# **Définition : UML**

## **Langage de modélisation**

« UML (en anglais Unified Modeling Language ou « langage de modélisation unifié ») est un langage de modélisation graphique à base de pictogrammes. Il est apparu dans le monde du génie logiciel, dans le cadre de la « conception orientée objet ». Couramment utilisé dans les projets logiciels, il peut être appliqué à toutes sortes de systèmes ne se limitant pas au domaine informatique.

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 ?

UML modélise donc le système logiciel suivant ces deux modes de représentation.

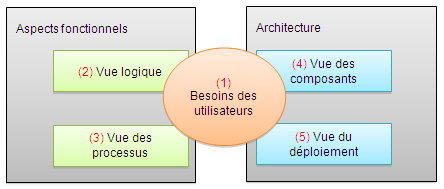
Nous allons donc dans un premier temps décrire ces différents aspects d’un logiciel grâce au schéma 4+1 vues et parcourir brièvement les différents diagrammes qui les composent.

Les 4+1 vues d’un système

Comme je vous le disais en introduction, la conception d’un logiciel est organisée autour des aspects fonctionnels et d’architecture. Ces deux aspects sont représentés par le schéma de 4 vues, axées sur les besoins des utilisateurs (parfois intitulé des cas d’utilisation), appelé **4+1 vues.**

Une première décomposition d’une problématique ou système peut donc être faite à l’aide de 4+1 vues. Le schéma ci-dessous montre les différentes vues permettant de répondre au mieux aux besoins des utilisateurs, organisées selon les deux aspects (fonctionnels et architecture). Chacune des vues est constituée de diagrammes.

Pour rappel, dans ce cours, nous nous concentrerons uniquement sur les besoins des utilisateurs, au centre de la figure.



Voici quelques exemples de diagramme qui peuvent être utilisés dans une démarche d’analyse et de conception d’un logiciel. Ne vous inquiétiez pas du nombre de diagrammes ci-dessous. Il s’agit ici simplement de vous familiariser à ces diagrammes.

Je vous rappelle que dans ce cours, mon objectif est d’avancer pas à pas dans l’analyse des besoins initiaux des utilisateurs, afin de travailler plus particulièrement les diagrammes suivants :

* 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 package ;
* le diagramme de cas d’utilisation ;
* le diagramme d’activité.

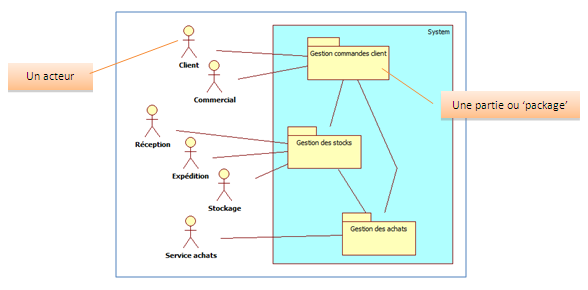
Les autres diagrammes pourront être vus lors d’un autre cours.

#### Les besoins des utilisateurs (1)

Cette partie représente le cœur de l’analyse. Il est composé de cas d’utilisation (que nous verrons plus tard). On y décrit le contexte, les acteurs ou utilisateurs du projet logiciel, les fonctionnalités du logiciel mais aussi les interactions entre ces acteurs et ces fonctionnalités. C’est d’ailleurs aussi le cœur de notre cours.

**Le besoin des utilisateurs peut être décrit à l’aide de deux diagrammes.**

* **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.

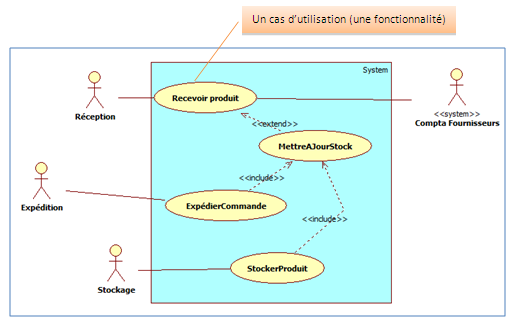


parties (ou packages) observables séparément :

1. La gestion des commandes client
2. La gestion des stocks
3. La gestion des achats

La boîte qui entoure les packages (la boîte bleu) correspond au système (c’est-à-dire le logiciel) qui est analysé.

* **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.



### L’aspect fonctionnel du logiciel

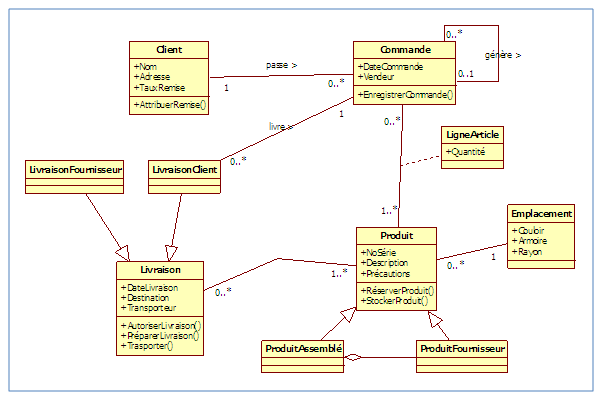
Pour rappel, cette partie du schéma 4+1 vues permet de définir qui utilisera le logiciel et pour quoi faire, comment les fonctionnalités vont se déroulement, etc.

#### 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.

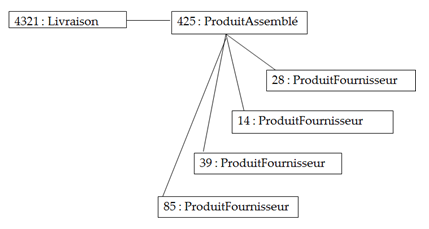
* **Le diagramme de classes**

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.



* **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.



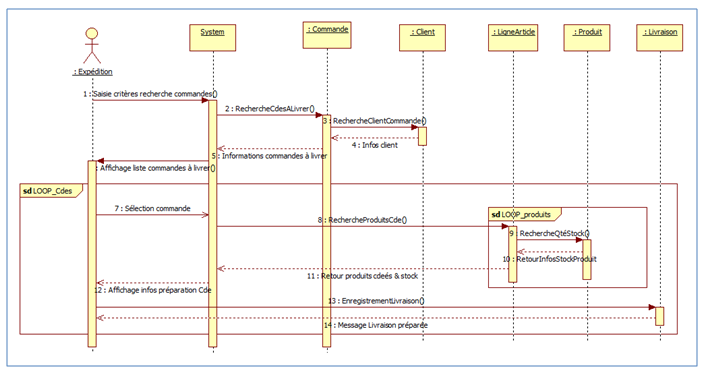
#### Vue des processus (3)

**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.

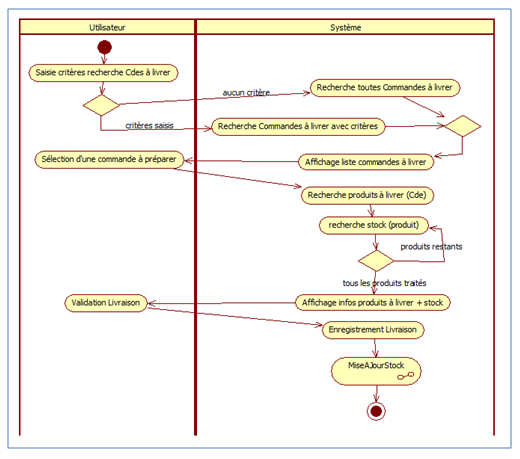
La vue des processus s’appuie sur plusieurs diagrammes.

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

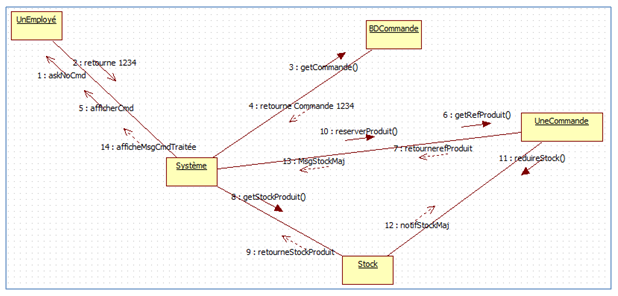


Ce diagramme ne sera pas étudié dans ce cours.

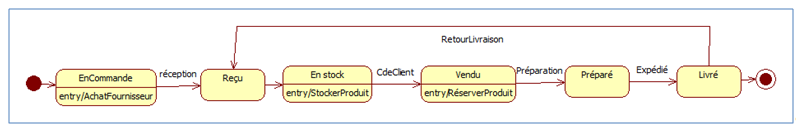
* **Le diagramme d’activité** représente le déroulement des actions, sans utiliser les objets. En phase d’analyse, il est utilisé pour consolider les spécifications d’un cas d’utilisation.

Un exemple de diagramme d’activité

* **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.

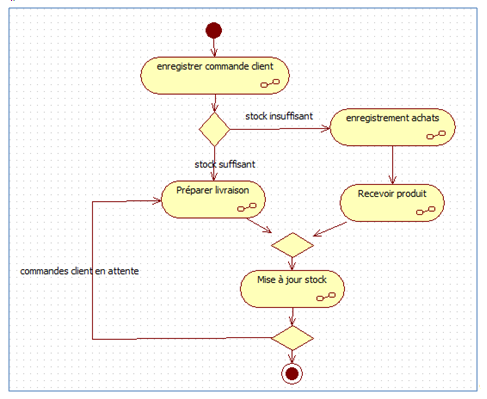
Un exemple de diagramme de collaboration (de communication)

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

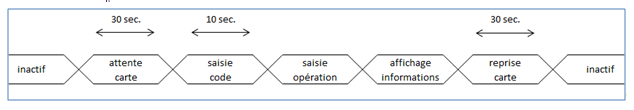
Un exemple de diagramme d’état-transition (objets de la classe produit)

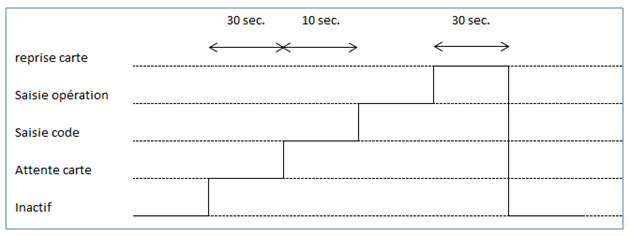
Ce diagramme ne sera pas étudié dans ce cours.

* **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

Un exemple de diagramme de temps avec un axe temporel par état

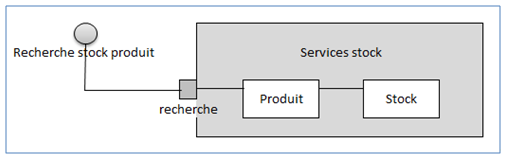
### L’aspect lié à l’architecture du logiciel

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.

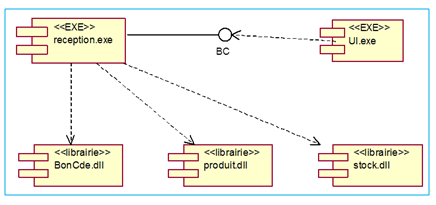
#### Vue des composants (4)

**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

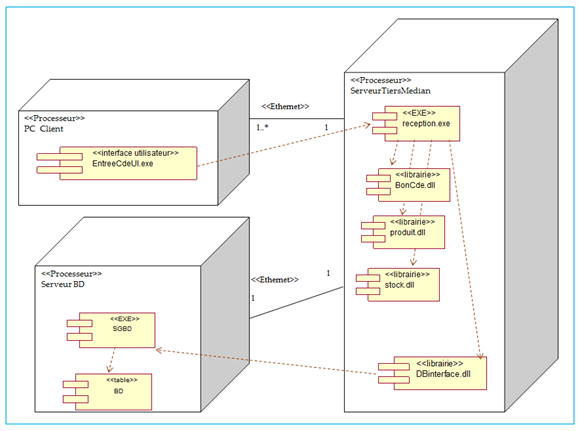
* **Le diagramme de composants** décrit tous les composants utiles à l’exécution du système (applications, librairies, instances de base de données, exécutables, etc.). Ce diagramme ne sera pas étudié dans ce cours.

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.

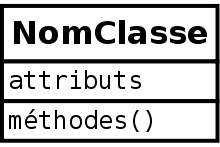
Un exemple de diagramme de déploiement

Voilà, je viens de vous proposer un aperçu des diagrammes utilisés dans la notation UML. Cela semble complexe, et je vous comprends. Mais rassurez-vous, je vous rappelle que nous n’en verrons que quatre dans ce cours, et je vous accompagnerai pas à pas, à partir d’un cas concret pour donner sens à ces diagrammes. Allez, c’est parti !

### 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.

## **Schéma d'une classe**

[](https://commons.wikimedia.org/wiki/File:ModeleDiagrammeDeClasseFR.svg?uselang=fr)

Modèle d'une simple classe.

Une classe est représentée par un rectangle séparée en trois parties :

* la première partie contient le nom de la classe
* la seconde contient les attributs de la classe
* la dernière contient les méthodes de la classe

La seconde et la dernière représentent le **comportement** de la classe.

### Première partie : le nom de la classe[[modifier](https://fr.wikipedia.org/w/index.php?title=Diagramme_de_classes&veaction=edit&vesection=2" \o "Modifier la section : Première partie : le nom de la classe) | [modifier le code](https://fr.wikipedia.org/w/index.php?title=Diagramme_de_classes&action=edit&section=2)]

[](https://commons.wikimedia.org/wiki/File:ModeleClasseAbstraite.svg?uselang=fr)

Modèle d'une classe abstraite.

[](https://commons.wikimedia.org/wiki/File:ModeleClasseTemplate.svg?uselang=fr)

Modèle d'une classe template.

Il est écrit dans le rectangle du haut.

Dans une classe classique, le nom est écrit en [romain](https://fr.wikipedia.org/wiki/Romain_(typographie)).

Le nom des [classes abstraites](https://fr.wikipedia.org/wiki/Classe_abstraite) est écrit en *italique*.

Les classes [template](https://fr.wikipedia.org/w/index.php?title=Classe_template&action=edit&redlink=1" \o "Classe template (page inexistante)) ont, dans leur angle supérieur droit, un rectangle dont la bordure est en pointillé et qui contient les types des paramètres.

### Seconde partie : les attributs

La syntaxe d'un attribut est la suivante :

*Visibilité* *nomAttribut* [*multiplicité*] : *typeAttribut* = *Initialisation ;*

#### Visibilité

La notion de visibilité indique qui peut avoir accès à l'attribut.

Elle ne peut prendre que 4 valeurs possibles :

|  |  |  |  |
| --- | --- | --- | --- |
| **Caractère** | **Rôle** | **Mot clé** | **Description** |
| + | accès public | public | Toutes les autres classes ont accès à cet attribut. |
| # | accès protégé | protected | Seules la classe elle-même et les classes filles (héritage) ont accès à cet attribut. |
| ~ | accès package | package | Classe visible uniquement dans le package. |
| - | accès privé | private | Seule la classe elle-même a accès à cet attribut. |

Afin de respecter le principe fondamental d'[encapsulation](https://fr.wikipedia.org/wiki/Encapsulation_(programmation)), tous les attributs devraient être *privés*.

Pour qu'un attribut privé ou protégé soit récupérable, on utilise en général un *getter* (ou *accesseur*); pour qu'il soit modifiable, on utilise en général un *setter* (ou*mutateur*).

#### Nom de l'attribut

Il ne doit pas comporter d'espaces, de signes de ponctuation ou d'accents. Pour remplacer les espaces, plusieurs conventions existent : on peut utiliser le symbole \_ ou changer la première lettre du mot en majuscule (exemple : *nom de l'objet* peut s'écrire *nomObjet* ou *nom\_objet*).

#### Multiplicité

La multiplicité représente le nombre de fois que la variable peut exister. Elle est représentée entre crochets.

Par exemple : si une personne possède 2 numéros de téléphone, on préfèrera nos Telephones[2] à noTelephone1 et noTelephone2.

#### Exemple d'attributs

*//attribut public prenom de type string (chaine de caractères)*

+ prenom : string

*//attribut privé age de type int (entier)*

- age : int