Plan :

Introduction :

I) Notion Analyse et de Conception

A. **Approche de la Décomposition Fonctionnelle**:
   a) Demarche
   b) Avantages et Inconvénients

B. Approche Orienté Objet :UML

- **Demarche**
- **Avantages**
- **Inconvénients**

C. Etude Comparative entre Merise et UML

II) UML

1) Approche 4 vues(Logique,Processus,Composant,Déploiement) + 1(Vue des besoins)

2) Les 13 Diagrammes UML et Classification des Diagrammes par Aspect (Fonctionnel ou Architecture) et par vue

Introduction :

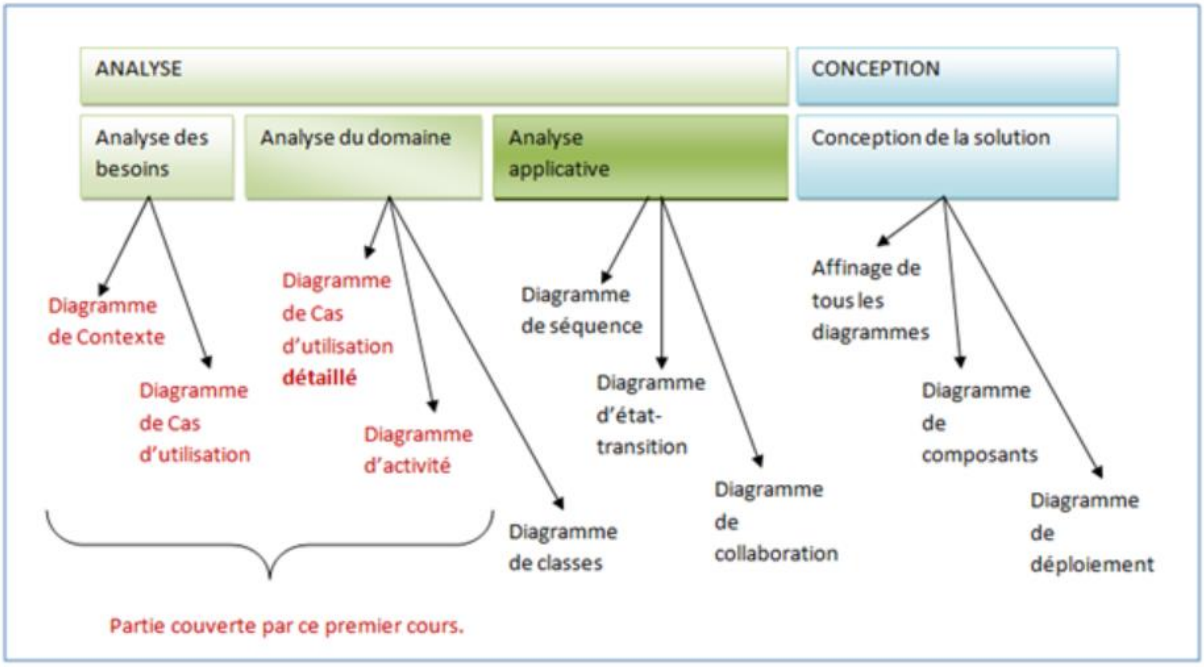
Le **Langage de Modélisation Unifié**, de l'anglais *Unified Modeling Language* (**UML**), est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est couramment utilisé en développement logiciel et en conception orientée objet.

I) Notion Analyse et de Conception

En ingénierie, une **méthode d'analyse et de conception** est un procédé qui a pour objectif de permettre de formaliser les étapes préliminaires du développement d'un système afin de rendre ce développement plus fidèle aux besoins du client. Pour ce faire, on part d'un énoncé informel (le besoin tel qu'il est exprimé par le client, complété par des recherches d'informations auprès des experts du domaine fonctionnel, comme les futurs utilisateurs d'un logiciel), ainsi que de l'analyse de l'existant éventuel (c'est-à-dire la manière dont les processus à traiter par le système se déroulent actuellement chez le client).

La phase d'analyse permet de lister les résultats attendus, en termes de fonctionnalités, de performance, de robustesse, de maintenance, de sécurité, d'extensibilité, etc.

La phase de conception permet de décrire de manière non ambiguë, le plus souvent en utilisant un langage de modélisation, le fonctionnement futur du système, afin d'en faciliter la réalisation.



A. **Approche de la Décomposition Fonctionnelle**:

1. La découpe fonctionnelle d'un problème informatique : une approche intuitive La découpe fonctionnelle d'un problème (sur laquelle reposent les langages de programmation structurée) consiste à découper le problème en blocs indépendants. En ce sens, elle présente un caractère intuitif fort.

2. La réutilisabilité du code Le découpage d'un problème en blocs indépendants (fonctions et procédures) va permettre aux programmeurs de réutiliser les fonctions déjà développées (à condition qu'elles soient suffisamment génériques). La productivité se trouve donc accrue.

3. Le revers de la médaille : maintenance complexe en cas d'évolution Le découpage en blocs fonctionnels n'a malheureusement pas que des avantages. Les fonctions sont devenues interdépendantes : une simple mise à jour du logiciel à un point donné, peut impacter en cascade une multitude d'autres fonctions. On peut minorer cet impact, pour peu qu'on utilise des fonctions plus génériques et des structures de données ouvertes. Mais respecter ces contraintes rend l'écriture du logiciel et sa maintenance plus complexe

4. Problèmes générés par la séparation des données et des traitements : Examinons le problème de l'évolution de code fonctionnel plus en détail... Faire évoluer une application de gestion de bibliothèque pour gérer une médiathèque, afin de prendre en compte de nouveaux types d'ouvrages (cassettes vidéo, CD-ROM, etc...), nécessite : ◦ de faire évoluer les structures de données qui sont manipulées par les fonctions, ◦ d'adapter les traitements, qui ne manipulaient à l'origine qu'un seul type de document (des livres). Il faudra donc modifier toutes les portions de code qui utilisent la base documentaire, pour gérer les données et les actions propres aux différents types de documents .

5. 1 ère amélioration : rassembler les valeurs qui caractérisent un type, dans le type Une solution relativement élégante à la multiplication des branches conditionnelles et des redondances dans le code (conséquence logique d'une trop grande ouverture des données), consiste tout simplement à centraliser dans les structures de données, les valeurs qui leurs sont propres. Par exemple, le délai avant rappel peut être défini pour chaque type de document. Cela permet donc de créer une fonction plus générique qui s'applique à tous les types de documents.

6. 2ème amélioration : centraliser les traitements associés à un type, auprès du type Pourquoi ne pas aussi rassembler dans une même unité physique les types de données et tous les traitements associés ? Que se passerait-il par exemple si l'on centralisait dans un même fichier, la structure de données qui décrit les documents et la fonction de calcul du délai avant rappel ? Cela nous permettrait de retrouver immédiatement la partie de code qui est chargée de calculer le délai avant rappel d'un document, puisqu'elle se trouve au plus près de la structure de données concernée. Ainsi, si notre médiathèque devait gérer un nouveau type d'ouvrage, il suffirait de modifier une seule fonction (qu'on sait retrouver instantanément), pour assurer la prise en compte de ce nouveau type de document dans le calcul du délai avant rappel. Plus besoin de fouiller partout dans le code... Écrit en ces termes, le logiciel serait plus facile à maintenir et bien plus lisible. Le stockage et le calcul du délai avant rappel des documents, serait désormais assuré par une seule et unique unité physique (quelques lignes de code, rapidement identifiables).

• Methode Merise

MERISE (Méthode d'Étude et de Réalisation Informatique pour les Systèmes d'Entreprise) est certainement le langage de spécification le plus répandu dans la communauté de l'informatique des systèmes d'information, et plus particulièrement dans le domaine des bases de données. Une représentation Merise permet de valider des choix par rapport aux objectifs, de quantifier les solutions retenues, de mettre en œuvre des techniques d'optimisation et enfin de guider jusqu'à l'implémentation. Reconnu comme standard, Merise devient un outil de communication. En effet, Merise réussit le compromis difficile entre le souci d'une modélisation précise et formelle, et la capacité d'offrir un outil et un moyen de communication accessible aux non-informaticiens. Un des concepts clés de la méthode Merise est la séparation des données et des traitements. Cette méthode est donc parfaitement adaptée à la modélisation des problèmes abordés d'un point de vue fonctionnel. Les données représentent la statique du système d'information et les traitements sa dynamique. L'expression conceptuelle des données conduit à une modélisation des données en entités et en associations.

perspective de la modélisation de bases de données.

a) **Demarche**

Merise propose une démarche, dite par niveaux, dans laquelle il s'agit de hiérarchiser les préoccupations de modélisation qui sont de trois ordres : la conception, l'organisation et la technique. En effet, pour aborder la modélisation d'un système, il convient de l'analyser en
premier lieu de façon globale et de se concentrer sur sa fonction : c'est-à-dire de s'interroger sur ce qu'il fait avant de définir comment il le fait. Ces niveaux de modélisation sont organisés dans
une double approche données/traitements. Les trois niveaux de représentation des données,
puisque ce sont eux qui nous intéressent, sont détaillés ci-dessous:

⬡ Niveau conceptuel : le modèle conceptuel des données (MCD) décrit les entités du monde réel, en terme d'objets, de propriétés et de relations, indépendamment de toute technique d'organisation et d'implantation des données. Ce modèle se concrétise par un schéma entités-associations représentant la structure du système d'information, du point de vue des données.

⬡ **Niveau logique : le modèle logique des données (MLD) précise le modèle conceptuel**

par des choix organisationnels. Il s'agit d'une transcription (également appelée dérivation) du MCD dans un formalisme adapté à une implémentation ultérieure, au niveau physique, sous forme de base de données relationnelle ou réseau, ou autres. Les
choix techniques d'implémentation (choix d'un SGBD) ne seront effectués qu'au niveau suivant.

⬡ **Niveau physique : le modèle physique des données (MPD) permet d'établir la manière**
concrète dont le système sera mis en place (SGBD retenu).

b) Avantanges et Inconvenients

-Pour petites bases de données, limitées à la troisième forme normale est généralement l'une des meilleures solutions du point de vue de l'architecture de base de données, mais pour les grandes bases de données, ce n'est pas toujours le cas. Il s'agit de choisir la balance entre deux options.

-En d'autres termes, il ressort clairement de ces avantages et les inconvénients que l'arbitrage sera effectué sur le niveau de la normalisation sur la base des tables de la base de données sont priés de lire ou d'écrire plus. Si une table (base de données) est écrite plus en détail que de lire, il est préférable de normaliser autant que possible. Inversement, si une table (base de données) est plus largement lue et écrite, il peut être sage d'être moins strict sur le respect des normes afin d'améliorer les performances d'accès aux données.

-Il faut être prudent lorsqu'on renonce à la forme normale. Il n'ya aucune garantie qu'une forme dénormalisée améliore le temps d'accès. En fait, la redondance peut provoquer une explosion des volumes de données qui peuvent réduire le rendement ou de saturer les disques durs.

-La normalisation des modèles de données a été popularisé notamment par la méthode Merise. La principale limitation de la normalisation est que les données doivent être dans la même base de données (en un seul diagramme).

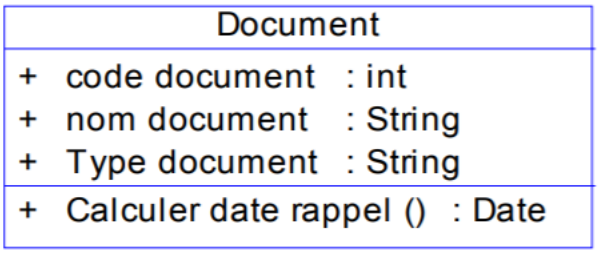
-Même si les échanges et la consultation entre concepteurs et utilisateurs sont formellement organisés, on a aussi reproché à Merise d'utiliser un formalisme jugé complexe (surtout pour les modèles de données), qu'il faut d'abord apprendre à manier, mais qui constitue ensuite un véritable langage commun, puissant et rigoureux pour qui le maîtrise.

-En outres, les limites de Merise sont telles que : elle est bien adaptée pour l'automatisation des tâches séquentielles de gestion pure. Toutefois, il est peu adapté pour les environnements distribués où plusieurs applications sont externes à un domaine d'interagir avec le modèle d'application. En d'autres termes, elle n'est pas en mesure de modéliser les données à caractère sémantique.

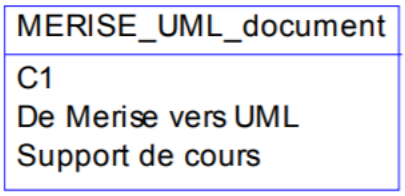
B. Approche Orienté Objet :UML

Les modifications que nous avons apportées à notre logiciel de gestion de médiathèque nous ont amenés à transformer ce qui était à l'origine une structure de données, manipulée par des fonctions, en une entité autonome, qui regroupe un ensemble de propriétés cohérentes et de traitements associés. Une telle entité s'appelle… un objet et constitue le concept fondateur de l'approche du même nom !

- Un objet est une entité aux frontières précises qui possède une identité (un nom).
- Un ensemble d'attributs caractérise l'état de l'objet.
- Un ensemble d'opérations (méthodes) en définissent le comportement.
- Un objet est une instance de classe (une occurrence d'un type abstrait).
- Une classe est un type de données abstrait, caractérisé par des propriétés (attributs et méthodes) communes à des objets et permettant de créer des objets possédant ces propriétés.



Classe : regroupement d'objets



Objet : instance d'une classe

✓ Demarche

UML est un langage qui permet de représenter des modèles, mais il ne définit pas le processus d'élaboration des modèles : UML n'est donc pas une méthode de modélisation. Cependant, dans le cadre de la modélisation d'une application informatique, les auteurs d'UML préconisent d'utiliser une démarche :

- itérative et incrémentale,

- guidée par les besoins des utilisateurs du système,

- centrée sur l'architecture logicielle.

D'après les auteurs d'UML, un processus de développement qui possède ces qualités devrait favoriser la réussite d'un projet.

-UNE DEMARCHE ITERATIVE ET INCREMENTALE Pour (comprendre et représenter) un système complexe, il vaut mieux s'y prendre en plusieurs fois, en affinant son analyse par étapes. Cette démarche doit aussi s'appliquer au cycle de développement dans son ensemble, en favorisant le prototypage. Le but est de mieux maîtriser la part d'inconnu et d'incertitudes qui caractérisent les systèmes complexes.

- UNE DEMARCHE PILOTEE PAR LES BESOINS DES UTILISATEURS

Avec UML, ce sont les utilisateurs qui guident la définition des modèles : Le périmètre du système à modéliser est défini par les besoins des utilisateurs (les utilisateurs définissent ce que doit être le système). Le but du système à modéliser est de répondre aux besoins de ses utilisateurs (les utilisateurs sont les clients du système). Les besoins des utilisateurs servent aussi de fil rouge, tout au long du cycle de développement (itératif et incrémental) : - à chaque itération de la phase d'analyse, on clarifie, affine et valide les besoins des utilisateurs. - à chaque itération de la phase de conception et de réalisation, on veille à la prise en compte des besoins des utilisateurs. - à chaque itération de la phase de test, on vérifie que les besoins des utilisateurs sont satisfaits.

- UNE DEMARCHE CENTREE SUR L'ARCHITECTURE Une architecture adaptée est la clé de voûte du succès d'un développement. Elle décrit des choix stratégiques qui déterminent en grande partie les qualités du logiciel (adaptabilité, performances, fiabilité...). Ph. Kruchten propose différentes perspectives, indépendantes et complémentaires, qui permettent de définir un modèle d'architecture (publication IEEE, 1995). Ph. Kruchten défend l'idée que l'architecture logicielle doit être une discipline à part entière. Il propose que plusieurs perspectives concourent à l'expression de l'architecture d'un système et il explique qu'il est nécessaire de garantir la séparation et l'indépendance de ces différentes perspectives. L'évolution de l'une des perspectives ne doit pas avoir d'impact (sinon limité) sur les autres.

✓ Avantage

UML est un langage formel et normalisé Il permet ainsi : - un gain de précision - un gage de stabilité - l'utilisation d'outils UML EST UN SUPPORT DE COMMUNICATION PERFORMANT Il cadre l'analyse et facilite la compréhension de représentations abstraites complexes. Son caractère polyvalent et sa souplesse en font un langage universel.

✓ Inconvénients

La mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation. Même si l'Espéranto est une utopie, la nécessité de s'accorder sur des modes d'expression communs est vitale en informatique. UML n 'est pas à l'origine des concepts objets, mais en constitue une étape majeure, car il unifie les différentes approches et en donne une définition plus formelle. Le processus (non couvert par UML) est une autre clé de la réussite d'un projet. L'intégration d'UML dans un processus n'est pas triviale et améliorer un processus est un tâche complexe et longue.

C. Etude Comparative entre Merise et UML

MERISE (*Méthode d'Etude et de Réalisation Informatique pour les Systèmes d'Entreprise*) est une méthode d'analyse et de réalisation des systèmes d'information qui est élaborée en plusieurs étapes : schéma directeur, étude préalable, étude détaillée et la réalisation.

Alors qu'**UML** (*UnifiedModeling Langage*), est un langage de modélisation des systèmes standard, qui utilise des diagrammes pour représenter chaque aspect d'un système c'est - à - dire : statique, dynamique, ... en s'appuyant sur la notion d'orienté objet qui est un véritable atout pour ce langage.

Tableau 1 : tableau comparatif UML - Merise

Merise	UML
méthode d'analyse et de conception de système d'information	langage de représentation d'un système d'information.
méthode de modélisation de données et traitements orienté bases de données relationnelles.	système de notation orienté objet.
relationnel	objet.
Franco-français	International
schéma directeur, étude préalable, étude détaillée et la réalisation.	langage de modélisation des systèmes standard, qui utilise des diagrammes pour représenter chaque aspect d'un systèmes ie: statique, dynamique,....en s'appuyant sur la notion d'orienté objet
plus adapté à une approche théorique	plus orientée vers la conception
du "bottom up" de la base de donnée vers le code	du "top down" du modèle vers la base de donnée.

II. UML

UML, c'est l'acronyme anglais pour « Unified Modeling Language ». On le traduit par «
Langage de modélisation unifié ». La notation UML est un langage visuel constitué d'un
ensemble de schémas, appelés des diagrammes, qui donnent chacun une vision différente du
projet à traiter. UML nous fournit donc des diagrammes pour représenter le logiciel à
développer : son fonctionnement, sa mise en route, les actions susceptibles d'être effectuées par
le logiciel, etc.
Réaliser ces diagrammes revient donc à modéliser les besoins du logiciel à développer. Ces
diagrammes sont au nombre de onze on distingue parmi eux, le diagramme de classe et le
diagramme de cas d'utilisation.
UML se base sur l'approche objet qui est une démarche qui s'organise autour de 4 principes
fondamentaux. C'est une démarche :

- ⬡ itérative et incrémentale ;
- ⬡ guidée par les besoins du client et des utilisateurs ;
- ⬡ centrée sur l'architecture du logiciel ;

qui décrit les actions et les informations dans une seule entité.

1) Approche 4 vues(Logique,Processus,Composant,Déploiement) + 1(Vue des besoins)

- La vue Logique

Cette vue concerne « l'intégrité de conception ». Cette vue de haut niveau se concentre sur l'abstraction et l'encapsulation, elle modélise les éléments et mécanismes principaux du système. Elle identifie les éléments du domaine, ainsi que les relations et interactions entre ces éléments « notions de classes et de relations » :

• les éléments du domaine sont liés au(x) métier(s) de l'entreprise,

• ils sont indispensables à la mission du système,

• ils gagnent à être réutilisés (ils représentent un savoir-faire). Cette vue organise aussi (selon des critères purement logiques), les éléments du domaine en "catégories" :

• pour répartir les tâches dans les équipes,

• regrouper ce qui peut être générique,

• isoler ce qui est propre à une version donnée, etc...

- ## La Vie des Processus

Cette vue concerne « l'intégrité d'exécution ». Cette vue est très importante dans les environnements multitâches ; elle exprime la perspective sur les activités concurrentes et parallèles. Elle montre ainsi :

• la décomposition du système en terme de processus (tâches).

• les interactions entre les processus (leur communication).

• la synchronisation et la communication des activités parallèles (threads).

- ### La Vue Des Composants

Cette vue concerne « l'intégrité de gestion ». Elle exprime la perspective physique de l'organisation du code en termes de modules, de composants et surtout des concepts du langage ou de l'environnement d'implémentation. Dans cette perspective, l'architecte est surtout concerné par les aspects de gestion du code, d'ordre de compilation, de réutilisation, d'intégration et d'autres contraintes de développement pur. Pour représenter cette perspective, UML fournit des concepts adaptés tels que les modules, les composants, les relations de dépendance, l'interface … Cette vue de bas niveau (aussi appelée « vue de réalisation »), montre ainsi :

• l'allocation des éléments de modélisation dans des modules (fichiers sources, bibliothèques dynamiques, bases de données, exécutables, etc...). Cette vue identifie les modules qui réalisent (physiquement) les classes de la vue logique.

- ### La Vue De Deploiement

Cette vue concerne « l'intégrité de performance ». Elle exprime la répartition du système à travers un réseau de calculateurs et de nœuds logiques de traitements . Cette vue est particulièrement utile pour décrire la distribution d'un système réparti. Elle montre :

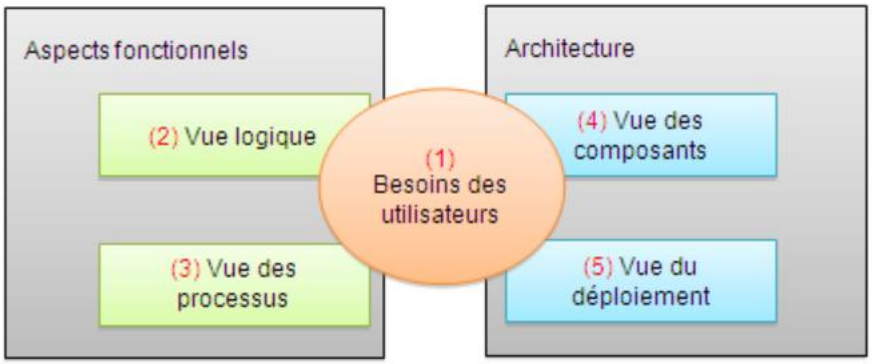
• la disposition et nature physique des matériels, ainsi que leurs performances.

• l'implantation des modules principaux sur les nœuds du réseau.

• les exigences en terme de performances (temps de réponse, tolérance aux fautes et pannes...).

- ## La Vue Des Besoins

Cette vue est particulière en ce sens qu'elle guide toutes les autres. Cette vue permet : • de trouver le « bon » modèle Les cas d'utilisation permettent de guider la modélisation. L'utilisation des scénarios et des cas d'utilisation s'avère plus rigoureuse et plus systématique que les entretiens et l'analyse des documents pour découvrir les abstractions du domaine. • d'expliquer et de justifier ses choix Il est en effet nécessaire d'expliquer le système, de justifier les choix qui ont guidé sa conception et son fonctionnement pour pouvoir le construire, le maintenir et le tester. Pour cela UML offre des concepts adaptés tels que les scénarios et les cas d'utilisation.



2) Les 13 Diagrammes UML et Classification des Diagrammes par Aspect (Fonctionnel ou Architecture) et par vue

Tout comme la construction d'une maison nécessite des plans à différents niveaux (vision extérieure, plan des différents étages, plans techniques…), la réalisation d'une application informatique ou d'un ensemble d'applications est basée sur plusieurs diagrammes. Comme je vous le disais dans le premier chapitre, le langage UML est constitué de diagrammes. À ce jour, il existe **13 diagrammes** « officiels ». Ces diagrammes sont tous réalisés **à partir du besoin des utilisateurs** et peuvent être regroupés selon les deux aspects suivants :

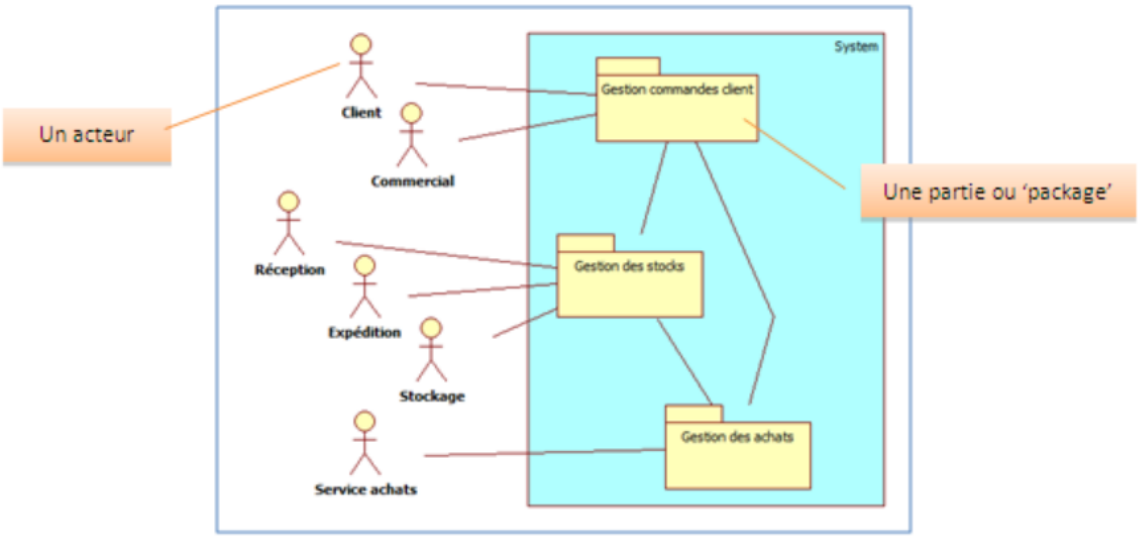
-Les aspects fonctionnels : Qui utilisera le logiciel et pour quoi faire ? Comment les actions devront-elles se dérouler ? Quelles informations seront utilisées pour cela ?

-Les aspects liés à l'architecture : Quels seront les différents composants logiciels à utiliser (base de données, librairies, interfaces, etc.) ? Sur quel matériel chacun des composants sera installé ?

- le diagramme de contexte ;

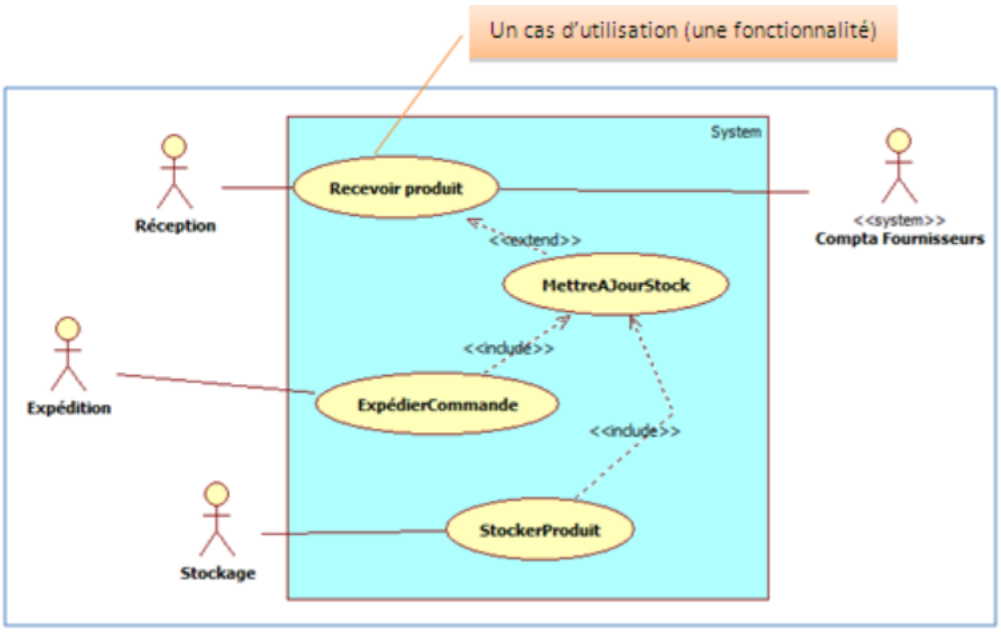
Ce diagramme n'est pas officiellement désigné comme diagramme UML. Il ne fait donc pas partie des 13 diagrammes « officiels », mais il est utile pour la définition des acteurs, avant de commencer à s'intéresser à d'autres aspects, tels que les packages et les cas d'utilisation.

- **Le diagramme de packages** permet de décomposer le système en catégories ou parties plus facilement observables, appelés « packages ». Cela permet également d'indiquer les acteurs qui interviennent dans chacun des packages.



Un exemple de diagramme de package

- **Le diagramme de cas d'utilisation** représente les fonctionnalités (ou dit cas d'utilisation) nécessaires aux utilisateurs. On peut faire un diagramme de cas d'utilisation pour le logiciel entier ou pour chaque package.



Un exemple de diagramme de cas d'utilisation pour un package (Gestion des stocks)

L’aspect fonctionnel du logiciel

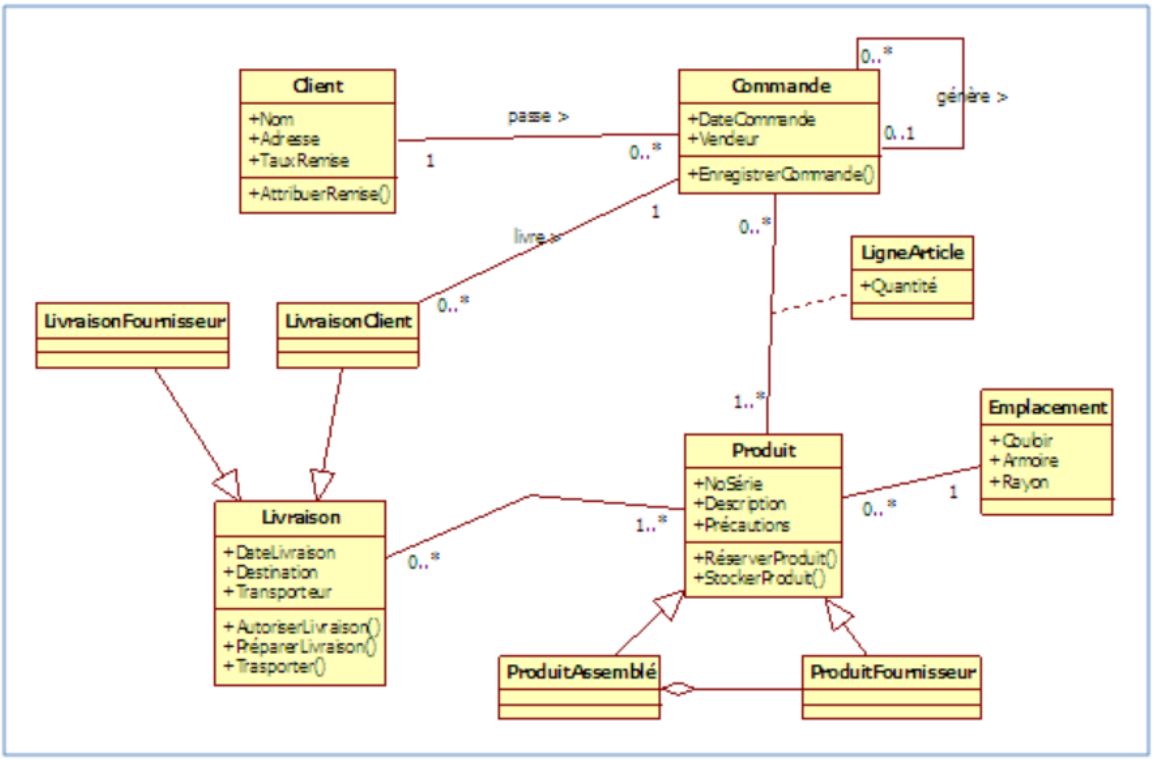
Pour rappel, cette partie du schéma 4+1 vues permet de définir qui utilisera le logiciel et pour quoi faire, comment les fonctionnalités vont se dérouler, etc.

Vue logique (2)

La vue logique a pour but d’identifier les éléments du domaine, les relations et interactions entre ces éléments. Cette vue organise les éléments du domaine en « catégories ». Deux diagrammes peuvent être utilisés pour cette vue.

- **Le diagramme de classes**

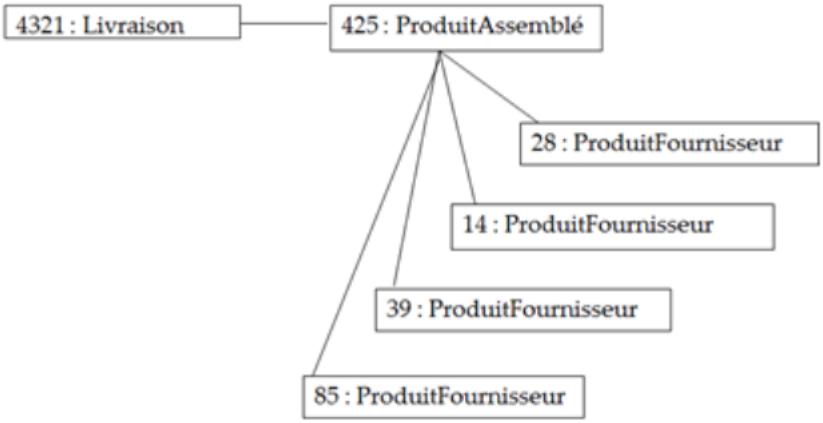
Dans la phase d’analyse, ce diagramme représente les entités (des informations) manipulées par les utilisateurs.
Dans la phase de conception, il représente la structure objet d’un développement orienté objet.
Ce diagramme ne sera pas étudié dans ce cours.



Un exemple de diagramme de classes (utilisé en phase d’analyse)

- **Le diagramme d'objets** sert à illustrer les classes complexes en utilisant des exemples d'instances.

Une instance est un exemple concret de contenu d'une classe. En illustrant une partie des classes avec des exemples (grâce à un diagramme d'objets), on arrive à voir un peu plus clairement les liens nécessaires. Ce diagramme ne sera pas étudié dans ce cours.



Un exemple de diagramme d'objet

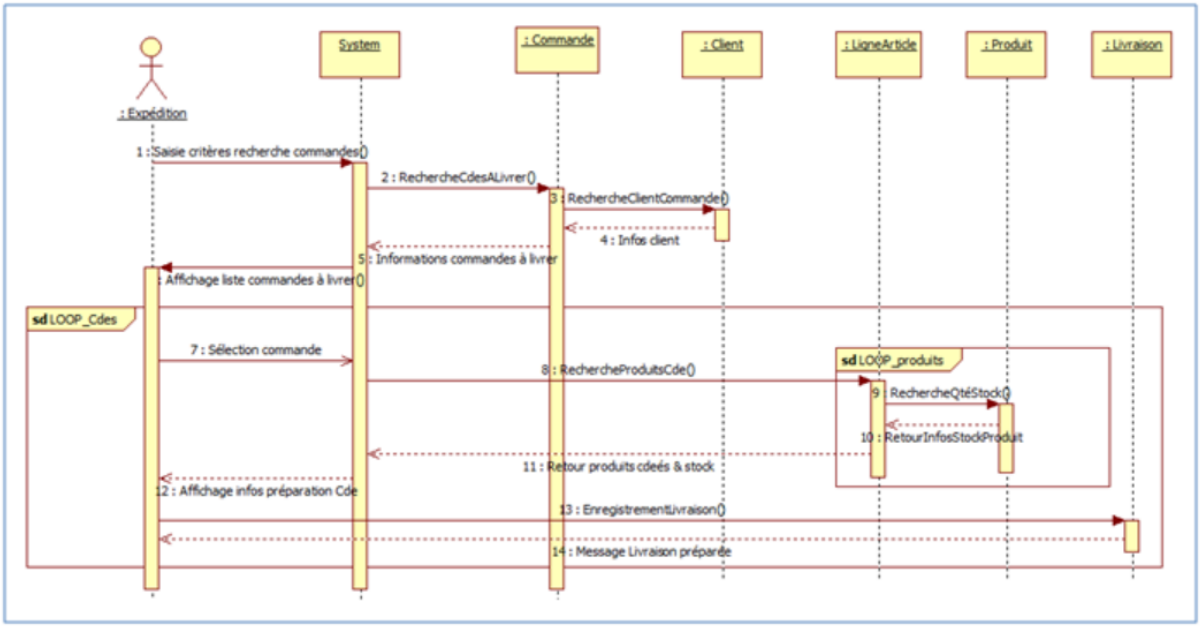
Vue des processus (3)

La vue des processus démontre :

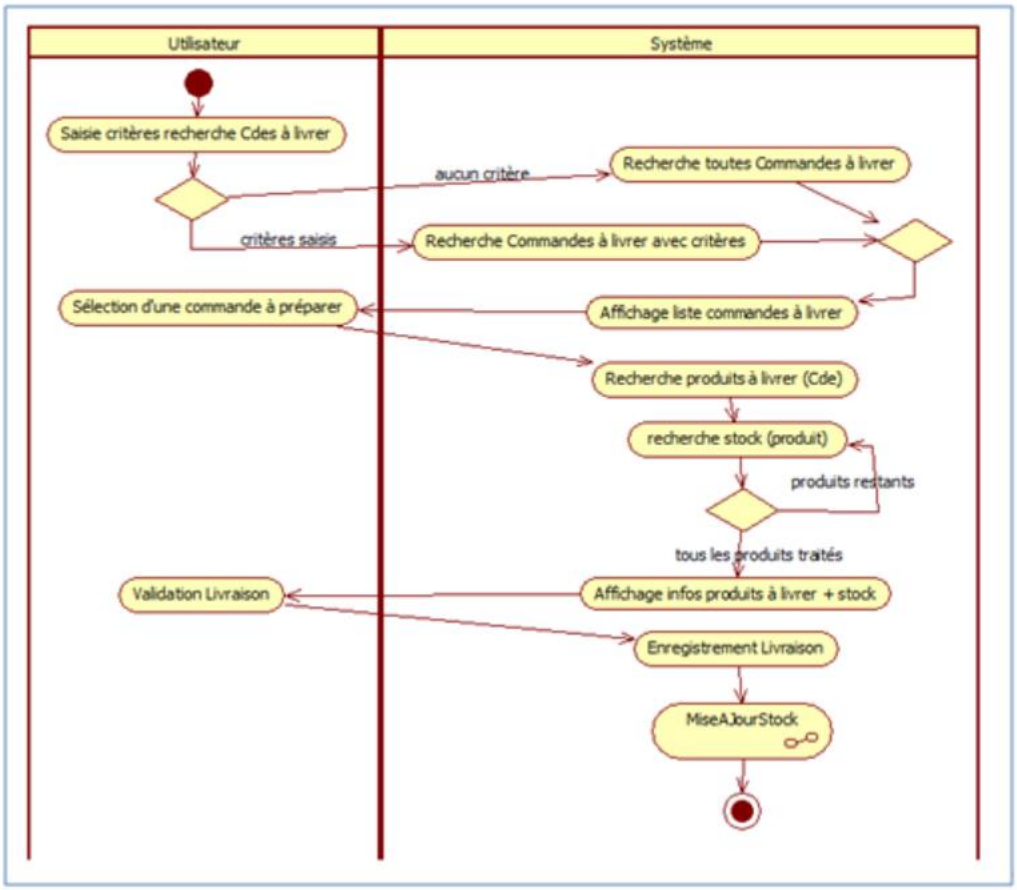
-la décomposition du système en processus et actions ;
-les interactions entre les processus ;
-la synchronisation et la communication des activités parallèles.

La vue des processus s'appuie sur plusieurs diagrammes.

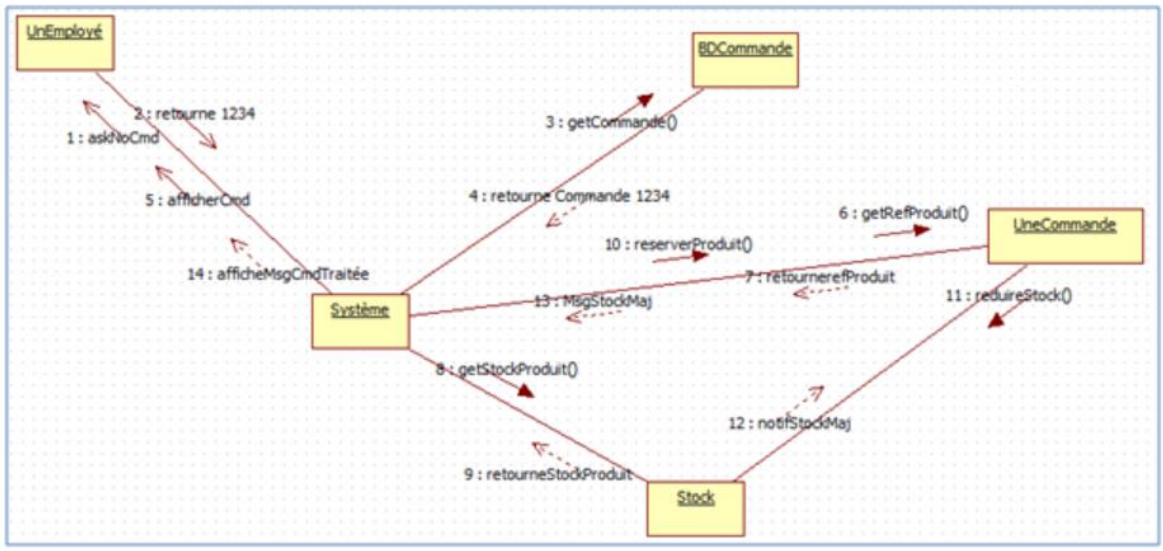
- **Le diagramme de séquence** permet de décrire les différents scénarios d'utilisation du système.



- **Le diagramme d'activité** représente le déroulement des actions, sans utiliser les objets. En phase d'analyse, il est utilisé pour consolider les spécifications d'un cas d'utilisation.

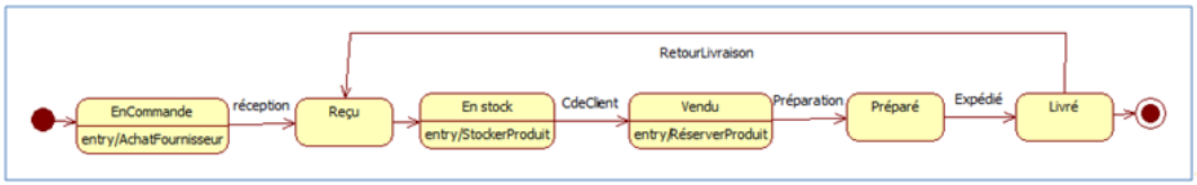


- **Le diagramme de collaboration** (appelé également diagramme de communication) permet de mettre en évidence les échanges de messages entre objets. Cela nous aide à voir clair dans les actions qui sont nécessaires pour produire ces échanges de messages. Et donc de compléter, si besoin, les diagrammes de séquence et de classes.

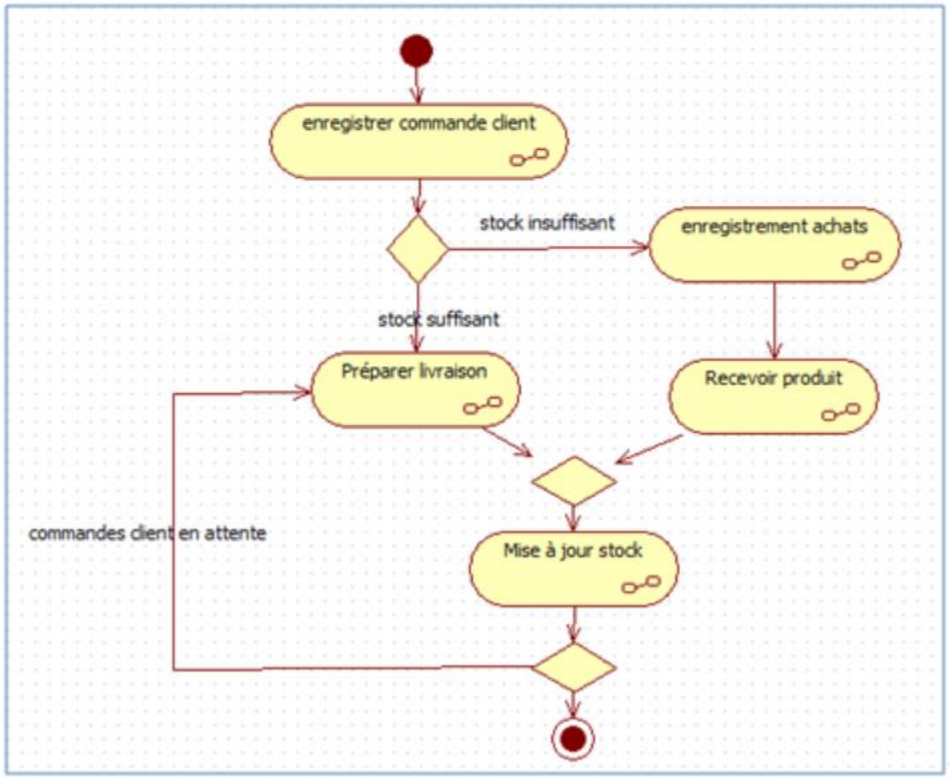


Un exemple de diagramme de collaboration (de communication)

- **Le diagramme d'état-transition** permet de décrire le cycle de vie des objets d'une classe.

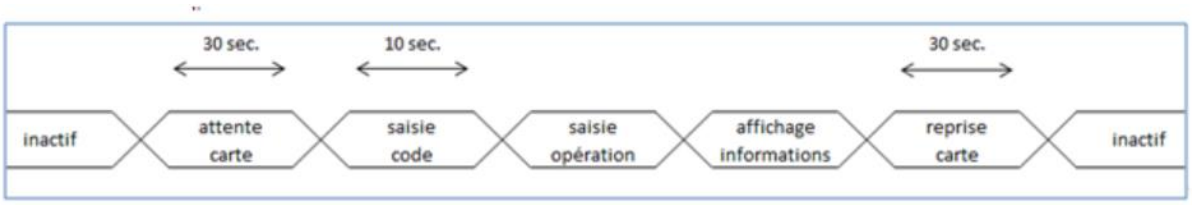


- **Le diagramme global d'interaction** permet de donner une vue d'ensemble des interactions du système. Il est réalisé avec le même graphisme que le diagramme d'activité. Chaque élément du diagramme peut ensuite être détaillé à l'aide d'un diagramme de séquence ou d'un diagramme d'activité. Ce diagramme ne sera pas étudié dans ce cours.



Un exemple de diagramme global d'interaction

- **Le diagramme de temps** est destiné à l'analyse et la conception de systèmes ayant des contraintes temps-réel. Il s'agit là de décrire les interactions entre objets avec des contraintes temporelles fortes. Ce diagramme ne sera pas étudié dans ce cours.



Un exemple de diagramme de temps avec un seul axe temporel

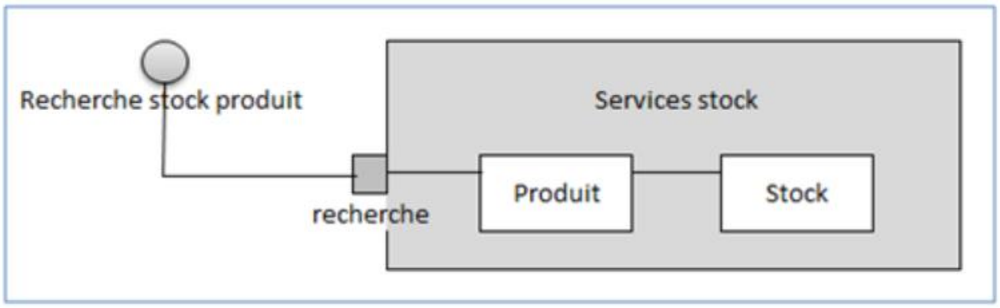
L'aspect lié à l'architecture du logiciel

Pour rappel, cette partie du schéma 4+1 vue permet de définir les composantes à utiliser (exécutables, interfaces, base de données, librairies de fonctions, etc.) et les matériels sur lesquels les composants seront déployés.

Vue des composants (4)

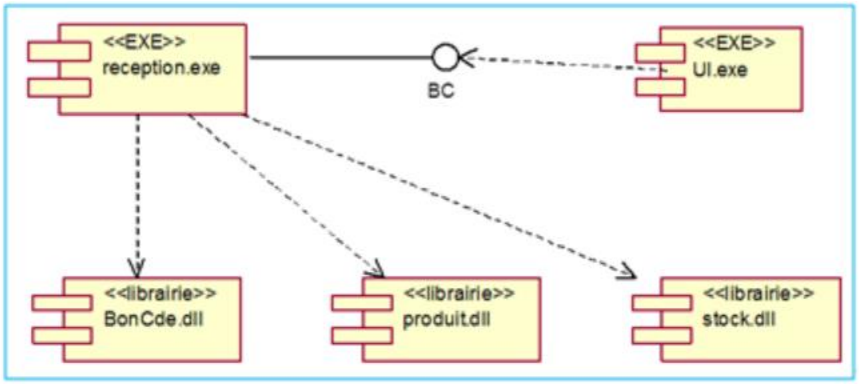
La vue des composants (vue de réalisation) met en évidence les différentes parties qui composeront le futur système (fichiers sources, bibliothèques, bases de données, exécutables, etc.). Cette vue comprend deux diagrammes.

- **Le diagramme de structure composite** décrit un objet complexe lors de son exécution. Ce diagramme ne sera pas étudié dans ce cours



Un exemple de diagramme de structure composite

- **Le diagramme de composants** décrit tous les composants utiles à l'exécution du système (applications, librairies, instances de base de données, exécutables, etc.). Ce diagramme ne sera pas étudié dans ce cours.

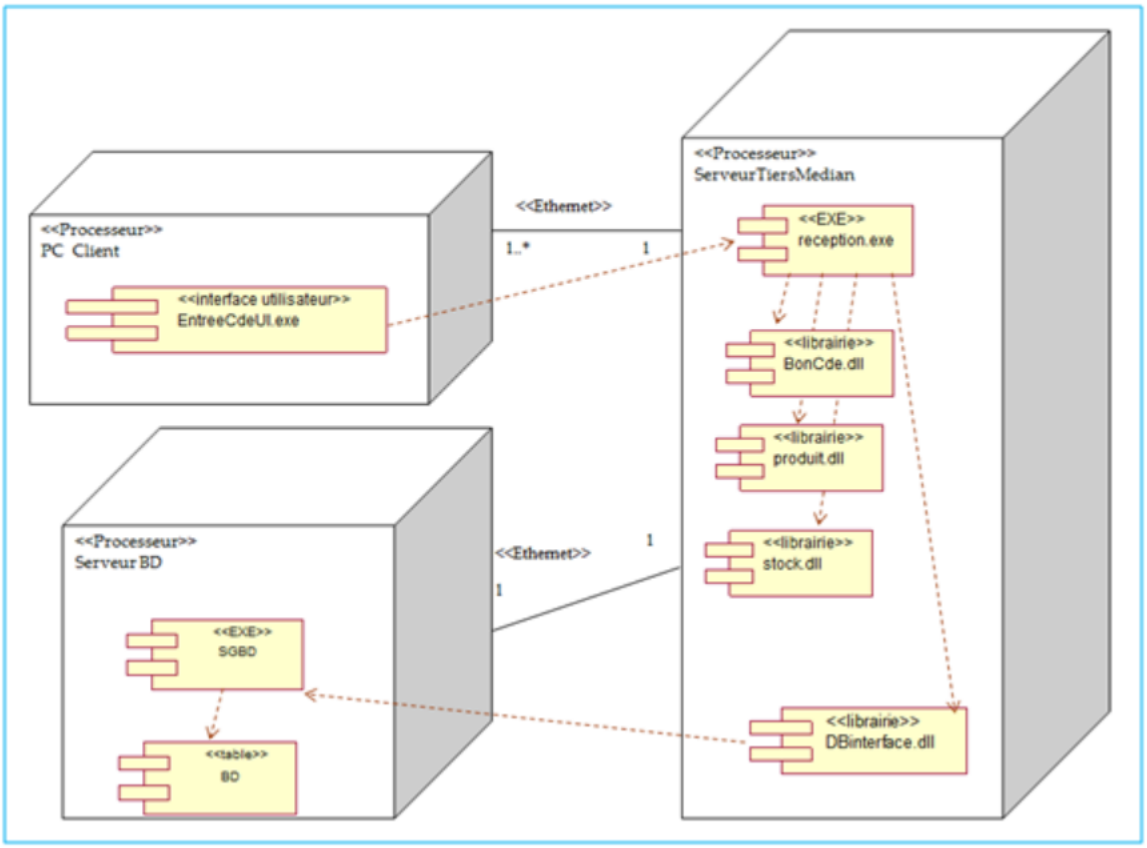


Un exemple de diagramme de composants

Vue de déploiement (5)

La vue de déploiement décrit les ressources matérielles et la répartition des parties du logiciel sur ces éléments. Il contient un diagramme :

- **Le diagramme de déploiement** correspond à la description de l'environnement d'exécution du système (matériel, réseau…) et de la façon dont les composants y sont installés. Ce diagramme ne sera pas étudié dans ce cours.



Un exemple de diagramme de déploiement

En résumé

- UML est constitué de 13 diagrammes qui représentent chacun un concept du système ou logiciel.
- Un logiciel peut être vu en considérant les aspects fonctionnels et les aspects d'architecture du logiciel. Ces deux aspects sont composés de 4 vues du logiciel à développer, organisés autour des besoins des utilisateurs. C'est le 4+1 vues.
- Chacune des 4+1 vues est constituée de diagrammes.