de profil.

Chapitre 04: Diagrammes UML comportementaux

1. Introduction

Alors que les diagrammes structurels (Chapitre 03) décrivent les éléments du système (le "Squelette"), les diagrammes comportementaux décrivent comment ces éléments évoluent et interagissent pour fournir les fonctionnalités attendues (le "Mouvement").

Les diagrammes comportementaux permettent de modéliser l'aspect dynamique du système à développer. Sept diagrammes sont utilisés pour la modélisation de cet aspect de système, chacun offrant une vue différente des interactions, des flux et des changements d'états au cours du temps.

2. Les objectifs de chapitre 04

À la fin de ce chapitre, l'étudiant sera capable de :

1. Définir les éléments clés de la modélisation dynamique (Acteurs, Cas d'Utilisation, Messages, Lignes de vie).
2. Interpréter le déroulement temporel d'un processus métier à travers un Diagramme de Séquence ou d'Activité.
3. Utiliser les Diagrammes de Cas d'Utilisation pour capturer et formaliser les besoins fonctionnels des utilisateurs.
4. Décomposer un processus complexe en scénarios nominaux et alternatifs pour en modéliser la dynamique (Diagramme de Séquence).
5. Élaborer l'ensemble des diagrammes comportementaux nécessaires pour décrire le cycle de vie complet d'une fonctionnalité système.

3. Diagramme des cas d'utilisation

- ❑ Représenter les scénarios dans lesquels votre système interagit avec des personnes, des organisations ou des systèmes externes.
- ❑ Déterminer les objectifs que votre système permet aux entités d'atteindre.
- ❑ Exprimer et organiser les exigences fonctionnelles dans un système.
- ❑ Préciser le contexte et les exigences d'un système.

3.1. Les Composants d'un (DCU)

1. Acteur
2. Cas d'utilisation
3. Relations

3.2.1. Acteur

- ✓ Un acteur est une entité extérieure qui interagit directement avec le système modélisé. Un acteur peut être une personne, une organisation ou un système externe
- ✓ Un acteur ce n'est pas une personne physique mais un rôle
- ✓ Plusieurs rôles sont possibles pour une même entité

Il existe deux types d'acteurs : les acteurs principaux (principaux utilisateurs du système) et les acteurs secondaires (ceux qui interagissent occasionnellement).



Figure 1: Exemple d'un acteur.

3.2.2. Cas d'utilisation

- ✓ Un service rendu à un acteur.
- ✓ Une fonctionnalité visible de l'extérieur.
- ✓ Une action déclenchée par un acteur (Verbe à l'infinitif)

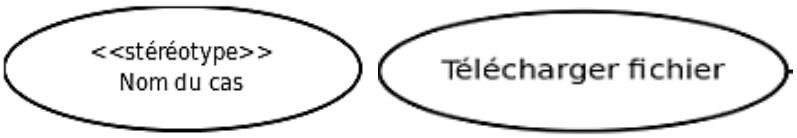


Figure 2: Exemple d'un cas d'utilisation.

3.2.3. Relations

Il existe trois types de relation

- ✓ Relation entre l'acteur et les cas d'utilisation (interaction)
- ✓ Relations entre les acteurs
- ✓ Relations entre cas d'utilisation

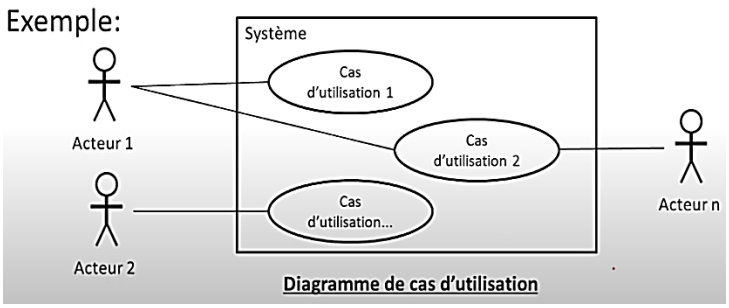


Figure 3: Exemple d'un diagramme de cas d'utilisation.

⇨ Relation entre l'acteur et les cas d'utilisation (interaction)

Une interaction permet de décrire les échanges entre un acteur et un cas d'utilisation. Elle se représente comme une association. Elle peut comporter des multiplicités comme toute association entre classes

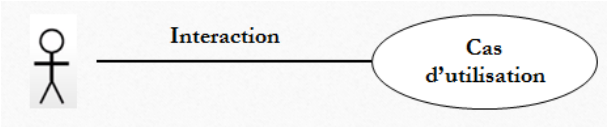


Figure 4: Exemple d'une interaction.

Si un acteur a la possibilité d'interagir plusieurs fois avec un cas d'utilisation, il est envisageable d'indiquer une multiplicité du côté du cas d'utilisation au niveau de l'association.

⇨ Relations entre les acteurs

Un seul type de relation entre les acteurs est possible: Un acteur A est une généralisation d'un acteur B lorsque l'acteur A peut être remplacé par l'acteur B. Cela signifie que tous les cas d'utilisation accessibles à l'acteur A sont également accessibles à l'acteur B, mais cela ne fonctionne pas dans le sens inverse.

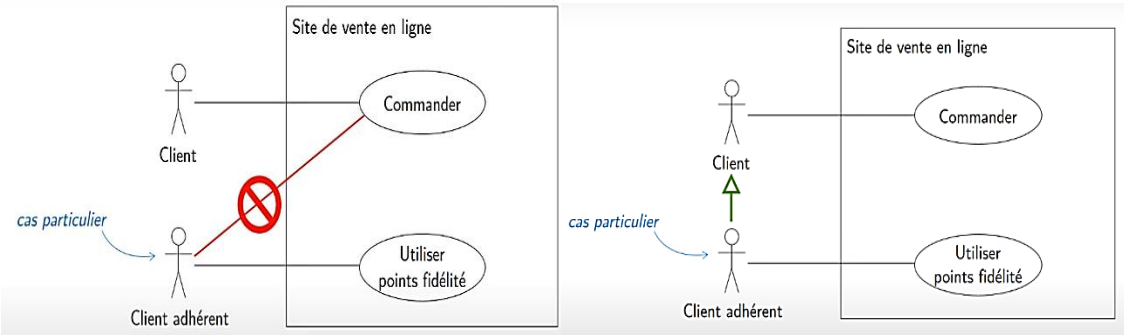


Figure 5: Exemple d'une relation de généralisation entre les acteurs.

⇨ Relations entre les cas d'utilisation

I. *Relation de généralisation*

Une relation de généralisation est utilisée pour indiquer que le cas d'utilisation (B) hérite de tout ce qui caractérise le cas (B), c'est-à-dire que le cas B est une sorte du cas A.



Figure 6: La relation de généralisation entre les cas d'utilisation.

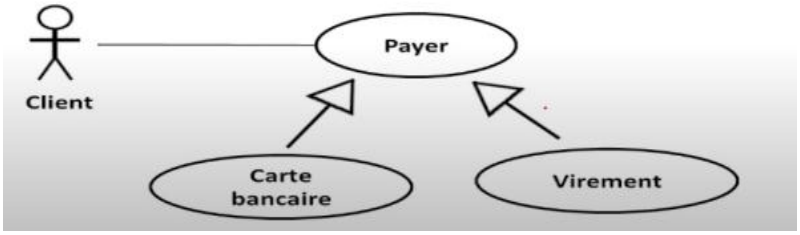


Figure 7: Exemple d'une relation de généralisation entre les cas d'utilisation.

II. *Relation d'inclusion*

Une relation « include » est utilisée pour indiquer que le cas d'utilisation source (A) contient toujours le cas d'utilisation inclus (B), c'est-à-dire que le cas B est *obligatoire* pour faire le cas A

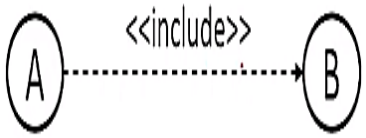


Figure 8: La relation d'inclusion.

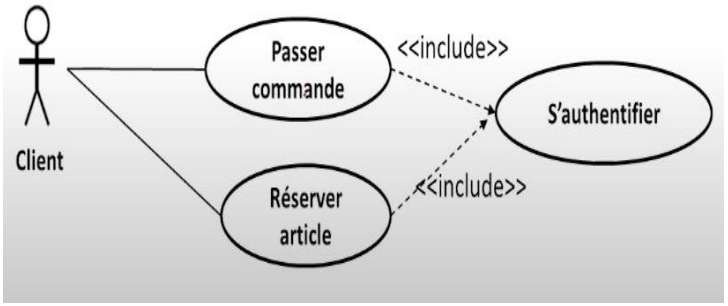


Figure 9: Exemple de la relation d'inclusion.

III. *Relation d'extension*

Une relation « extend » est utilisée pour indiquer que le cas d'utilisation source (B) n'est pas toujours nécessaire au cas d'utilisation principale (A), c'est-à-dire que le cas B est une partie *optionnelle* du cas A



Figure 10: La relation d'extension.

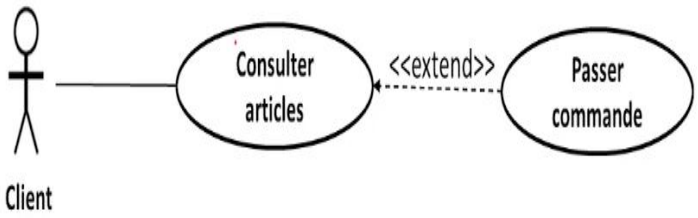


Figure 11: Exemple de la relation d'extension.

3.2. Paquets des cas d'utilisation

Le paquet est un autre élément optionnel extrêmement utile dans les diagrammes complexes en découpant le système en paquets (sous-systèmes). Un sous-système doit avoir un nom et regrouper une famille de fonctionnalités clairement identifiable.

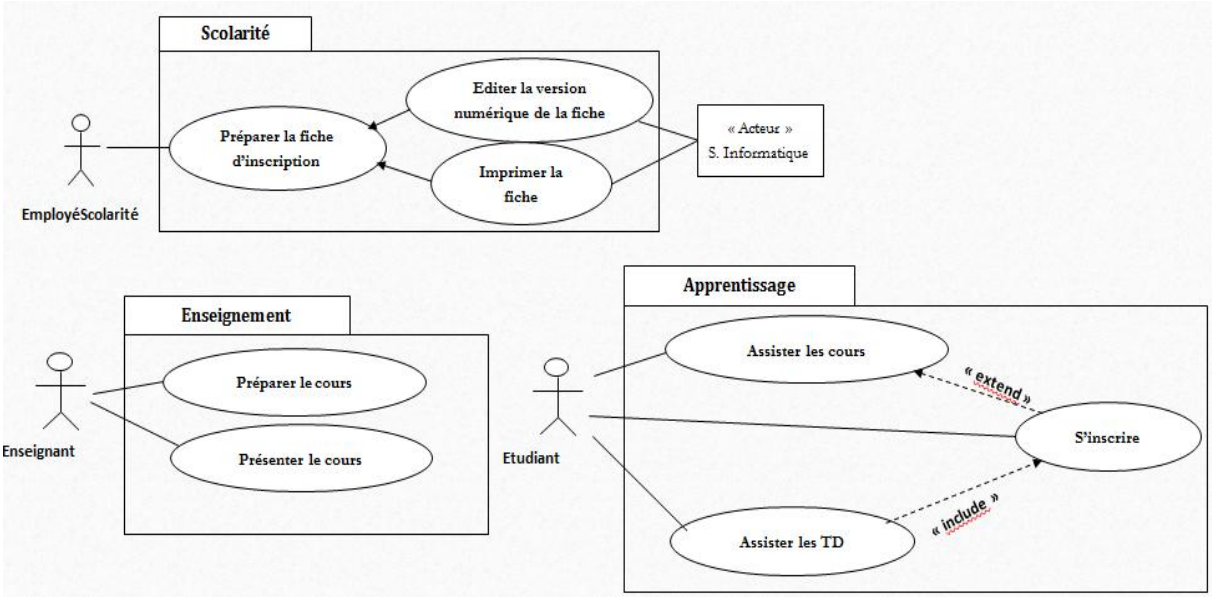


Figure 12: Exemple des paquets des cas d'utilisation.

3.3. Description textuelle d'un cas d'utilisation

Une description textuelle des interactions entre l'acteur et le système et les actions que le système doit réaliser doit être associée à chaque cas d'utilisation.

Il n'existe pas de norme établie pour la description textuelle des cas. On y trouve généralement pour chaque cas :

- Son nom, Un bref résumé de son déroulement, Le contexte dans lequel il s'applique, Les acteurs qu'il met en jeu, puis
- Une description détaillée, faisant apparaître le déroulement nominal de toutes les interactions (scénario nominal), Les variantes du scénario nominal (scénarii alternatifs), Les cas nécessitant des traitements d'exceptions (cas d'erreurs, scénarii d'exception),
- Les effets du déroulement sur l'ensemble du système,

Cette approche textuelle fournit une documentation détaillée et claire pour chaque cas d'utilisation dans un diagramme de cas d'utilisation.

Il n'existe pas de norme établie pour la description textuelle des cas. Voici comment vous pourriez structurer une description textuelle dans un diagramme de cas d'utilisation :

- Titre du Cas d'utilisation : Commencez par le titre du cas d'utilisation pour indiquer clairement le scénario que vous décrivez.
- Objectif : Décrivez l'objectif global du cas d'utilisation. Quel est le résultat attendu pour l'acteur ou le système après l'exécution du scénario ?
- Acteurs : Identifiez les acteurs impliqués dans le cas d'utilisation. Les acteurs peuvent être des utilisateurs humains, d'autres systèmes, ou des entités externes qui interagissent avec le système.
- Préconditions : Énumérez les conditions qui doivent être remplies avant que le scénario du cas d'utilisation puisse commencer. Par exemple, des données spécifiques doivent être disponibles, ou certains états du système doivent être atteints.
- Déclencheur : Indiquez ce qui déclenche le début du scénario. Il peut s'agir d'une action d'un acteur ou d'un événement particulier.
- Scénario principal (nominal): Divisez le scénario en étapes séquentielles. Chaque étape doit décrire une action spécifique effectuée par un acteur ou par le système.
- Extensions (le cas échéant) : Décrivez les scénarios alternatifs ou les branches qui peuvent se produire en fonction de certaines conditions. Cela inclut les chemins d'erreur et les actions alternatives.
- Postconditions : Indiquez les conditions qui seront remplies après la fin du scénario. Cela peut inclure des changements d'état dans le système, des notifications à l'acteur, ou d'autres résultats.
- Remarques : Ajoutez des remarques supplémentaires ou des informations pertinentes pour clarifier certains points ou spécifications importantes.

3.3.1. Exemple 01 d'une description textuelle d'un cas d'utilisation

Un exemple partiel de description textuelle pour un cas d'utilisation « Enregistrer un nouveau client »

- Titre du Cas d'utilisation : Enregistrer un nouveau client
- Objectif : Permet à un employé d'enregistrer les informations d'un nouveau client dans le système.
- Acteurs : Employé
- Préconditions : L'employé est authentifié dans le système.
- Déclencheur : L'employé sélectionne l'option "Enregistrer un nouveau client" dans le menu.
- Scénario principal :

    1. Le système affiche le formulaire d'enregistrement du client.
    2. L'employé remplit les informations du nouveau client.
    3. L'employé valide le formulaire.
    4. Le système enregistre les informations du client dans la base de données.
    5. Le système affiche un message de confirmation.

- Postconditions : Les informations du nouveau client sont enregistrées dans la base de données.

- Extensions :
  - Si l'employé quitte la page avant de valider, le système annule l'opération d'enregistrement.
  - Si des informations obligatoires sont manquantes, le système affiche un message d'erreur et demande à l'employé de les fournir.

3.3.2. Exemple 02 d'une description textuelle d'un cas d'utilisation

La description textuelle du cas d'utilisation « Inscription » :

- Titre du Cas d'utilisation : : Inscription
- Objectif : Permet à un étudiant de s'inscrire
- Acteurs : Etudiant, Employé de scolarité
- Préconditions
  - L'étudiant est valable pour le niveau demandé.
  - Le dossier d'inscription de l'étudiant est retenu.
- Scénario nominal (Principale)
  1. L'étudiant dépose son dossier d'inscription.
  2. L'employé vérifie si le dossier est complet.
  3. L'étudiant saisit les informations de formulaire d'inscription.
  4. L'employé vérifie si les informations saisies sont correctes.
  5. L'employé édite une fiche d'inscription.
  6. Le système informatique imprime la fiche d'inscription.
  7. La fiche d'inscription est passée pour signature.
  8. L'étudiant récupère sa fiche d'inscription.
- Scénario alternatifs
  - En (2) : si le dossier est incomplet, l'étudiant est averti et invité à compléter son dossier.
  - En (4) : si les informations sont erronées, elles sont redemandées à l'étudiant.
- Postconditions
  - La fiche est préparée.
  - L'étudiant est inscrit au niveau demandé.
- Remarques (Rubriques optionnelles)

  La fiche d'inscription est préparée dans un délai de 03 jours au maximum.

4. Diagramme de séquence

- ❑ Spécifie le comportement d'un cas d'utilisation ou d'une partie de celui-ci
- ❑ Décrit comment et dans quel ordre plusieurs objets fonctionnent ensemble.
- ❑ 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

4.1. Avantages des diagrammes de séquence

- ❖ Représenter les détails d'un cas d'utilisation UML
- ❖ Modéliser le déroulement logique d'une procédure, fonction ou opération complexe
- ❖ Voir comment les objets et les composants interagissent entre eux pour effectuer un processus.
- ❖ Schématiser et comprendre le fonctionnement détaillé d'un scénario existant ou à venir

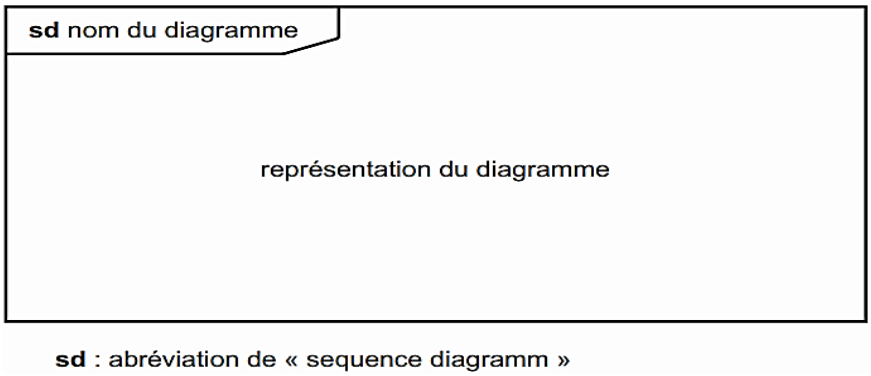


Figure 13: Représentation d'un diagramme de séquence.

4.2. Les Composants d'un (DS)

1. Acteur
2. Objets (instances des classes)
3. Messages (cas d'utilisation, appel d'opération)
4. Lignes de vie
5. Les fragments dans un DS

4.2.1. Acteur:

Montre les entités qui interagissent avec le système ou qui sont extérieures à lui



Figure 14: Représentation d'un acteur.

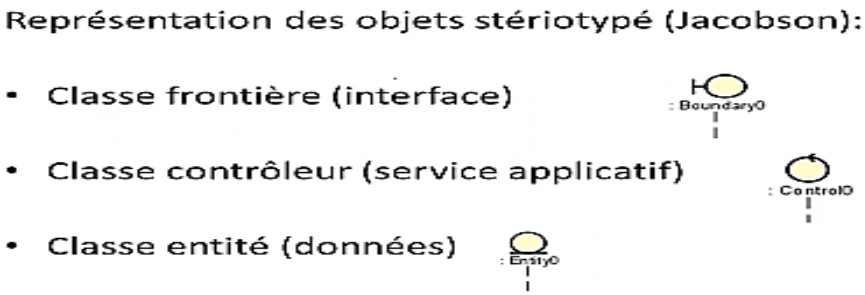


Figure 15: Représentation des objets stéréotypé.

4.2.2. Objets

Les objets représentent les instances de classes ou de composants qui participent à l'interaction. Un objet est matérialisé par un rectangle et une barre verticale appelée ligne de vie d'objet.

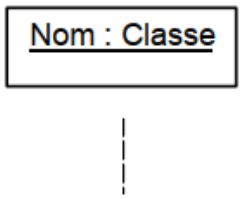


Figure 16: Représentation d'un objet.

- La réception des messages provoque une période d'activité (rectangle vertical sur la ligne de vie) marquant le traitement du message.
- Un objet peut être actif plusieurs fois

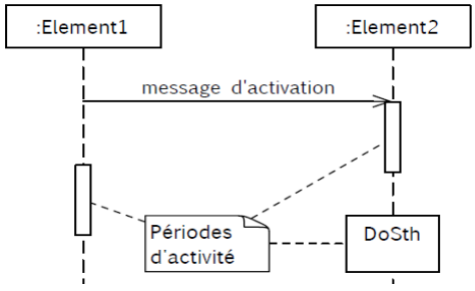


Figure 17: Représentation d'une période d'activité.

4.2.3. Messages

Il existe quatre types de message

- ✓ Message synchrone
- ✓ Message asynchrone
- ✓ Message de routeur
- ✓ Message réflexive

⇨ ***Message synchrone***

- L'émetteur reste en attente de la réponse à son message avant de poursuivre ses actions c-t-d: émetteur bloqué en attente de retour.
- La flèche avec extrémité pleine symbolise ce type de message.

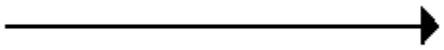


Figure 18: Représentation d'un message synchrone .

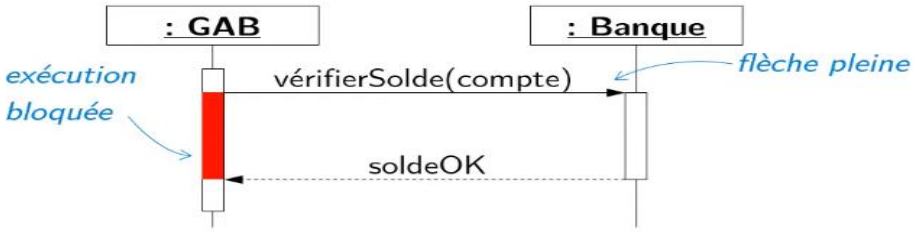


Figure 19: Exemple d' un message synchrone .

⇨ ***Message asynchrone***

- l'émetteur n'attend pas la réponse à son message, il poursuit l'exécution de ses opérations c-t-d: émetteur non bloqué, continue son exécution.
- C'est une flèche avec une extrémité non pleine qui symbolise ce type de message.

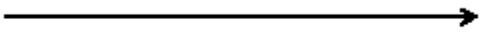


Figure 20: Représentation d'un message asynchrone .

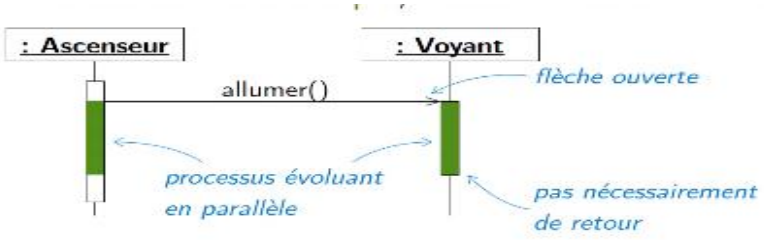


Figure 21: Exemple d'un message asynchrone .

- ***Message de retour***

Un message de retour peut être implicite (cas général) ou explicite. Un message de retour explicite est représenté par une flèche en pointillée en parallèle avec le message émis.



Figure 22: Représentation d'un message de retour .

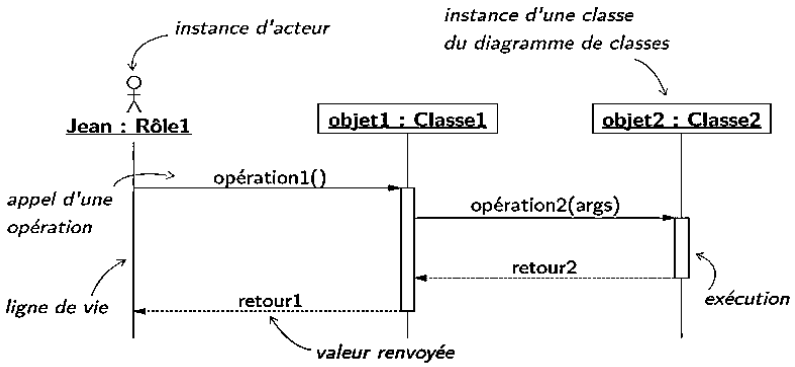


Figure 23: Exemple d'un message de retour .

- ***Message réflexive***

Un objet peut également s'envoyer un message

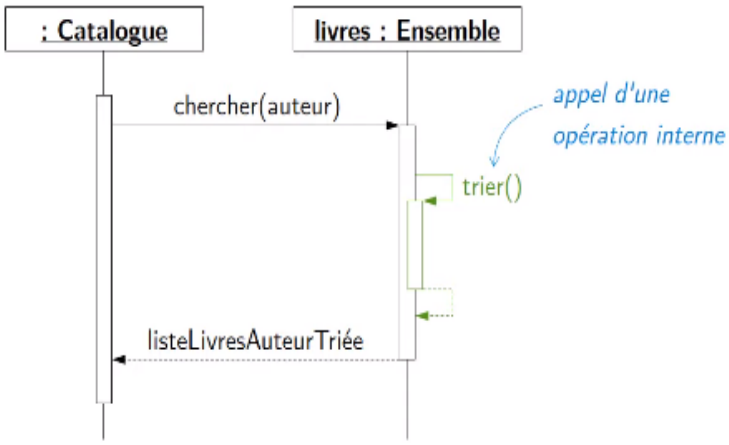


Figure 24: Représentation d'un message réflexive .

4.2.4. Lignes de vie

Chaque objet a une "ligne de vie" verticale qui représente sa durée de vie pendant l'interaction. Les messages (appel de méthode, envoi de message, etc.) sont tracés le long de ces lignes de vie.

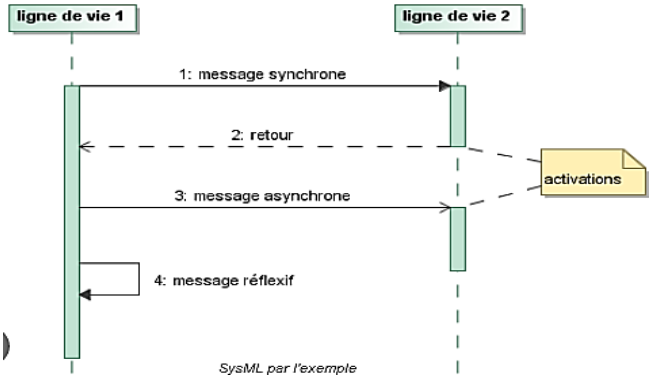


Figure 25: Représentation d'une ligne de vie .

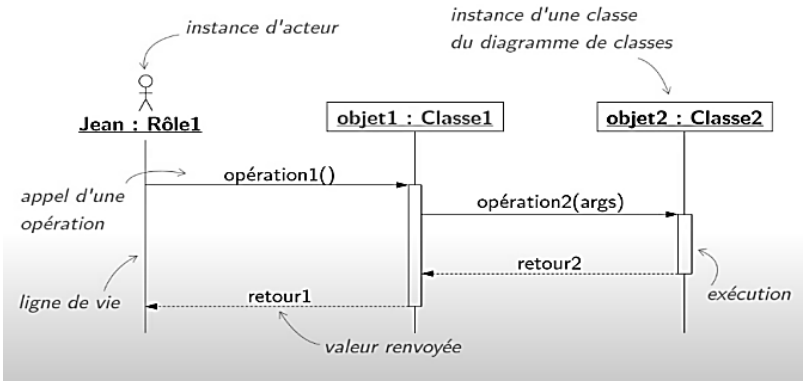


Figure 26: Exemple d'un diagramme de séquence.

4.3. Création et destruction d'un 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 ».

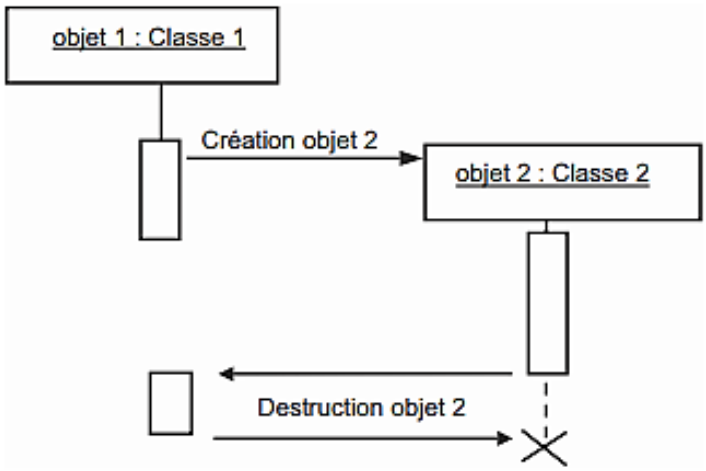


Figure 27: Exemple de création et destruction d'un objet.

4.4. Contrainte temporelle

La flèche qui symbolise un message peut être représentée *en oblique* lorsque l'émission d'un message requiert une certaine durée

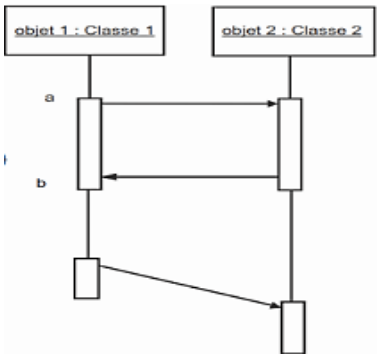


Figure 28: Représentation d'une contrainte temporelle.

4.5. Les fragments dans un DS

Il existe deux types de fragments:

- Fragment d'interaction
- Fragment combiné

4.5.1. *Fragment d'interaction*

- Permet de décomposer une interaction complexe en fragments suffisamment simples pour être compris.
- Se représente globalement comme un diagramme de séquence dans un rectangle avec indication dans le coin à gauche du nom du fragment.

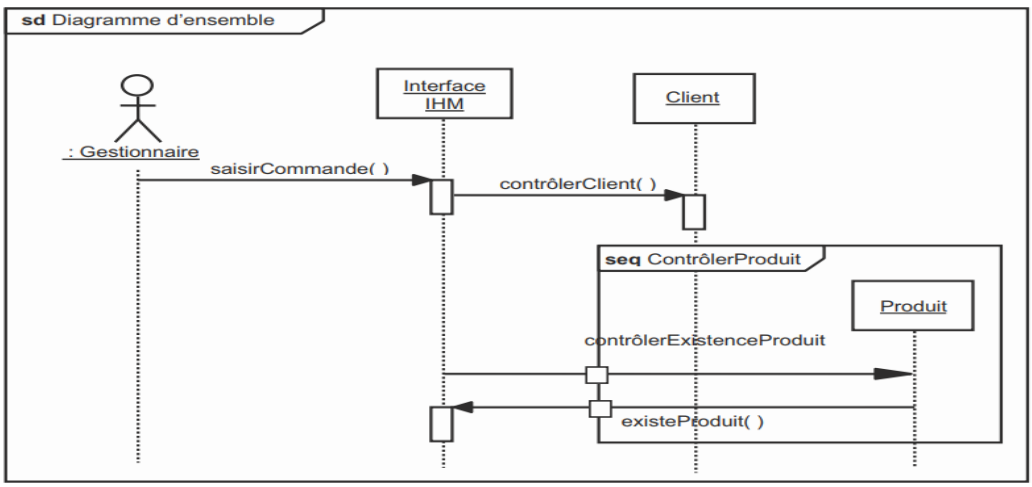


Figure 29: Représentation d'une fragment d'interaction.

4.5.2. Fragment combiné

Correspond à un fragment d'interaction auquel on applique un opérateur d'interaction. Il est représenté par un rectangle dont le coin supérieur gauche contient un pentagone ou figure le type de la combinaison

Treize opérateurs ont été définis dans UML , on peut les classifier comme suit:

- Opérateurs de branchement (choix et boucles) : opt (optional), alt (alternative), loop, break.
- Opérateurs contrôlant l'envoi en parallèle de messages: par (parallel) , critical.
- Opérateurs contrôlant l'envoi de messages: ignore/consider, assert (assertion)/neg(negative).
- Opérateurs fixant l'ordre d'envoi des messages: seq (weak sequencing/ strict (strict sequencing).
- Opérateur de référence: ref (reference)

5. ***Opérateurs de branchement***

- Alternative (alt) : est utilisée lorsqu'il existe un test de contrôle avec une ou plusieurs alternatives possibles. Une seule séquence peut se produire à la fois.

- Optional (opt) : est utilisée lorsqu'il existe un test de contrôle sans alternative

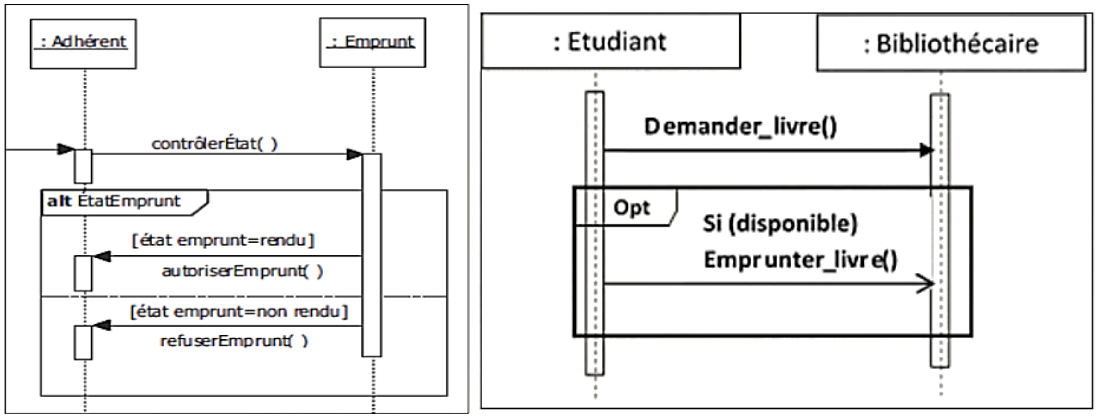


Figure 30: Les opérateurs de branchement (alt et opt).

- Loop (loop) : est utilisée pour une instruction de boucle qui permet d'exécuter une séquence d'interaction tant qu'une condition est satisfaite.

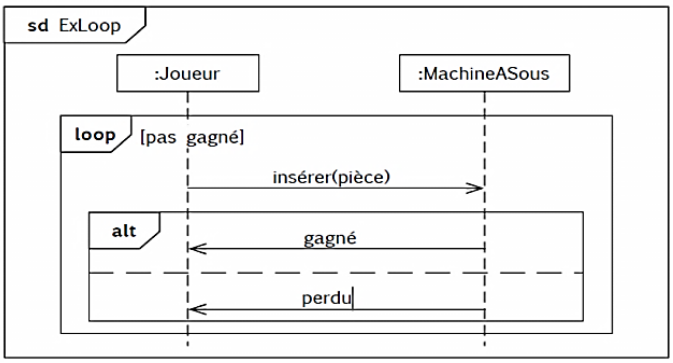


Figure 31: L'opérateur de branchement (loop).

- Break (break) : correspondant à une situation exceptionnelle correspondant à un scénario de rupture par rapport au scénario général.

  La figure 32 présente un exemple de l'opérateur de branchement « break ». L'utilisateur, lorsque le distributeur lui demande son code, peut choisir de rentrer son code ou de consulter l'aide. S'il choisit de consulter l'aide, le flot d'interaction relatif à la saisie du code est interrompu. Les interactions de l'opérateur break sont « exécutées ».

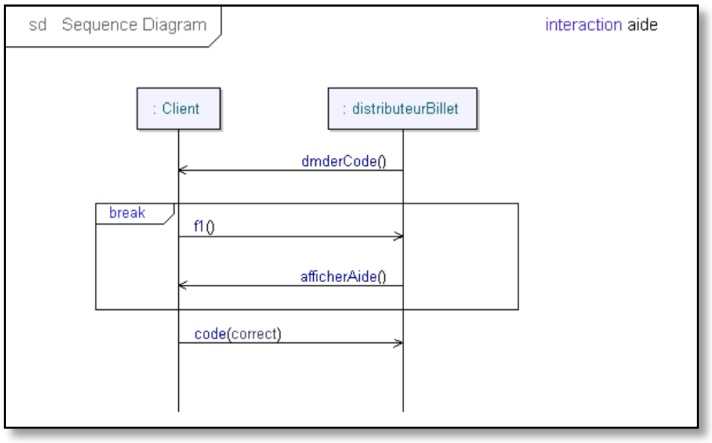


Figure 32: L'opérateur de branchement (break).

⇨ ***<u>Opérateurs contrôlant l'envoi en parallèle de messages</u>***

- Parallel (par) : Les événements des fragments peuvent être entrelacés (se déroulent en parallèle).
- Critical : Indique que les messages de fragment ne doivent pas être entrelacés avec d'autres messages.

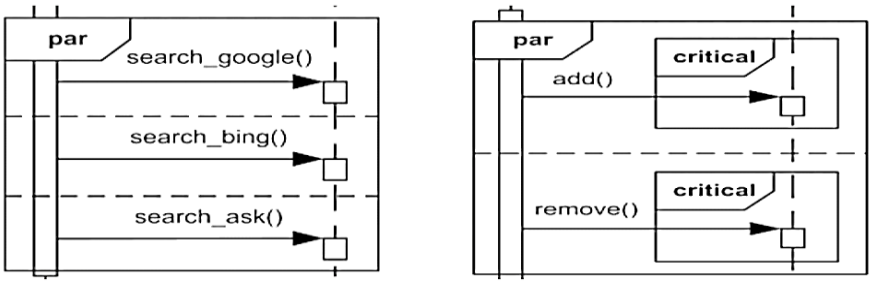


Figure 33: Exemple des opérateurs (par et critical).

⇨ ***Opérateurs contrôlant l'envoi de messages***

- Ignore/consider : permet de montrer que certains messages peuvent être soit absents sans avoir un effet sur le déroulement des interactions (ignore), soit obligatoirement présents (consider).

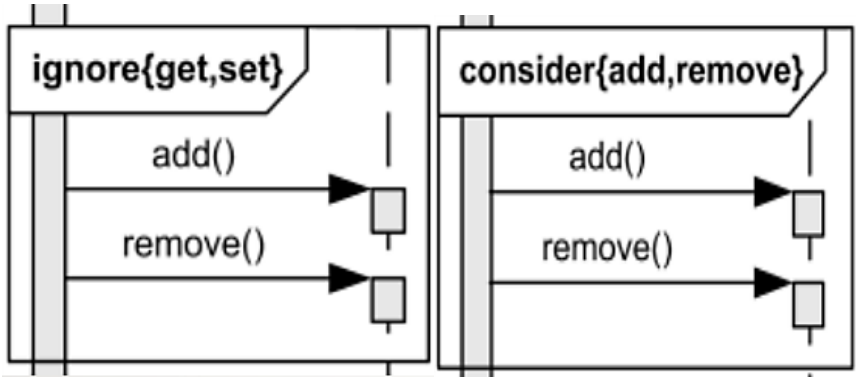


Figure 34: Exemple des opérateurs (Ignore/consider).

- Negative (neg) : indiquer qu'une séquence d'interactions est invalide.
- Assertion (assert) : 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.

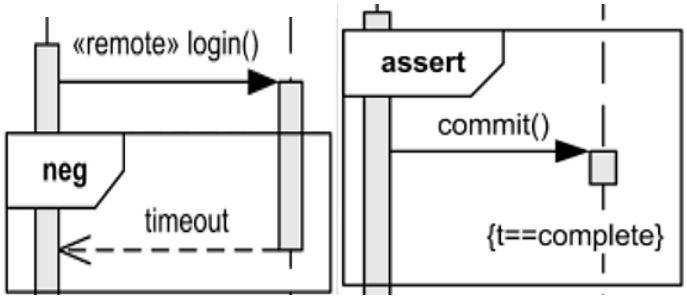


Figure 35: Exemple des opérateurs (neg/assert).

⇨ ***Opérateurs fixant l'ordre d'envoi des messages***

- Strict: est utilisé quand l'ordre d'exécution des opérations doit être strictement respecté.

- Weak: est utilisé quand l'ordre d'exécution des opérations n'a pas d'importance.

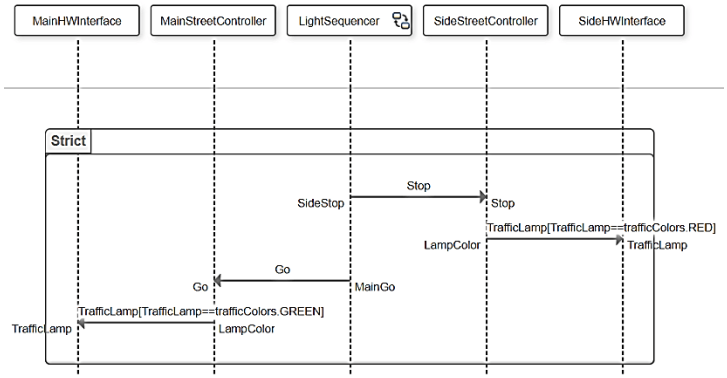


Figure 36: Exemple des opérateurs (strict).

⇨ ***Opérateur de référencement « Ref »***

- Rref: permet d'appeler une séquence d'interactions
  décrite par ailleurs constituant ainsi une sorte de sous-diagramme de séquence.

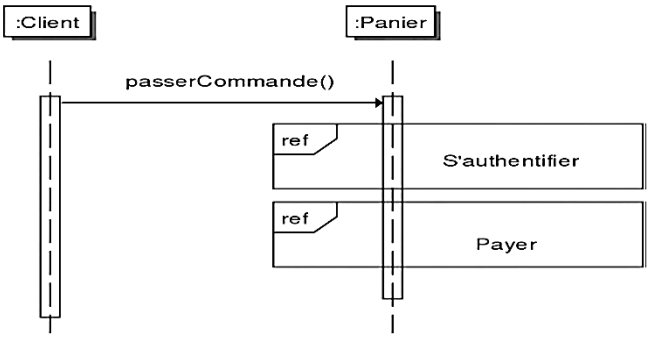


Figure 37: Exemple des opérateurs (ref).

4.6. Messages Perdus / Trouvés

- Le Message Trouvé : C'est un message dont on connaît le destinataire (une ligne de vie du diagramme), mais dont l'émetteur est inconnu ou hors du cadre du diagramme.
- Le Message Perdu : C'est un message dont on connaît l'émetteur (une ligne de vie du diagramme), mais dont le destinataire est inconnu ou hors du cadre.

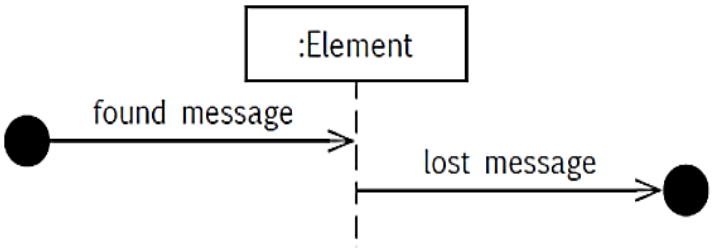


Figure 38: Exemple des messages (Trouvé/ Perdu).

5. Diagramme d'activités DA

- Visualise un graphe d'activités qui modélise le comportement interne d'une méthode, d'un cas d'utilisation ou plus généralement d'un processus impliquant un ou plusieurs classificateurs.
- 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.
- Ne représente ni la collaboration ni le comportement des objets.
- Représente les activités sous forme de rectangles.

5.1. Avantages des diagrammes d'activités

- Aident à collaborer pour comprendre un même procédé et un même comportement.
- Démontrer la logique d'un algorithme
- Décrire les étapes effectuées dans un cas d'utilisation d'UML
- Illustrer un processus métier ou un flux de travail entre les utilisateurs et le système
- Simplifier et améliorer n'importe quel processus en clarifiant les cas d'utilisation complexes

5.2. Représentation d'un diagramme d'activités

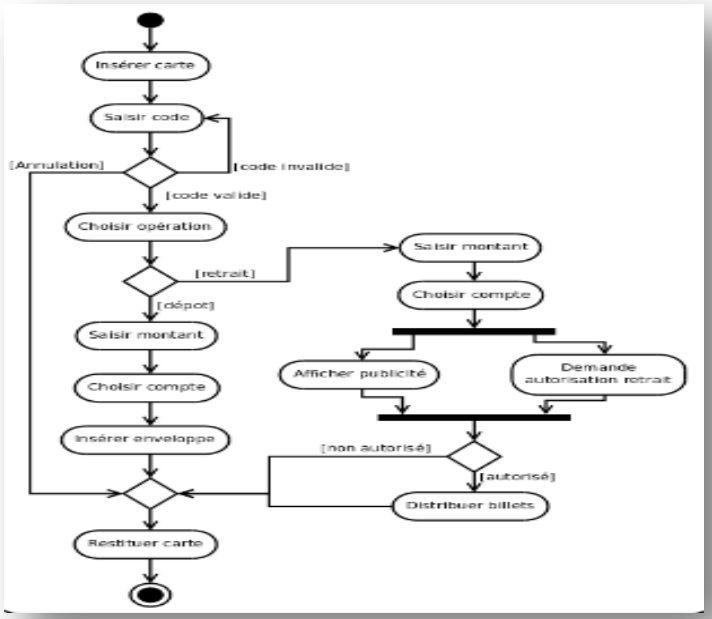


Figure 39: Représentation d'un diagramme d'activités.

5.3. Les Composants d'un (DA)

5.3.1. Action

- Une action est le plus petit traitement qui puisse être exprimé en UML.
- 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 distinguons ainsi graphiquement les actions associés à une communication: send signal, accept event et accept time event. Cela permet de mieux mettre en valeur les échanges entre les diagrammes de la spécification.

5.3.2. Activités :

- 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 (*behavior* en anglais) et à ce titre peut être associée à des paramètres.
  - ⇨ Une activité est un traitement complexe et décomposable en activités ou en actions. Une activité peut être interrompue par un événement. Une action est un traitement simple et non décomposable. Elle ne peut pas être interrompue.
- Une activité peut regrouper des nœuds et des arcs, c'est ce que nous appelons « groupe d'activités »
- Un diagramme d'activités est lui-même un groupe d'activités

5.3.3. Nœuds d'activité:

- 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.
  - les nœuds objets.
  - et les nœuds de contrôle.

5.3.4. Les transitions:

- 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.

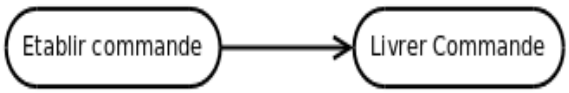


Figure 40: Représentation d'une transition.

5.4. Nœuds d'activité

- Il existe trois familles de nœuds d'activités :
  - les nœuds d'exécutions.
  - les nœuds de contrôle.
  - les nœuds d'objets.

5.4.1. Les nœuds d'exécutions

➪ ***Nœud d'action***

- Un nœud d'action est un nœud d'activité exécutable qui constitue l'unité fondamentale de fonctionnalité exécutable dans une activité.
- L'exécution d'une action représente une transformation ou un calcul quelconque dans le système modélisé.
- Certaines actions de communication ont une notation spéciale comme montré dans la figure 41.

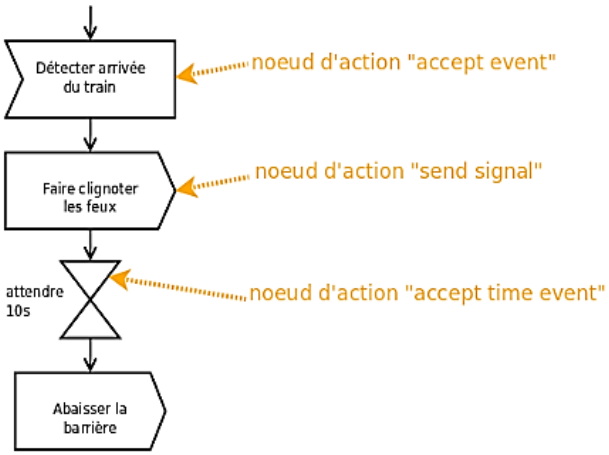


Figure 44: Représentation d'un nœud d'action.

5.4.2. Nœud de contrôle

- 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



Figure 45: Un nœud initial.

Nœud de fin d'activité

- Un nœud final est un nœud de contrôle possédant un ou plusieurs arcs entrants et aucun arc sortant.
- Il existe deux types de nœud finaux :
  - ✓ Nœud de fin d'activité
  - ✓ Nœud de fin de flot
- 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.



Figure 46: Nœud de fin d'activité.

- 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.



Figure 47: Nœud de fin de flot.

⌗ Nœud de décision

❖ 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.

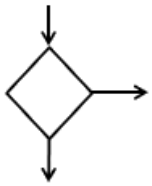


Figure 48: Nœud de décision.

⌗ Nœud de fusion

❖ 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.

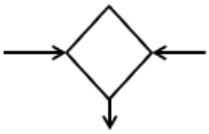


Figure 49: Nœud de fusion.

⌗ Nœud de bifurcation

❖ 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. On apparie généralement un nœud de bifurcation avec un nœud d'union pour équilibrer la concurrence.

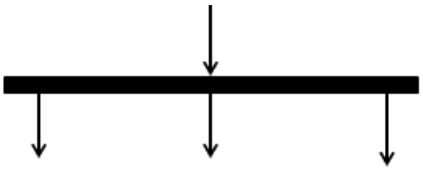


Figure 50: Nœud de bifurcation.

⌗ Nœud d'union

❖ 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.

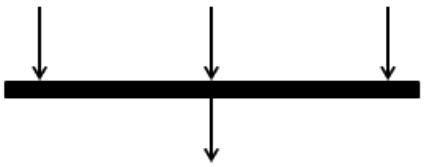


Figure 51: Nœud de bifurcation.

❖ Graphiquement, il est possible de fusionner un nœud de bifurcation et un nœud d'union, et donc d'avoir un trait plein possédant plusieurs arcs entrants et sortants

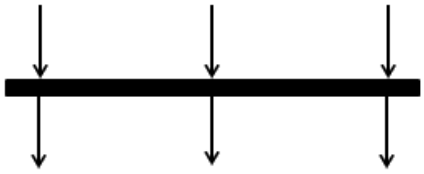


Figure 52: fusion d'un nœud de bifurcation et un nœud d'union.

5.4.3. Les nœuds d'objets

- Un nœud d'objet permet de définir un flot d'objets (*i.e.* un flot de données) dans un diagramme d'activités. Ce nœud représente l'existence d'un objet généré par une action dans une activité et utilisé par d'autres actions.
- Graphiquement, un tel nœud d'objet est représenté par un rectangle dans lequel est mentionné le type de l'objet.
- Des arcs viennent ensuite relier ce nœud d'objet à des activités sources et cibles. Le nom d'un état, ou d'une liste d'états, de l'objet peut être précisé entre crochets après ou sous le type de l'objet.

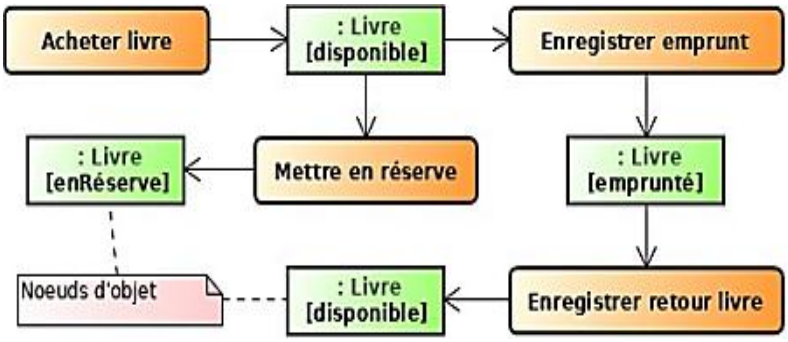


Figure 53: Exemple des nœuds d'objets.

5.5. Pin d'entrée ou de sortie

- Pour spécifier les valeurs passées en argument à une activité et les valeurs de retour, on utilise des nœuds d'objets appelés pins (*pin* en anglais) d'entrée ou de sortie. L'activité ne peut débuter que si l'on affecte une valeur à chacun de ses pins d'entrée. Quand l'activité se termine, une valeur doit être affectée à chacun de ses pins de sortie.
- Les valeurs sont passées par copie : une modification des valeurs d'entrée au cours du traitement de l'action n'est visible qu'à l'intérieur de l'activité.

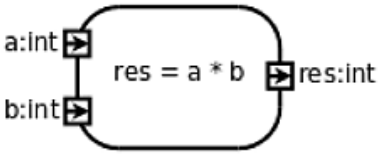


Figure 54: Exemple des pin d'entrée ou de sortie.

5.6. Nœud tampon central

Un nœud tampon central est un nœud d'objet qui accepte les entrées de plusieurs nœuds d'objets ou produit des sorties vers plusieurs nœuds d'objets. Les flots en provenance d'un nœud tampon central ne sont donc pas directement connectés à des actions. Ce nœud modélise donc un tampon traditionnel qui peut contenir des valeurs en provenance de diverses sources et livrer des valeurs vers différentes destinations.

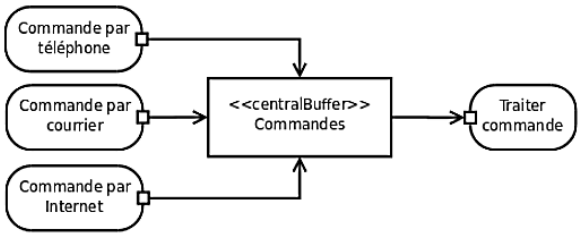


Figure 55: Exemple d'un nœud tampon central.

5.7. Nœud de stockage des données

- Un nœud de stockage des données est un nœud tampon central particulier qui assure la persistance des données. Lorsqu'une information est sélectionnée par un flux sortant, l'information est dupliquée et ne disparaît pas du nœud de stockage des données comme ce serait le cas dans un nœud tampon central. Lorsqu'un flux entrant véhicule une donnée déjà stockée par le nœud de stockage des données, cette dernière est écrasée par la nouvelle.

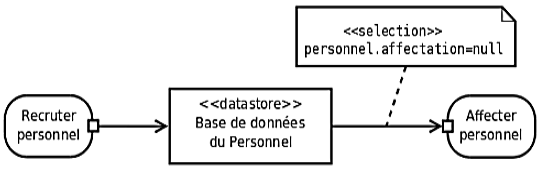


Figure 56: Exemple d'un nœud de stockage des données.

5.8. Les partitions ou couloirs d'activités

- Les partitions, souvent appelées couloirs ou lignes d'eau (*swimlane*) 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 de tâches. Dans ce cas, la classe en question est responsable de l'implémentation du comportement des nœuds inclus dans ladite partition.

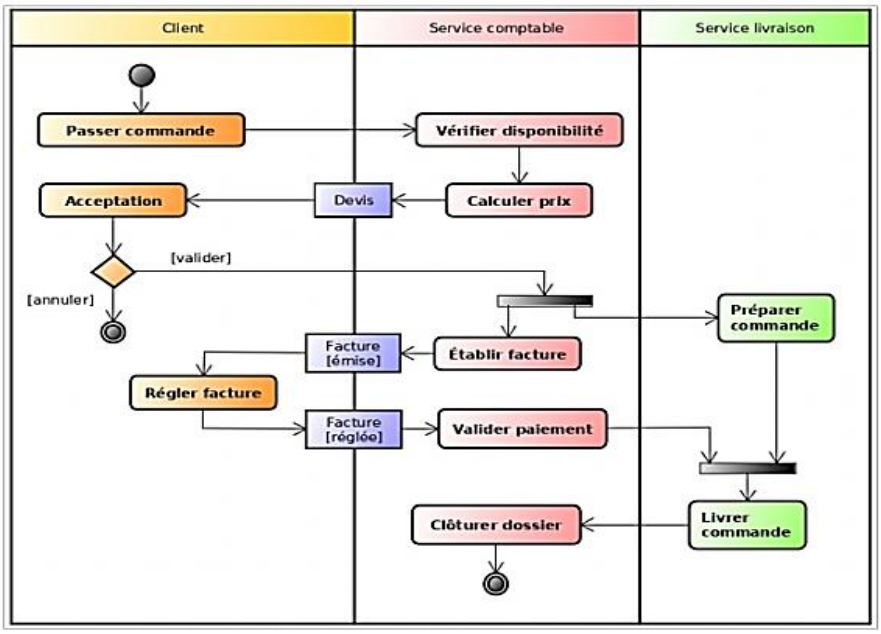


Figure 57: Exemple des partition (couloirs).

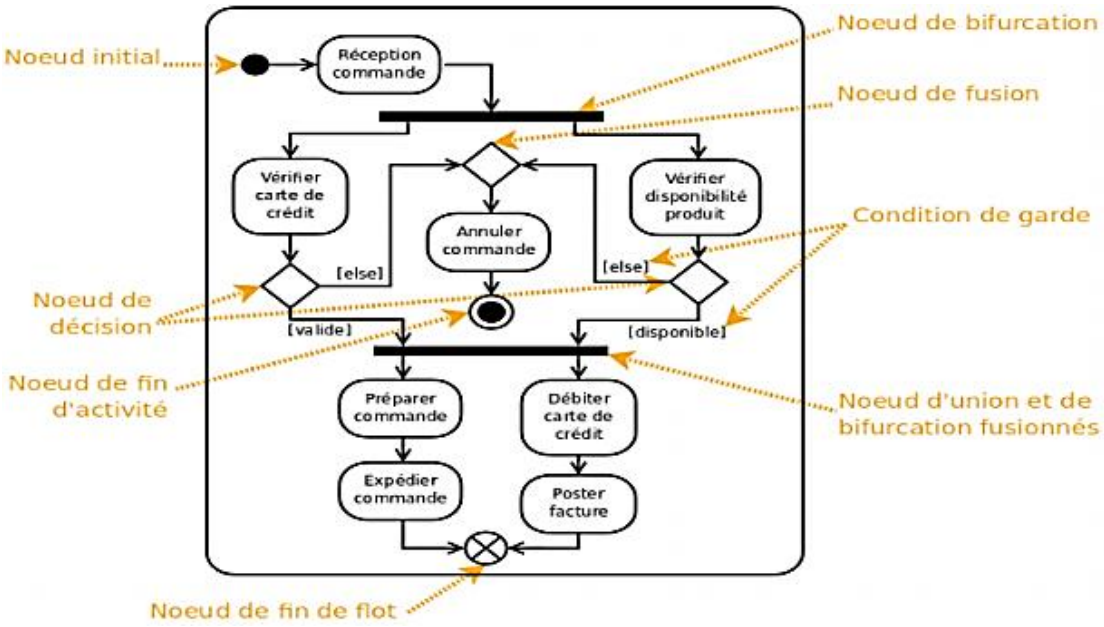


Figure 58: Exemple d'un diagramme d'activité.

6. Diagramme d'états-transitions DET

- Décrire le comportement dynamique d'une entité (logiciel, composant, objet...)
- 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.
- Décrire le cycle de vie d'un objet depuis sa création jusqu'à sa destruction.
- Modéliser la dynamique d'un système quelconque ou d'un algorithme.

6.1. Avantages d'un DET

- Le diagramme d'états-transitions 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.
- Ils montrent comment le système réagit aux événements, aux conditions ou aux stimuli externes en changeant d'état.
- Plus accessibles et compréhensibles pour différentes parties prenantes, y compris les personnes non techniques. Ils facilitent la communication entre les membres d'une équipe de développement, les gestionnaires de projet et d'autres intervenants.

6.2. Représentation d'un diagramme d'ET

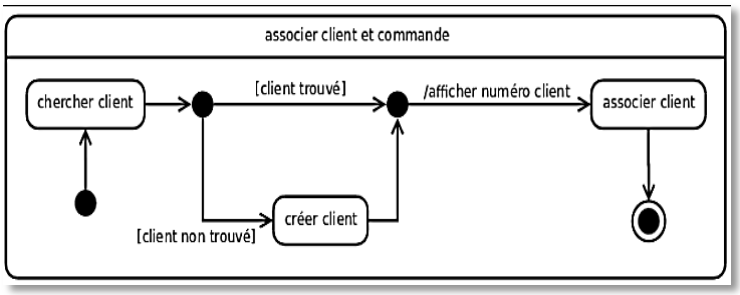


Figure 59: Représentation d'un diagramme d'ET.

6.3. Les Composants d'un (DET)

6.3.1. État

- Un état se représente graphiquement dans un diagramme d'états-transitions par un rectangle aux coins arrondis

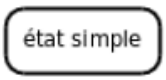


Figure 60: Représentation d'un état dans un diagramme d'ET.

- Un objet peut passer par une série d'états pendant sa durée de vie. Un état représente une période dans la vie d'un objet pendant laquelle ce dernier attend un événement ou accomplit une activité
- Dans le cas d'un diagramme d'états-transitions simple (sans transition concurrente), il ne peut y avoir qu'un seul état actif à la fois.

⇨ ***État initial***

❖ L'état initial est un pseudoétat 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.



Figure 61: Représentation graphique de l'état initial.

⇨ ***État final***

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



Figure 62: Représentation graphique de l'état final.

6.3.2. Événement

- Les diagrammes d'états-transitions 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.

⇨ **Événement de type signal (signal)**

❖ Un événement de signal est un événement qui représente la spécification d'un stimulus asynchrone entre objets. C'est la réception d'un message asynchrone.

⇨ **Événement d'appel (call)**

❖ Un événement d'appel représente la réception de l'appel d'une opération par un objet. Les paramètres de l'opération sont ceux de l'événement d'appel.

⇨ **Événement de changement (change)**

❖ Un événement de changement est généré par la satisfaction (*i.e.* passage de faux à vrai) d'une expression booléenne sur des valeurs d'attributs. Il s'agit d'une manière déclarative d'attendre qu'une condition soit satisfaite.

❖ Un événement de changement est évalué continuellement jusqu'à ce qu'il devienne vrai, et c'est à ce moment-là que la transition se déclenche. Il est désigné par le mot-clé « when ».

⇨ **Événement temporel (after ou when)**

❖ Les événements temporels sont générés par le passage du temps. Ils sont spécifiés soit de manière absolue (date précise) en utilisant le mot clé « when (date = <date>) », soit de manière relative (temps écoulé) en utilisant le mot clé « after (<durée>) ». Par défaut, le temps commence à s'écouler dès l'entrée dans l'état courant.

6.3.3. Transition

- Une transition définit la réponse d'un objet à l'occurrence d'un événement. Elle lie, généralement, deux états *E*1 et *E*2 et indique qu'un objet dans un état *E*1 peut entrer dans l'état *E*2 et exécuter certaines activités, si un événement déclencheur se produit et que la condition de garde est vérifiée.
- 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.
- Syntaxe d'une transition

  nom-événement '(' liste-param-événement ')' '[' garde ']' '/' activité.

- Lorsque l'événement se produit, si la condition est vérifiée, alors l'action est effectuée

⇨ ***Condition de garde***

- Une transition peut avoir une condition de garde. Il s'agit d'une expression logique sur les attributs de l'objet, associé au diagramme d'états-transitions, ainsi que sur les paramètres de l'événement déclencheur.
- La condition de garde est évaluée uniquement lorsque l'événement déclencheur se produit. Si l'expression est fausse à ce moment-là, la transition ne se déclenche pas, si elle est vraie, la transition se déclenche et ses effets se produisent.

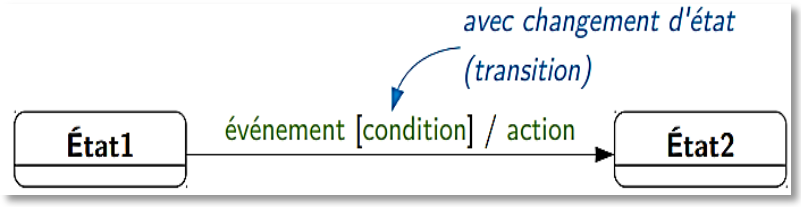


Figure 63: Syntaxe d'une transition.

⇨ ***Effet d'une transition***

- Lorsqu'une transition se déclenche son effet s'exécute. Il s'agit généralement d'une activité qui peut être:

- ✓ une opération primitive comme une instruction d'assignation ;
- ✓ l'envoi d'un signal ;
- ✓ l'appel d'une opération ;
- ✓ une liste d'activités, etc.

- Lorsque l'exécution de l'effet est terminée, l'état cible de la transition devient actif.

Transition externe

- Une transition externe est une transition qui modifie l'état actif. Il s'agit du type de transition le plus répandu.

⇨ ***Transition d'achèvement***

- Une transition dépourvue d'événement déclencheur explicite se déclenche à la fin de l'activité continue dans l'état source. Elle peut contenir une condition de garde qui est évaluée au moment où l'activité continue dans l'état s'achève, et non pas ensuite.

⇨ ***Transition interne***

- 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

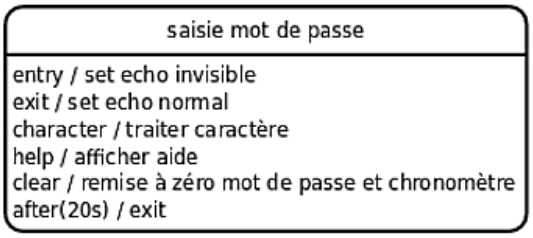


Figure 64: Représentation de la saisie d'un mot de passe dans un état unique en utilisant des transitions internes.

6.3.4. Point de jonction:

- Les points de jonction permettent 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.
- les points de jonction sont représentés par un petit cercle plein.

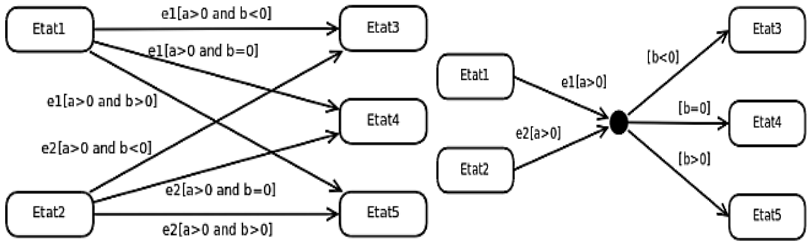


Figure 65: Un DET sans point de jonction. Et son équivalent utilisant un point de jonction.

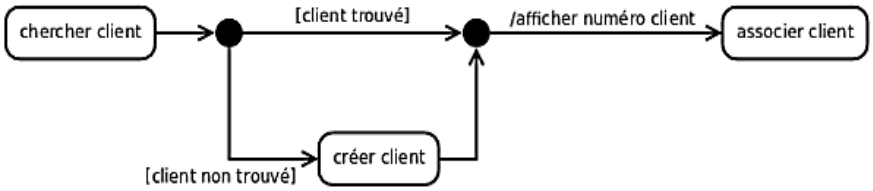


Figure 66: Exemple d'utilisation de deux points de jonction pour représenter une alternative.

6.4. États composites

- Un état composite est un état qui regroupe un ensemble de sous-états.

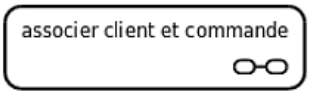


Figure 67: Notation abrégée d'un état composite.

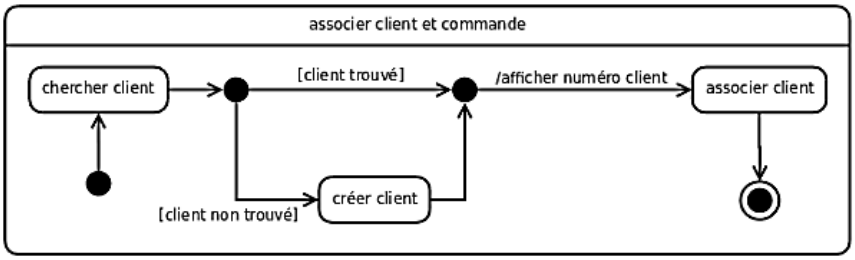


Figure 68: Exemple d'état composite modélisant l'association d'une commande à un client.

➪ ***État orthogonal***

C'est un état composite dans lequel plusieurs états sont actifs simultanément (concurrence/parallélisme)

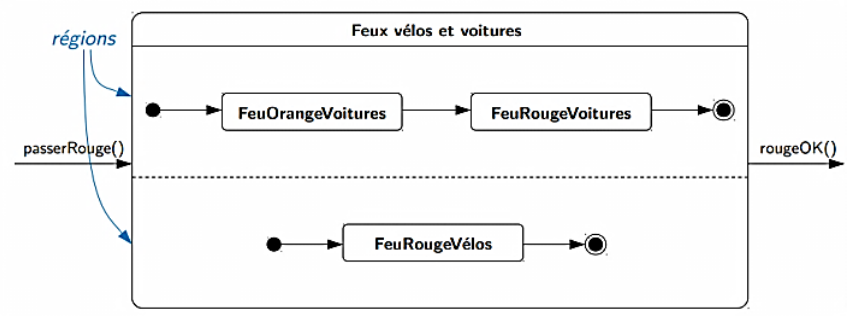


Figure 69: Un état orthogonal.

- Les points de connexion: sont des points d'entrée et de sortie portant un nom, et situés sur la frontière d'un état composite.
- Ils sont respectivement représentés par un cercle vide et un cercle barré d'une croix.

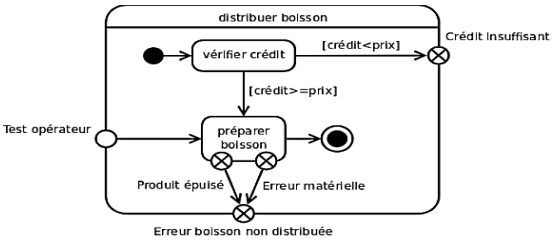


Figure 70: Exemple d'utilisation de points de connexion.

- Toutes les régions concurrentes d'un état composite orthogonal doivent atteindre leur état final pour que l'état composite soit considéré comme terminé.

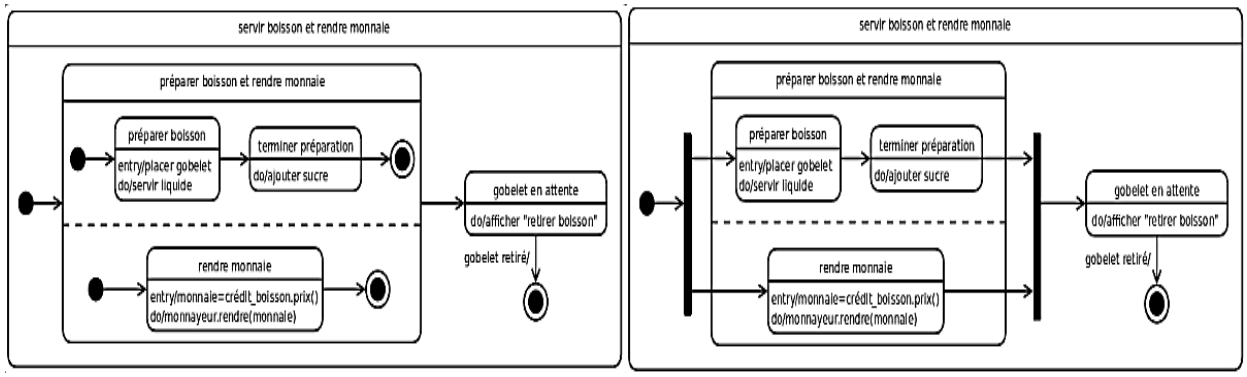


Figure 71: Exemple d'utilisation de transitions complexes.

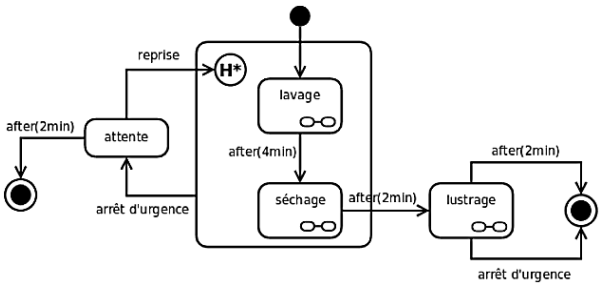


Figure 72: Exemple d'un DET.

7. Diagramme de communication (collaboration)

- En UML 1.x : Diagramme de collaboration.
- En UML 2.0 : Diagramme de communication.
- Montre des interactions entre objets (instances de classes et acteurs). Il s'intéresse à la structure de collaboration entre objets (séquencement, itération, concurrence, etc.)
- La chronologie des interactions est indiquée par la numérotation de messages pour indiquer leur ordre d'envoi.
- Equivalent au diagramme de séquences. Cependant, l'aspect temporel n'apparaît pas, mais l'aspect chronologique est présent.

7.1. Avantages d'un DC

- 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
- En illustrant les interactions entre les objets, les diagrammes de communication aident à définir clairement les responsabilités de chaque objet. Cela contribue à une conception plus modulaire et facilite la compréhension des rôles et des fonctions de chaque composant.

7.2. Représentation d'un DC

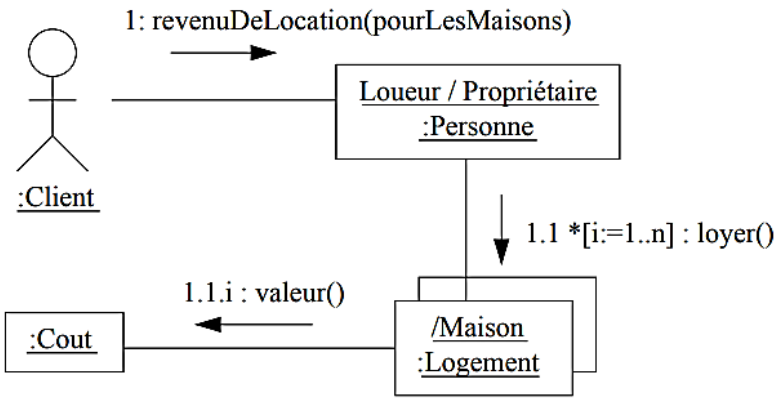


Figure 73: Exemple d'un diagramme de communication.

7.3. Les Composants d'un (DC)

7.3.1. Les objets

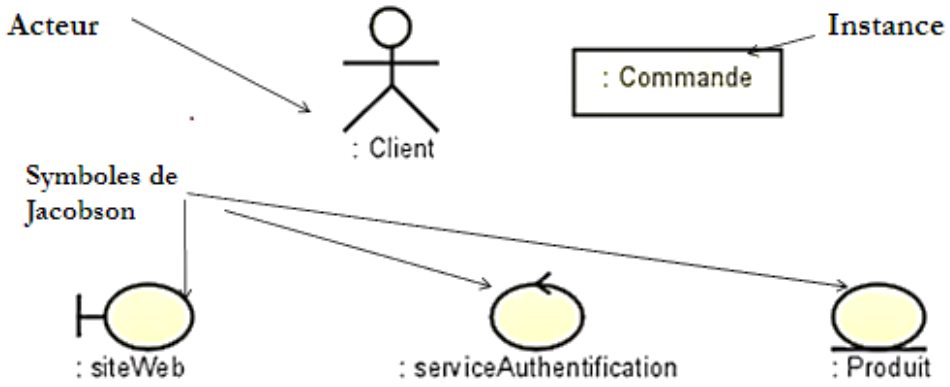


Figure 74: Exemple des objets dans un diagramme de communication.

7.3.2. Les liens

- Connexion entre deux objets
- Permet de faire circuler les messages entre ces deux objets

7.3.3. Les messages

- Déclenche l'exécution d'une opération de l'objet destinataire

[prédécesseurs /] [garde] [séquence] [itération] [résultat :=] signatureMessage (arguments)

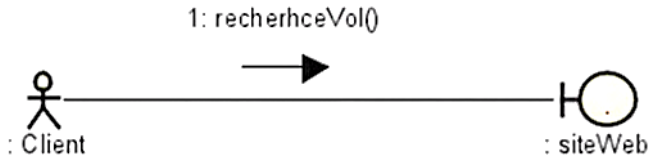


Figure 75: Exemple d'un message.

7.3.4. La numérotation

- Ensemble de numéros ordonnant l'envoi des messages (1 puis 2 puis 3 ...)
- Séquencement: 1 (appel initial) puis 1.1 (premier appel imbrique) puis 1.2 (second sous-appel) puis 2 (appel du même niveau que le numéro 1).
- Concurrence: 1,1.1a, 1.1b, 1.2, 1.3 (les points 1.1a et 1.1b peuvent être exécutés en parallèle)

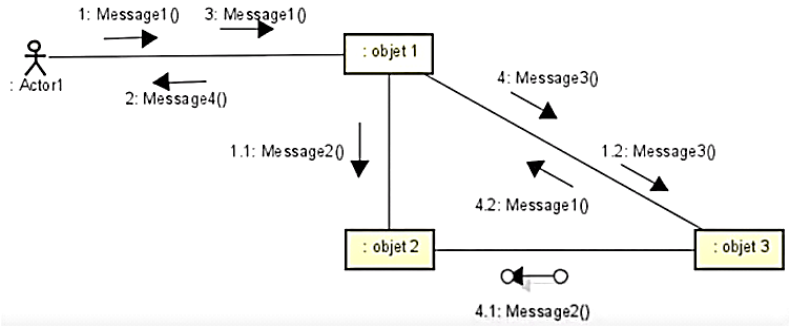


Figure 76: Exemple de la numérotation.

7.3.5. Conditions

- Consiste à l'envoi des messages conditionnels.

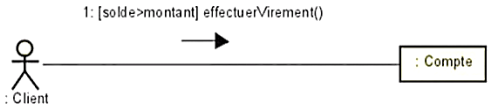


Figure 77: Exemple d'une condition.

7.3.6. Itération



Figure 78: Exemple d'une itération.

7.3.7. Arguments

- Représentent les paramètres de la méthode à exécuter.



Figure 79: Exemple des arguments.

7.3.8. Résultat

- Représente la valeur retournée par le message.

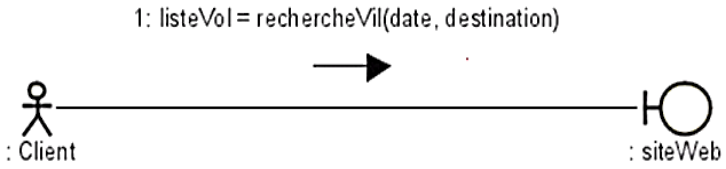


Figure 80: Exemple de représentation du résultat.

7.4. Les contraintes associées

- Un objet crée porte la contrainte {nouveau)

- Un objet détruit porte la contrainte {détruit)

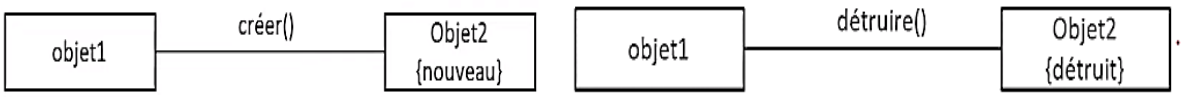


Figure 81: création et destruction d'un objet.

7.5. Exemple d'un DS et équivalent au DC

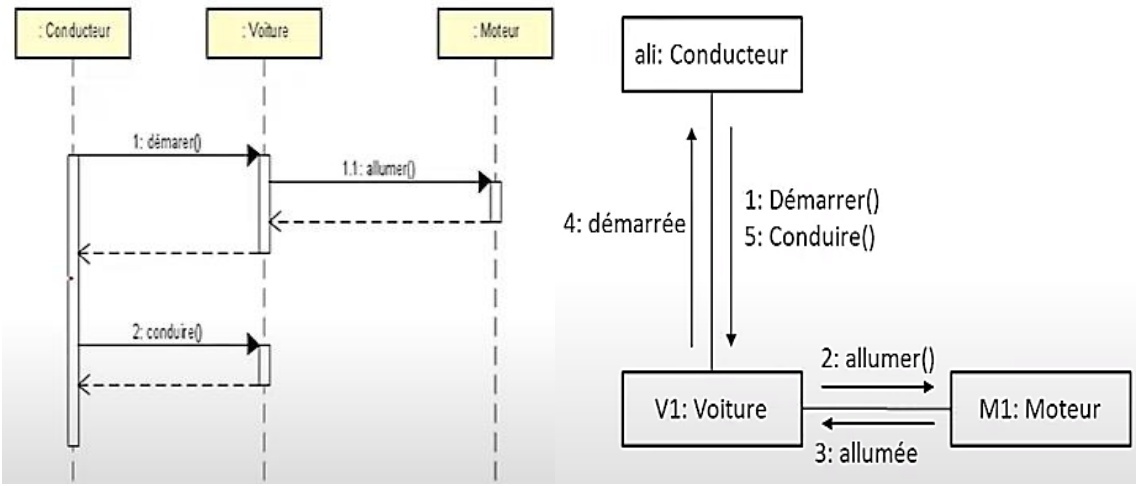


Figure 82: Exemple d'un DS et équivalent au DC.

8. Diagramme d'interaction globale

- Permet de représenter une vue générale des interactions décrites dans le diagramme de séquence et des flots de contrôle décrits dans le diagramme d'activité.
- Privilégie la vue générale des flux de contrôle dans lesquels les nœuds sont des interactions ou des utilisations d'interactions (opérateur ref).
- Utile pour décomposer de multiples scénarios et activités en diagrammes d'interaction plus petits

8.1. Avantages d'un DIG

- Fournissent une vision simple de l'activité au sein d'une modélisation.
- Offrent un degré élevé de navigabilité entre les diagrammes.
- Permettent d'utiliser la plupart des annotations d'un diagramme d'activité, ainsi que des éléments supplémentaires pour plus de clarté.

8.2. Représentation d'un DIG

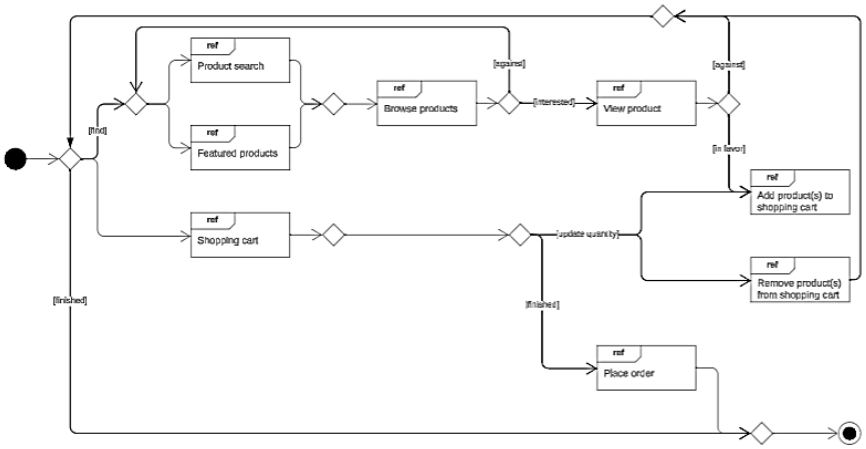


Figure 83: Exemple d'un diagramme d'interaction globale.

8.3. Les Composants d'un (DIG)

Le diagramme d'interaction globale utilise les concepts du diagramme d'activité auquel on ajoute deux compléments :

- Les fragments d'interaction du diagramme de séquence
  - Il s'agit de la notion de fragment d'interaction vue dans le diagramme de séquence mais qui ne doit pas être détaillé à ce niveau.

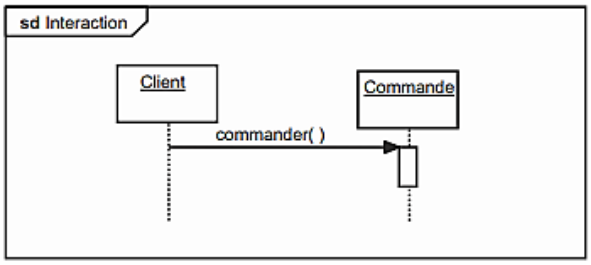


Figure 84: Exemple de fragment d'interaction.

- Les utilisations de fragments d'interaction
  - Il est aussi possible de faire appel à des fragments d'interaction à l'aide de l'opérateur ref.

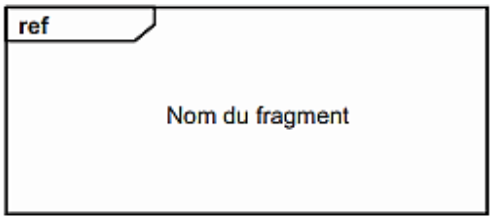


Figure 85: Exemple de fragment d'interaction avec l'opérateur ref.

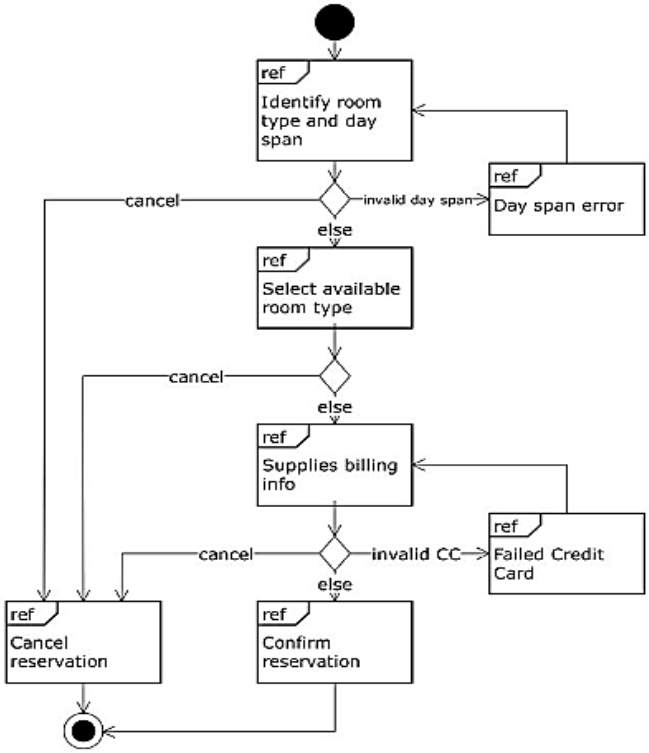


Figure 86: Exemple d'un DIG.

9. Diagramme de temps

- 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.
- permet de mieux représenter des changements d'états et des interactions entre objets liés à des contraintes de temps.
- le temps s'écoule de la gauche vers la droite.

9.1. Avantages d'un DT

❖ Ils permettent de représenter graphiquement l'évolution temporelle des états ou des événements d'un système. Cela facilite la compréhension de la séquence chronologique des actions ou des changements dans le système.

❖ Ils peuvent être utilisés pour analyser les performances d'un système en examinant le temps nécessaire pour passer d'un état à un autre. Cela est utile pour l'optimisation des performances

9.2. Représentation d'un DT

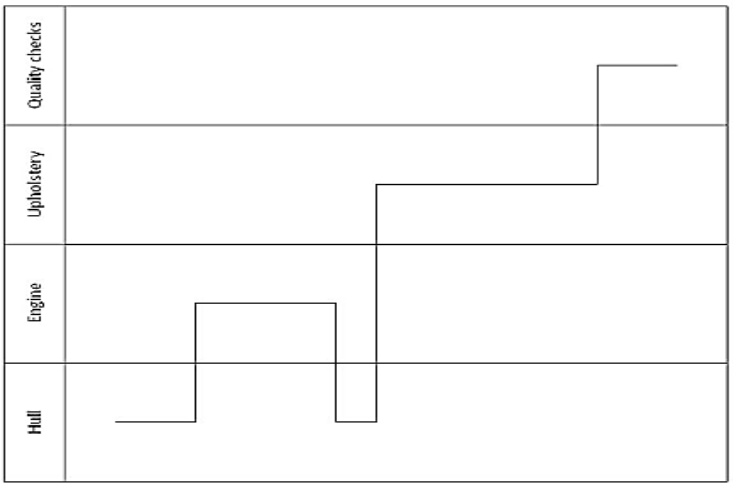


Figure 87: Exemple d'un DT pour un processus de fabrication des bateaux.

9.3. Les Composants d'un (DT)

Le diagramme de temps utilise trois concepts de base :

✓ **Ligne de vie** : Elle représente l'objet que l'on veut décrire. Elle se dessine de manière horizontale. Plusieurs lignes de vie peuvent figurer dans un diagramme de temps.

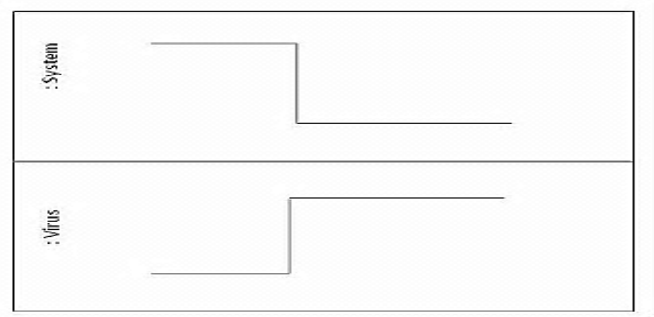


Figure 88: Exemple des lignes de vies.

- ✓ **État ou ligne de temps conditionnée**: Les différents états que peut prendre l'objet d'étude sont listés en colonne permettant ainsi de suivre le comportement de l'objet ligne par ligne (une ligne pour un état).

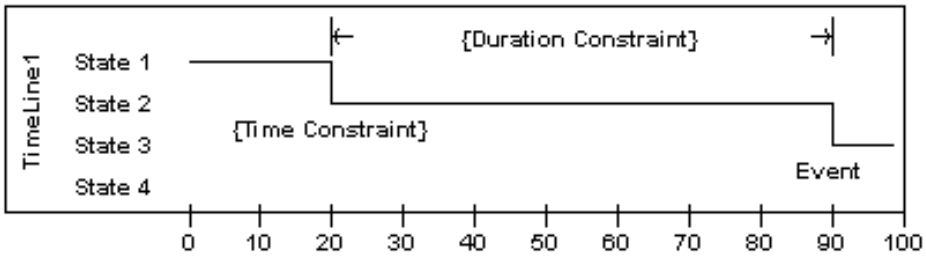


Figure 89: Exemple d'une ligne de temps conditionnée.

- ✓ **États linéaires** : Il s'agit du même concept que le précédent, mais la représentation de la succession des états est faite de manière linéaire à l'aide d'un graphisme particulier.

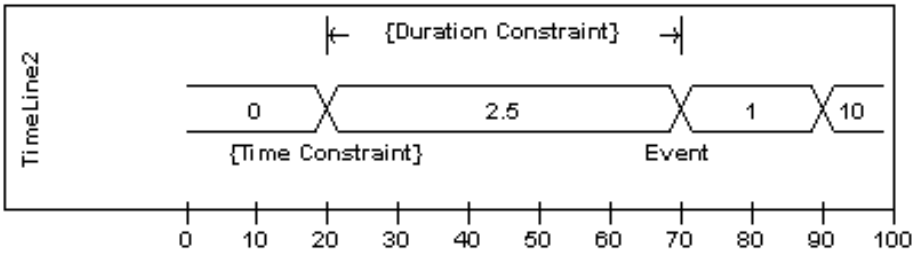


Figure 90: Exemple d'un état linéaire.

9.4. Contrainte de durée

Représente un intervalle de durée, ou la période de temps pendant laquelle un état ou une valeur doit être en vigueur.

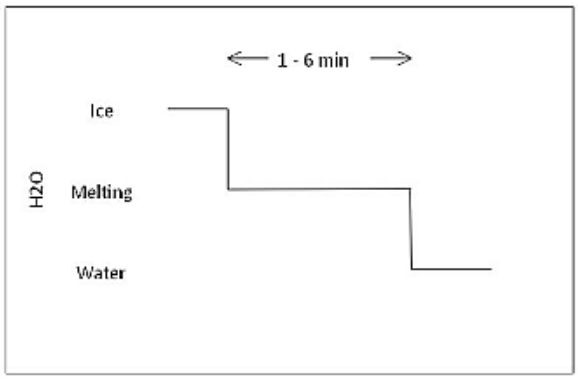


Figure 91: Exemple de la contrainte de durée.

9.5. Événement de destruction

Le symbole "X" marque la fin d'une ligne de vie.

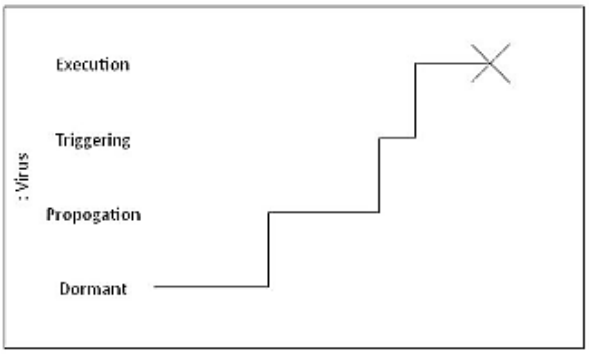


Figure 92: Exemple d'événement de destruction.

9.6. Exemple d'un DT

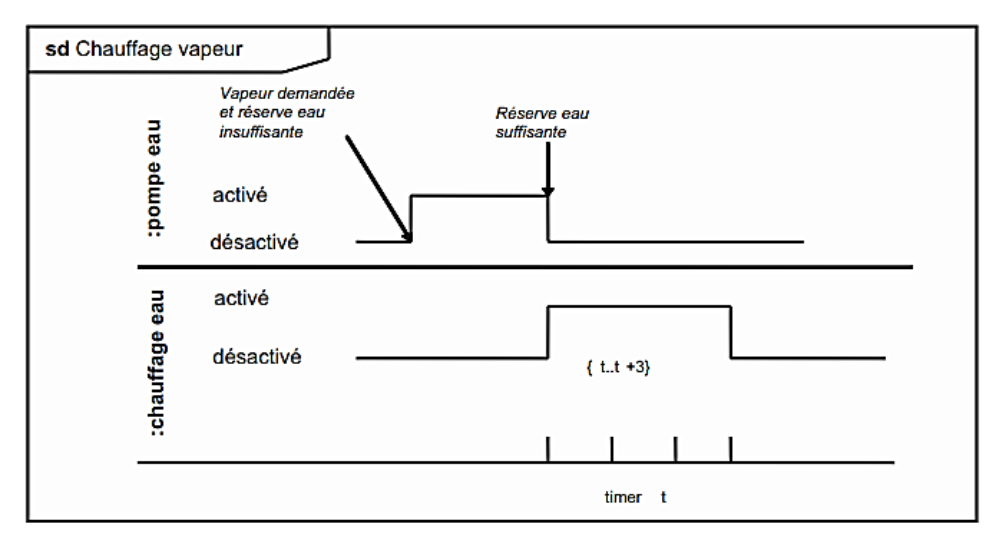


Figure 93: Exemple de diagramme de temps avec représentation « en escalier ».

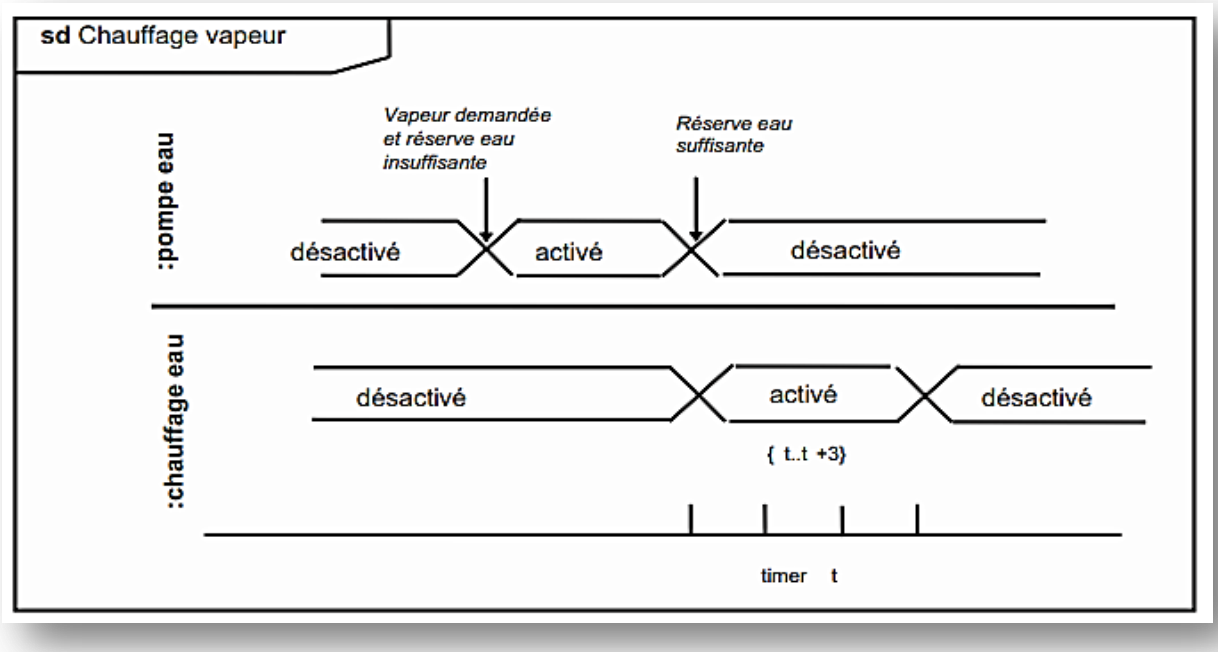


Figure 94: Exemple de diagramme de temps avec représentation linéaire.

Chapitre 05: Étude de cas SI

1. Introduction :

L'étude de cas constitue une approche pédagogique essentielle pour comprendre le fonctionnement réel des systèmes d'information. Elle permet de passer de la théorie à la pratique en appliquant les concepts appris sur des systèmes concrets, tels que la gestion de documents numériques, la banque, ou les systèmes de réservation.

L'étude de cas offre un contexte riche où l'analyse, la conception et la modélisation d'un SI peuvent être pratiquées de manière intégrée. Elle permet notamment de travailler sur les diagrammes UML (Use Case, Classe, Séquence, Activité, Composant) afin de représenter visuellement les interactions, les structures et les processus du système.

2. Les objectifs de chapitre 05

A la fin de ce chapitre, l'étudiant seras capable de :

- ✓ Identifier et comprendre les concepts clés d'un SI et les diagrammes UML.
- ✓ Expliquer les interactions entre acteurs et composants.
- ✓ Construire et analyser des diagrammes UML pour représenter le système.
- ✓ Concevoir une modélisation complète et cohérente adaptée au système étudié.

3. Outil de modélisation utilisé

Dans cette étude de cas, nous avons utilisé Visual Paradigm comme outil de modélisation UML. Visual Paradigm est un logiciel professionnel de conception et de modélisation qui permet de créer différents types de diagrammes, notamment les diagrammes de cas d'utilisation, les diagrammes de classes, les diagrammes de séquence et autres diagrammes UML.

4. Présentation générale du système de gestion de pharmacie en ligne

4.1. Problématique

La gestion pharmaceutique traditionnelle repose sur une interaction physique exclusive, imposant aux clients des déplacements souvent contraignants. Ce mode de fonctionnement présente des limites majeures : perte de temps, difficultés d'accès pour les personnes à mobilité réduite, et lourdeur administrative en cas de retours ou d'échanges.

Face à ces contraintes, la mise en place d'un système de pharmacie en ligne s'impose comme une solution d'optimisation. Ce projet vise à digitaliser le processus de vente pour offrir une accessibilité accrue, un suivi des commandes en temps réel et une gestion sécurisée des médicaments, affranchissant ainsi le client des contraintes géographiques.

Le système met en interaction trois acteurs principaux, à savoir le client, l'administrateur et le livreur. Les interactions entre ces acteurs et le système constituent les principales fonctionnalités du système et peuvent être représentées sous forme de cas d'utilisation et séquences.

Le système est organisé en plusieurs modules logiciels :

- Gestion des comptes et utilisateurs
- Gestion des commandes et du panier
- Gestion des médicaments et du stock
- Gestion des paiements
- Gestion des livraisons

Ces modules communiquent entre eux et permettent de gérer les entités suivantes :

- Client, Administrateur, Livreur
- Commande : chaque commande contient une ou plusieurs lignes de commande, associées à des médicaments
- Panier : espace temporaire pour regrouper les produits avant la finalisation de la commande
- Paiement : peut être effectué par carte bancaire ou à la livraison

Cette organisation permet de représenter les modules internes et externes ainsi que les relations entre entités et composants.

Le système est distribué sur plusieurs nœuds physiques :

- Clients (PC, smartphone, tablette) : exécutent un navigateur qui héberge l'interface utilisateur du système. Les clients communiquent avec le serveur web via HTTP/HTTPS.
- Serveur Web / Application : héberge tous les modules internes (comptes, commandes, panier, médicaments, paiements, livraisons). Il communique avec la base de données et les services externes.
- Serveur Base de données : contient un MySQL qui stocke toutes les informations nécessaires (clients, commandes, lignes de commande, panier, médicaments, livreurs, paiements). La communication avec le serveur Web se fait via TCP/IP.
- Services externes :
    - API bancaire pour traiter les paiements, accessible via HTTPS

Les flux d'informations principaux sont :

- Le client envoie des requêtes au serveur web via le navigateur
- Le serveur web accède aux données dans MySQL via TCP/IP
- Le serveur web communique avec l'API bancaire via HTTPS

4.2. Acteurs et fonctionnalités principales

Trois acteurs principaux interagissent avec le système :

- **Client** : Il peut créer un compte, se connecter, mettre à jour son profil, rechercher des médicaments, ajouter des produits au panier, passer et confirmer des commandes, annuler une commande si nécessaire, suivre l'état des commandes et effectuer le paiement (soit par carte

bancaire en ligne, soit à la livraison). Le système envoie automatiquement des notifications par SMS ou email pour informer le client de l'état de ses commandes.

- **Administrateur** : Il gère les médicaments (ajouter, modifier, supprimer), supervise les commandes des clients, consulte et met à jour les informations des livreurs, et peut visualiser les utilisateurs connectés pour suivre l'activité du système.
- **Livreur** : Il consulte les commandes à livrer, affiche les adresses des clients, met à jour le statut des commandes (livré ou non livré), mentionne la raison d'un échec de livraison si nécessaire, et peut gérer ou mettre à jour son profil personnel et sa présence.

4.3. Objectifs du système

- ✓ Permettre aux clients de créer un compte et de passer des commandes en ligne.
- ✓ Fournir aux clients la possibilité de suivre l'état des commandes.
- ✓ Permettre aux administrateurs de gérer les fournisseurs et stocks.
- ✓ Réduire le temps et l'effort nécessaires pour obtenir des médicaments.
- ✓ Garantir une interface sécurisée et fiable pour toutes les transactions.

4.4. Les Digramme UML

4.4.1. Acteurs principaux

Dans le système de gestion de pharmacie en ligne, trois acteurs principaux interagissent avec les différents modules du portail. Chaque acteur a des responsabilités spécifiques.

⇨ **Client**

Le client peut créer un compte utilisateur pour accéder au portail et passer des commandes de médicaments en ligne. Il peut également consulter l'état de ses commandes et mettre à jour son profil personnel avec ses informations, comme le nom, l'adresse et tout identifiant nécessaire.

⇨ **Administrateur**

L'administrateur peut gérer la disponibilité des médicaments en ajoutant, modifiant ou supprimant des articles selon les besoins. Il peut également consulter les informations des clients et mettre à jour la présence ou les informations des livreurs pour superviser correctement le processus de livraison.

⇨ **Livreur**

Le livreur peut consulter les adresses de livraison pour les commandes attribuées. Il peut mettre à jour son profil personnel, suivre sa présence et confirmer la livraison des commandes effectuées.

4.4.2. La description textuelle des cas d'utilisation

1. Cas d'utilisation : Login (Se connecter)

- Titre du cas d'utilisation : Login
- Objectif : Permettre au client de se connecter au portail de la pharmacie en ligne pour accéder à ses services.

- Acteurs : Client
- Préconditions :
    - Le client doit déjà posséder un compte enregistré.
- Scénario nominal (principal) :
    - Le client saisit son nom d'utilisateur et son mot de passe.
    - Le système vérifie les informations et authentifie le client.
    - Le client est redirigé vers la page d'accueil.
- Scénarios alternatifs :
    - Si le client n'a pas de compte, il peut demander la création d'un nouveau compte.
- Postconditions : Le client est connecté et accède à la page d'accueil du portail.

2. Cas d'utilisation : Créer un compte

- Titre du cas d'utilisation : Créer un compte
- Objectif : Permettre au client de créer un compte sur le portail de la pharmacie en ligne afin de pouvoir passer des commandes et accéder aux services.
- Acteurs : Client
- Préconditions :
    - Toutes les informations requises (nom, adresse, identifiant, etc.) doivent être fournies pour créer un compte.
- Scénario nominal (principal) :
    - Le client saisit toutes les informations personnelles nécessaires.
    - Le système valide les informations et crée le compte.
    - Le client est redirigé vers la page de login.
- Scénarios alternatifs :
    - Si le client possède déjà un compte, il peut se connecter directement via la page de login.
- Postconditions : Le compte est créé et le client est dirigé vers la page de connexion pour se connecter.

3. Cas d'utilisation : mettre à jours le profil

- Titre du cas d'utilisation : mettre à jours le profil
- Objectif **:** Permettre au client de modifier ou mettre à jour ses informations personnelles dans le portail de la pharmacie en ligne.
- Acteurs : Client
- Préconditions :
    - Le client doit être authentifié.
    - Le client doit avoir fourni des informations valides.
- Scénario nominal (principal) :
    - Le client accède à son profil.
    - Le client modifie ou met à jour ses informations personnelles.
    - Le système enregistre les modifications et confirme la mise à jour.
    - Le client est redirigé vers la page d'accueil.
- Scénarios alternatifs :

- Le client peut simplement consulter les informations sans les modifier.

- Postconditions : Les informations du client sont mises à jour et il est dirigé vers la page d'accueil.

4. Cas d'utilisation : Rechercher un médicament

- Titre du cas d'utilisation : Rechercher un médicament
- Objectif : Permettre au client de rechercher un médicament dans le catalogue de la pharmacie en ligne.
- Acteurs : Client
- Préconditions :
  - Le client doit être authentifié.
  - Le système doit contenir des médicaments enregistrés.
- Scénario nominal (principal) :
  - Le client accède à la barre de recherche.
  - Le client saisit le nom du médicament ou un mot-clé.
  - Le système affiche la liste des médicaments correspondants.
- Scénarios alternatifs :
  - Aucun médicament correspondant n'est trouvé → le système affiche un message approprié.
- Postconditions : Les résultats de recherche sont affichés et le client peut sélectionner un médicament.

5. Cas d'utilisation : Passer une commande

- Titre du cas d'utilisation : Passer une commande
- Objectif : Permettre au client de passer une commande de médicaments sur le portail de la pharmacie en ligne.
- Acteurs : Client
- Préconditions :
  - Le client doit être authentifié.
  - Au moins un médicament doit être sélectionné.
- Scénario nominal (principal) :
  - Le client sélectionne l'option « Passer une commande ».
  - Le système ajoute les médicaments sélectionnés au panier.
  - Le client vérifie le contenu du panier.
  - Le système enregistre la commande.
- Scénarios alternatifs :
  - Le client peut modifier ou supprimer des médicaments du panier avant la confirmation.
  - Le client peut annuler le processus de commande et être redirigé vers la page d'accueil.
- Postconditions : La commande est créée, enregistrée par le système, et le client est redirigé vers la page de confirmation de commande.

6. Cas d'utilisation : Consulter les commandes

- Titre du cas d'utilisation : Consulter les commandes
- Objectif : Permettre au client de consulter les commandes qu'il a passées sur le portail.
- Acteurs : Client

- Préconditions :
  - Le client doit être authentifié.
- Scénario nominal (principal) :
  - Le client accède à la section “Mes commandes”.
  - Le client peut consulter ou supprimer des produits dans ses commandes.
  - Le système met à jour l’affichage en conséquence.
- Scénarios alternatifs :
  - Le client peut être redirigé vers la page d’accueil s’il souhaite quitter la consultation.
- Postconditions : Le système doit afficher le nombre de commandes effectuées par le clien.

7. Cas d'utilisation : Ajouter au panier

- Titre du cas d’utilisation : Ajouter au panier
- Objectif : Permettre au client d’ajouter des produits à son panier et de consulter les produits ajoutés.
- Acteurs : Client
- Préconditions :
  - Des produits doivent être sélectionnés et disponibles pour être ajoutés au panier et le client a passé une commande.
- Scénario nominal (principal) :
  - Le client sélectionne un produit.
  - Le client passe une commande.
  - Le client ajoute le produit au panier.
  - Le système affiche les produits ajoutés dans le panier.
- Scénarios alternatifs :
  - Le panier est vide, le client est redirigé vers la page de passage des commandes.
- Postconditions : Le client est redirigé vers les options de confirmation ou d’annulation de la commande.

8. Cas d'utilisation : Confirmer une commande

- Titre du cas d’utilisation : Confirmer une commande
- Objectif : Permettre au client de confirmer une commande.
- Acteurs : Client
- Préconditions :
  - Au moins une commande doit être présente pour la confirmation.
- Scénario nominal (principal) :
  - Le client sélectionne l’option de confirmation.
  - Le système affiche la commande confirmée.
- Scénarios alternatifs :
  - Le système redirige vers la page du panier.
- Postconditions : Le client est dirigé vers la page de mise à jour de la livraison.

9. Cas d'utilisation : Annuler une commande

- Titre du cas d’utilisation : Annuler une commande
- Objectif : Permettre au client d’annuler une commande passée.

- Acteurs : Client
- Préconditions :
  - Au moins une commande doit être présente pour annulation.
- Scénario nominal (principal) :
  - Le client sélectionne l'option d'annulation.
  - Le système supprime la commande.
- Scénarios alternatifs :
  - Le client est redirigé vers la page du panier.
- Postconditions : Le système retourne à la page du panier.

10. Cas d'utilisation : Ajouter les informations de livraison

- Titre du cas d'utilisation : Ajouter les informations de livraison
- Objectif : Permettre au client de modifier l'adresse de livraison.
- Acteurs : Client
- Préconditions :
  - Le client doit saisir une adresse valide.
- Scénario nominal (principal) :
  - Le client accède à la page de livraison.
  - Le client confirmer ou modifie l'adresse.
  - Le système enregistre l'information.
- Scénarios alternatifs :
  - Le système redirige vers la page de confirmation de commande.
- Postconditions : Le système dirige vers la page de paiement.

11. Cas d'utilisation : Suivre une commande

- Titre du cas d'utilisation : Suivre une commande
- Objectif : Permettre au client de consulter l'état de sa commande.
- Acteurs : Client
- Préconditions :
  - Le client doit être authentifié.
  - Une commande doit être enregistrée et confirmée pour être suivie.
- Scénario nominal (principal) :
  - Le client choisit l'option de suivi.
  - Le système affiche l'état actuel de la commande.
- Scénarios alternatifs :
  - Aucune commande n'est disponible pour le client.
- Postconditions : Le client est redirigé vers la mise à jour de livraison.

12. Cas d'utilisation : Effectuer un paiement

- Titre du cas d'utilisation : Effectuer un paiement
- Objectif : Permettre au client de régler le montant de la commande en choisissant un mode de paiement.
- Acteurs : Client
- Préconditions :
  - Une commande doit être confirmée.

- Scénario nominal (principal) :
    - Le client lance l'option « Effectuer un paiement ».
    - Le système propose les modes de paiement disponibles.
    - Le client choisit un mode de paiement.
- Scénarios alternatifs :
    - Le client annule l'opération de paiement.
- Postconditions : Un mode de paiement est sélectionné et le système déclenche le cas d'utilisation correspondant.

13. Cas d'utilisation : Paiement par carte bancaires

- Titre du cas d'utilisation : Paiement par carte bancaires
- Objectif : Permettre au client de saisir ou mettre à jour ses informations bancaires pour le paiement.
- Acteurs : Client
- Préconditions :
    - Le client doit fournir des informations bancaires valides.
- Scénario nominal (principal) :
    - Le client accède à l'interface de paiement en ligne.
    - Il saisit les informations bancaires.
    - Le système enregistre les données.
- Scénarios alternatifs :
    - Les informations bancaires sont incorrectes.
    - Le paiement est refusé par la banque.
- Postconditions : Le paiement est effectué avec succès et la commande est validée, le client est redirigé vers la page d'accueil.

14. Cas d'utilisation : Payer à la livraison

- Titre du cas d'utilisation : Payer à la livraison
- Objectif : Permettre au client de choisir le paiement à la livraison.
- Acteurs : Client
- Préconditions :
    - La commande doit être confirmée.
- Scénario nominal (principal) :
    - Le client sélectionne l'option de paiement à la livraison.
    - Le système enregistre l'option choisie.
- Scénarios alternatifs :
    - Le client est redirigé vers la page de paiement.
- Postconditions : La commande est validée et le client est redirigé vers la page d'accueil.

15. Cas d'utilisation : Consulter les utilisateurs connectés

- Titre du cas d'utilisation : Consulter les utilisateurs connectés
- Objectif : Permettre à l'administrateur de visualiser les utilisateurs connectés.
- Acteurs : Administrateur
- Préconditions :
    - L'utilisateur doit être administrateur.
- Scénario nominal (principal) :
    - L'administrateur accède à la liste des utilisateurs connectés.

- Le système affiche les utilisateurs actifs.
- Scénarios alternatifs :
  - Le système redirige vers la page d'accueil.
- Postconditions : Retour à la page d'accueil.

16. Cas d'utilisation : Mettre à jour les médicaments

- Titre du cas d'utilisation : Mettre à jour les médicaments
- Objectif : Permettre à l'administrateur de gérer les médicaments du système.
- Acteurs : Administrateur
- Préconditions :
  - L'administrateur est authentifié.
  - L'accès aux données de stock est autorisé.
- Scénario nominal (principal) :
  - L'administrateur affiche la liste des médicaments.
  - Le système propose les actions de gestion des médicaments.
- Scénarios alternatifs :
  - L'administrateur quitte la gestion des médicaments.
- Postconditions : Une action de gestion des médicaments est déclenchée.

17. Cas d'utilisation : Ajouter un médicament

- Titre du cas d'utilisation : Ajouter un médicament
- Objectif : Permettre à l'administrateur d'ajouter un nouveau médicament au stock.
- Acteurs : Administrateur
- Préconditions :
  - Les informations du médicament sont disponibles (nom, quantité, références).
- Scénario nominal (principal) :
  - L'administrateur saisit les informations du médicament.
  - Le système valide les données.
  - Le système ajoute le médicament au stock.
- Scénarios alternatifs :
  - Les informations saisies sont incorrectes ou incomplètes.
- Postconditions : Le médicament est ajouté au stock.

18. Cas d'utilisation : Modifier les informations du médicament

- Titre du cas d'utilisation : Modifier les informations du médicament
- Objectif : Permettre à l'administrateur de modifier la disponibilité et les détails d'un médicament existant.
- Acteurs : Administrateur
- Préconditions :
  - Le médicament existe dans le système.
- Scénario nominal (principal) :
  - L'administrateur sélectionne un médicament.
  - Il modifie la disponibilité ou les détails.
  - Le système enregistre les modifications.
- Scénarios alternatifs :
  - Le médicament sélectionné n'existe plus.
- Postconditions : Les informations du médicament sont mises à jour.

19. Cas d'utilisation : Supprimer un médicament

- Titre du cas d'utilisation : Supprimer un médicament
- Objectif : Permettre à l'administrateur de supprimer un médicament du système.
- Acteurs : Administrateur
- Préconditions :
  - Le médicament existe dans le système.
- Scénario nominal (principal) :
  - L'administrateur sélectionne le médicament à supprimer.
  - Le système demande une confirmation.
  - L'administrateur confirme la suppression.
  - Le système supprime le médicament du stock.
- Scénarios alternatifs :
  - L'administrateur annule la suppression.
- Postconditions : Le médicament est supprimé du système.

20. Cas d'utilisation : Gérer les commandes

- Titre du cas d'utilisation : Gérer les commandes
- Objectif : Permettre à l'administrateur de gérer les commandes des clients.
- Acteurs : Administrateur
- Préconditions :
  - L'administrateur est authentifié.
  - Des commandes doivent être enregistrées.
- Scénario nominal (principal) :
  - L'administrateur accède à la liste des commandes.
  - Il gère ou met à jour leur statut.
- Scénarios alternatifs :
  - Redirection vers la page d'accueil.
- Postconditions : Retour à la page d'accueil.

21. Cas d'utilisation : Consulter les informations du livreur

- Titre du cas d'utilisation : Consulter les informations du livreur
- Objectif : Permettre à l'administrateur de visualiser les informations du livreur.
- Acteurs : Administrateur
- Préconditions :
  - L'administrateur est authentifié.
  - Les informations du livreur doivent être disponibles.
- Scénario nominal (principal) :
  - L'administrateur affiche les informations du livreur.
- Scénarios alternatifs :
  - Le système redirige vers la page d'accueil.
- Postconditions : Retour à la page d'accueil.

22. Cas d'utilisation : Gérer la présence du livreur

- Titre du cas d'utilisation : Gérer la présence du livreur
- Objectif : Permettre à l'administrateur de gérer l'assiduité du livreur.
- Acteurs : Administrateur
- Préconditions :

  - Les informations du livreur doivent être disponibles.
- Scénario nominal (principal) :
  - L'administrateur met à jour l'assiduité selon les jours travaillés.
- Scénarios alternatifs :
  - Redirection vers la page d'informations du livreur.
- Postconditions : Retour à la page d'informations du livreur.

23. Cas d'utilisation : Consulter les commandes à livrer

- Titre du cas d'utilisation : Consulter les commandes à livrer
- Objectif : Permettre au livreur de vérifier les commandes à livrer.
- Acteurs : Livreur
- Préconditions :
  - Livreur est authentifié.
  - Des commandes doivent être enregistrées.
- Scénario nominal (principal) :
  - Le livreur consulte la liste des commandes à livrer.
- Scénarios alternatifs :
  - Le système redirige vers la page d'accueil.
- Postconditions : Retour à la page d'accueil.

24. Cas d'utilisation : Consulter l'adresse de livraison du client

- Titre du cas d'utilisation : Consulter l'adresse de livraison du client
- Objectif : Permettre au livreur d'afficher l'adresse du client pour effectuer la livraison.
- Acteurs : Livreur
- Préconditions :
  - Livreur est authentifié.
  - L'adresse doit être correctement enregistrée.
- Scénario nominal (principal) :
  - Le livreur affiche l'adresse de livraison.
- Scénarios alternatifs :
  - Le système redirige vers la page d'accueil.
- Postconditions : Retour à la page d'accueil.

25. Cas d'utilisation : Mettre à jour le statut de la commande

- Titre du cas d'utilisation : Mettre à jour le statut de la commande
- Objectif : Permettre au livreur de mettre à jour le statut d'une commande après tentative de livraison.
- Acteurs : Livreur
- Préconditions :
  - Une commande est sélectionnée.
  - Le livreur a consulté les détails de la commande.
- Scénario nominal (principal) :
  - Le livreur sélectionne une commande.
  - Le livreur choisit l'option de mise à jour du statut.
  - Le système demande le type de statut à appliquer.
- Scénarios alternatifs :
  - aucun

- Postconditions : Le système attend la sélection du statut spécifique (*Livré* ou *Non livré*).

26. Cas d'utilisation : Mettre à jour le statut " livré"

- Titre du cas d'utilisation : Mettre à jour le statut " livré"
- Objectif : Permettre au livreur d'indiquer qu'une commande a été livrée avec succès.
- Acteurs : Livreur
- Préconditions :
  - La commande est effectivement livrée.
- Scénario nominal (principal) :
  - Le livreur sélectionne l'option « Livré ».
  - Le système enregistre le statut *Livré*.
- Scénarios alternatifs :
  - Redirection vers la page de contact de l'adresse de livraison.
- Postconditions : Le système retourne à la page de contact de livraison.

27. Cas d'utilisation : Mettre à jour le statut " Non livré"

- Titre du cas d'utilisation : Mettre à jour le statut " livré"
- Objectif : Permettre au livreur d'indiquer qu'une commande n'a pas pu être livrée.
- Acteurs : Livreur
- Préconditions :
  - Le livreur est en mesure d'identifier un échec de livraison.
- Scénario nominal (principal) :
  - Le livreur sélectionne l'option « Non livré ».
  - Le système enregistre le statut *Non livré*.
- Scénarios alternatifs :
  - Redirection vers la page de contact de l'adresse de livraison.
- Postconditions : Le système retourne à la page de contact de livraison.

28. Cas d'utilisation : Mentionner la raison du statut "Non livré"

- Titre du cas d'utilisation : Mentionner la raison du statut "Non livré"
- Objectif : Permettre au livreur d'indiquer la raison pour laquelle la commande n'a pas été livrée.
- Acteurs : Livreur
- Préconditions :
  - Le livreur doit avoir identifié la raison du non-livraison.
- Scénario nominal (principal) :
  - Le livreur saisit la raison du non-livraison.
  - Le système enregistre l'information.
- Scénarios alternatifs :
  - Redirection vers la page "Non livré".
- Postconditions : Le système retourne vers la page "Non livré".

29. Cas d'utilisation : Mettre à jour le profil

- Titre du cas d'utilisation : Mettre à jour le profil
- Objectif : Permettre au livreur de consulter ou modifier son profil.
- Acteurs : Livreur
- Préconditions :
  - Livreur est authentifié.

- Le livreur doit posséder un profil enregistré.

- Scénario nominal (principal) :
  - Le livreur accède à son profil.
  - Il modifie ses informations si nécessaire.
  - Le système enregistre les changements.
- Scénarios alternatifs :
  - Redirection vers la page d'accueil.
- Postconditions : Retour à la page d'accueil.

4.4.3. Le diagramme de cas d'utilisation

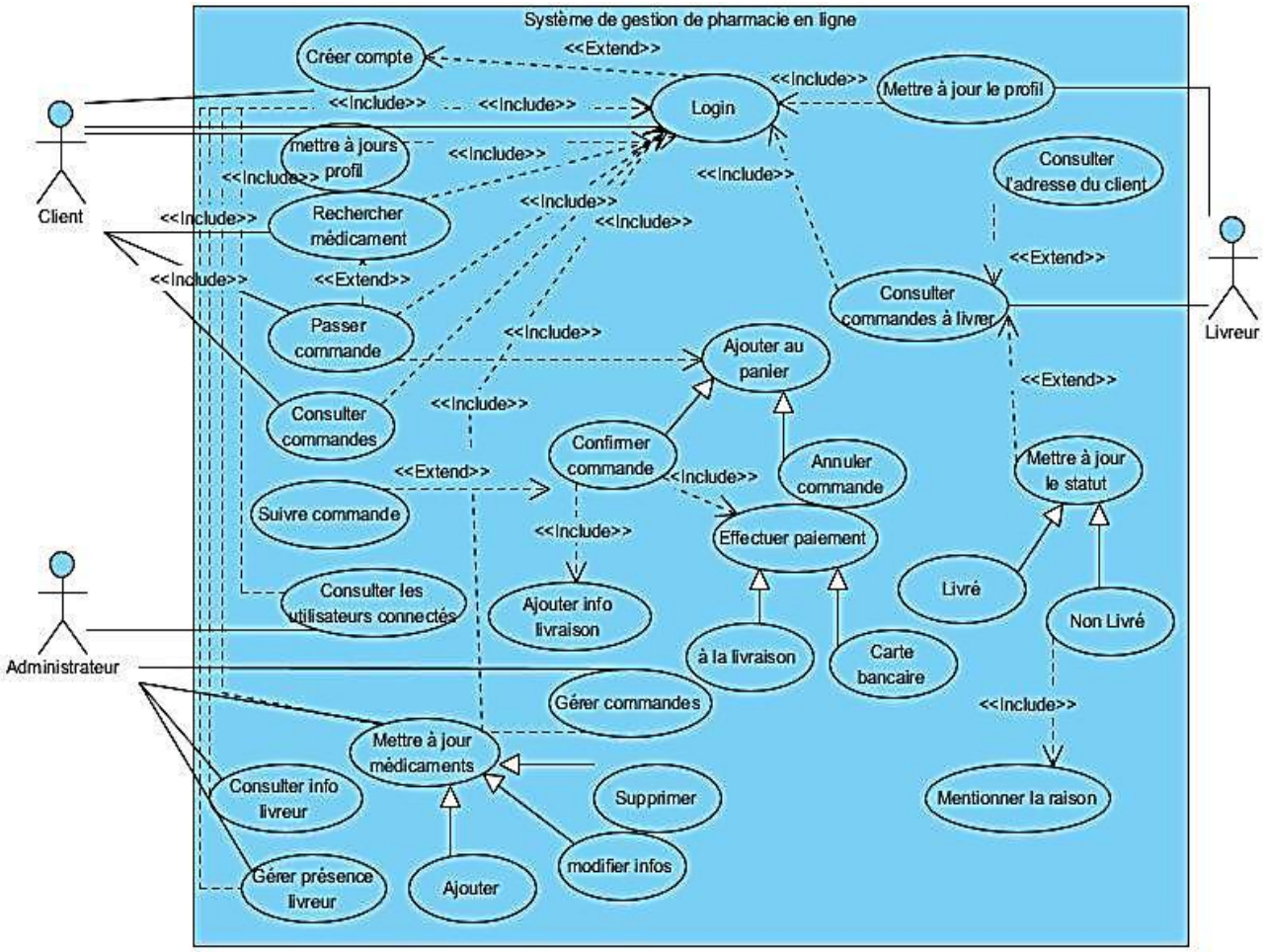


Figure 1 : Diagramme de cas d'utilisation UML.

4.4.4. Le diagramme de classe

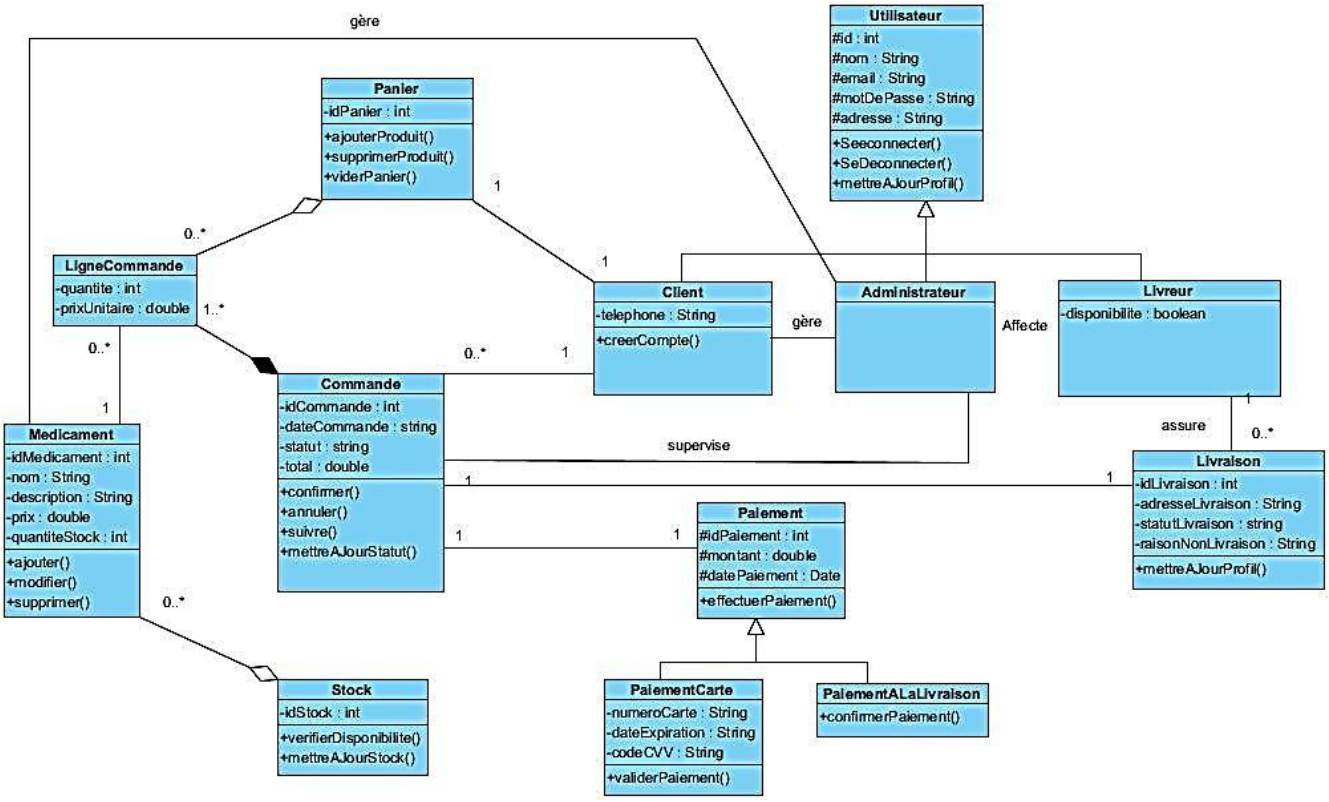


Figure 2 : Le diagramme de classe.

4.4.5. Le diagramme d'activité

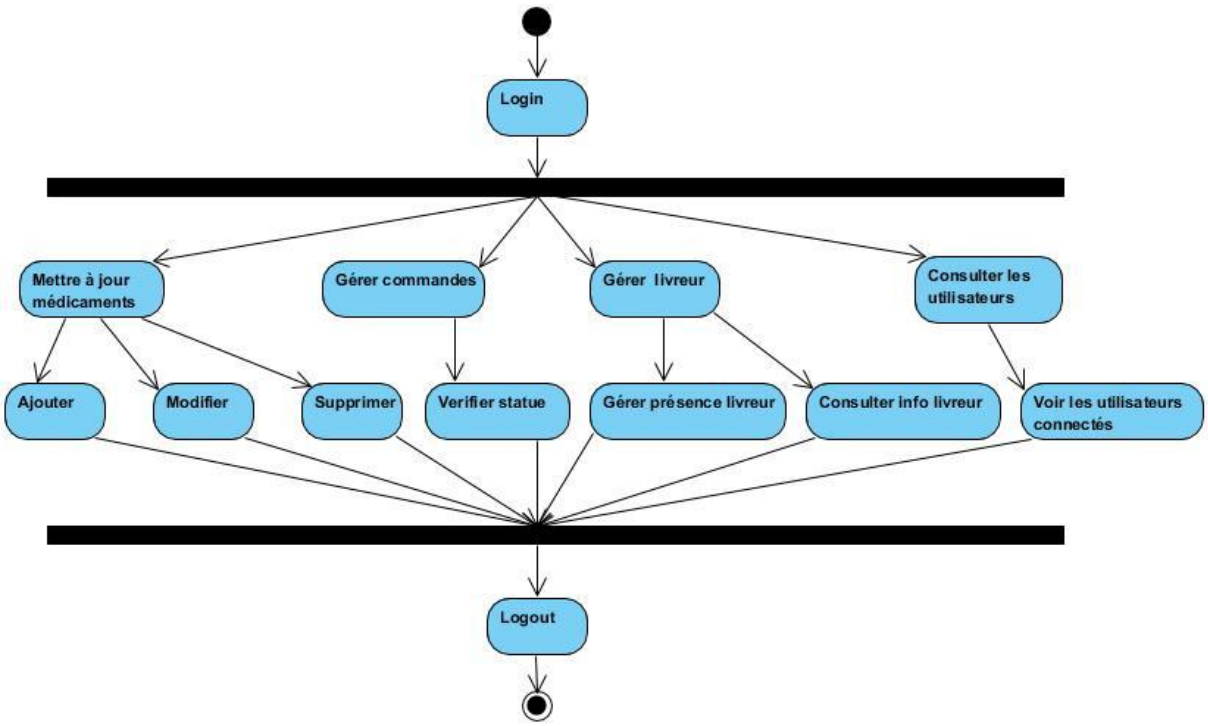


Figure 3: Diagramme d'activité UML – Admin.

4.4.6. Le diagramme de séquence (login)

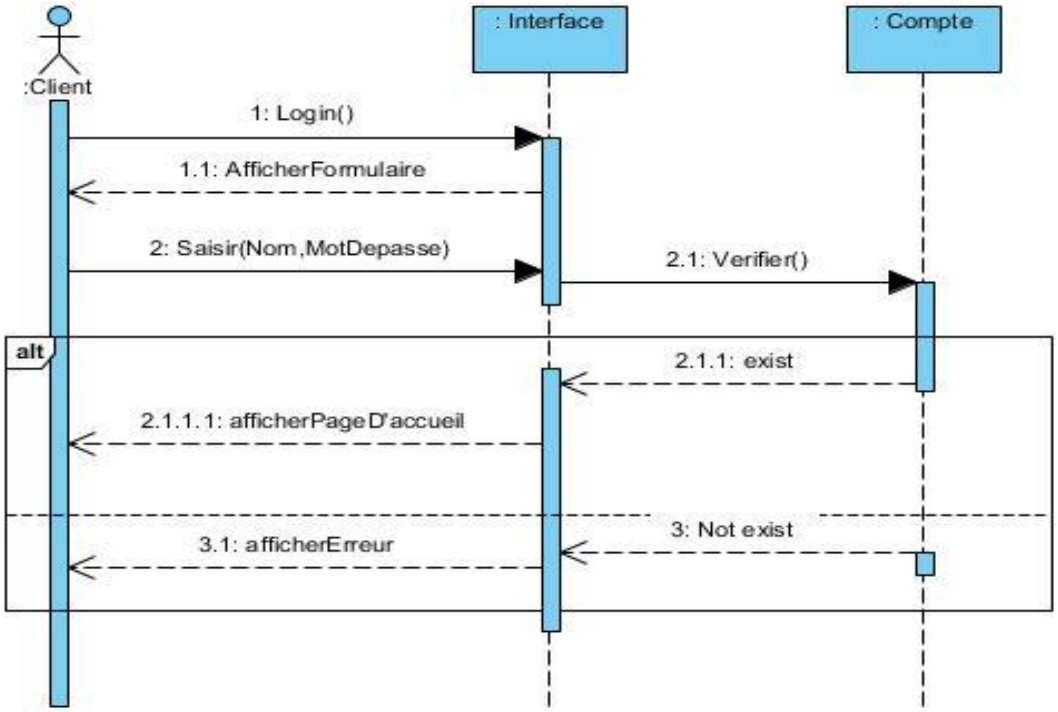


Figure 4 : Le diagramme de séquence (login).

4.4.7. Le diagramme de séquence (client)

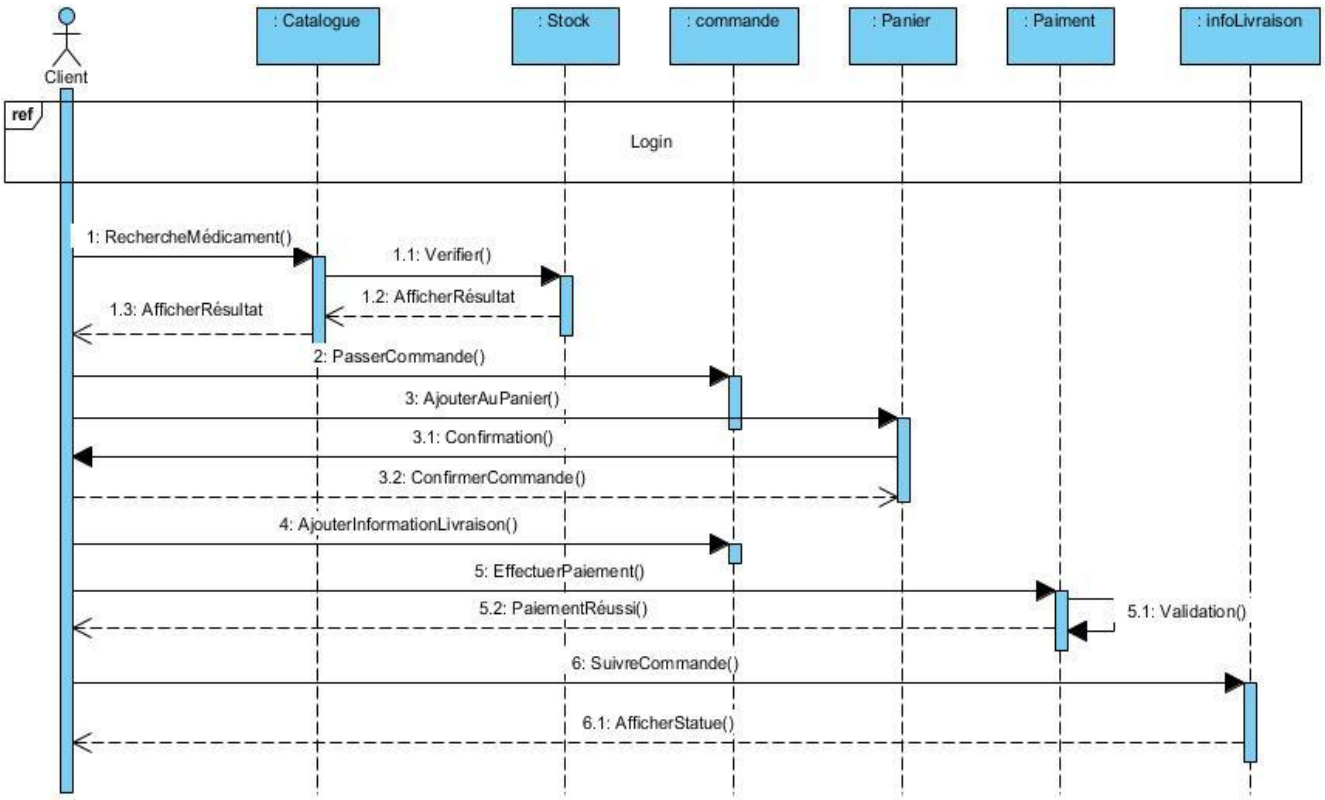


Figure 5 : Le diagramme de séquence (client).

4.4.8. Le diagramme de séquence (admin)

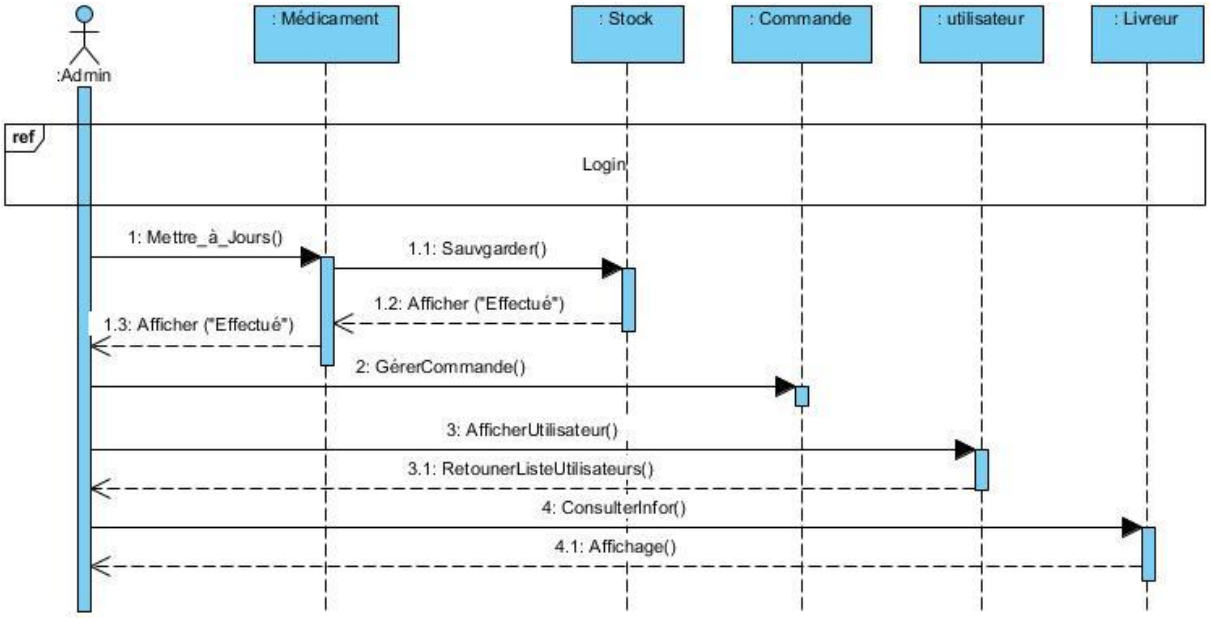


Figure 6: Le diagramme de séquence (admin).

4.4.9. Le diagramme de séquence (livreur)

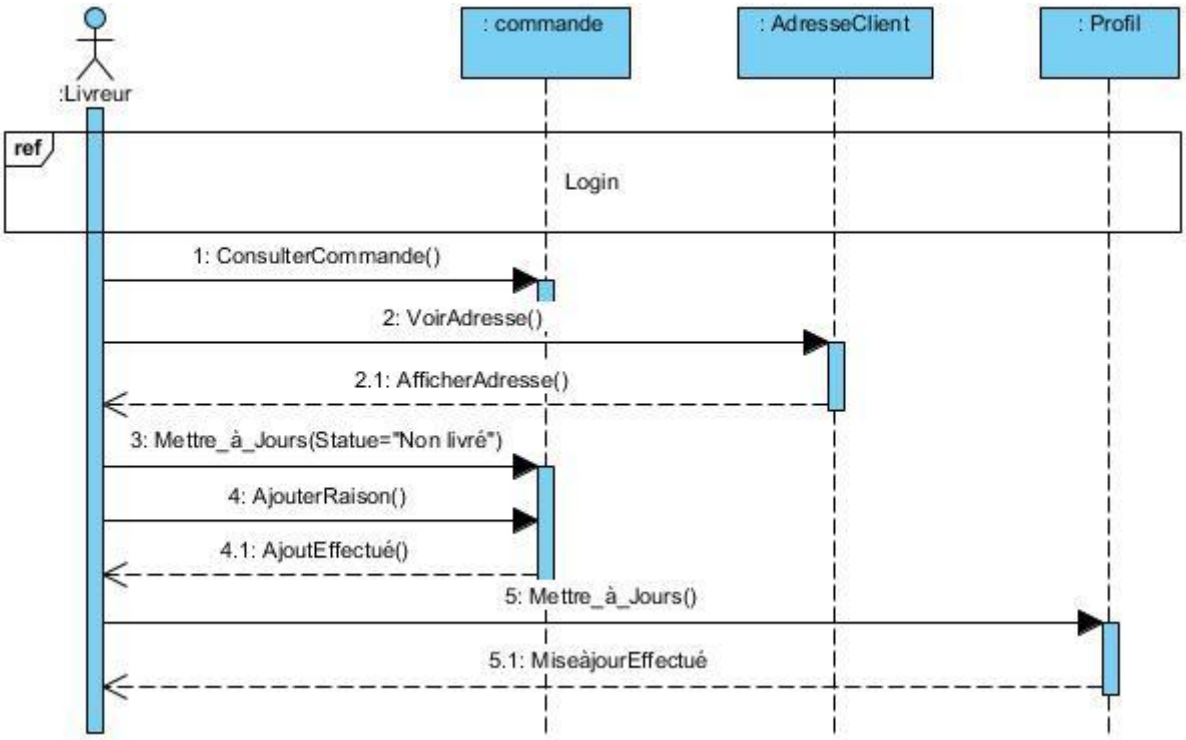


Figure 7: Le diagramme de séquence (livreur).

4.4.10. Le diagramme de séquence (Passer commande)

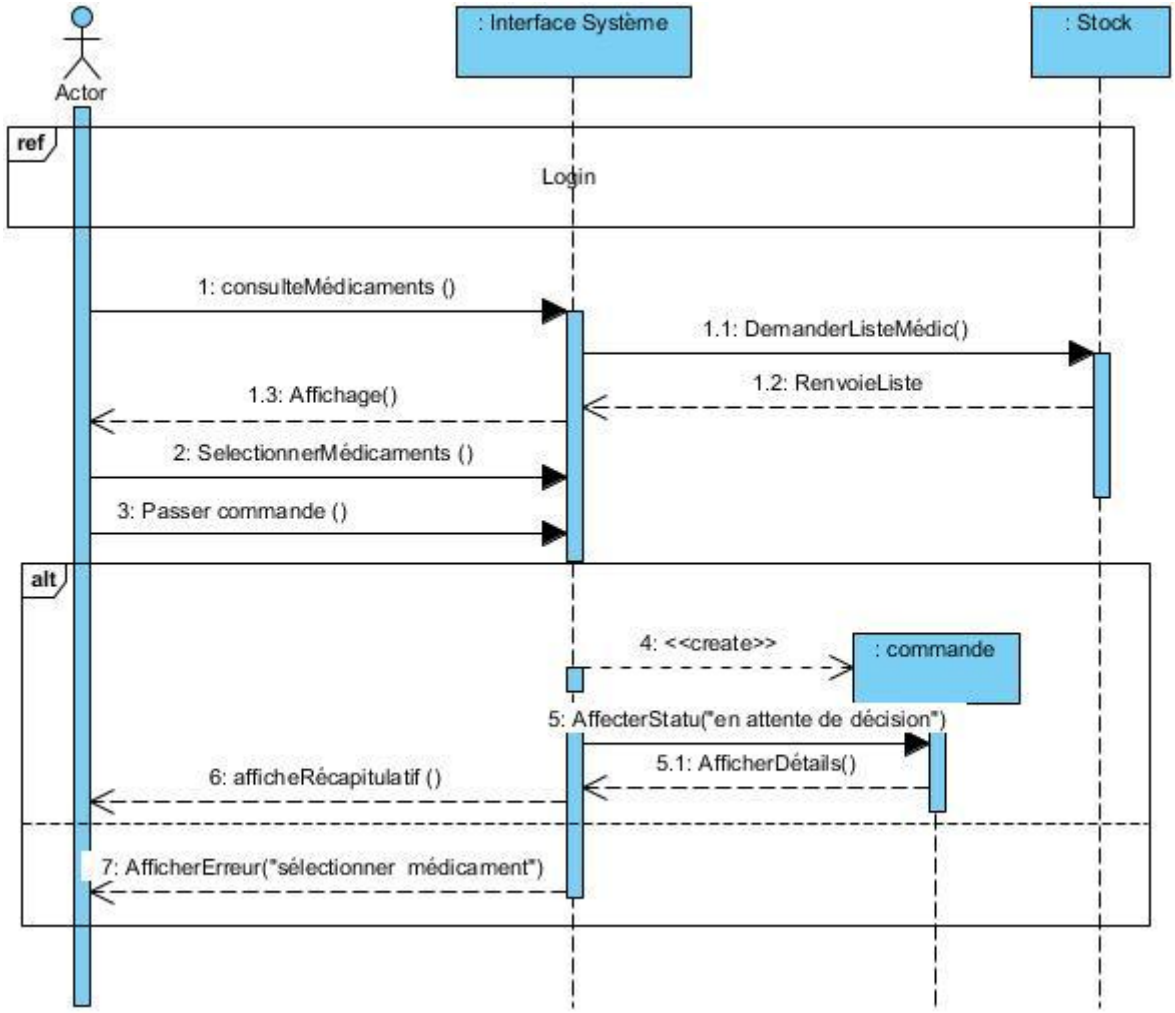


Figure 8: Le diagramme de séquence (Passer commande).

4.4.11. Le diagramme de composant

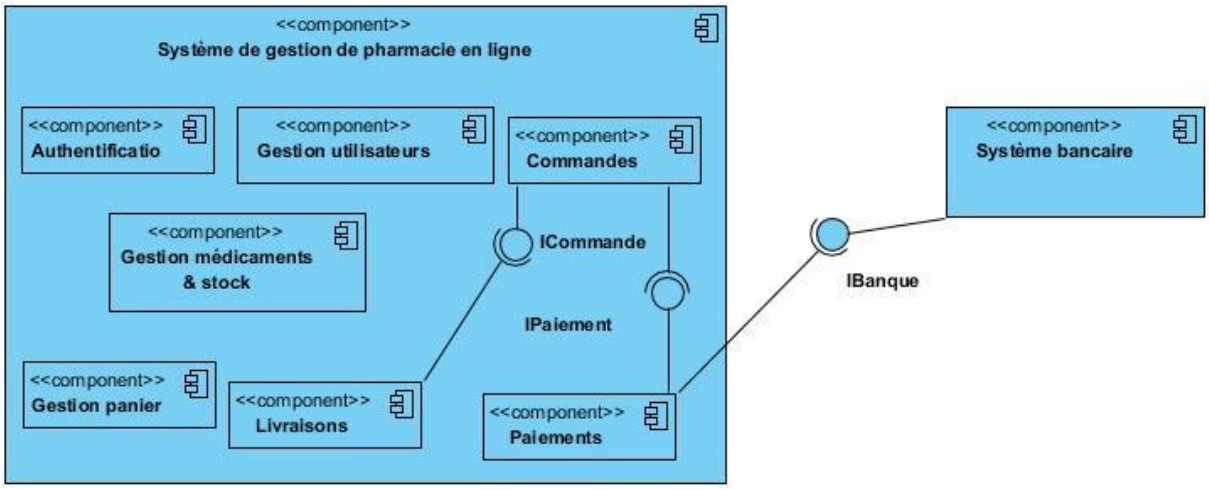


Figure 9: Le diagramme de composant.

4.4.12. Le diagramme de déploiement

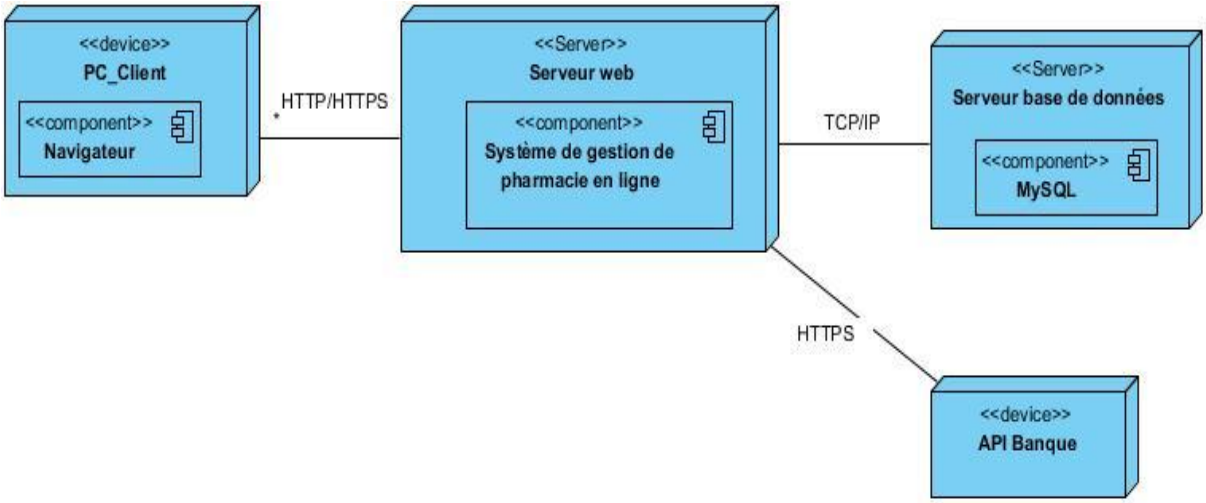


Figure 10: Le diagramme de déploiement.

Chapitre 06: Introduction à AUML

1. Introduction

L'approche orientée objet (OO) permet de concevoir des systèmes en représentant les entités comme des objets possédant des attributs (données) et des méthodes (comportements). Chaque objet peut interagir avec d'autres objets via des messages ou des appels de méthodes. Cette approche est très efficace pour modéliser des systèmes traditionnels, comme une application de gestion de bibliothèque ou un système de gestion de commandes.

Cependant, certains systèmes modernes nécessitent que les entités soient autonomes, capables de prendre des décisions, de coopérer et de réagir dynamiquement à leur environnement. C'est le cas des systèmes multi-agents, où chaque agent agit de manière indépendante tout en collaborant avec d'autres agents pour atteindre des objectifs communs.

2. Les objectifs de chapitre 06

A la fin de ce chapitre, l'étudiant seras capable de :

- ✓ Comprendre le concept des systèmes multi-agents et l'intérêt d'AUML.
- ✓ Identifier les agents, leurs rôles et leurs interactions dans un système.
- ✓ Découvrir les principaux types de diagrammes AUML et leur utilité.
- ✓ Se familiariser avec l'extension AUML pour la modélisation orientée agent.

3. Exemple concret

Imaginons un système de pharmacie en ligne :

- Avec l'orienté objet, on aurait des classes comme Client, Commande, Panier et Médicament. Les interactions sont directes : le client appelle des méthodes pour ajouter un médicament au panier ou passer une commande.
- Avec une approche multi-agent, le Client devient un agent autonome capable de :
  - prendre des décisions (ex. choisir un mode de paiement),
  - communiquer avec un agent Livreur pour organiser la livraison,
  - recevoir des notifications de l'agent Administrateur sur la disponibilité des médicaments.

Chaque agent possède son comportement propre, peut réagir à des événements (retard de livraison, rupture de stock) et coopérer avec d'autres agents, ce qui rend le système plus dynamique et intelligent.

Ainsi, AUML (Agent UML) est une extension d'UML qui permet de modéliser ces agents, leurs interactions et leurs comportements autonomes, tout en s'appuyant sur les concepts connus de l'orienté objet.

4. Définition

Un agent est défini comme un programme informatique autonome capable de percevoir son environnement, de prendre des décisions et d'agir pour atteindre ses objectifs, tout en réagissant aux changements de cet environnement.

AUML combine l'intérêt croissant pour les méthodes de conception basées sur les agents avec l'utilisation répandue d'UML pour le développement orienté objet.

La programmation orientée agent (AOP) est une spécialisation de la programmation orientée objet (POO) dont la façon ou le système est conçu :

« Un système informatique est vu comme étant composé de modules de communications, chacun avec sa propre manière de gérer les messages. »
[Shoham, 1993]

Les principales extensions d'AUML par rapport à UML sont :

- Diagramme de classes d'agent qui est une reformulation du diagramme de classes d'objets
- Diagramme de séquence qui permet une meilleure modélisation des interactions entre agents
- Diagramme de collaboration qui complémente le diagramme de séquences en proposant une autre lecture et vision des interactions entre agents.

5. Caractéristiques principales d'un agent logiciel

⇨ Autonome
L'agent décide lui-même de ses actions.
⇨ Réactif
L'agent perçoit son environnement et réagit aux événements.
⇨ Capable d'agir (Proactif)
L'agent ne se contente pas de réagir, il prend des initiatives pour atteindre ses objectifs.
⇨ Sociable
L'agent communique et coopère avec d'autres agents.

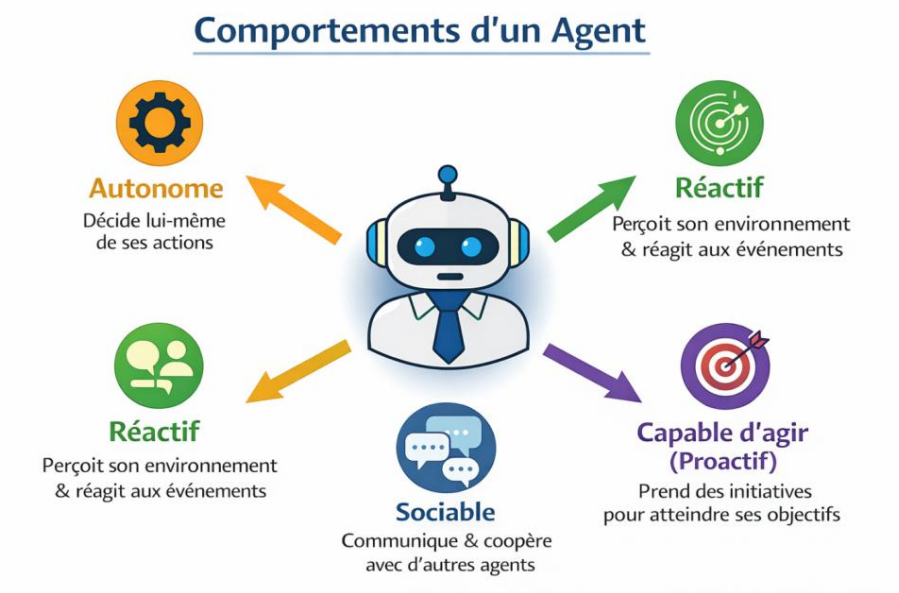


Figure 1 : Caractéristiques principales d'un agent logiciel.

6. AUML comme extension d'UML

AUML permet de :

- représenter les rôles que jouent les agents au lieu de simples instances d'objets ;
- modéliser des protocoles d'interaction entre agents (par exemple des envois de messages, des dialogues avec conditions) ;
- annoter des messages plus riches que de simples appels de méthode ; les messages peuvent inclure des actes de communication explicites (*performative*).

En AUML, une « ligne de vie » dans un diagramme d'interaction peut représenter un rôle d'agent, plutôt qu'une instance unique simple de classe.

7. Du Diagramme de Classes au Diagramme de Classes d'Agent

Le diagramme de classes UML standard est conçu pour montrer des classes avec leurs attributs (état) et méthodes (comportement). Cependant, cette structure est insuffisante pour un agent qui possède des caractéristiques sociales et organisationnelles.

Pour correspondre aux caractéristiques des agents, le diagramme de classes a été étendu. On l'appelle Diagramme de Classes d'Agent. Il reprend la notation graphique des classes mais y ajoute des compartiments spécifiques pour décrire la dimension sociale.

Il contient les éléments spécifiques suivants :

1. **Identificateur de l'agent (Agent Identifier)** : Contrairement à un objet qui est identifié par son adresse mémoire, un agent possède un identifiant sémantique unique qui permet de le reconnaître dans le système multi-agent (ex: Agent_Vendeur_01).

2. **Rôles (Roles)** : C'est le concept central. Il définit le comportement d'un agent au sein d'une société d'agents. Un même agent peut jouer plusieurs rôles.
    - *Exemple :* Vendeur, Négociateur, Médiateur.
3. **Organisation (Organization)** : Définit les relations structurelles entre les rôles. Cela permet de modéliser des architectures sociales similaires aux organisations humaines.
    - *Types d'organisations :* Hiérarchies (Chef/Subordonné), Groupes d'intérêt, Troupeaux, Équipes.

Dans AUML, la représentation d'un agent modifie le rectangle classique de la classe :

- En-tête : Contient le nom de l'agent et son/ses rôles (ex: AgentA / Rôle: Vendeur).
- Compartiments : En plus des attributs et méthodes, on peut trouver des compartiments pour les "Capacités" (Capabilities) ou les "Services" offerts.

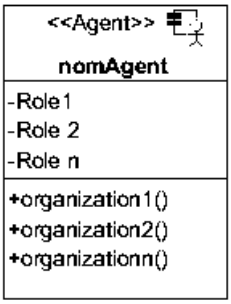


Figure 2: exemple de représentation d'un agent.

8. Exemple d'une représentation de capacités

L'agent ***sum*** a deux capacités d'exprimer le fait qu'il est en mesure de faire des additions et soustractions.

- *sum* est un agent logiciel.
- Il appartient à une organisation appelée Machine à calculer.
- Son rôle est d'effectuer des opérations arithmétiques : addition et soustraction.
- Il est identifié par un id.

⇨ L'agent représente une entité autonome capable d'agir selon ses compétences.

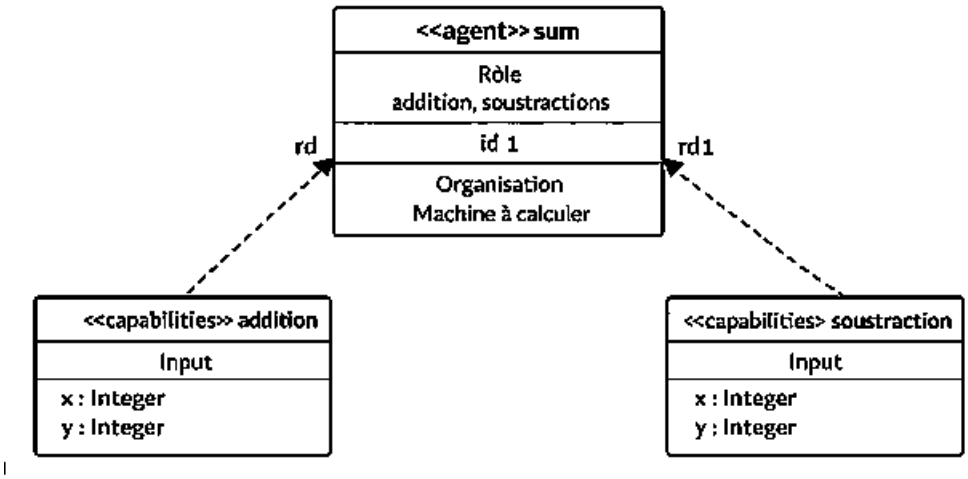


Figure 3 : représentation de capacités.

Figure 3 présente Les capacités d'agent. Elles décrivent ce que l'agent est capable de faire.

- La capacité soustraction est modélisée séparément.
- Elle prend en entrée deux valeurs :
  - x : Integer
  - y : Integer

Cela signifie que l'agent peut exécuter une soustraction sur deux entiers grâce à cette capacité.

9. Intérêt de représentation de capacités

- Séparer l'agent de ses capacités permet :
  - une meilleure modularité,
  - une réutilisation des capacités par d'autres agents,
  - une modélisation claire des compétences de chaque agent.

10. Protocoles d'interaction

- Les protocoles d'interaction sont au cœur des systèmes multi-agents.
- Ils décrivent des conversations, c'est-à-dire des séquences structurées de messages échangés entre plusieurs agents.
- Ces conversations respectent un ensemble de règles partagées par les agents, comparables à des conventions sociales.

11. Caractéristiques d'un protocole d'interaction

- Un protocole possède un nom (ex. : protocole de commande, protocole de négociation).
- Il définit les participants et leur assigne des rôles précis (demandeur, répondant, initiateur, etc.).
- Il spécifie le comportement de l'interaction, c'est-à-dire :

- les messages pouvant être échangés,
- l'ordre possible de ces messages,
- les conditions (guards) qui contrôlent l'envoi des messages.

12. Structures de contrôle dans les protocoles

- AUML permet de représenter des structures avancées :
  - AND : échanges parallèles,
  - OR : choix inclusifs,
  - XOR : choix exclusifs entre plusieurs messages ou actions.
- Ces structures facilitent la modélisation d'interactions complexes entre agents.

⇨ Cette richesse de représentation n'est pas disponible dans UML classique, ce qui souligne l'intérêt d'AUML pour les systèmes multi-agents.

13. Messages et actes de communication

- Dans AUML, un message entre agents n'est pas un simple appel de méthode.,Il est considéré comme un acte de communication porteur de sens.
- Chaque message exprime une intention de l'agent émetteur, par exemple :
  - *inform* (informer),
  - *request* (demander),
  - *agree* (accepter),
  - *refuse / failure* (refuser ou signaler un échec).
- L'agent récepteur interprète le message selon les règles du protocole.
- Cette approche s'appuie sur la théorie des actes de langage :
  - communiquer, c'est agir,
  - informer, demander ou promettre sont des actions à part entière.
- AUML permet de représenter graphiquement ces actes dans les diagrammes d'interaction.

14. Avantages et limites d'AUML

⇨ ***Avantages***

- AUML offre une représentation intuitive et visuelle des protocoles d'interaction entre agents.
- Il est aligné avec les spécifications FIPA et OMG pour la modélisation des agents.

⇨ **Limites**

- Les diagrammes AUML peuvent être semi-formels et parfois ambigus si on ne précise pas la sémantique exacte.

- Ils ne fournissent pas toujours une description suffisamment formelle pour des preuves de propriétés (comme l'absence de blocage).

15. Applications

AUML est particulièrement utilisé dans :

- Les systèmes multi-agents (MAS)
- La simulation d'environnement complexe (ex. systèmes de transport, logistique)
- Les systèmes distribués intelligents
- Les applications où l'autonomie, la coopération et l'interaction dynamique entre composants sont essentielles

16. Exemple

Dans un système de pharmacie en ligne (chapitre précèdent), un agent *Client* peut décider de passer une commande, un agent *Administrateur* peut surveiller les stocks et un agent *Livreur* peut organiser et confirmer la livraison. Chaque agent poursuit ses propres objectifs tout en collaborant avec les autres pour assurer le bon fonctionnement global du système.

Dans AUML, un agent peut jouer différents rôles selon le contexte.

Par exemple, un même agent *Client* peut jouer le rôle de :

- *Demandeur* lors du passage d'une commande
- le rôle de *Payeur* lors du paiement.

Les interactions entre agents ne sont plus vues comme de simples appels de méthodes, mais comme des échanges de messages porteurs d'intentions, tels que demander, informer, accepter ou refuser.

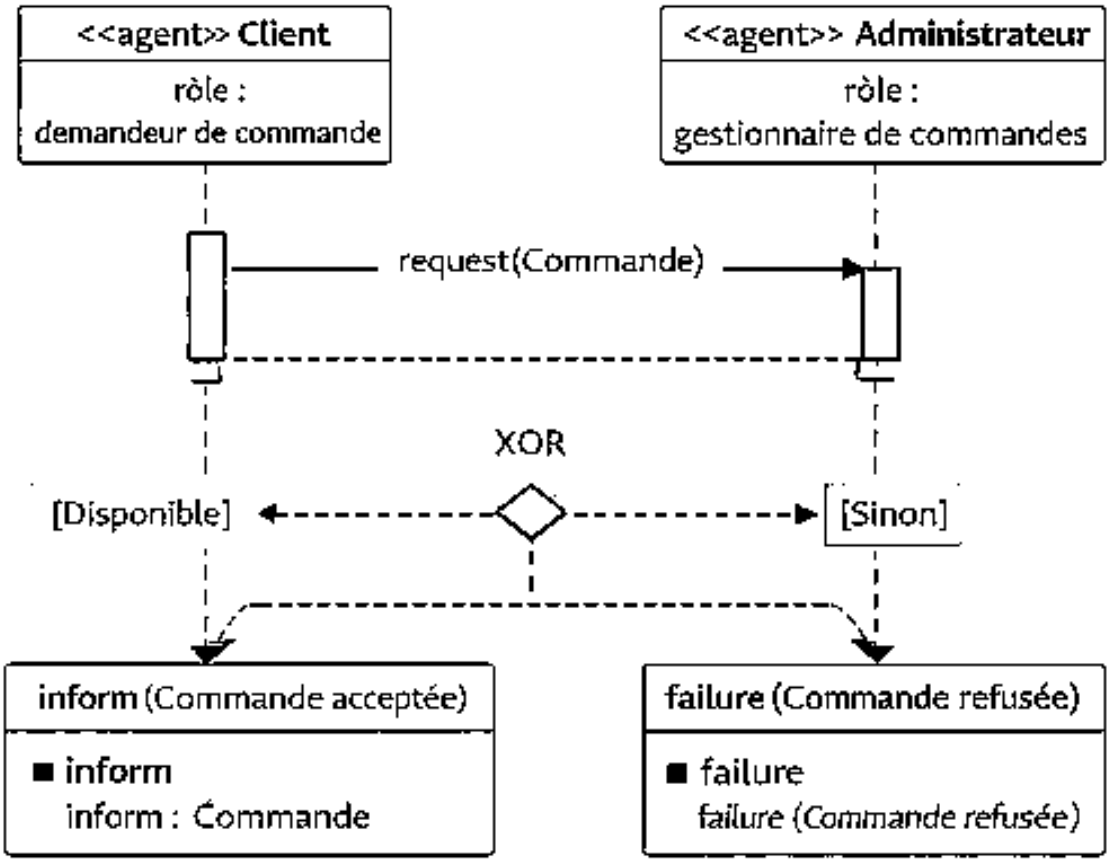


Figure 4 : Exemple d'un diagramme de Séquence AUML.

Bibliographie

[1]. Sundaramoorthy, S. (2022). UML Diagramming: A Case Study Approach. CRC Press.

[2]. Yedjour, H. (2019). Introduction à la Programmation Orientée Objet - Application en langage C++. (Support de cours universitaire).

[3]. Delannoy, C. (2014). Programmer en Java. 9e édition. Paris : Éditions Eyrolles.

[4]. Roques, P. (2011). UML 2 par la pratique : Études de cas et exercices corrigés. 7e édition. Paris : Éditions Eyrolles.

[5]. Gabay, J., & Gabay, D. (2008). UML 2 Analyse et conception : Mise en œuvre guidée avec études de cas. Paris : Dunod.

[6]. Bersini, H. (2007). L'orienté objet. Paris : Éditions Eyrolles.

[7]. Booch, G., Rumbaugh, J., & Jacobson, I. (2005). The Unified Modeling Language User Guide. 2nd Edition. Addison-Wesley.

[8]. Gabay, J. (2004). Merise et UML : Pour la modélisation des systèmes d'information. Paris : Dunod.

[9]. Leduc, T., & Leduc, D. (2000). Le langage Java : programmer par l'exemple. Paris : Éditions Technip.

Annexe A : Séries de Travaux Dirigés (TD)

Série TD N°01

Exercice 01

- Quelle est la différence entre les méthodes procédurale et les méthodes orienté objet ?
- Citez les avantages de l'approche OO.
- Quelle sont les points fort de l'UML ?

Exercice 02

- La voiture de Ali a pour numéro d'immatriculation AB-165-RE, elle pèse 1,5T, elle est verte. Elle a 55000 kilomètres au compteur. La voiture peut démarrer, s'arrêter, accélérer, décélérer, et tourner.
- La voiture de Ahmed a pour numéro d'immatriculation AC-271-BB, elle pèse 2T, elle est noire, elle a 90000 kilomètres au compteur. Cette voiture peut démarrer, s'arrêter, accélérer, décélérer, et tourner.

Travail demandé :

- Déterminer les objets correspondants.
- Déduire la classe correspondante.

Exercice 03

Des interviews d'experts métier ont permis de mettre en évidence les éléments suivants :

- Un vol a un aéroport de départ et un aéroport d'arrivée ;
- Un vol a une heure de départ et une heure d'arrivée, ainsi qu'une date de départ et une d'arrivée ;
- Un vol peut comporter des escales dans des aéroports ;
- Les escales interviennent dans un ordre déterminé ;
- Une escale a une heure d'arrivée et une heure de départ ;
- Chaque aéroport a un nom ;
- On peut ouvrir (et fermer) à la réservation chacun des vols.

Travail demandé :

- Déduire les classes correspondantes en déterminant le type de chaque attribut.

Exercice 04

Il est demandé de représenter les différentes classes pour une gestion technique de documents. Chaque document est composé d'un ou plusieurs feuillets. Un feuillet comporte du texte et des objets géométriques qui constituent deux types d'objets graphiques supportant des opérations de type : sélectionner, copier, couper, coller et déplacer.

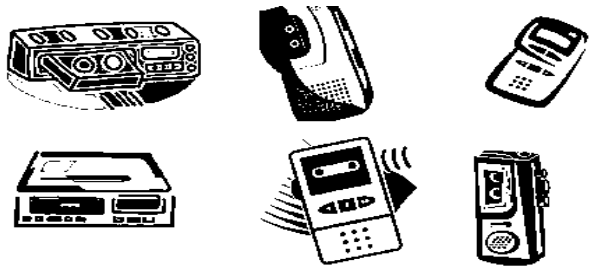
Exercice 05

Un robot se déplace dans un environnement composé de zones, de murs et de portes. Le robot Mars est en mouvement. Il est lié à une instance mondeCourant parmi les mondes possibles où peut évoluer le robot. Le robot peut manipuler des objets se trouvant dans le monde dans lequel il évolue.
À l'instant qui nous intéresse, le robot Mars est en mouvement et mondeCourant est lié aux zones z1 et z2. La zone z2 est composée de deux murs (m1 et m2) et d'une porte. La largeur de la porte est d'un mètre.

- Déterminer les objets correspondants.
- Déduire les classes correspondantes.

Exercice 06

1. Déterminer la classe « enregistreurs » corresponde aux objets suivants :



2. Basant sur la classe précédente « enregistreurs », décrire l'enregistreur suivant :



Solution (Série TD N°01)

Exercice 01

- **Quelle est la différence entre les méthodes procédurale et les méthodes orienté objet ?**

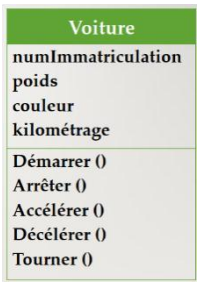
  Alors que la programmation procédurale traite les programmes comme un ensemble de données sur lesquelles agissent des procédures, la programmation orientée objet, elle, relève d'une conception définie comme des « messages » échangés par des entités appelées objets.

- **Citez les avantages de l'approche OO.**
  - ✓ Reflète plus finement les objets du monde réel (Basés sur le monde réel)
  - ✓ Facile à maintenir le code
  - ✓ Réutilisation des composantes
  - ✓ Simplifier le passage conceptuel/physique
  - ✓ Conduit à des modèles plus stables
  - ✓ Structure indépendante des fonctions
  - ✓ Encapsule la complexité
- **Quelle sont les points fort de l'UML ?**
  - ✓ Un langage formel et normalisé
  - ✓ Clair et précis
  - ✓ Encourage l'utilisation d'outils
  - ✓ Support de communication performant
  - ✓ L'analyse devient plus facile
  - ✓ Facilite la compréhension de représentations abstraites complexes

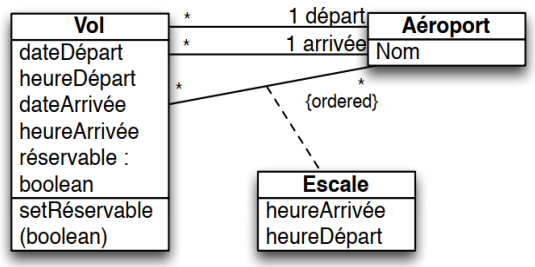
Exercice 02

Voiture 1 (mohamed **)**
Num= AB-165-RE Poids = 1,5T Couleur= verte Kilométrage=55000
démarrer, s'arrêter, accélérer, décélérer, et tourner.

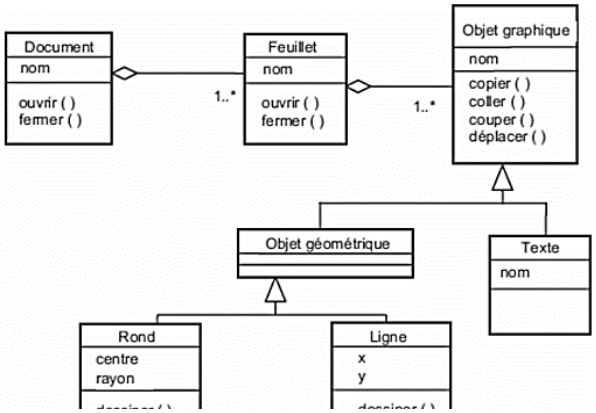
Voiture 2 (Imene **)**
Num= AC-271-BB Poids = 2T Couleur= noire Kilométrage=90000
démarrer, s'arrêter, accélérer, décélérer, et tourner.



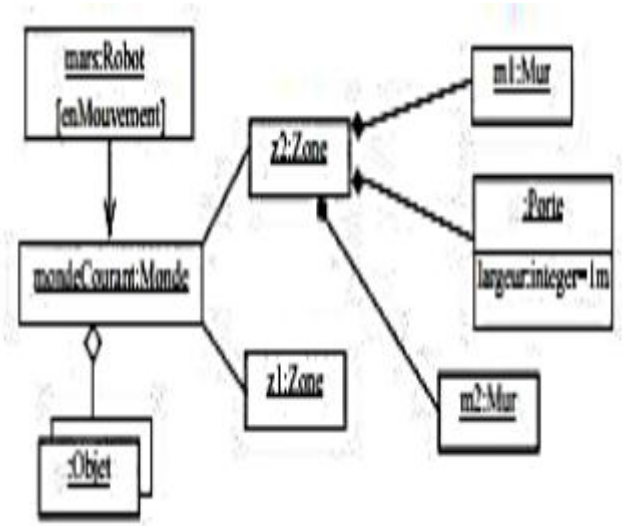
Exercice 03



Exercice 04



Exercice 05



Exercice 06

ENREGISTREUR
Code : string Marque : string Modèle : string Longueur : real Largeur : real Couleur : string
Enregistrer() Ecouter() Stopper() MarquerPause() Avancer() Ejecter()

ENREGISTREUR
Code : OLY-S830 Marque : Olympus Modèle : Microcassette Longueur : 5.2 pouces Largeur : 2.2 pouces Couleur : string
Enregistrer() Ecouter() Stopper() MarquerPause() Avancer() Ejecter()

Annexe A : Séries de Travaux Dirigés (TD)

Série TD N°02

Exercice 01

Pour chaque situation ci-dessous, proposez un diagramme de classe.

1. Une librairie vend des livres, caractérisés par leur auteur et leur nombre de pages ; certains livres possèdent également d'autres caractéristiques : une fourchette des âges pour les livres pour enfants, et la discipline et le niveau pour les livres scolaires.
2. On considère une entreprise, et on suppose qu'un chef dirige plusieurs salariés (les subordonnés) et que le chef est lui-même un salarié.
3. On considère une université, et les personnes y travaillant qui peuvent être des étudiants ou des enseignants.

Exercice 02

Soit un système d'information qui concerne le suivi des personnels d'un ensemble d'agences locales. Chaque agence se trouve dans une région, chaque région est pilotée par une direction régionale. La direction régionale se charge d'un ensemble d'agences locales. Une direction régionale est caractérisée par un code et un libellé.

Dessinez **le diagramme de classe** correspondant.

Exercice 03

Il est demandé d'établir le diagramme de classe pour la gestion des activités sportives des membres inscrits. Chaque membre possède une carte avec numéro unique, sur laquelle est mentionné son nom, son prénom, sa date de naissance. Un membre peut s'inscrire à une seule activité sportive et possède alors une date d'inscription. Une activité sportive possède un numéro séquentiel et la discipline (judo, natation, …). Un membre peut participer à plusieurs compétitions sportives et une compétition est ouverte à la participation de plusieurs membres. Chaque compétition est identifiée par un numéro séquentiel, une date et le lieu où elle est organisée. A la fin de chaque compétition, on note le résultat de chaque membre (bonne prestation, moyen, mauvais).

Exercice 04

Dans une société de transport, on voudrait gérer les bus de ramassage scolaire et les conducteurs. Un lycéen est un enfant, il est caractérisé par son nom, son âge et son sexe. Les informations qui caractérisent le conducteur sont les mêmes que pour le lycéen, avec en plus le numéro de son permis. Quant au bus, on a besoin de connaître son numéro d'immatriculation, sa date de mise en service, nombre d'années de service, et le poids total.

Dessinez **le diagramme de classe** correspondant.

Exercice 05

Une entreprise souhaite informatiser la gestion de son parc informatique (ordinateurs, imprimantes, etc.) pour en optimiser la maintenance. Un ordinateur est caractérisé par son numéro d'inventaire, son adresse réseau (adresse IP), son modèle, la date de son acquisition, la date de la prochaine maintenance planifiée et le système d'exploitation installé. Sur chaque ordinateur est installé un ensemble de logiciels caractérisés par un numéro de licence, un nom et une version. Grâce à un système de mots de passe, chaque ordinateur peut être utilisé par plusieurs employés mais, pour des raisons de sécurité des données, un employé n'a le droit d'utiliser qu'un seul ordinateur. Un employé est caractérisé par son nom, son prénom et sa fonction dans l'entreprise. Les ordinateurs sont reliés à un certain nombre de périphériques en réseau (imprimantes, scanners, etc.). Chaque périphérique est caractérisé par un numéro d'inventaire, son adresse IP, son type, son modèle, sa date d'acquisition et la date de la prochaine maintenance planifiée. Les périphériques pouvant servir à plusieurs ordinateurs simultanément, un indice de priorité est affecté à chaque ordinateur pour chaque périphérique auquel il est connecté. Chaque ordinateur et chaque périphérique est localisé dans un bureau donné. Les bureaux sont caractérisés par un numéro de bureau et le numéro du bâtiment dans lequel ils se trouvent. Un numéro de bureau est unique dans un bâtiment donné.

Dessinez **le diagramme de classe** correspondant.

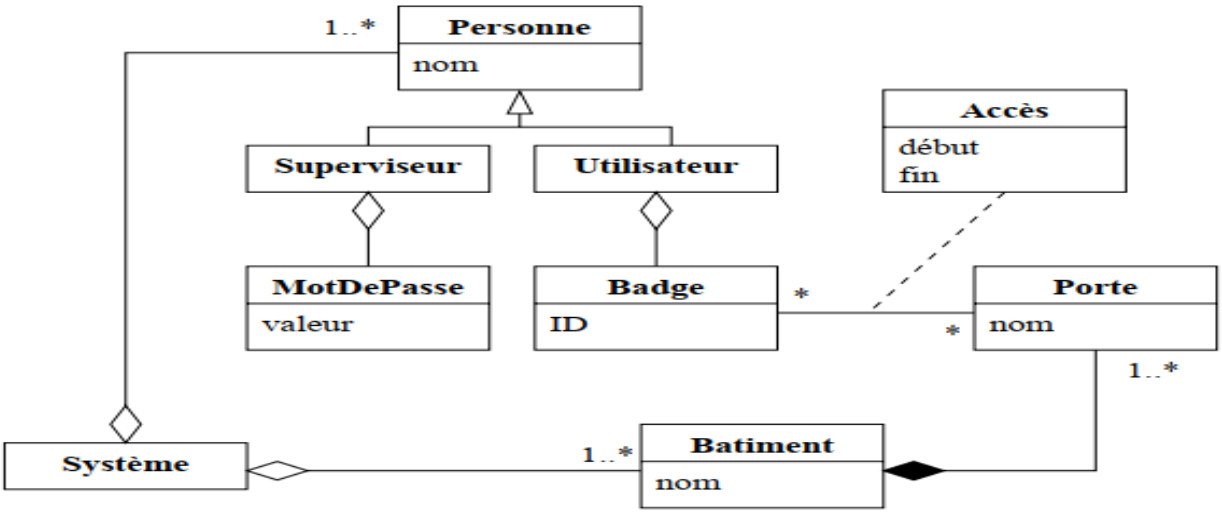
Exercice 06

Un hôtel est composé d'au moins deux chambres. Chaque chambre dispose d'une salle d'eau qui peut être une douche ou une salle de bain. Un hôtel héberge des personnes. Il peut employer du personnel et il est impérativement dirigé par un directeur. On ne connaît que le nom et le prénom des employés, des directeurs et des occupants. Certaines personnes sont des enfants et d'autres des adultes (faire travailler des enfants est interdit). Un hôtel a les caractéristiques suivantes : une adresse, un nombre de pièces et une catégorie. Une chambre est caractérisée par le nombre et de lits qu'elle contient, son prix et son numéro. On veut pouvoir savoir qui occupe quelle chambre à quelle date. Pour chaque jour de l'année, on veut pouvoir calculer le loyer de chaque chambre en fonction de son prix et de son occupation (le loyer est nul si la chambre est inoccupée). La somme de ces loyers permet de calculer le chiffre d'affaires de l'hôtel entre deux dates.

Donnez **un diagramme de classes** pour modéliser le problème de l'hôtel.

Série TD N°03

Exercice 01



Le système gère un seul bâtiment contenant trois portes. Le système est géré par une personne nommée Mohamed. Deux utilisateurs peuvent accéder au bâtiment :

- Aymen a accès à la première (8h-18h) et seconde porte (12h-24h)
- Imane a accès à la troisième porte toute la journée.

Dessinez **le diagramme d'objets** correspondant à cette situation.

Exercice 02

En utilisant le diagramme de classe correspondant à l'exercice N° 2 de la série TD précédente, Créez **un diagramme d'objets** correspondant au texte suivant :

La direction Dir_Reg_centrales avec un code 0112 et un libellé STAR pilote deux agences locales Ag_Voyage (code = 10, nom= « Nour ») et Ag_Location (code=25).

Exercice 03

Dessinez **le diagramme** d'objets correspondant à la situation suivante: Said et Amir sont des programmeurs spécialisés respectivement dans les langages C# et Java. Sabrine est un concepteur UML. Ils font partie de l'équipe « AMB » qui représente l'entreprise "TechSoft". Le logo de l'équipe «AMB» est «AMB*». Tous les développeurs utilisent un seul ordinateur.

Exercice 04

Un objet nommé C245 de classe Avion et en état détresse est en relation avec Mina, une tour de contrôle. Un autre avion anonyme dont l'état est à terre est aussi lié à Mina. La tour de contrôle communique avec CP12, une caserne de pompiers.

Dessinez **le diagramme d'objets** correspondant à la situation décrite ci-dessus.

Série TD N°04

Exercice 01

Le composant Forme.java implémente l'interface DessinerForme et dépond de l'interface ColorerForme du composant Graphique.java. Représentez **le diagramme de composants** (en utilisant les connecteurs d'assemblage puis en utilisant les connecteurs d'interfaces).

Le composant fraudeAgent.dll réalise deux interfaces Fraudes et DetailsFraudes. Ce sous-système contient les classes agentDesFraudes, reglementDesFraudes et RechercheDePattern. Représentez **le diagramme de composants** correspondant.

Exercice 02

Le composant gestionReservation.dll contient les classes Reservation, Client et Passager. Représentez **le diagramme de composants** correspondant.

Le composant reservation.java dépend de l'interface resForm du composant component.java. Représentez **le diagramme de composants**.

Exercice 03

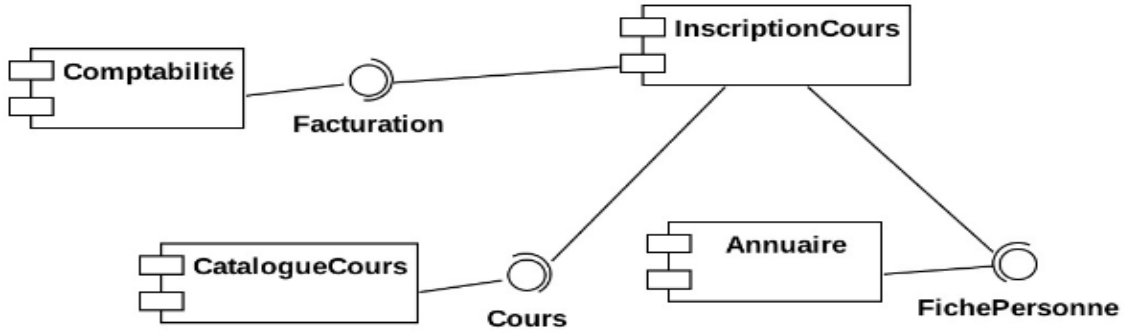
On considère une application constituée des fichiers suivants :

- Un code source registre.cpp utilise un programme exécutable registre.exe
- Des librairies dynamiques personne.dll et cours.dll. Les librairies à liens dynamiques sont utilisées
  lors de l'exécution d'une application

Donnez **le diagramme de composants** correspondant.

Exercice 04

Que pouvez-vous dire de ce schéma (à quoi ça sert, qu'est-ce que ça représente, quelle est la méthode de représentation utilisée…) :



Série TD N°05

Exercice 01

Un système est constitué d'un serveur (SGBD) autour duquel gravitent des PC pilotant l'ouverture et la fermeture de portes. Le nombre de PC n'est pas précisé. Chaque PC peut être le maître de 10 portes au plus.

Trois terminaux X jouent le rôle de console pour accéder au système ; une imprimante est reliée au serveur. Le serveur et les PC sont reliés par une liaison RNIS; les terminaux X et le serveur communiquent via TCP/IP.

Donner le **diagramme de déploiement** correspondant

Exercice 02

Dessinez **le diagramme de déploiement** correspondant à l'application suivante :

Un serveur principal (Processeur Pentium, Mémoire vive : 16 Go, Ecran 17 pouces, tour de disques durs montés en RAID 2TO) : dans ce serveur on doit installer le serveur web IIS pour déployer l'application. Un serveur pour installer le SGBD (Oracle) : ce serveur va communiquer avec le serveur principal via le protocole TCP/IP

Un troisième serveur de backup: pour assurer la réplication de la BD.

Un poste client: qui contient un navigateur internet pour se connecter avec le serveur web via l'internet et par des requêtes HTTP.

Exercice 03

La réalisation de l'application de réservation de vol se base sur la technologie JEE, cette application sera séparée en couches:

- Couche présentation: jsf, RicheFaces,..
- Couche métier: java
- Couche persistante: Hibernate

Les composants utilisés pour la création de l'application sont:

Serveur Web (Apache Tomcat) : c'est où le système va être installé, le serveur contient les composants qui exécutent le code java installé en interrogeant un serveur de base de données.

Serveur de base de données (MySQL)

PC client : qui contient un navigateur internet qui va se connecter avec le serveur web via l'internet et par des requêtes HTTP.

Représenter **le diagramme de déploiement** pour cette application.

Exercice 04

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 lequel 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 est déployé le composant: «ejb.jar». Il est packagé 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 PL/SQL : «scripts.sql ».

Dessinez **le diagramme de déploiement** correspondant

Exercice 05

L'architecture de l'application Smarteam est constituée des ressources suivantes:

Client web représente les navigateurs web avec lesquels se connectent les différents collaborateurs à l'application.

Serveur UNIX/ WINDOWS représente le serveur dans lequel est déployé l'application. Ce serveur héberge les deux serveurs :

- le serveur web Apache
- le serveur application Tomcat

Les deux serveurs sont fusionnés en Apache-Tomcat version 4.2. Le serveur UNIX ou Windows doit être accessible aux utilisateurs sur un réseau LAN ou WAN.

Comme une installation préalable du JDK 9.1.exe compatible avec Tomcat est exigée.

SGBD est un système de gestion de base de données MySQL assurant le stockage des données communique avec le serveur UNIX/ WINDOWS via TCP.

Serveur SMTP achemine les emails générés par l'application à destination des utilisateurs lors de la création de leurs profils avec une connexion SMTP/POP3.

Dessinez **le diagramme de déploiement** correspondant l'application Smarteam.

Série TD N°06

Exercice 01

Considérons une station-service de distribution d'essence. Les clients se servent de l'essence et le pompiste remplit les cuves. 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 ?

Amin, dont le métier est pompiste, peut se servir de l'essence pour sa voiture. Pour modéliser cette activité de « Amin ».

- Doit-on définir un nouvel acteur ? Comment modélise-t-on ça ?

Lorsqu'Amin vient avec son camion-citerne pour remplir les réservoirs des pompes.

- Est-il considéré comme un nouvel acteur ? Comment modélise-t-on cela ?

Certains pompistes sont aussi qualifiés pour opérer des opérations de maintenance en plus des opérations habituelles des pompistes telles que le remplissage des réservoirs. Ils sont donc réparateurs en plus d'être pompistes.

- Modéliser **le diagramme des cas d'utilisation** correspondant?

Exercice 02

Un système informatique doit permettre à des acheteurs potentiels de préparer l'achat de chevaux (mais pas l'achat proprement dit). L'achat d'un cheval concerne soit une jument soit un étalon. Dans le premier cas, on doit impérativement examiner l'état de maternité du cheval, et éventuellement vérifier que la jument n'a pas un jeune poulain en ce moment. Que l'on souhaite acheter un étalon ou une jument, on doit effectuer un examen des vaccinations. En outre l'acheteur peut souhaiter, lors de la préparation de son achat, consulter le caractère du cheval ou bien en connaître la robe. Toutes les informations en rapport à la filiation d'un cheval sont obtenues en consultant la base de données externe des haras nationaux.

Donnez **un diagramme des cas d'utilisation** pour le système de préparation avant achat.

Exercice 03

Il est demandé d'élaborer **le diagramme des cas d'utilisation (DCU)** pour une bibliothèque universitaire qui 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.

Exercice 04

Le déroulement normal d'utilisation d'une caisse enregistreuse est le suivant :

1. Un client arrive à la caisse avec des articles
2. Le caissier enregistre le numéro d'identification de chaque article, ainsi que la quantité si celle-ci est supérieure à 1
3. La caisse affiche le prix de chaque article et son libellé
4. Lorsque tous les articles ont été enregistrés, le caissier signale la fin de la vente
5. La caisse affiche le total des achats
6. Le client choisit son mode de paiement :
   - Liquide : le caissier encaisse l'argent et la caisse indique le montant éventuel à rendre au client
   - Chèque : le caissier note l'identité du client et la caisse enregistre le montant sur le chèque
   - Carte de crédit : un terminal bancaire fait partie de la caisse, il transmet la demande à un centre d'autorisation multi-banques
7. La caisse enregistre la vente et imprime un ticket
8. Le caissier transmet le ticket imprimé au client
9. Un client peut présenter des coupons de réduction avant le paiement. Lorsque le paiement est terminé, la caisse transmet les informations relatives aux articles vendus au système de gestion des stocks. Tous les matins, le responsable du magasin initialise les caisses pour la journée.

- Donnez **un diagramme de cas d'utilisation** pour la caisse enregistreuse.
- Donnez une description textuelle d'un cas d'utilisation « enregistrer article ».

Exercice 05

Une entreprise qui vend et répare des voitures désire exploiter un logiciel de gestion des réparations sur son intranet. Elle dispose déjà d'un logiciel comptable. Les factures de réparation sont donc imprimées et gérées par ce logiciel comptable. Par contre, le logiciel de gestion des réparations doit communiquer avec celui-ci pour transmettre les réparations à facturer.

Le logiciel de gestion des réparations est destiné en priorité au chef d'atelier. Il doit lui permettre de saisir les fiches de réparation et le travail effectué par les divers employés de l'atelier. Pour effectuer leur travail, les mécaniciens et autres employés de l'atelier vont chercher des pièces de rechange au magasin. Lorsque le logiciel sera installé, les magasiniers ne fourniront des pièces que pour les véhicules pour lesquels une fiche de réparation est ouverte. Ils saisiront directement les pièces fournies depuis un terminal installé au magasin.

Lorsqu'une réparation est terminée, le chef d'atelier va essayer la voiture. Si tout est en ordre, il met la voiture sur le parc clientèle et ferme la fiche de réparation informatisée. Les fiches de réparation fermées doivent pouvoir être importées par le comptable dans le logiciel comptable.

À partir de ces informations, Etablissez **un diagramme des cas d'utilisation**.

Série TD N°07

Exercice 01

Un acteur adhérent peut emprunter un exemplaire d'une œuvre donnée. L'emprunt se fait de la façon suivante : la méthode emprunter est appelée ; s'il reste des exemplaires dans la bibliothèque, l'un des exemplaires associés à l'œuvre est extrait via la méthode extraireExemplaire, une instance de la classe Prêt est créée, puis l'exemplaire extrait de la bibliothèque est attribué à l'adhérent grâce à l'opération attribuer. S'il restait un exemplaire, l'œuvre retourne « OK »et dans le cas contraire, elle retourne « Pas OK ».

- Modéliser **le diagramme de séquence** correspondant

Exercice 02

Soit la description textuelle du cas d'utilisation « retrait d'espèces »:

1. Le guichetier saisit le numéro de compte du client ; Si le numéro de compte n'est pas valide, le système affiche un message d'erreur. Sinon, l'application valide le compte auprès du système central.
2. Le guichetier demande un retrait de 100 euros.
3. Le système guichet interroge le système central pour s'assurer que le compte est suffisamment approvisionné. Le système central vérifie le solde du compte. S'il est suffi, Le système central effectue le débit du compte et notifie au guichetier qu'il peut délivrer le montant demandé ; Sinon, il affiche un message « solde insuffisant ».

- Donner **le diagramme de séquences** associé à cette description textuelle dans le cas de la rubrique « enchaînement nominal » puis dans le cas « enchaînement alternative».

Exercice 03

Le déroulement de réservation de vol est le suivant:

Le Client sélectionne un vol. Une validation de nombre de place automatique du système se déclenche. Si le nombre de places réservés est inférieur au nombre de places disponibles dans un vol, la création de la réservation se lance, sinon une erreur s'affichera au client.

Le déroulement de création de vol est le suivant:

Le gestionnaire affecte un avion à un vol: il doit sélectionner un avion disponible et l'affecter au vol. Ensuite il doit sélectionner un aéroport de départ et un autre d'arrivée. Puis il crée un vol après avoir spécifié les autres informations requis.

- Elaborez **les diagrammes de séquences** relatifs à: « réserver dans un vol » et « créer un vol ».

Exercice 04

Pour passer une commande, le client fournit son numéro de téléphone (qui va permettre de l'identifier) et le contenu de sa commande qu'on représente sous la forme d'une liste de couples (nom d'une pizza, quantité).

Le système commence par vérifier que le client existe dans la base des clients et conserve l'objet client qui lui correspond. Le système vérifie ensuite que la commande est faisable (elle peut être fabriquée dans un Point Pizza qui fait toutes les pizzas spéciales demandées) puis crée la commande dans le système et en indique le numéro et le prix au client.

Pour vérifier que la commande est faisable, le système récupère la liste des Points Pizza pour chaque spécialité de la commande puis vérifie que l'intersection de ces listes n'est pas vide, c'est-à-dire qu'il existe au moins un Point Pizza qui peut satisfaire la commande. C'est une façon de faire un peu naïve, on peut trouver d'autres méthodes.

Le système crée ensuite une commande en créant les objets Détails Commande correspondant à chaque pizza (avec l'attribut état initialisé à en Attente) et en ajoutant chaque couple (pizza, détails) à l'objet commande créé. Il attribue ensuite le client à la commande puis ajoute la commande aux commandes du client. Il conserve l'objet commande créé (quand on crée un objet, on le renvoie nécessairement à celui qui l'a créé) puis il renvoie le numéro et le prix de la commande au client.

Donner le diagramme de séquence pour le cas d'utilisation « Passer une commande »

Exercice 05

Un organisme de formation assure une cinquantaine de stages différents dans plusieurs spécialités. Les inscriptions des candidats se font de la manière suivante :

Le candidat à la formation envoie sa demande d'inscription au responsable de formation après étude, la demande sera acceptée ou refusée en fonction des prérequis du stage demandé. Dans le cas d'une demande acceptée, le candidat est inscrit et dans le cas contraire le responsable de formation lui transmettra une lettre de regret.

Si 10 jours avant le début du stage, le nombre de candidats est supérieur ou égal à 5, le responsable de formation établit des convocations qu'il transmettra aux inscrits. Si par contre le nombre d'inscrits au stage est inférieur à 5, la session est alors reportée par le responsable de formation qui établira des lettres de report à l'intention des candidats.

Les annulations d'une inscription de la part d'un candidat inscrit doivent être faites par écrit au plus tard 10 jours avant le début du stage au responsable qui lui délivrera une décision d'annulation.

A chaque fin d'un stage de formation, le responsable de formation clôture la formation en établissant un PV de fin de stage et une liste des inscrits présents à tout ou une partie du stage ainsi que des attestations de formations. La liste est envoyée au comptable pour établir une facture pour chaque candidat. Le PV de fin de stage est classé.

Les factures sont transmises au responsable de formation pour signature qui les transmettra accompagnées des attestations aux inscrits.

Dessinez le diagramme de séquence.

Annexe B : Séries de Travaux Pratiques (TP)

TP01 Initialisation au langage Java

Le langage de programmation java

Java est donc un langage de programmation, un langage dit compilé. C'est-à-dire que ce que vous allez écrire n'est pas directement compréhensible et utilisable par votre ordinateur. Nous devrons donc passer par une étape de compilation (étape obscure où votre code source est entièrement transformé).

Points fort de java

- **Popularité** : Java est un langage de programmation Orienté Objet très utilisé
- **portabilité** : une fois votre programme créé, il fonctionnera automatiquement sous Windows, Mac, Linux, etc.
- **Diversité**: On peut faire de nombreux types de programmes avec Java
- **Interactivité**: permet de définir des interfaces graphiques (GUI)

Structure d'un programme Java

```
public class NomClasse {
    /*
     * ceci est un commentaire
     */
public static void main (String[] args)
{
    System.out.print("Welcome");
}
}
```

Les variables de type numérique

- Le type byte (1 octet) peut contenir les entiers entre -128 et +127.

```
byte temperature;
temperature = 64;
```

- Le type short (2 octets) contient les entiers compris entre -32768 et +32767.

```
short vitesseMax;
vitesseMax = 32000;
```

- Le type int (4 octets) va de -2 × 109 à 2 × 109 (2 et 9 zéros derrière . . . ce qui fait déjà un joli nombre).

```
int temperatureSoleil;
temperatureSoleil = 15600000;
```

- Le type long (8 octets) peut aller de -9 × 1018 à 9 × 1018 (encore plus gros. . .).

```
long anneeLumiere;
anneeLumiere = 9460700000000000;
```

- Le type float (4 octets) est utilisé pour les nombres avec une virgule flottante.

```
float pi;
pi = 3.141592653f;
```

- Le type double (8 octets) est identique à float, si ce n'est qu'il contient plus de chiffres derrière la virgule et qu'il n'a pas de suffixe.

```
double division;
division = 0.333333333333333333333333
33333333333333333334;
```

Les variables de type caractère

- Le type char contient UN caractère stocké entre apostrophes [' '], comme ceci :

```
char caractere;
caractere = 'A';
```

Les variables de type String

- Le type String permet de gérer les chaînes de caractères, c'est-à-dire le stockage de texte. Il s'agit d'une variable d'un type plus complexe que l'on appelle objet

```
Exemple :
String phrase1;
phrase1 = "Modélisation avec UML";
String phrase2 = new String();
phrase2 = "Modélisation avec UML 2";
String phrase3 = "Modélisation avec UML3";
String phrase4 = new String("Modélisation avec
UML 4!");
```

Les variables de type booléen

- Le type boolean, lui, ne peut contenir que deux valeurs : true (vrai) ou false (faux), sans guillemets.

```
boolean question;
question = true;
```

Les constantes

Déclaration d'une variable non modifiable →
Déclaration d'une constante

```
Final type nom_constante = valeur;
```

Les commentaires



L'écriture

- System,out,print (`` votre message``)

Pour faire un saut à la ligne :

- System,out,println (`` votre message``)
- System,out,print (`` votre message \n``)

Lecture en java (La classe Scanner)

Pour que Java puisse lire ce que vous tapez au clavier, vous allez devoir utiliser un objet de type Scanner.

La lecture des entrées clavier se fait via l'objet Scanner.

- Ce dernier se trouve dans le package java.util que vous devrez importer.
- Pour pouvoir récupérer ce vous allez taper dans la console, vous devrez initialiser l'objet Scanner avec l'entrée standard, System.in.
- Il y a une méthode de récupération de données pour chaque type (sauf les char) : nextLine() pour les String, nextInt(),..,etc

Exemple 1 (type String)

```
Scanner sc = new Scanner(System.in);
System.out.println("Veuillez saisir un mot:");
String str = sc.nextLine();
System.out.println("Vous avez saisi : " + str);
```

```
Problems  Javadoc  Declaration  Console
<terminated> NomClasse [Java Application] C:\Program Files\
Veuillez saisir un mot :
Bonjour
Vous avez saisi : Bonjour
```

Exemple 2 (autres types)

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
double d = sc.nextDouble();
long l = sc.nextLong();
byte b = sc.nextByte();
next<Type de variable commençant par une majuscule>
```

Exemple 3 (type Char)

```
System.out.println("Saisissez une lettre :");
Scanner sc = new Scanner(System.in);
String str = sc.nextLine();
char carac = str.charAt(0); /*récupérer le
 premier caractère saisi */
System.out.println("Vous avez saisi le
caractère : " + carac);
```

Les opérateurs

Operateur	Rôle
==	Tester l'égalité
!=	Tester l'inégalité
<	Strictement inférieur
<=	Inférieur ou égal.
>	Strictement supérieur.
>=	Supérieur ou égal.
&&	Le ET / AND logique (court circuit)
&	Le ET / AND logique
||	Le OU / OR logique (court circuit)
|	Le OU / OR logique
!	Le Non logique
?	Les conditions ternaires
^	Le OU exclusif

Remarque :

L'esperluette simple (&) évalue toujours les deux arguments alors que l'esperluette double (&&) n'évaluera le deuxième argument que si le premier argument est vrai.

Les structures de contrôles

Il existe plusieurs sortes de structures conditionnelles qui vous permettent de n'exécuter que certains morceaux de code. :

- la structure if... elseif... else ;
- la structure switch... case... default ;
- la structure ? :

La structure de contrôle (IF / else)

```
If (//condition)
{
... //Exécution des instructions si la condition est remplie
}
else
{
... //Exécution des instructions si la condition n'est pas remplie
}
```

La structure de contrôle (IF / elseif / else)

```
If (//condition)
{
//Exécution des instructions si la condition est vrai
}
elseif (//condition)
{
 //Exécution des instructions si la condition est vrai
}
Else
```

```
{
//Exécution des instructions si les autres conditions
n'sont pas remplie
}
```

La structure de contrôle (Switch)

```
switch (/*Variable*/)
{
case /*Argument*/:
/*Action*/;
break;
case /*Argument*/:
/*Action*/;
break;
default:
/*Action*/;
}
```

La structure de contrôle (?)

```
Variable = (condition) ? /*Action*/ : /*Action*/;
```

```
int x = 10, y = 20;
int max = (x < y) ? y : x ; //Maintenant, max vaut 20
```

Si la condition renvoie true , la valeur qui se trouve après le ? sera affectée ; sinon, la valeur se trouvant après le symbole : sera affectée.

Les boucles

- ***La boucle while***

```
while (/* Condition */)
{
//Instructions à répéter
}
```

- ***La boucle do... while***

```
do{
//instructions
} while(a < b);
```

Ces boucles fonctionnent exactement de la même manière Sauf que La boucle « do... while » s'exécutera au moins une fois, contrairement à la boucle « while »

- ***La boucle for***

```
for(int i = 1; i <= n; i++)
{
/*actions
}
```

Note:

Dans les boucles "while" ou "do ..while":

- déclaration d'une variable avant la boucle
- initialisation de cette variable
- incrémentation de celle-ci dans la boucle.

Dans la boucle « for », tous ces étapes sont dans la condition elle-même.

Les instructions break /continue

- ***Break***

Sortir d'un bloc d'instructions sans traiter les instructions restantes dans ce bloc et le contrôle du programme reprend à l'instruction suivante qui suit le bloc

- ***Continue***

Toutes les déclarations après l'instruction continue sont omises et la boucle continue avec la prochaine itération.

TP01 / Exercice 01

Écrire un programme calculant la somme des *n* premiers termes de la "série harmonique", c'est-à-dire la somme :

$$Ar(n)=1+1/2+1/3+1/4+\ldots\ldots+1/n$$

La valeur de *n* sera lue par l'utilisateur.

TP01 / Exercice 02

Écrire un programme qui calcule les racines carrées de nombres fournis en donnée. Il s'arrêtera lorsqu'on lui fournira la valeur 0. Il refusera les valeurs négatives. Son exécution se présentera ainsi :

```
donnez un nombre positif : 2
sa racine carrée est : 1.4142135623730951
donnez un nombre positif : -3
svp positif
donnez un nombre positif : 5
sa racine carrée est : 2.23606797749979
donnez un nombre positif : 0
```

TP01 / Exercice 03

Il s'agit d'écrire un programme qui lire une chaîne de caractères (une instance de la classe String) puis indique s'il s'agit ou non d'un palindrome.

Note : Un palindrome est un mot qui se lit de la même façon de gauche à droite et de droite à gauche (ex: ICI, ETE, RADAR).

Modélisation Objet avec UML
(MOU)

TP 02

Les tableaux



Déclaration des tableaux

```
int monTableau[ ] = new int[10];
```

```
int [ ] monTableau = new int[10];
```

Remplissage d'un tableau

```
int [] monTableau = new int[10];
monTableau[5] = 23;
```

```
int [] monTableau = {5,8,6,0,7};
```

```
monTableau = new int[]{11,13,17,19,23,29};
```

Parcourir un tableau

```
for(int i=0 ; i<monTableau.length ; i++)
    {
        //Action sur monTableau[i]

    }
```

Tableaux à plusieurs dimensions

```
int[ ][ ] matrice=new int[5][6];
```

Parcourir un tableau

```
for (int i=0 ; i<matrice.length ;i++)
        {
        for (int j=0 ; j<matrice[i].length ;j++)
              {
              //Action sur matrice[i][j]
              }
        }
```

Lecture d'une matrice

```
int t22[][] ;
t22 = new int[5][2];

for( int i = 0; i< t22.length; ++i)
{
  for( int j = 0; j<t22[i].length; ++j)
  {
     t22[i][j]=sc.nextInt();


  }
}
```

TP02 / Exercice 01

Écrire un programme permettant de calculer la somme, le produit et la moyenne d'un tableau de dix réels.

TP02 / Exercice 02

Écrire un programme permettant de déterminer la valeur maximale et minimale, avec leurs positions dans un tableau de 8 entiers.



TP02 / Exercice 03

Soit T un tableau de N=10 entiers naturels. Ranger les nombres pairs au début du tableau T et les nombres impairs à la fin.

TP02 / Exercice 04

Écrire un programme permettant de remplir un tableau à deux dimensions (3*5) à partir du clavier, et tester l'existence d'une certaine valeur saisie par l'utilisateur,
Si elle existe, déterminer leur nombre d'apparition

Annexe B : Séries de Travaux Pratiques (TP)

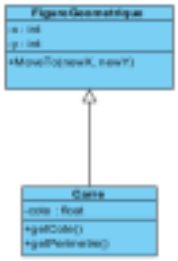
Modélisation Objet avec UML
(MOU)

Tp 05

Java et la Programmation Orientée Objet
(part III)

Travaille demandé

- Créer un nouveau projet: TP05
- Créer les classes avec le constructeurs paramétré et redéfinir le constructeur par défaut



Travaille demandé

- Créer un nouveau projet: TP05
- Créer les classes avec le constructeurs paramétré et redéfinir le constructeur par défaut
- Créer une classe test
- Créer une instance de la classe FigureGeométrique avec x=3, y=5
- Faire l'appel à la méthode MoveTo avec les coordonnées (1,1) puis afficher les valeurs precedantes et les nouvelles valeurs,
- Créer un objet de la classe carré avec (x=2,y1,cote=4)
- Calculer et afficher le périmètre
- Faire l'appel à la méthode MoveTo avec les coordonnées (2,4) , puis afficher les nouvelles valeurs du carée

- Créer une classe cercle qui hérite de la classe FigureGeométrique
- La méthode ToString affiche :

« Cercle: Position (x , y), Rayon = rayon »

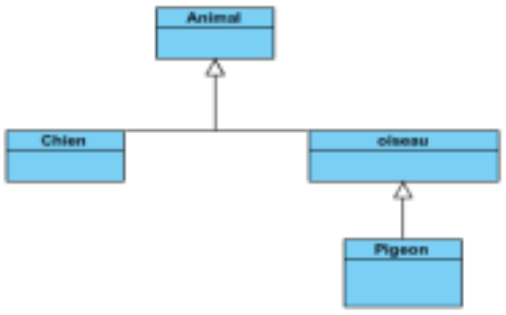
- Ajouter la méthode ToString dans la classe carré et FigureGeométrique
- Dans la classe test, créer une listes contient 3 objets, (FigureGeométrique , carré , cercle) et afficher les information de chaque objet en utilisant la méthode ToString
- Comment afficher le forme et les valeurs des attributs de chaque objet?,

Polymorphisme

@Override

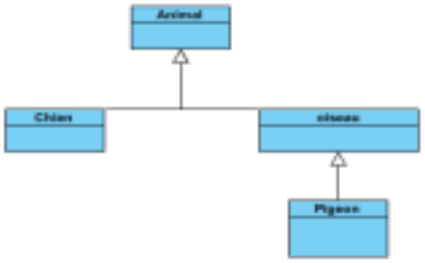
- L'annotation @Override informe explicitement les développeurs (et le compilateur) que la méthode redéfinit une méthode héritée. Cela rend le code plus lisible et maintenable, surtout dans des projets collaboratifs.
- Si une méthode est censée redéfinir une méthode existante mais qu'il y a une erreur (par exemple, une faute de frappe dans le nom de la méthode ou une mauvaise signature), le compilateur génère une erreur lorsque @Override est utilisé.

Polymorphisme



Travaille demandé

- Créer un nouveau projet: TP06
- Créer les 4 classes: Animal, Chien, Oiseau, Pigeon;



Travaille demandé

- Implémenter la méthode : deplacer() dans chaque classe. Cette méthode permet d'afficher:

Dans la classe Animal: « Je me déplace »

Dans la classe Chien : « Je marche »

Dans la classe Oiseau: « Je vole »

Dans la classe Pigeon: « Je vole surtout en ville »



Polymorphisme



Travaille demandé

- Créer une classe « Test »
- Dans cette classe, créer 4 objets de type: Animal, Chien, Oiseau, Pigeon
- Appeler la méthode « deplacer » avec chaque objet
- Que remarquez-vous?



```
class Animal {
   void deplacer() {
   System.out.println("Je me déplace");
}
```

```
class Chien extends Animal {

  void deplacer() {
    System.out.println("Je marche");
  }

}
```

```
class Oiseau extends Animal {

  void deplacer(){
    System.out.println("Je vole");
  }

}
```

```
class Pigeon extends Oiseau {

  void deplacer() {
    System.out.println("Je vole surtout en ville");
  }

}
```

```
public class Test {
public static void main(String[] args) {

   Animal a1 = new Animal();

   Animal a2 = new Chien();

   Animal a3 = new Pigeon();

   a1.deplacer();

   a2.deplacer();

   a3.deplacer();

   }

}
```

L'affichage

Je me déplace

Je marche

Je vole surtout en ville

Polymorphisme

Le Java, l'une des meilleures les plus connues et utilisées est @Override. Elle est utilisée en complément de polymorphisme pour indiquer que la méthode annotée est une substitution d'une méthode de la classe mère. Si l'annotation est présente sur une méthode, le compilateur va vérifier que la signature de la méthode est bien identique à celle de la méthode mère.

Polymorphisme

```
class Animal {
  void deplacer() {
  System.out.println("Je me déplace");
}
class Chien extends Animal {
  @Override
  void deplacer() {
    System.out.println("Je marche");
  }
}
```

Classe abstraite

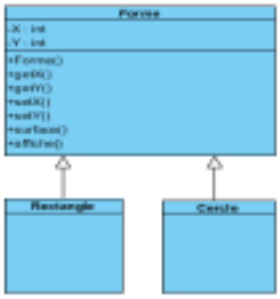


Travaille demandé

❑Dans le projet TP06, créer un nouveau package :

LesClassesAbstraites

❑Créer les 3 classes: Forme, Rectangle, Cercle



Travaille demandé

❑Implémenter la classe Forme qui possède deux méthodes abstraites: Surface() et Afficher(). Les autres méthodes sont implémenté,

Modélisation Objet avec UML (MOU)

Tp 07

AUML

Contexte

Dans le cadre du développement d'une pharmacie en ligne (système multi-agents), nous souhaitons modéliser le processus d'achat. Contrairement à une approche objet classique, les entités ici sont autonomes (Agents) et communiquent via un protocole défini.

Scénario

Un agent Client souhaite acheter un ou plusieurs médicaments. Il contacte l'agent Administrateur du système.
Le Client envoie une demande contenant la liste des médicaments.
L'Administrateur consulte ses stocks (processus interne).

Deux issues sont possibles :

Cas A : Les médicaments sont disponibles.
L'Administrateur accepte la commande et informe le client.
Cas B : Rupture de stock. L'Administrateur refuse la commande et signale l'échec.

Travaille demandé

- Identification : Listez les agents impliqués et définissez le rôle de chacun.
- Sémantique : Identifiez les actes de communication nécessaires (ex demander, informer, refuser) pour chaque étape du scénario.
- Modélisation : Réalisez le Diagramme de Séquence AUML correspondant, en utilisant les notations appropriées pour les messages et les structures de contrôle (choix).

Agents et Rôles identifiés

Agent	Rôle	Description
Agent Client	Demandeur (Initiateur)	Il a l'initiative de l'interaction. Son but est d'acquérir un produit.
Agent Admin	Gestionnaire (Participant)	Il reçoit la demande. Il a la capacité de décision (vérifier le stock) et de réponse (valider/refuser).

L'échange repose sur trois actes performatifs :
- request (Demande) : Utilisé par le Client pour demander à l'Admin d'effectuer l'action "Traiter la commande".
- inform (Information) : Utilisé par l'Admin pour confirmer que la commande est validée (l'action est faite).
- failure (Echec) : Utilisé par l'Admin pour signaler qu'il ne peut pas satisfaire la demande (Rupture de stock).

Le Diagramme de Séquence AUML

