I. Introduction

Les méthodes d'analyse orientées objet sont initialement issues des milieux industriels. La préoccupation dominante de leurs auteurs est le génie logiciel, c'est-à-dire les principes et les techniques permettant d'augmenter la rigueur et la qualité quand on construit une application informatique. Initialement, UML(Unified Modeling Language) est le résultat de la fusion de trois méthodes orientées objet.

La méthode OOD, object oriented design, de **grady booch** a été conçu à la demande du ministère de la défense des États-Unis. L'objectif était de préparer de façon rigoureuse la structuration des programmes écrits en langage ADA ou C++.

La méthode OMT object modeling technique de **James Rumbaugh** en 1996 a été mise au point à général électrique. Ses auteurs ont été largement influencés par les applications d'informations industrielles (automates ,contrôle de processus ..). En plus des principes des langages à objet, ils ont été inspirés par les techniques de modélisation conceptuelle des méthodes d'analyse des années 1980.

La méthode OOSE, object oriented software engineering d'**IVAR Jacobson** en 1993 est d'origine universitaire (informatique temps réel) et industrielle. Comme OMT, la méthode a emprunté aux méthodes d'analyse antérieures la technique de modélisation conceptuelle, enrichie des aspects comportementaux. Son originalité consiste à faire reposer l'analyse sur l'expression par l'utilisateur de la façon dont il pense utiliser le futur système.

Événement considérable et presque miraculeux, les trois "gourous" qui régnaient chacun sur l'une des trois méthodes se mirent d'accord pour définir une méthode commune qui fédérerait leurs apports respectifs (on les surnomme depuis "the Amigos"). C'est de cet effort de convergence qu'est né UML, "Unified Modeling Language", l'adjectif "unified" étant là pour bien marquer qu'UML "unifie" et donc remplace les méthodes antérieures.

Dans ce chapitre nous allons détailler le langage UML ainsi qu'un processus de développement.

II. Le langage UML

II.1. Définition d'UML

C'est un langage de modélisation graphique à base de pictogrammes, conçu pour représenter, spécifier les artefacts de systèmes logiciels, de plus il est destiné à comprendre et décrire des besoins spécifiés et documentés des systèmes, esquissé des architectures logicielles, concevoir des solutions et communiquer des points de vue, comme il peut être appliqué à toutes sortes de systèmes ne se limitant pas au domaine informatique.

UML résulte de l'unification de techniques ayant fait leurs preuves pour l'analyse et la conception de grands logiciels et de systèmes complexes. [1][2]

- ***UML est une norme***

Il est nécessaire qu'une méthode objet soit définie de manière rigoureuse et unique afin de lever les ambiguïtés. De nombreuses méthodes objet ont été définies, mais aucune n'a su s'imposer en raison du manque de standardisation. C'est pourquoi l'ensemble des acteurs du monde informatique a fondé en 1989 l'**OMG** (**O**bject **M**anagement **G**roup), une organisation à but non lucratif, dont le but est de mettre au point des standards garantissant la compatibilité entre des applications programmées à l'aide de langages objet et fonctionnant sur des réseaux hétérogènes (de différents types).

A partir de 1997, UML est devenue une norme de l'OMG, ce qui lui a permis de s'imposer en tant que méthode de développement objet et être reconnue et utilisée par de nombreuses entreprises.

- ***UML est un langage de modélisation objet***

UML comble une lacune importante des technologies objet, il permet d'exprimer, d'élaborer et de modéliser au sens de la théorie des langages, de ce fait il contient les éléments constitutifs de ce derniers : concepts, une syntaxe et une sémantique.

- ***UML décrit un méta modèle***

La puissance et l'intérêt d'UML est qu'il normalise la sémantique des concepts qu'il véhicule, il repose sur un méta modèle pour permettre à n'importe qui de déchiffrer son intention de manière non équivoque, il est donc primordiale de s'accorder sur la

sémantique des éléments de modélisation, bien avant de s'intéresser à la manière de les présenter.

II.2. Points fort d'UML

Il permet ainsi :

- un gain de précision.
- un gage de stabilité.
- l'utilisation d'outils.
- Il cadre l'analyse et facilite la compréhension de représentations abstraites complexes. Son caractère polyvalent et sa souplesse en font un langage universel. [3]

II.3. Points faibles d'UML

- La mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation.

- L'intégration d'UML dans un processus n'est pas triviale, et améliorer un processus est une tâche complexe et longue. [3]

II.4. Les diagrammes

II.4.1. Définition d'un diagramme

Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du Modèle.

Chaque type de diagramme UML possède une structure et véhicule une sémantique précise.

II.4.2. Les Différents types de diagrammes

- ***Diagrammes structurels ou diagrammes statiques***

- ***Diagramme de classe :*** class diagram

Les diagrammes de classes expriment de manière générale la structure statique d'un système, en termes de classes et de relations entre ses classes. Outre les classes, ils représentent un ensemble d'interfaces et de paquetages, ainsi que leurs relations.

Les diagrammes de classes contiennent généralement les éléments suivant :

- ✓ **Les classes :** Une classe est la description d'un ensemble d'objet qui partage les mêmes attributs, les mêmes opérations, les mêmes relations et la même sémantique. Une classe est symbolisée par un rectangle.

✓ **Attribut :** Un attribut est une propriété nommée d'une classe qui décrit un ensemble de valeurs que les instances de cette propriété peuvent prendre. Une classe peut ne pas avoir, comme elle peut avoir un ou plusieurs attributs.

✓ **Opération :** Une opération est une abstraction de ce que peut réaliser un objet et qui est réalisable par tous les objets de la classe. Une classe peut ne pas avoir comme elle peut avoir plusieurs opérations.

✓ **Les relations d'association d'agrégation et de composition**

Une association : représente une relation sémantique durable entre deux classes.

Une agrégation est un particulier d'association non symétrique exprimant une relation de contenance.

Une composition est une agrégation plus forte.

➢ ***Diagramme d'objets :*** object diagram

Les diagrammes d'objets servent, d'une part à inventorier les objets (i.e les instances de classe) composant une application à un instant donné ainsi que les relations, d'autre part à donner une image statique des relations entre ces objets. Ils peuvent également être mis en œuvre pour tester la pertinence d'un diagramme de classe.

➢ ***Diagramme de composant :*** component diagram

Les diagrammes de composants servent à représenter la configuration logicielle ainsi que les relations d'un système, on permettant également de représenter les programmes, les sous programmes et les interrelations.

➢ ***Diagramme de déploiement :*** deployment diagram

Les diagrammes de déploiement représentent un ensemble de nœuds ainsi que leurs relations. On les utilise pour illustrer la vue de déploiement statique d'une architecture. Les diagrammes de déploiement sont apparentés aux diagrammes de composant car un nœud englobe généralement un ou plusieurs composants.

➢ ***Diagramme de cas d'utilisation :*** use case diagram

Les diagrammes de cas d'utilisation représentent un ensemble de cas d'utilisation, d'acteurs et leurs relations. Ils représentent la vue statique des cas d'utilisation d'un système et sont particulièrement importants dans l'organisation et la modélisation des comportements d'un système.

✓ **Les cas d'utilisation :** Les cas d'utilisation décrivent, sous la forme d'actions et de réactions, le comportement, ou tout simplement ce que fait le système du point de vue de l'utilisateur, encore appelé acteur. On recense, de la sorte, l'ensemble des fonctionnalités d'un système en examinant les besoins fonctionnels de chaque acteur.

✓ **Les acteurs :** Un acteur représente un ensemble cohérent de rôles joués par les utilisateurs des cas d'utilisation en interaction avec ces cas d'utilisation. En règle générale, un acteur représente un rôle qu'un homme, une machine ou même un autre système joue avec le système. Il existe 4 grandes catégories d'acteurs :

- les acteurs principaux. Cette catégorie regroupe les personnes qui utilisent les fonctions principales du système.
- les acteurs secondaires. Cette catégorie regroupe les personnes qui effectuent des tâches administratives ou de maintenance.
- le matériel externe. Cette catégorie regroupe les dispositifs matériels autres que les ordinateurs comme les périphériques.
- les autres systèmes. Cette catégorie regroupe les systèmes avec lesquels le système interagit.

✓ **Les relations entre les cas d'utilisations:** UML définit trois types de relations standardisées entre cas d'utilisation, détaillées ci après :

- **La relation d'inclusion** : formalisée par le mot clé « include », le cas d'utilisation de base en incorpore explicitement un autre de façon obligatoire.
- **La relation d'extension :** formalisée par le mot clé « extend », le cas d'utilisation de base en incorpore explicitement un autre, de façon optionnelle.
- **La relation de généralisation ou spécialisation :** Les cas d'utilisation descendant hérite de la description de leur parent commun. Chacun entre eux peut néanmoins comprendre des interactions spécifiques supplémentaires. [2]

- ***Diagrammes comportementaux ou diagrammes dynamiques***

➢ ***Diagramme d'activité :*** activity diagram

Le diagramme d'activité est attaché à une catégorie de classe et décrit le déroulement des activités de cette catégorie. Le déroulement s'appelle "flot de contrôle". Il indique la part prise par chaque objet dans l'exécution d'un travail. Il sera enrichi par les conditions de Séquencement.

➢ ***Diagramme d'états-transitions :*** State machine diagram

Ils ont pour rôle de représenter les traitements (opérations) qui vont gérer le domaine étudié. Ils définissent l'enchaînement des états de classe et font donc apparaître

l'ordonnancement des travaux. Le diagramme d'états-transition est associé à une classe pour laquelle on gère différents états : il permet de représenter tous les états possibles ainsi que les événements qui provoquent les changements d'état.

- ***Diagramme de séquence :*** Séquence diagram

Un diagramme de séquence met en évidence le classement des messages par ordre chronologique. On forme un diagramme de séquence en plaçant d'abord les objets qui participent à l'interaction en haut du diagramme. Le long de l'axe des abscisses. En générale. On place l'objet qui débute l'interaction à gauche, puis on continue en progressant vers la droite, les objets les plus subordonnés étant tout à fait à droite. On place ensuite les messages envoyés et reçus par ces objets le long de l'axe des ordonnées, par ordre chronologique, du haut vers le bas. Cela donne au lecteur une indication visuelle claire du flot de contrôle dans le temps.

En générale, les diagrammes de séquence contiennent :

- ✓ **l'objet :** est une manifestation concrète d'une abstraction à laquelle on peut appliquer un ensemble d'opérations et qui possède un état capable de mémoriser les effets de ces opérations. On représente un objet en soulignant son nom.
- ✓ **Le lien:** est une liaison sémantique entre objets, en générale, il s'agit d'une instance d'une association. Chaque fois qu'une classe est reliée à une autre par une association, il peut y avoir un lien entre les instances des deux classes, et chaque fois qu'un lien existe entre deux objets, le premier objet peut envoyer un message au deuxième.
- ✓ **Le message :** est la spécification d'une communication entre objets, qui transporte des informations et qui s'affiche dans le but de déclencher une activité. La réception d'une instance de message peut être considérée comme une instance d'un événement.

- ***Diagramme de collaboration :*** collaboration diagram

Les diagrammes de collaboration (tout comme les diagrammes de séquence) sont des cas particuliers de diagrammes d'interactions qui représentent une vue dynamique du système. Les diagrammes de collaboration présentent un ensemble de rôles joués par des objets dans un contexte particulier, ainsi que les liens entre ces objets. [4][5]

III. Le choix de la méthode

Il existe plusieurs méthodes de développement logiciel construites sur UML comme la méthode : UP, RUP, TTUP, UP agile, XP, 2TUP …..

Parmi ses méthodes notre choix est basé sur la méthode UP (Unified Process).

III.1. Le processus unifié

III.1.1. Définition du processus unifié

Le processus unifié est un processus de développement logiciel itératif, centré sur l'architecture, Piloté par des cas d'utilisation et orienté vers la diminution des risques .C'est un patron de processus pouvant être adaptée à une large classe de systèmes logiciels, à différents domaines d'application, à différents types d'entreprises, à différents niveaux de compétences et à différentes tailles de l'entreprise. [2]

III.1.2. Les caractéristiques du processus unifié

- ***UP est itératif et incrémental***

Le projet est découpé en itérations ou étapes de courte durée qui permettent de mieux suivre l'avancement globale. A la fin de chaque itération une partie exécutable du système finale est produite, de façon incrémentale (par ajout).

La figure I.1 illustre l'itération d'UP.

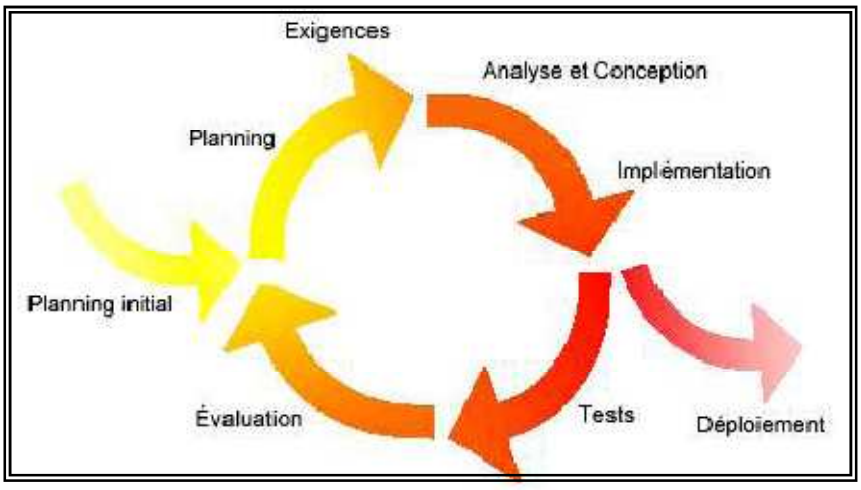

Figure I.1 : Déroulement Du Processus Unifié

❖ ***UP est centré sur l'architecture***

Tout système complexe doit être décomposé en partie modulaire afin d'en faciliter la maintenance et l'évolution. Cette architecture (fonctionnelle, logique, matérielle, etc.) doit être modéliser en UML, et pas seulement documentée en texte.

❖ ***UP est guidé par les cas d'utilisation d'UML***

Le but principal d'un système d'informatique est de satisfaire les besoin de client. Le processus de développement sera donc accès sur l'utilisateur. Le cas d'utilisation permettent d'illustrer ces besoins. Ils détectent puis décrivent les besoin fonctionnel et leur ensemble constitue le modèle de cas d'utilisation qui dicte les fonctionnalités complètes du système. [2]

❖ ***UP est piloté par les risques***

Les risques majeurs du projet doivent être identifiés au plus tôt mais surtout levés le plus rapidement. Les mesures à prendre dans ce cadre déterminent l'ordre des itérations

III.1.3. Cycle de vie du processus unifié

L'objectif d'un processus unifié est de maîtriser la complexité des projets informatiques en diminuant les risques. UP est un ensemble de principes génériques adapté en fonctions des spécificités des projets.

✓ **L'architecture bidirectionnelle:** UP gère le processus de développement par deux axes. (figure I.2).

✓ **L'axe vertical :** représente les principaux enchaînements d'activités, qui regroupent les activités selon leur nature. Cette dimension rend compte l'aspect statique du processus qui s'exprime en termes de composants, de processus, d'activités, d'enchaînements, d'artefacts et de travailleurs.

✓ **L'axe horizontal :** représente le temps et montre le déroulement du cycle de vie du processus; cette dimension rend compte de l'aspect dynamique du processus qui s'exprime en terme de cycles, de phases, d'itérations et de jalons.

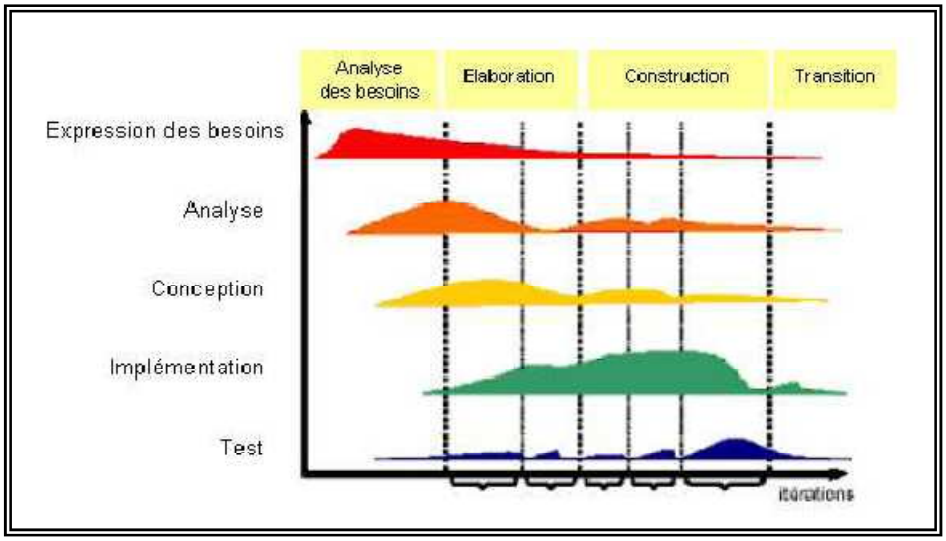

Figure I.2 : L'architecture bidirectionnelle

Pour mener efficacement un tel cycle, les développeurs ont besoins de toutes les représentations du produit logiciel.

- un modèle de cas d'utilisation.

- un modèle d'analyse : détailler les cas d'utilisation.

- un modèle de conception : finissant la structure statique du système sous forme de sous systèmes, de classes et interfaces.

- un modèle d'implémentation : intégrant les composants

- un modèle de déploiement : définissant les nœuds physiques des ordinateurs

- un modèle de test : décrivant les cas de test vérifiant les cas d'utilisation

- une représentation de l'architecture.

III.1.4. Les activités

- ***Expression des besoins***

L'expression des besoins comme son nom l'indique, permet de définir les différents besoins :

- inventorier les besoins principaux et fournir une liste de leurs fonctions ;

- recenser les besoins fonctionnels (du point de vue de l'utilisateur) qui conduisent à l'élaboration des modèles de cas d'utilisation ;

- appréhender les besoins non fonctionnels (techniques) et livrer une liste des exigences.

Le modèle de cas d'utilisation présente le système du point de vue de l'utilisateur et représente sous forme de cas d'utilisation et d'acteur, les besoins du client

- ***Analyse***

L'objectif de l'analyse est d'accéder à une compréhension des besoins et des exigences du client. Il s'agit de livrer des spécifications pour permettre de choisir la conception de la solution.

Un modèle d'analyse livre une spécification complète des besoins issus des cas d'utilisation et les structure sous une forme qui facilite la compréhension (scénarios), la préparation (définition de l'architecture), la modification et la maintenance du futur système. Il s'écrit dans le langage des développeurs et peut être considéré comme une première ébauche du modèle de conception.

- ***Conception***

La conception permet d'acquérir une compréhension approfondie des contraintes liées au langage de programmation, à l'utilisation des composants et au système d'exploitation. Elle détermine les principales interfaces et les transcrit à l'aide d'une notation commune.

Elle constitue un point de départ à l'implémentation :

- elle décompose le travail d'implémentation en sous-système
- elle créée une abstraction transparente de l'implémentation

- ***Implémentation***

L'implémentation est le résultat de la conception pour implémenter le système sous formes de composants, c'est-à-dire, de code source, de scripts, de binaires, d'exécutable et d'autres éléments du même type.

Les objectifs principaux de l'implémentation sont de planifier les intégrations des composants pour chaque itération, et de produire les classes et les sous-systèmes sous formes de codes sources.

- ***Test***

Les tests permettent de vérifier des résultats de l'implémentation en testant la construction. Pour mener à bien ces tests, il faut les planifier pour chaque itération, les implémenter en créant des cas de tests, effectuer ces tests et prendre en compte le résultat de chacun.

III.1.5. Les phases

- ***Analyse des besoins***

L'analyse des besoins donne une vue du projet sous forme de produit fini. Cette phase porte essentiellement sur les besoins principaux (du point de vue de l'utilisateur), l'architecture générale du système, les risques majeurs, les délais et les coûts.

- ***Elaboration***

L'élaboration reprend les éléments de la phase d'analyse des besoins et les précise pour arriver à une spécification détaillée de la solution à mettre en œuvre.

L'élaboration permet de préciser la plupart des cas d'utilisation, de concevoir l'architecture du système et surtout de déterminer l'architecture de référence.

- ***Construction***

La construction est le moment où l'on construit le produit. L'architecture de référence se métamorphose en produit complet. Le produit contient tous les cas d'utilisation que les chefs de projet, en accord avec les utilisateurs ont décidé de mettre au point pour cette version.

- ***Transition***

Le produit est en version bêta. Un groupe d'utilisateurs essaye le produit et détecte les anomalies et défauts. Cette phase suppose des activités comme la formation des utilisateurs clients, la mise en œuvre d'un service d'assistance et la correction des anomalies constatées.

Tout simplement la phase de transition permet de faire passer le système informatique des mains des développeurs à celles des utilisateurs finaux. [4]

IV. Conclusion

Le langage UML nous apporte une aide à toutes les étapes d'un projet, comme il nous offre ainsi de nombreux avantages pour l'analyse et la conception d'un système, Le couple UML et le processus unifié propose une approche pour conduire la réalisation de systèmes orienté objet.

Le chapitre suivant définit la conception de notre système.