Sommaire

Chapitre II :

Chapitre III

Liste des figures

Aujourd'hui les systèmes informatique simples ou complexes jouent un rôle important dans plusieurs domaines d'applications (par exemple la médecine, l'architecture, les mathématiques, etc ...). La modélisation des systèmes est une phase laborieuse pour la conception et la validation des systèmes. Dans notre mémoire on s'intéresse à la modélisation. Dans ce contexte, plusieurs langages de modélisation existent dans la littérature ; par exemple UML (Uni-er modeling language), MERISE. Dans notre travail, nous avons appliqué la modélisation en utilisant les diagrammes UML Ces diagrammes ont montré leurs importances et efficacités depuis leur apparition en 1990.

notre problématique repose sur complémentation de conception appelleront avec une conception multicouche.

cette idée a démontré leur nécessité sur les projets de développements énorme ou nous a avons besoin de réduire les information , dans notre objective et de garder touts les information sans perdre les détailles avec une organisation de travaille au cours des phases de développement de génie logiciel le surcharge des information influent sur la compréhension du projet . pour avoir une vue global de projet nous avons besoin d'afficher le minimum des information pour entrée dans les détaille et gardé la vue globale nous avons introduit la multicouches de conception et cette fonctions lorsque on zoom on peut voir plus de détail

nous avons au départ, commencer par le diagramme de classe uml ,comme une conception graphique pour avoir une vue complète entre les classes et leur relations (association)

si on veux voir le code source on fait un zoom avec le mouse wheel dans la zone d'affichage (paper).

Chapitre I

Généralité

Les applications WEB

1.1 Introduction

Le développement des applications WEB présente certaines particularités, au niveau technique et ergonomique. Cette spécificité nous oblige, au moment de la conception, à préconiser des méthodes de conception et des méthodes de travail dédiées à ce genre d'applications.

Ce chapitre met en claire cette particularité des applications WEB en mettant l'accent sur leurs caractéristiques.

1.2 Définition d'une application WEB

Une application Web est un ensemble de pages qui interagissent avec les utilisateurs, les unes avec les autres, ainsi qu'avec les différentes ressources d'un serveur Web, notamment les bases de données.

Une application Web est un site Web qui contient des pages et dont le contenu est partiellement ou totalement indéterminé. Le contenu final d'une page est déterminé uniquement lorsque l'utilisateur requiert une page depuis le serveur Web. Le contenu final d'une page variant d'une requête à une autre en fonction des actions de l'utilisateur, ce type de page est appelé page dynamique. Les applications Web sont construites de manière à répondre à différents types de défis et de problèmes.

1.3 Sites ou applications web?

Les applications web reposent sur des technologies sous-jacentes qui rendent leur contenu dynamique et qui permettent à l'utilisateur de modifier l'état applicatif sur le serveur. La distinction entre sites web et applications web est subtile, puisqu'elle réside dans le dernier cas en la possibilité d'affecter l'état applicatif sur le serveur via un navigateur. Sans cela, il est inapproprié de parler d'application web. Seuls les systèmes qui offrent cette possibilité, et les serveurs d'application qui utilisent un serveur web pour l'interaction utilisateur, peuvent être considérés comme des applications web. Pour toutes les applications web, aussi simples soient-elles, l'utilisateur doit transmettre plus que de simples requêtes de navigation ; en général, il peut saisir du texte simple, sélectionner des boutons d'options, voire donner des informations binaires ou de fichier.

La distinction devient encore plus subtile dans le cas des moteurs de recherche, pour lesquels des critères de recherche relativement sophistiqués sont saisis. Les sites web que sont les moteurs de recherche ne font qu'accepter ces critères, qu'ils emploient dans une espèce de requête sur une base de données afin de renvoyer des résultats. Quand l'utilisateur en a terminé avec le système, il n'y a pas de différence notable dans l'état du moteur de recherche, excepté, bien sûr, dans les fichiers journaux et les compteurs d'accès. Il en va tout autrement pour les applications web qui, par exemple, acceptent en ligne des informations d'inscription. Ainsi l'état d'un site web dans lequel un utilisateur s'inscrit à une formation sera-t-il modifié quand ce dernier en aura fini avec l'application.

L'architecture d'un site web est tout ce qu'il y a de simple. Elle contient trois composants principaux : un serveur web, une connexion réseau et des navigateurs clients. Les applications web contiennent, en outre, un serveur d'application qui rend possible le traitement d'une logique et d'un état applicatif.

1.4 Utilisations courantes des applications Web

Les applications Web peuvent être utilisées de diverses façons par les visiteurs d’un site et les développeurs, notamment pour :

Permettre aux utilisateurs de trouver rapidement et facilement des informations sur un site Web riche en contenu. Ce type d’applications Web permet aux visiteurs du site de rechercher, d’organiser et de parcourir le contenu à leur convenance.

Collecter, enregistrer et analyser des données fournies par les visiteurs du site. Auparavant, les données saisies dans des formulaires HTML étaient envoyées sous forme de courriel aux employés ou sous forme d’applications CGI pour le traitement. Une application Web peut enregistrer les données d’un formulaire directement dans une base de données et créer des rapports Web pour l’analyse. Les exemples incluent des pages de banques en ligne, de contrôle des stocks, des sondages et des formulaires de commentaires.

Mettre à jour des sites Web dont le contenu change souvent. Une application Web évite au créateur d’avoir à mettre fréquemment à jour le code HTML du site. Les fournisseurs de contenu tels que les rédacteurs en chef alimentent l’application Web et celle-ci met automatiquement le site à jour.

1.5 Le processus de développement

On ne développe pas une application comme on suit une recette. Les compétences et les aptitudes des acteurs impliqués sont pleinement sollicitées. Cela ne diminue en rien l'importance de la définition du projet. Mais si les efforts consentis par une équipe de développement peuvent suffire pour mener un projet à terme, ils seront mieux encore rétribués si dès l'abord le processus a été convenablement défini.

Un processus de développement assure quatre fonctions [9] :

- ✓ Déterminer l'ordre des activités d'une équipe,
- ✓ Spécifier les artefacts à développer,
- ✓ Guider la tâche des développeurs et de l'équipe dans son ensemble,
- ✓ Offrir des critères pour le contrôle et l'évaluation des produits et des activités

du projet.

Un processus de développement logiciel peut prendre la forme d'un ensemble de documents ou de fichiers hypertexte. Le processus définit les chaînes de processus métier (workflows), les activités, les artefacts et les rôles des acteurs impliqués dans le processus de développement. Un acteur, dans ce sens, est un rôle joué par un individu au cours du processus. Dans beaucoup d'équipes, des individus assurent les rôles de plusieurs acteurs.

Un workflow est un ensemble d'activités qui produit à terme un résultat concret et observable. Ainsi, la détermination des besoins, la modélisation par cas d'utilisation, l'analyse, la conception. Un workflow définit un ensemble d'activités dans lesquelles des acteurs s'investissent. Les activités représentent que font les acteurs pour produire les artefacts de sortie du workflow. Un artefact désigne tout élément d'information qui est produit par les acteurs du processus. Les artefacts peuvent être du code, graphique, schémas de base de données, documents texte, diagrammes, modèles, etc.… Ils ont une propriété importante : ils peuvent être soumis à un contrôle de versions car ils évoluent souvent tout au long du processus ; un historique précis de ces suites de modifications est en effet critique.

Le processus décrit ici est basé sur le processus unifié de Rational (RUP) pour Rational Unified Process [15] et le ICONIX Unified Process [16]. Tous deux découlent de processus et de méthodologies orientés objet fondés sur les travaux de Grady Booch, Ivar Jacobson et Jim

Rumbaugh [14]. Ces deux processus ne sont pas les seuls processus formels utilisés aujourd'hui, il existe bien d'autres processus.

Comme d'autres, ces deux processus sont itératifs .Cela veut dire que chaque phase du processus, analyse des besoins, analyse et conception, implémentation, test et évaluation, est répétée et raffinée jusqu'à ce que le processus réponde aux besoins et qu'il puisse être déployé. C'est une orientation toute différente du processus traditionnel de cycle de vie en cascade, dans lequel chaque phase du projet était achevée avant que l'équipe ne passe à la suivante. Les processus itératifs proviennent de cette constatation que, bien souvent, le développement de logiciels ne se déroule pas ainsi.

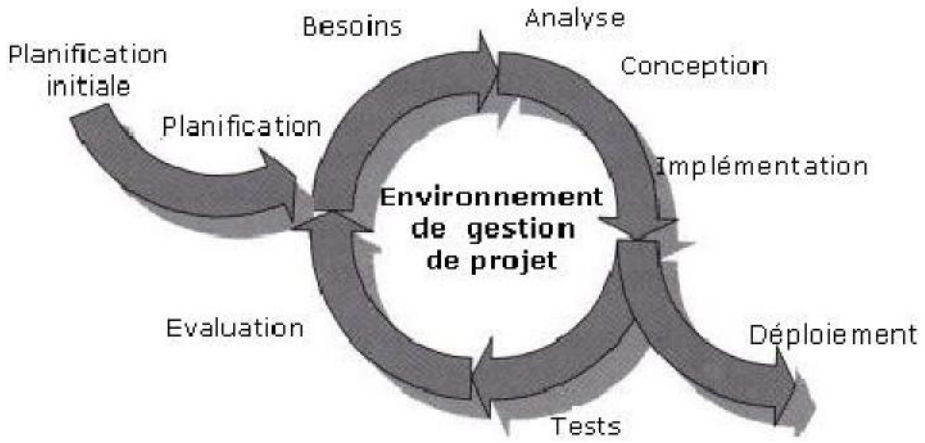

Figure 1.1 Un processus itératif [14]

Alors que les processus décrits dans les manuels sont abstraits, la réalisation même d'un processus de développement est un processus en soi. Or, on ne saurait imposer une méthodologie à une équipe tout en espérant qu'elle se conforme à une organisation, sans qu'un effort d'adaptation soit entrepris. Jim Conallen [17] défini un certain nombre de particularités qui peuvent influencer la qualité d'un processus de développement logiciel :

1. Constitution de l'entreprise et organisation : Les grandes entreprises avec en leur sein des groupes conséquents de talents spécialisés seront plus efficaces en suivant un processus rigoureux. Dans de telles organisations, les membres de l'équipe ne jouent qu'un rôle et ne réalisent qu'un ensemble restreint d'activités bien en accord avec leur compétence et leur expérience. Le nombre important de personnes impliquées renforce la nécessité de

communications formelles entre les membres de l'équipe, ce qui se traduira par des artefacts détaillés et complets.

2. Les petites équipes, quant à elles, pourront préférer un processus moins contraignant. Ainsi, des équipes dont les membres ont déjà collaboré avec succès sur d'autres projets et dans lesquelles une dynamique de groupe a instauré une qualité de communication satisfaisante n'auront peut-être pas besoin de toute la rigueur d'un processus surchargé d'artefacts. Que ces types de structures reposent sur des compétences éprouvées ne fera néanmoins pas l'économie d'un processus bien défini. Simplement, certains aspects du processus, tels que les réunions officielles, n'auront pas la même importance.

3. Nature de l'application. Le domaine de l'application peut influencer considérablement la structure du processus de développement. Ainsi le contrôle de qualité deviendra-t-il une priorité dans des secteurs qui engagent la vie humaine, tels que les appareillages médicaux, les engins spatiaux ou les contrôles de centrales nucléaires. Dans tous ces cas, le rythme des itérations sera avant tout défini par la réalisation des objectifs fixés par l'assurance qualité.

4. Les compétences de l'équipe de développement. Le processus doit être adapté afin de tenir compte de la compétence de l'équipe. Des équipes peu expérimentées pourront tirer profit d'un processus plus encadré où les revues de groupe seront essentielles. Pour ce genre d'équipe, le processus de
développement doit aussi devenir un processus d'apprentissage.

5. Les priorités relatives entre les fonctionnalités, le délai de livraison, le taux de défauts acceptable, etc. C'est l'importance relative de tout ce qui caractérise la livraison du système final qui détermine les éléments importants du processus de développement. Par exemple, dans le cas où le but prioritaire du projet est d'être le premier sur le marché, les tests qualité pourront être allégés, au profit de la capacité à s'adapter à de nouveaux besoins qui sera déterminante - quitte à ne répondre qu'à un minimum de besoins dans la première version. Le processus ainsi adapté répondra au mieux à la priorité accordée aux délais.

1.5.1 Le modèle

La communication est essentielle dans le processus; c'est précisément une des attributions du modèle du système que de servir de mécanisme de communication. Le modèle est une représentation abstraite du système à développer, du système en développement et du système développé. Le modèle évolue donc avec le système en jouant un rôle majeur dans chaque phase du projet.

Le modèle est un ensemble d'artefacts représentant chacun une vue du système. Il est utilisé par presque tous les membres de l'équipe, depuis les commanditaires dont les conditions de travail seront améliorées par le système, jusqu'aux développeurs responsables de l'implémentation de ses composants. Chaque participant dans le processus de développement utilise ou enrichit le modèle différemment La communication est facilitée parce que tous les membres sont partie prenante à l'élaboration du même modèle, même si chaque participant en a une vision particulière.

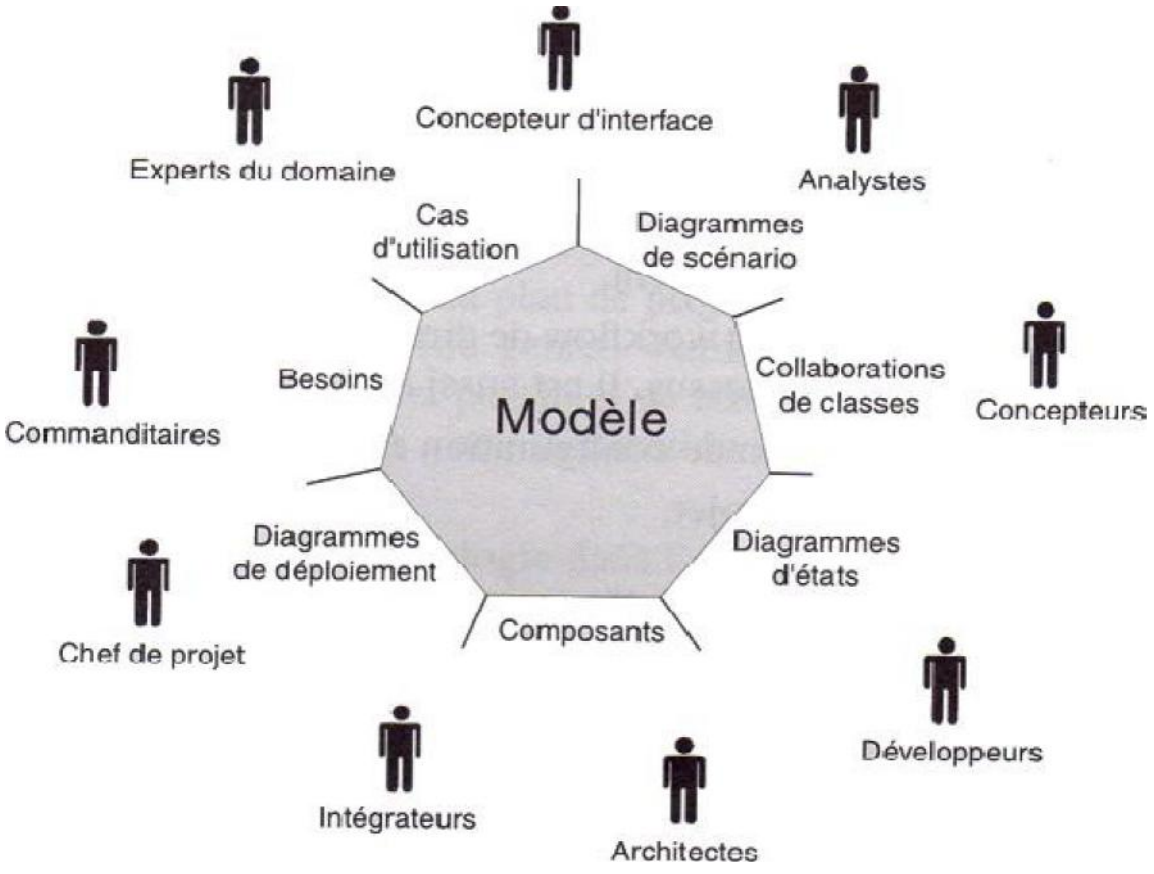

Figure 1.2 Le modèle, un artefact Central du processus De développement

Comme les systèmes deviennent de plus en plus complexes, leur compréhension dépasse les capacités d'un seul individu. La construction d'un modèle abstrait permet d'y remédier. Un bon modèle pourra ainsi servir à déterminer quels composants sont associés à quels cas d'utilisation et en quelle qualité. Le modèle pourra aussi aider à prédire les conséquences que des modifications de besoins pourraient avoir sur le système.

Le modèle présente notamment l'atout de faciliter la traçabilité du système, à savoir la possibilité de partir d'un de ses éléments et de suivre ses interactions et liens avec d'autres parties du modèle. Par exemple, les cas d'utilisation dans le modèle sont associés aux réalisations de cas d'utilisation et aux diagrammes de séquence (scénario), qui à leur tour sont associés aux classes de base et aux collaborations qui implémentent le scénario. La traçabilité permet ainsi aux chefs de projet de naviguer dans le modèle pour trouver les solutions aux problèmes rencontrés pendant le processus de développement.

1.5.2 Le workflows

Les principaux workflows du processus de développement logiciel sont les suivants : la direction de projet, les besoins, l'analyse, la conception, l'implémentation, les tests, le déploiement, la configuration et la gestion des changements. [20]

la direction de projet

Le workflow de direction de projet est responsable de la gestion globale de l'application, incluant la gestion des ressources et des budgets, et assume aussi le rôle de porte-parole du projet. Le workflow de direction de projet produit les artefacts suivants :

6. Planning du projet.
7. Planning des itérations.
8. Gestion du risque.
9. Contrôle de l'avancement.

La collecte des besoins

On appelle besoin un énoncé de ce que le système devrait faire. L'ensemble des besoins est défini dans la spécification des besoins. Définir les besoins pour un système ne va pas de soi : même en partant d'une vision commune, les représentations mentales d'un système diffèrent d'un individu à l'autre. Pire encore, les divergences de conception peuvent aussi se manifester parmi les développeurs.

L’analyse

L'analyse et la conception sont souvent fortement associées ; nous préférons les séparer dans deux workflows distincts. En effet, bien que, dans la majorité des cas, les mêmes individus participent aux deux workflows, les activités et les objectifs sont très différents. L'analyse est le processus qui, en partant de l'examen des besoins, produit un modèle conceptuel du système à développer. Les artefacts de l'analyse incluent les classes et les collaborations détaillées, les diagrammes de séquence, les diagrammes d'états et les diagrammes d'activités. La conception opère sur les mêmes artefacts, mais seulement après que l'architecture y a été intégrée. Au moment de l'analyse, ces artefacts peuvent encore être discutés avec des experts du domaine et des représentants des utilisateurs, alors qu'une fois l'architecture appliquée au moment de la conception, ils deviennent le plus souvent trop techniques pour eux.

La conception

Le workflow de conception traite les artefacts produits durant l'analyse en leur appliquant l'architecture. Le but principal de la conception est de rendre le modèle de l'analyse réalisable sous forme de logiciel. C'est alors la première fois que les concepts abstraits du métier sont confrontés au monde logiciel. Dans certains cas, l'introduction d'une architecture affecte tant le modèle que deux modèles séparés du système sont maintenus : un modèle d'analyse et un modèle de conception. Ces modèles ne sont que deux vues différentes du même système. Dans le cas où deux modèles doivent coexister, le workflow de gestion de configuration et des changements a une nouvelle responsabilité : assurer la cohérence des deux modèles.

L’implémentation

L'implémentation d'applications web implique souvent la maîtrise simultanée de nombreuses technologies. Pour le développement côté client, les langages et les techniques sont principalement HTML, JavaScript, Java, ActiveX et d'éventuelles technologies objet distribuées. Sur le serveur, les langages et les technologies sont plus variés avec des langages de troisième génération et des langages orientés objet (C/C++, Java, Smalltalk, Ada, Eiffel, PHP), mais aussi des technologies à base de composants (JavaBeans, COM), et enfin des technologies traditionnelles de bases de données. On retrouve ainsi sur le serveur toutes les technologies d'un système client serveur classique.

Les tests

De nombreux tests différents sont effectués sur le système; chacun d'entre eux essaie de déterminer une propriété du système. Un test de performance évalue la capacité du système à fonctionner rapidement sous la pression d'une forte charge. Un test de charge soumet le système

à une utilisation intense pour établir son point de rupture ou simplement sa courbe de performance relative. Les tests fonctionnels vérifient que des fonctions particulières ont bien été implémentées telles qu'elles sont définies dans les spécifications des besoins. Les tests fonctionnels découlent souvent directement des cas d'utilisation.

Outre les tests d'évaluation de propriétés du système, d'autres tests permettent de valider certaines étapes du processus :

10. Test unitaire : il est effectué par le développeur sur une partie limitée du système qu'il a développé.
11. Test d'intégration : il valide la compatibilité des différentes interfaces des composants.
12. Test système : il vérifie que l'ensemble des besoins est satisfait.
13. Test d'acceptation : c'est le test officiel par lequel la communauté des utilisateurs valide le système. Si le système est accepté, il est prêt à être déployé.

Le déploiement

Selon les cas, le déploiement d'une application web peut s'avérer très facile ou au contraire très complexe. Ainsi, le déploiement d'une application intranet simple qui fonctionne sur un serveur et bénéficie d'un réseau existant pourra être très aisé. Seul le serveur doit être configuré car les clients possèdent probablement des navigateurs adaptés. Si l'application est conçue pour n'utiliser que les possibilités de base du client, il n'y a rien de plus à faire.

La gestion de configuration et des changement

Si la gestion de configuration et des changements est un workflow en soi, c'est parce que son rôle est vital dans un processus itératif. La raison en est que la possibilité d'incorporer des changements dans le processus en cours, d'une façon contrôlée, est la condition même du succès de la convergence des itérations. Chaque changement est un petit ajustement par rapport à l'orientation fixée au projet.

Le contrôle des changements fournit au chef de projet des informations utiles qui lui permettent de valider la qualité des parties du projet. Des changements fréquents et non prévus dans certains secteurs indiqueraient probablement que ce sont là des secteurs à haut risque, qui n'ont pas fait l'objet d'une analyse suffisante en amont.

1.5.3 Les risques

Le processus de développement vise avant tout à cerner au plus tôt tout risque qu'encourt un projet. Le risque, c'est l'inconnu ; et ce, aussi bien dans les secteurs de l'application qui reposent sur des technologies non éprouvées que dans des domaines où l'équipe de développement manque d'expérience. C'est le plus tôt possible dans le processus que le chef de

projet, l'architecte et les membres clés de l'équipe doivent examiner l'ensemble des artefacts de l'analyse des besoins, incluant les cas d'utilisation, à la recherche de sources de risques.

1.6 Contenu des sites web

1.6.1 Page dynamique et page statique

Un site Web est un ensemble de pages statiques et dynamiques. Une page statique n'est pas modifiée lorsqu'un visiteur la consulte : le serveur Web transmet la page au navigateur qui la sollicite sans la modifier. A l'inverse, une page Web dynamique est modifiée (construite) par le serveur avant d'être transmise au navigateur qui la sollicite. C'est pourquoi cette page est dite dynamique.

Vous pouvez par exemple créer une page pour afficher les résultats du programme de mise en forme et faire en sorte que certaines informations (telles le nom et les résultats de l'employé) soient déterminées lorsqu'une page est sollicitée par un employé donné.

I I. Architecture des applications graphiques

Pour développer une application graphique interactive, le programmeur peut s'appuyer sur plusieurs niveaux logiciels et d'outils dont le rôle s'est clarifié et s'est stabilisé depuis une dizaine d'années. Ces niveaux et outils, du plus prés du matériel jusqu'au plus abstrait, sont :

– les system de fenêtrage

– les boites a outils

– les langages graphiques

– les squelettes d'applications (application frameworks)

– les éditeurs d'interface (interface builders)

Les figures(1.3) montrent quelques exemples d'éditeurs graphiques professionnels et a quel point leur présentation est semblable. On distingue une barre de menu, une boite d'outils, et une fenêtre principale, parfois décorée d'objets de contrôle. Ce qui varie d'une application a une autre, c'est la nature des objets affiches dans la vue principale, le mode`le graphique utilisé pour leur affichage, les dispositifs d'entrée et les modalités d'interaction

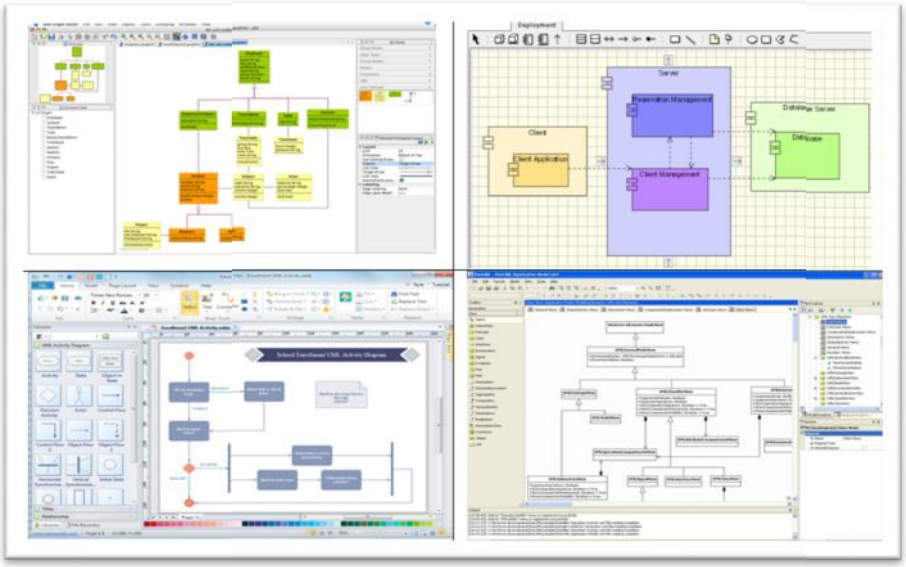

FIG. 1.3 - *Editeurs graphiques sur Windows*

II.1 Vocabulaire et concepts

Pour décrire les éditeurs, nous employons des termes et des concepts qui sont utilisés par la communauté scientifique et technique, parfois en anglais, et dont nous donnons ici un définition.

Un dispositif d'entrée, ou plus simplement dispositif, est un objet qui a une facette physique, une facette numérique et qui gère la communication entre un utilisateur et un ordinateur. Il transforme des signaux physiques e´mis par des capteurs en informations numériques, permet a l'ordinateur de connaitre l'e´tat des capteurs et notifie l'ordinateur lorsque cet e´tat physique change. Les stations de travail disposent toutes d'une souris et d'un clavier comme dispositifs d'entrée.

Un curseur est un petit objet graphique qui visualise la position d'un dispositif sur l'e´cran d'un ordinateur. Un curseur de texte existe dans un éditeur de texte et visualise la position ou le prochain caractère sera inséré lorsque l'on tapera au clavier.

Une action est une modification de l'état d'un dispositif dans un contexte donné, destiné a modifier l'état d'une application graphique interactive. Le fait d'appuyer sur le bouton d'une souris (cliquer) lorsque le curseur est placé sur un objet graphique est une action.

Une manipulation ou une interaction est une suite d'actions destinées a spécifier une commande et ses arguments. Par exemple, appuyer sur le bouton de la souris et le relâcher lorsque le curseur est placé sur un icone dans le Finder est une manipulation qui déclenche la commande (voir ci-dessous) ((sélectionner)) avec comme argument l'icone sous le curseur. Une commande est initiée par une action initiale et est termine´e par une action terminale.

Une commande est une fonction avec ses arguments, déclenchée par une action terminale. Par exemple, l'action consistant a‘ cliquer sur le menu ((Quitter)) déclenche la commande ((exit)) de l'application.

Une modalité d'interaction définit selon quelle interprétation une manipulation sera transformée en commande. Par exemple, la suite d'actions commençant par un bouton de souris appuyé, suivi de mouvements de la souris et terminé par le relâchement du bouton de la souris est une manipulation qui peut être interprétée comme du ((clique et tire)) ou comme du ((dessin a main levée)) ou comme de la ((reconnaissance de geste)).

La manipulation directe a été introduite par B.Shneiderman [22] pour qualifier une manipulation interactive ayant les caractéristiques suivantes :

1. représentation continue des objets d'intérêt.

2. actions physiques plutôt que syntaxe complexe.

3. actions rapides, incrémentales et réversibles dont l'effet sur l'objet est rendu immédiatement visible.

Ces caractéristiques donnent a l'interface des propriétés intéressantes, en particulier en terme de vitesse d'apprentissage, de vitesse d'exécution des tâches et de mémorisation du processus opératoire.

Nous utilisons dans cette thèse la notion de manipulation directe dans un sens plus précis que celui de Shneiderman, et qui a été défini par Karsenty [16] par opposition a manipulation indirecte. En effet, en se rapportant a la définition de Shneiderman, la manipulation d'un menu ou d'un bouton poussoir peut être interprété comme directe, suivant le sens qu'on donne a l'((objet d'intérêt)) de la définition de Shneiderman. Cependant, la manipulation étant destinée principalement a déclencher une action extérieure a‘ l'objet graphique qu'est le menu ou le bouton, Karsenty la qualifie de manipulation indirecte. Le terme manipulation directe est réservé

a l'interaction avec les ((vrais)) objets d'intérêt, c'est-a`-dire ceux présentés dans la vue principale.

II.2.Décomposition des éditeurs en couches:

Suivant leur complexité, on peut distinguer entre deux et un dizaine de couches graphiques dans les éditeurs. Les plus simples, ceux qui ne font que de la visualisation passive, distinguent déjà deux couches graphiques : le fond et la couche de visualisation des données.

II.2.1.Fond

Le fond des applications graphiques a une fonction qui peut être neutre, décorative, support de métaphore ou de schéma. Dans les deux premiers cas, la nature du graphique de la couche est inde´pendante de la se´mantique de l'application graphique.

Le choix peut être motivé par :

Des raisons ergonomiques : un fond gris neutre pour une application utilisée intensivement permet de ne pas fatiguer les yeux.

des raisons décoratives : le logo de la société qui a créé le programme.

pour appuyer une métaphore : dans une application iconique comme le Finder du Macintosh, le fond de l'écran est censé supporter la métaphore du bureau, et dans HyperCard de Apple [23], le graphique du fond est généralement utilisé pour appuyer une métaphore .

ou pour représenter un fond de carte: dans les systèmes de contrôle aérien, le fond de l'écran affiche les balises et les pistes qui guident les avions.

Les systèmes de fenêtrage disposent de certaines primitives spéciales pour gérer le fond. Dans X par exemple, lorsqu'une fenêtre est créée , il est possible de lui associer un fond qui a une couleur, un motif ou une image. Le système de fenêtrage se charge d'afficher le fond lorsque la fenêtre est effacée.

II.2.2. Visualisation passive

La couche de visualisation passive des données (VP), comme son nom l'indique, se contente de visualiser une structure graphique. C'est cette couche qui affiche les objets

graphiques principaux sur les quel l'utilisateur veut agir. L'affichage peut nécessiter un modèle graphique particulier, implanté par une extension le cas échéant.

II.2.3 Gestion de la sélection

Lorsqu'un éditeur doit gérer la sélection, il a le choix entre l'utilisation d'attributs graphiques particuliers ou l'utilisation d'objets graphiques particuliers exemple :

dans les éditeurs de texte: la portion de texte sélectionnée est généralement représentée avec une couleur de fond et de texte différente du texte non sélectionne.

dans les éditeurs schématiques/iconiques: les objets sélectionnés sont généralement

représentés avec des couleurs spécifiques ou grisés.

dans les éditeurs graphiques:

les objets sélectionnés sont ((décorés)) d'éléments graphiques qui permettent de les distinguer (les ((poignées))). Ces éléments graphiques ont généralement une sémantique particulière, a` la fois graphique ils s'affichent rapidement et peuvent s'effacer rapidement et interactive ils sont réactifs au pointeur et la manipulation qu'ils déclenchent de´pend de l'emplacement du pointeur lorsque la manipulation est commencée.

Dans les deux premiers cas, la visualisation de la sélection est généralement faite directement par l'objet graphique de la couche de visualisation passive, tandis que dans le dernier cas, il est intéressant de disposer d'une couche spécifique pour gérer la sélection. C'est la couche de gestion de la sélection. Le comportement de cette couche.

La raison pour laquelle les deux premières catégories peuvent se passer d'utiliser une couche de gestion de la sélection tient au fait que leur visualisation n'utilise pas tous les attributs graphiques disponibles dans le modèle graphique des systèmes de fenêtrage. Le texte utilise deux couleurs, une pour le texte lui même et une pour le fond, tandis que les éditeurs schématiques associent des couleurs conventionnelles a` leurs éléments et peuvent donc décider d'utiliser une couleur/ texture également conventionnelle pour représenter l'état sélectionné.

Les éditeurs graphiques généraux ne peuvent pas toujours utiliser un attribut graphique libre pour visualiser le fait qu'un objet graphique est sélectionné car ils permettent justement aux objets graphiques d'utiliser tous les attributs disponibles.

C'est la raison pour laquelle la sélection est visualisée par des objets graphiques particuliers, qui sont conçus pour se distinguer graphiquement des objets de la couche de visualisation passive. La méthode la plus efficace est certainement celle utilisée par PhotoShop [Adobe Systems Incorporated93a] qui trace la sélection en pointillés animés. Il est alors impossible de confondre la sélection avec le dessin.

II.2.4. Manipulation directe

Un des objectifs déclarés de certains concepteurs d'architectures permettant la manipulation directe est d'offrir un mode de manipulation conforme a la réalité on prend un objet et on le déplace. Les Pacers en sont un exemple en 2D (décrits plus loin). En 3D, plusieurs boîtes a' outils comme Upfront permettent une manipulation en perspective. Cependant, dans notre monde physique, les objets opaques cachent les objets placés derrière eux, tandis que les objets virtuels sur un écran n'ont pas nécessairement cet inconvénient. Il suffit qu'ils se dessinent au-dessus des autres objets pendant la manipulation directe. Leur allouer une couche permet ce résultat.

Lorsqu'un objet graphique est manipulé interactivement et que sa forme se modifie continument a‘ l'écran, les boîtes a` outils Interviews et Fresco préconisent de modifier la structure graphique utilisée pour sa visualisation. Garnet permet de choisir entre la manipulation des objets graphiques et la création d'objets intérimaires avec lesquels sera visualisée la manipulation. Unidraw et ET++ créent des objets graphiques particuliers pour visualiser la manipulation directe. Il existe plusieurs raisons pour utiliser une couche spéciale pour la manipulation directe (MD). En particulier lorsque la sémantique graphique doit être relâchée, schématique ou adaptée.

Sémantique graphique relâchée :

Les éditeurs graphiques, comme Canvas et Idraw(fait avec Unidraw), utilisent une sémantique graphique relâchée pendant la manipulation directe. Les objets sélectionnés, lorsqu'ils sont manipulés, sont représentés par des rectangles englobant leur forme initiale, permettant ainsi a la manipulation de se faire rapidement dans la plupart des cas. Illustrator

n’affiche pas que des rectangles, il affiche les courbes de construction du dessin, tracées de façon optimise´es avec une épaisseur d’un pixel.

Sémantique graphique schématique : Dans les interfaces iconiques comme le Finder du Macintosh, les icones sont tracés en grisé lorsqu’ils sont déplacés. Ce mode de dessin n’accélère pas vraiment les performances mais permet de distinguer durant la manipulation les objets manipulés des objets en place.

Sémantique graphique adaptée : En dehors du modèle graphique, on peut vouloir

modifier le comportement des objets durant leur manipulation directe. Par exemple,

les Pacers sont des objets graphiques qui peuvent être manipulés directement et qui adaptent la qualité de leur affichage a la vitesse de la manipulation. S’ils n’arrivent pas a s’afficher suffisamment rapidement, ils dégradent leur qualité. Dans l’article original, ces objets sont directement les objets graphiques de la visualisation qui sont utilisés pendant la manipulation directe. En utilisant l’architecture multicouche, les Pacers pourraient devenir les objets de la manipulation directe et s’adapter plus facilement aux manipulations qu’on leur applique.

II.2.5 Autres utilisations des couches

L’usage des couches n’est pas limité. Nous avons décrit ici celles qui sont utilisées de façon semblable dans beaucoup d’applications graphiques interactives, mais d’autres utilisations intéressantes existent. Pour n’en citer que quelques unes :

– la séparation explicite d’un dessin en couches, comme le permet Canvas ou PhotoShop .

– le placement d’un modèle numérisé sous la couche de visualisation pour permettre de décalquer virtuellement, comme sous Illustrator .

– le placement d’une zone translucide au-dessus de la couche de visualisation pour délimiter la zone sur laquelle un outil est actif , ce sont les See Through Tools [23]

– le retour des manipulations d’un autre utilisateur dans un éditeur collecticiel [23].

III. Architecture multicouche

La navigation à plusieurs niveaux permet aux utilisateurs d'explorer de vastes espaces d'information. Aider Les utilisateurs ont un sens des données, c'est le but même de la visualisation. Cependant, la croissance des taille des ensembles de données entraîne l'incapacité de transmettre toutes les informations avec Une seule illustration statique. L'interface est alors liée pour afficher seulement un sous-ensemble des données et nécessite des fonctionnalités de navigation multiscalaire pour permettre aux téléspectateurs de manière interactive Reconfigurez la partie de la représentation examinée. Nous définissons Navigation multiscalaire en tant que navigation dans de très grands espaces d'information où les utilisateurs doivent afficher les jeux de données à différents grossissements ou échelles.

La navigation est un concept qui provient du monde physique. Il a ensuite été porté vers les mondes électroniques, Lorsque des moniteurs permettant d'explorer visuellement les informations virtuelles stockées dans les disques durs des ordinateurs ont été introduits. Certains concepts de navigation dans le monde physique, Tels que les repères et les itinéraires ont donc été évalués et parfois appliqués dans ces nouveaux espaces[4]. Bien que la navigation dans les mondes virtuels soit régie par des lois complètement différentes de la navigation dans le monde physique, Dans les mondes virtuels, la locomotion, la visibilité des objets et les formes ne sont pas régies par les lois de la physique et peuvent être modifiées dans un souci de navigation et de visibilité. Si cette nouvelle liberté offre de nouvelles opportunités pour résoudre la navigation Problèmes [6], Spence a reconnu que, cependant, ils "ont sans aucun doute posé Défis pour le concepteur de l'interaction. "[1]

III.1.Interfaces multi-échelles

Dans cette section, nous présentons les principaux schémas d'interface introduits dans la littérature Pour soutenir la navigation à plusieurs niveaux. Cockburn [35] a introduit une classification Mettant en vedette quatre stratégies principales pour mélanger des vues détaillées et contextuelles. Les quatre schémas d'interface sont: Interfaces de zoom, Aperçu + détail, Focus + context, Techniques basées sur la queue.

III.2. Zoom sur l'interface utilisateur:

En mode informatique, une interface utilisateur zoom ou une interface utilisateur zoomable (ZUI, zoo-ee prononcée) est un environnement graphique où les utilisateurs peuvent

modifier l'échelle de la zone visualisée afin de voir plus de détails ou moins et parcourir différents documents. Un ZUI est un type d'interface utilisateur graphique (GUI). Les éléments d'information apparaissent directement sur un bureau virtuel infini (généralement créé à l'aide de graphiques vectoriels), au lieu de Windows. Les utilisateurs peuvent parcourir la surface virtuelle en deux dimensions et zoomer sur les objets d'intérêt. Par exemple, au fur et à mesure que vous zoomez sur un objet texte, il peut être représenté sous forme de petit point, puis une vignette d'une page de texte, puis une page pleine taille et enfin une vue agrandie de la page.

Les ZUI utilisent le zoom en tant que métaphore principale pour la navigation à travers des informations hypertextes ou multi variées. Les objets présents à l'intérieur d'une page agrandie peuvent à leur tour être zoomés eux-mêmes pour révéler des détails supplémentaires, permettant une nidification récursive et un niveau arbitraire de zoom.

Lorsque le niveau de détail présent dans l'objet redimensionné est changé pour s'adapter aux informations pertinentes dans la taille actuelle, au lieu d'être une vue proportionnelle de l'objet entier, on appelle le zoom sémantique. [1]

Certains considèrent le paradigme ZUI comme un successeur flexible et réaliste de l'interface GUI traditionnelle, étant une interface Post-WIMP. Cependant, peu d'efforts sont actuellement consacrés au développement de ZUI, alors qu'il existe des efforts continus pour développer d'autres types d'interfaces graphiques.

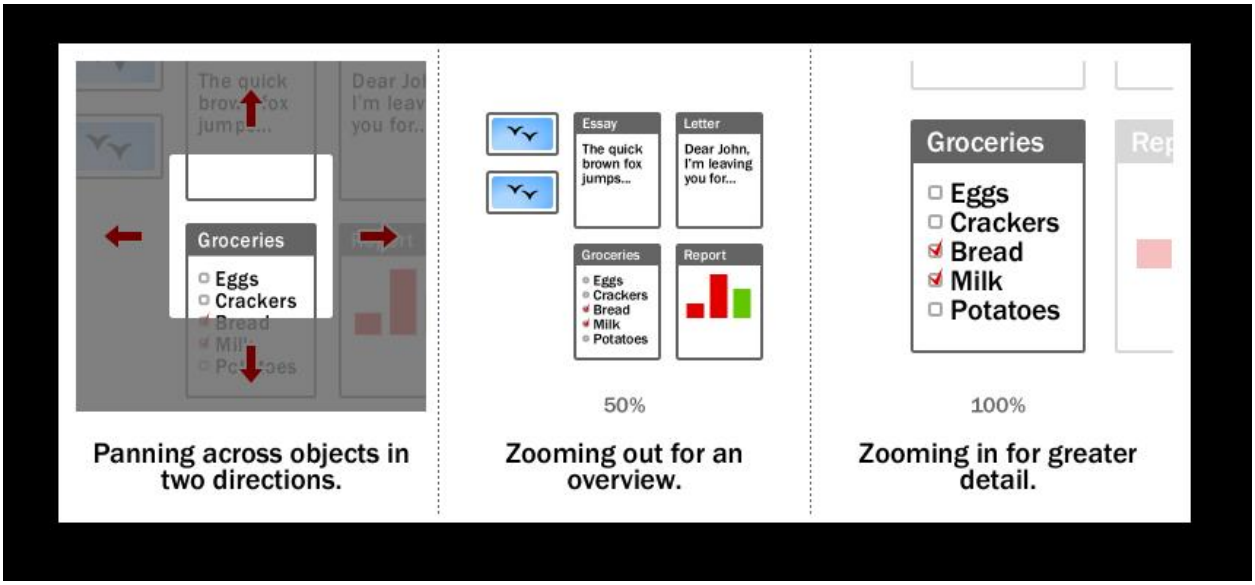

FIG. 1.4 - Exemple de ZUI

Application :

Les fonctionnalités de zoom étaient une caractéristique précoce du logiciel générateur de fractale afin que les utilisateurs puissent agrandir une zone pour révéler plus de détails. Lorsque les opérateurs de saisie de données saisissent des données à partir d'images numériques de documents numérisés, une interface utilisateur de zoom permet de discerner l'écriture de main.

Historique :

Ivan Sutherland a présenté le premier programme pour zoomer et créer des structures graphiques avec contraintes et instanciation, sur un CRT dans son programme Sketchpad en 1962. [2]

Une interface plus générale a été réalisée par l'Architecture Machine Group dans les années 1970 au MIT. Le suivi manuel, l'écran tactile, le joystick et le contrôle vocal ont été utilisés pour contrôler un plan infini de projets, de documents, de contacts, de vidéos et de programmes interactifs. L'une des instances de ce projet s'appelait Spatial Dataland. [3]

Un autre environnement GUI des années 70 qui utilisait l'idée de zoom était Smalltalk au Xerox Parc, qui avait des "ordinateurs de bureau" infinis (seulement plus tard inventé par Apple Computer), qui pourrait être agrandi à partir d'une vue des oiseaux après que l'utilisateur l'ait reconnu Une miniature de la configuration de la fenêtre pour le projet.

L'effort le plus long pour créer un ZUI a été le projet Pad ++ lancé par Ken Perlin, Jim Hollan et Ben Bederson à l'Université de New York et a continué à l'Université du Nouveau-Mexique sous la direction de Hollan. Après Pad ++, Bederson a développé Jazz, puis Piccolo, [4] et maintenant Piccolo2D [5] à l'Université du Maryland, College Park, qui est maintenu en Java et C #. Les efforts les plus récents de ZUI incluent Archy par le défunt Jef Raskin, ZVTM développé à l'INRIA (qui utilise la technique Sigma lens [6]) et le ZUI simple de l'environnement et de la langue de programmation Squeak Smalltalk. Le terme ZUI lui-même a été inventé par Franklin Servan-Schreiber et Tom Grauman alors qu'ils travaillaient ensemble chez Sony Research Laboratories. Ils développaient la première bibliothèque d'interface utilisateur de zoom basée sur Java 1.0, en partenariat avec le professeur Ben Bederson, l'Université du Nouveau-Mexique et le professeur Ken Perlin, de l'Université de New York.

Avant la disponibilité des boîtes à outils ZUI, la fonctionnalité de postes de travail virtuels de plusieurs gestionnaires de fenêtres offrait certains avantages organisationnels des ZUI. Les postes de travail virtuels diffèrent des ZUI car ils ne fournissent pas une métaphore physique du

zoom continu, mais une collection de conteneurs de bureau séparés et de taille fixe. Les fonctions des postes de travail virtuels sont disponibles par défaut dans KDE, GNOME et Mac OS X Leopard, et par un complément dans Microsoft Windows XP.

GeoPhoenix, un Cambridge, MA, démarrage associé au MIT Media Lab, fondé par Julian Orbanes, Adriana Guzman, Max Riesenhuber, a sorti le premier Zoomspace professionnel commercialisé en 2002-3 sur le portable Sony CLIÉ PDA, avec Ken Miura de Sony

En 2006, Hillcrest Labs a introduit le système de navigation de télévision HoME, la première interface graphique et zoom pour la télévision. [7]

En 2007, Live Labs de Microsoft a publié une UI de zoom pour la navigation Web appelée Microsoft Live Labs Deepfish pour la plate-forme Windows Mobile 5.

L'iPhone d'Apple (créé en juin 2007) utilise une forme stylisée de ZUI, dans laquelle le panoramique et le zoom sont effectués via une interface tactile. Un ZUI plus pleinement réalisé est présent dans l'écran d'accueil iOS (depuis iOS 7), avec un zoom de l'écran d'accueil en dossiers et finalement vers les applications. L'application photo diffuse d'une seule photo à des moments, à des collections, à des années. Et de même dans l'application de calendrier avec des vues de jour, de mois et d'année. [8] Il ne s'agit pas d'une implémentation complète de ZUI car ces opérations sont appliquées à des espaces délimités (tels que des pages Web ou des photos) et ont une portée limitée de zoom et de panoramique.

Franklin Servan-Schreiber a fondé Zoomorama, basé sur le travail qu'il a effectué chez Sony Research Laboratories au milieu des années 90. Le navigateur Zooming pour Collage d'images haute résolution a été publié dans Alpha en octobre 2007. Le navigateur de Zoomorama est basé sur Flash. Le développement de ce projet a été arrêté en 2010.

Aperçu + détail

Les techniques de Aperçu + détail fournissent à la fois une vue détaillée et un aperçu de l'espace d'information simultanément, chacun dans des espaces de présentation distincts. L'aperçu standard + interface de détail, comme Cockburn le nomme [35], est celui Mis en œuvre par Google Maps, qui introduit un petit Aperçu de la carte en bas à droite de la fenêtre, fournissant aux utilisateurs Avec plus d'informations contextuelles. Cette technique a été largement mise en œuvre Dans les jeux vidéo, plus particulièrement les jeux de stratégie ou les jeux de sport où une Une petite représentation de l'ensemble de la carte aide les utilisateurs à

situer leur emplacement actuel, Ou des applications de bureau telles que les outils d'édition d'image.

Focus + Contexte

Le troisième schéma d'interface, appelé focus + context, intègre la mise au point et le contexte dans un seul affichage, affichant la mise au point sans problème dans son contexte environnant. Les techniques Focus + context ont été inspirées par les lentilles fisheye utilisées par les photographes

Conclusion:

Le développement web constitue un domaine en expansion, les entreprises, les particuliers tout le monde est concerné. En parallèle, les technologies évoluent vers des modèles facilitant visiblement la tâche des développeurs web, des solutions d'intégration et de conception d'architectures web sont proposées pour accélérer le processus de développement. Ce chapitre nous a présenté quelques aspects de ce domaine. Nous nous sommes focalisés sur le coté fonctionnel pour permettre une appropriation avec le thème de ce mémoire à savoir le rapprochement des méthodes de conception des applications web afin de proposer des solutions fonctionnellement exploitables.

Chapitre II

UML

1. I. Présentation générale d'UML

I.1. Introduction

Le génie logiciel et la méthodologie s'efforcent de couvrir tous les aspects de la vie du logiciel. Issus de l'expérience des développeurs, concepteurs et chefs de projets, ils sont en constante évolution, parallèlement à l'évolution des techniques informatiques et du savoir-faire des équipes. Comme toutes les tentatives de mise à plat d'une expérience et d'un savoir-faire,

les méthodologies ont parfois souffert d'une formalisation excessive, imposant aux développeurs des contraintes parfois contre-productives sur leur façon de travailler. Avec la mise en commun de l'expérience et la maturation des savoir-faire, on voit se développer à présent des méthodes de travail à la fois plus proches de la pratique réelle des experts et moins contraignantes.

UML, qui se veut un instrument de capitalisation des savoir-faire puisqu'il propose un langage qui soit commun à tous les experts du logiciel, va dans le sens de cet assou-plissement des contraintes méthodologiques.

UML signifie Unified Modeling Language. La justification de chacun de ces mots nous servira de fil conducteur pour cette présentation.

I.2.historique des méthodes de conception

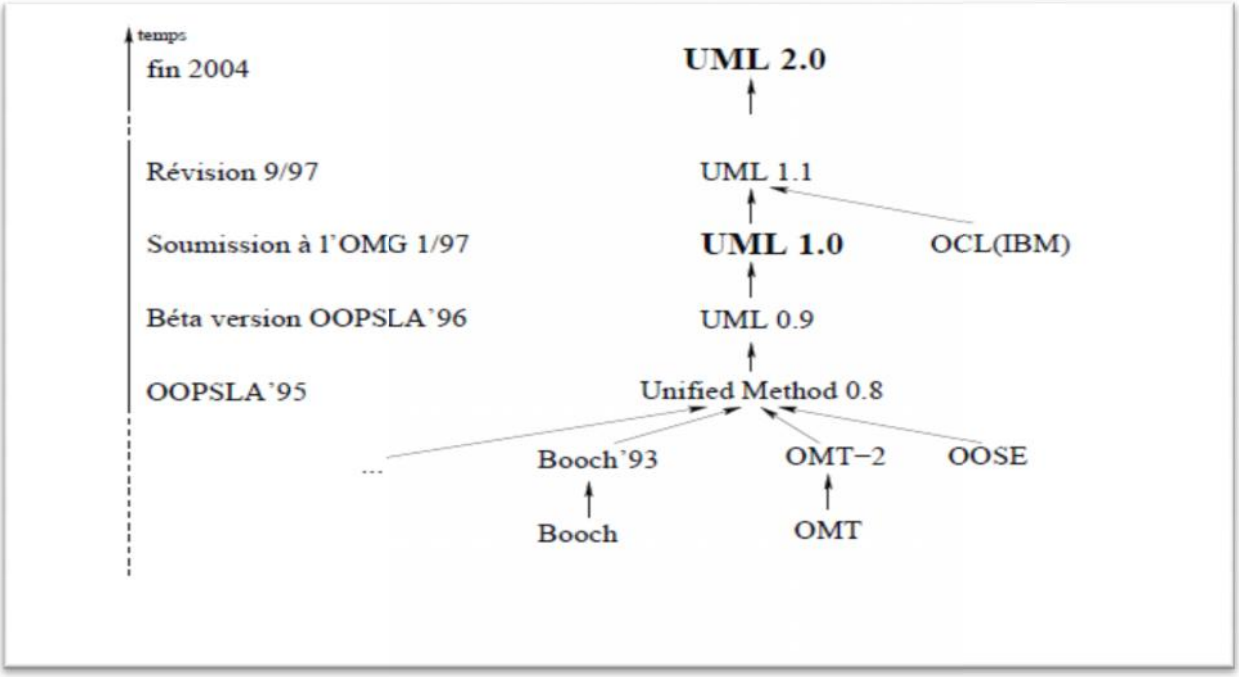

FIG.2.1 Historique de la constitution d'UML

À chacune des différentes phases de la conception d'un logiciel correspondent des problèmes ou des contraintes différentes. Naturellement, ces niveaux ont fait l'objet de recherches méthodologiques considérables depuis les années 80. Il en résulte que de nombreuses méthodes de développement ou d'analyse de logiciel ont vu le jour, chacune plus ou moins spécialisée ou adaptée à une démarche particulière, voire à un secteur industriel particulier (bases de données, matériel embarqué, ...) [url1]. Celles-ci ayant été développées indépendamment les unes des autres, elles sont souvent partiellement redondantes ou incompatibles entre elles lorsqu'elles font appel à des notations ou des terminologies différentes, voire à des faux amis.

De plus, à chaque méthode correspond un ou plusieurs moyens (plus ou moins formel) de représentation des résultats. Celui-ci peut être graphique (diagramme synoptique, plan physique d'un réseau, organigramme) ou textuel (expression d'un besoin en langage naturel, jusqu'au listing du code source). Dans les années 90, un certain nombre de méthodes orientées objets ont émergé, en particulier les méthodes :

– OMT de James RUMBAUGH [12]

– BOOCH de Grady BOOCH [13],

– OOSE (Object Oriented Software Engineering) de Ivar JACOBSON à qui l'on doit les Use cases [13]

En 1994, on recensait plus de 50 méthodologies orientées objets. C'est dans le but de remédier à cette dispersion que les « poids-lourds » de la méthodologie orientée objets ont entrepris de se regrouper autour d'un standard. En octobre 1994, Grady Booch et James Rumbaugh se sont réunis au sein de la société RATIONAL [url3] dans le but de travailler à l'élaboration d'une méthode commune qui intègre les avantages de l'ensemble des méthodes reconnues, en corrigeant les défauts et en comblant les déficits. Lors de OOPSLA'95 (Object Oriented Programming Systems, Languages and Applications, la grande conférence de la programmation orientée objets), ils présentent UNIFIED METHOD V0.8. En 1996, Ivar Jacobson les rejoint.

Leurs travaux ne visent plus à constituer une méthodologie, mais un langage 2. Leur initiative a été soutenue par de nombreuses sociétés, que ce soit des sociétés de développement (dontMicrosoft, Oracle, Hewlet-Packard, IBM qui a apporté son langage de contraintes OCL ...) ou des sociétés de conception d'ateliers logiciels. Un projet a été déposé en janvier 1997 à l'OMG 3 en vue de la normalisation d'un langage de modélisation. Après amendement, celui-ci a été accepté en novembre 97 par l'OMG sous la référence UML-1.1. La version UML-2.0 est annoncée pour la fin 2004.

I.3.A quoi sert UML ?

UML utilise l'approche objet en présentant un langage de description universel. Il permet grâce à un ensemble de diagrammes très explicites, de représenter l'architecture et le fonctionnement des systèmes informatiques complexes en tenant compte des relations entre les concepts utilisés et l'implémentation qui en découle.

UML est avant tout un support de communication performant, qui facilite la représentation et la compréhension de solutions objet :

.Sa notation graphique permet d'exprimer visuellement une solution objet, ce qui facilite la comparaison et l'évaluation de solutions.

· L'aspect formel de sa notation, limite les ambiguïtés et les incompréhensions.

· Son indépendance par rapport aux langages de programmation, aux domaines d'application et aux processus, en fait un langage universel.

UML est donc bien plus qu'un simple outil qui permet de "dessiner" des représentations mentales... Il permet de parler un langage commun, normalisé mais accessible, car visuel.

Il représente un juste milieu entre langage mathématique et naturel, pas trop complexe mais suffisamment rigoureux, car basé sur un méta modèle. Une autre caractéristique importante d'UML, est qu'il cadre l'analyse. UML permet de représenter un système selon différentes vues complémentaires : les diagrammes.

I.4.Que ce qu'un diagramme UML ?

Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle ; c'est une perspective du modèle.

Chaque type de diagramme UML possède une structure (les types des éléments de modélisation qui le composent sont prédéfinis) et véhicule une sémantique précise (il offre toujours la même vue d'un système). Combinés, les différents types de diagrammes UML offrent une vue complète des aspects statiques et dynamiques d'un système. Les diagrammes permettent donc d'inspecter un modèle selon différentes perspectives et guident l'utilisation des éléments de modélisation (les concepts objet), car ils possèdent une structure.

Une caractéristique importante des diagrammes UML, est qu'ils supportent l'abstraction. Cela permet de mieux contrôler la complexité dans l'expression et l'élaboration des solutions objet.

UML opte en effet pour l'élaboration des modèles, plutôt que pour une approche qui impose une barrière stricte entre analyse et conception. Les modèles d'analyse et de conception ne diffèrent que par leur niveau de détail, il n'y a pas de différence dans les concepts utilisés.

UML n'introduit pas d'éléments de modélisation propres à une activité (analyse, conception.) le langage reste le même à tous les niveaux d'abstraction.

Cette approche simplificatrice facilite le passage entre les niveaux d'abstraction. L'élaboration encourage une approche non linéaire, les "retours en arrière" entre niveaux d'abstraction différents sont facilités et la traçabilité entre modèles de niveaux différents est assurée par l'unicité du langage. Il s'agit d'une tâche très complexe, qui nécessite une approche itérative, car il est plus efficace de construire et valider par étapes, ce qui est difficile à cerner et maîtriser.

C'est donc avec beaucoup d'intérêt que nous avons pris connaissance de cette proposition de notre encadreur à utiliser comme langage de modélisation UML

I.5.Avantages et inconvénients d'UML

I.5.1.Les points forts d'UML

UML est un langage formel et normalisé : Il permet le gain de précision, encourage l'utilisation d'outils et constitue à cet effet un gage de stabilité.

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.

I.5.2.Les points faibles d'UML

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. Or, l'intégration d'UML dans un processus n'est pas triviale et améliorer un processus est une tâche complexe et longue. Les auteurs d'UML sont tout à fait conscients de l'importance du processus, mais l'acceptabilité industrielle de la modélisation objet passe d'abord par la disponibilité d'un langage d'analyse.

I.6.Modeling : analyse et conception

Une bonne méthodologie de réalisation de logiciels suppose une bonne maîtrise de la distinction entre l'analyse et la conception [14]. Le lecteur verra qu'en pratique, le respect d'une distinction entre des phases d'analyse et de conception rigoureusement indépendantes n'est pas tenable, mais il est important d'avoir en tête la différence lorsqu'on s'apprête à réaliser un logiciel. Encore une fois, il est important de garder à l'esprit qu'UML n'offre pas une

méthodologie pour l'analyse et la conception, mais un langage qui permet d'exprimer le résultat de ces phases. Du point de vue des notations employées en UML, les différences entre l'analyse et la conception se traduisent avant tout par des différences de niveau de détail dans les diagrammes utilisés. On peut ainsi noter les différences suivantes :

– Dans un diagramme de classes d'analyse, les seules classes qui apparaissent servent à décrire des objets concrets du domaine modélisé.

Dans un diagramme de classes de conception, par opposition, on trouve aussi toutes les classes utilitaires destinées à assurer le fonctionnement du logiciel.

– Dans un diagramme de classes d'analyse, on peut se contenter de faire apparaître juste la dénomination des classes, avec parfois le nom de quelques attributs et méthodes quand ceux-ci découlent naturellement du domaine modélisé. Dans un diagramme de classes de conception, par opposition, tous les attributs et toutes les méthodes doivent apparaître de façon détaillée, avec tous les types de paramètres et les types de retour.

– Dans un diagramme de séquence d'analyse, les communications entre les principaux

objets sont écrits sous forme textuelle, sans se soucier de la forme que prendront ces échanges lors de la réalisation du logiciel. Dans un diagramme de séquence de conception, par opposition, les échanges entre classes figurent sous la forme d'appels de méthodes dont les signatures sont totalement explicitées. Les étapes permettant de passer de diagrammes d'analyse à des diagrammes de conception et les motivations de la formalisation progressive.

I.7.Language : méthodologie ou langage de modélisation

Il est important de bien faire la distinction entre une méthode qui est une démarche d'organisation et de conception en vue de résoudre un problème informatique, et le formalisme dont elle peut user pour exprimer le résultat (voir le glossaire en annexe).

Les grandes entreprises ont souvent leurs propres méthodes de conception ou de réalisation de projets informatiques. Celles-ci sont liées à des raisons historiques, d'organisation administrative interne ou encore à d'autres contraintes d'environnement (défense nationale, ...) et il n'est pas facile d'en changer. Il n'était donc pas réaliste de tenter de standardiser une méthodologie de conception au niveau mondial. UML n'est pas une méthode, mais un langage. Il peut donc être utilisé sans remettre en cause les procédés habituels de conception de l'entreprise et, en particulier, les méthodes plus anciennes telles que celle proposée par OMT sont tout à fait utilisables.

D'ailleurs, la société RATIONAL (principale actrice de UML) propose son propre processus de conception appelé OBJECTORY [JBR97a] et entièrement basé sur UML.

Ainsi, UML facilite la communication entre clients et concepteurs, ainsi qu'entre équipes de concepteurs. De plus, sa sémantique étant formellement [JBR97a], cela accélère le développement des outils graphiques d'atelier de génie logiciel permettant ainsi d'aller de la spécification (haut niveau) en UML vers la génération de code (JAVA, C++, ADA, ...). De plus, cela autorise l'échange électronique de documents qui deviennent des spécifications exécutables en UML.

UML ne se contente pas d'homogénéiser des formalismes existants, mais apporte également un certain nombre de nouveautés telles que la modélisation d'architectures distribuées ou la modélisation d'applications temps-réel avec gestion du multi-tâches.

II.Différentes vues et diagrammes d'UML

Toutes les vues proposées par UML sont complémentaires les unes des autres, elles permettent de mettre en évidence différents aspects d'un logiciel à réaliser. On peut organiser une présentation d'UML autour d'un découpage en vues, ou bien en différents diagrammes, selon qu'on sépare plutôt les aspects fonctionnels des aspects architecturaux, ou les aspects statiques des aspects dynamiques. Nous adopterons plutôt dans la suite un découpage en diagrammes, mais nous commençons par présenter les différentes vues, qui sont les suivantes :

1 - la vue fonctionnelle: interactive, qui est représentée à l'aide de diagrammes de cas et de diagrammes des séquences. Elle cherche à appréhender les interactions entre les différents acteurs/utilisateurs et le système, sous forme d'objectif à atteindre d'un côté et sous forme chronologique de scénarios d'interaction typiques de l'autre.

2 - la vue structurelle: ou statique: réunit les diagrammes de classes et les diagrammes de packages. Les premiers favorisent la structuration des données et tentent d'identifier les objets/composants constituant le programme, leurs attributs, opérations et méthodes, ainsi que les liens ou associations qui les unissent. Les seconds s'attachent à regrouper les classes fortement liées entre elles en des composants les plus autonomes possibles. A l'intérieur de chaque package, on trouve un diagramme de classes.

3 - la vue dynamique: qui est exprimée par les diagrammes d'états. Cette vue est plus algorithmique et orientée « traitement », elle vise à décrire l'évolution (la dynamique) des objets complexes du programme tout au long de leur cycle de vie. De leur naissance à leur mort, les objets voient leurs changement d'états guidés par les interactions avec les autres

objets. Le diagramme d'activité est une sorte d'organigramme correspondant à une version simplifiée du diagramme d'états. Il permet de modéliser des activités qui se déroulent en parallèle les unes des autres, quand ce parallélisme peut poser problème. En général, les diagrammes d'états à eux seuls ne permettent pas défaire apparaître les problèmes spécifiques posés par la synchronisation des processus en concurrence, pour assurer la cohérence du comportement et l'absence d'inter blocage. Etablir un diagramme d'activité peut aider à mettre au point un diagramme d'états.

Outre les diagrammes précédemment mentionnés, il existe aussi les diagrammes suivants, que nous ne présenterons pas, dans la mesure où ils relèvent plus spécifiquement de la conception ou de l'implémentation.

1 - les diagrammes de collaboration: en appoint de la vue fonctionnelle. Proches des scénarios ou diagrammes de séquences, ces diagrammes insistent moins sur le séquencement chronologique des événements. En numérotant les messages pour conserver l'ordre, ils insistent sur les liens entre objets émetteurs et récepteurs de messages, ainsi que sur des informations supplémentaires comme des conditions d'envoi ou des comportements en boucle, ce que ne permettent pas les diagrammes de séquence, trop linéaires.

2 - les diagrammes de déploiement: spécifiques de l'implémentation, qui indiquent sur quelle architecture matérielle seront déployés les différents processus qui réalisent l'application.

3- Le diagramme des cas (vue fonctionnelle)

3.1 Les cas d'utilisation

Le diagramme des cas est un apport d'Ivar Jacobson à UML.

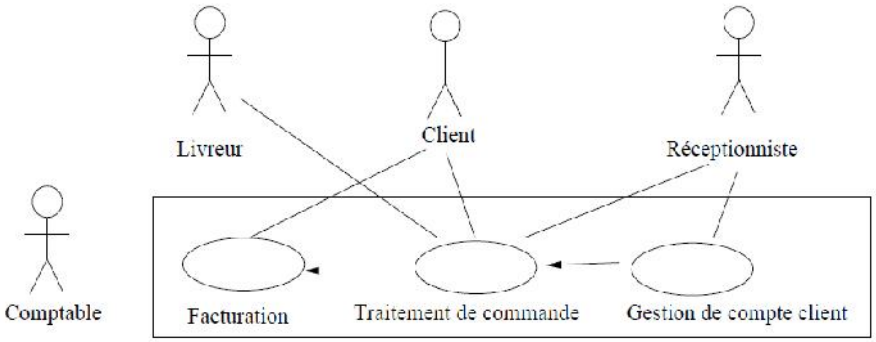

FIG.2.2:Exemple de diagramme de cas

Un cas d'utilisation (use case) modélise une interaction entre le système informatique à développer et un utilisateur ou acteur interagissant avec le système. Plus précisément, un cas

d'utilisation décrit une séquence d'actions réalisées par le système qui produit un résultat observable pour un acteur. Il y a en général deux types de description des use cases :

– une description textuelle de chaque cas

– le diagramme des cas, constituant une synthèse de l'ensemble des cas ;

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, les cas nécessitant des traitements d'exceptions, les effets du déroulement sur l'ensemble du système, etc.

Liens entre cas d'utilisation : **include** et **extend**

Il est parfois intéressant d'utiliser des liens entre cas (sans passer par un acteur), UML en fournit de deux types : la relation utilise (include) et la relation étend (extend). Utilisation de cas : La relation utilise (include) est employée quand deux cas d'utilisation ont en commun une même fonctionnalité et que l'on souhaite factoriser celle-ci en créant un sous-cas, ou cas intermédiaire, afin de marquer les différences d'utilisation.

Extension de cas (extend) : Schématiquement, nous dirons qu'il y a extension d'un cas d'utilisation quand un cas est globalement similaire à un autre, tout en effectuant un peu plus de travail (voire un travail plus spécifique). Cette notion – à utiliser avec discernement – permet d'identifier des cas particuliers (comme des procédures à suivre en cas d'incident) dès le début ou lorsque l'attitude face à un utilisateur spécifique du système doit être spécialisée ou adaptée. Il s'agit grosso modo d'une variation du cas d'utilisation normale.

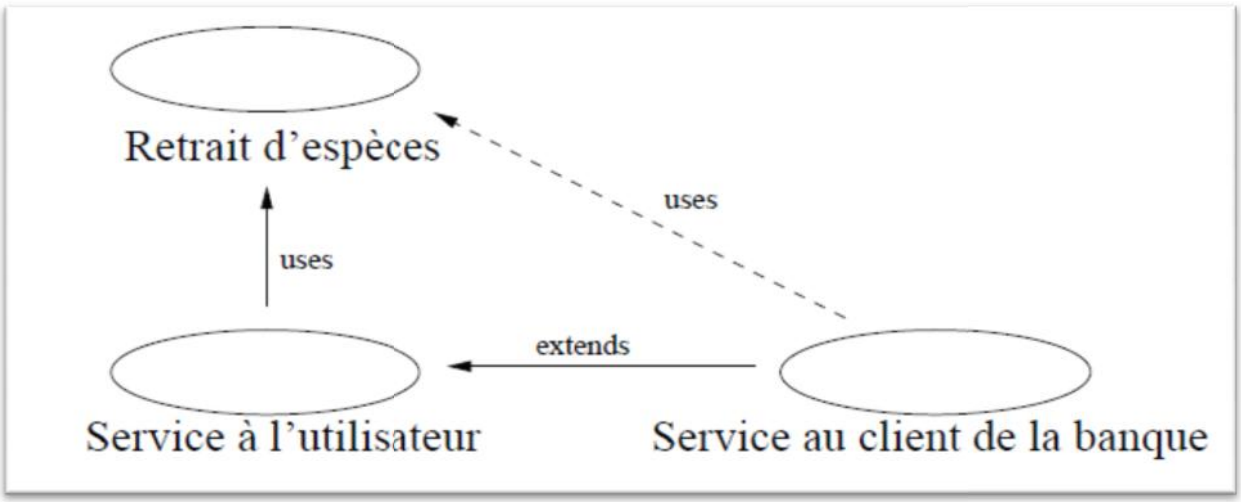

FIG.2.3.Exemple de diagramme d'utilisation

Par exemple, dans le cas d'un distributeur automatique de billets dans une banque, les utilisateurs du distributeur qui sont clients de la banque peuvent effectuer des opérations qui ne sont pas accessibles à l'utilisateur normal (par exemple, consultation de solde).
On dira que le cas « service au client de la banque » étend le cas « service à l'utilisateur ». Mais on peut dire aussi que les deux types de clients peuvent effectuer des retraits, si bien que les cas « service au client de la banque » et « service à l'utilisateur » utilisent tous les deux le cas « retrait d'espèces ». On représente cet exemple sur la figure.

4 Le diagramme des classes (vue structurelle)

4.1 Introduction au diagramme des classes

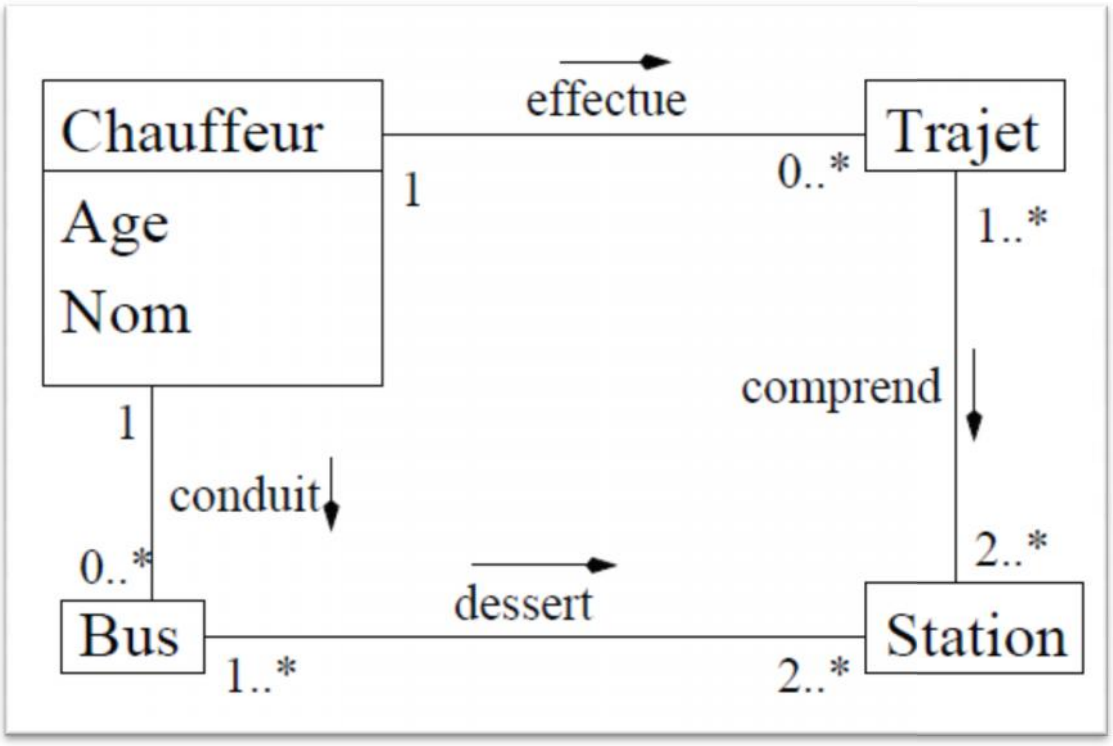

FIG.2.4.Exemple de diagramme de Classe

Un diagramme des classes décrit le type des objets ou données du système ainsi que les différentes formes de relation statiques qui les relient entre eux. On distingue classiquement deux types principaux de relations entre objets :

– les associations, bien connues des vieux modèles entité/association utilisés dans la conception des bases de données depuis les années 70 .

– les sous-types, particulièrement en vogue en conception orientée objets, puisqu'ils s'expriment très bien à l'aide de l'héritage en programmation.

La figure (2.4) présente un exemple de diagramme de classes très simple, tel qu'on pourrait en rencontrer en analyse. On voit qu'un simple coup d'oeil suffit à se faire une première idée des entités modélisées et de leurs relations. Nous allons examiner successivement chacun des éléments qui le constituent. Auparavant, nous introduirons les packages.

4.2 Description d'une classe

L'ensemble des éléments qui décrivent une classe sont représentés sur la figure (2.5) On notera que l'on peut spécifier le package d'appartenance de la classe, dans lequel figure sa description complète, au dessus du nom de la classe.

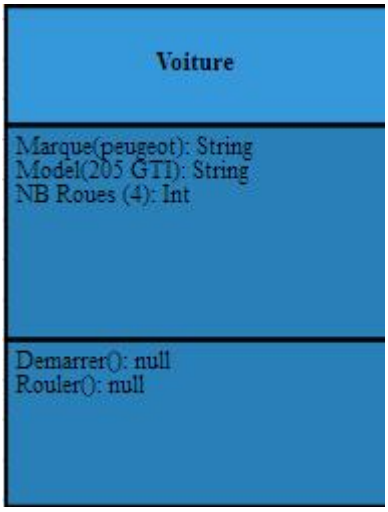

FIG.2.5. Classe

4.2.1 Les attributs

Pour une classe, un attribut est une forme dégénérée d'association entre un objet de la classe et un objet de classe standard : c'est une variable qui lui est en général propre (dans certains cas elle peut être commune à la classe et non particulière à l'objet, on parle d'attributs de classes).

Dans le cadre d'un diagramme de classes UML en analyse, il faut s'interdire de représenter une variable propre à une classe sous forme d'attribut si la variable est elle-même instance d'une classe du modèle. On représente donc les relations d'attribution entre classes sous forme

d'associations et ce n'est qu'au moment du passage à la conception que l'on décidera quelles associations peuvent être représentées par des attributs.
En revanche, plus on se rapproche de la programmation, plus il faut envisager l'attribut comme un champ, une donnée propriété de l'objet, alors qu'une association est bien souvent une référence vers un autre objet. La notation complète pour les attributs est la suivante :

<visibilité> <nomAttribut> :<type> = <valeur par défaut>

La visibilité (ou degré de protection) indique qui peut avoir accès à l'attribut (ou aux opérations dans le cas des opérations). Elle est symbolisée par un opérateur :
« + » pour une opération publique, c'est-à-dire accessible à toutes les classes (a priori associée à la classe propriétaire) .
« # » pour les opérations protégées, autrement dit accessibles uniquement par les sous-classes de la classe propriétaire .
« - » pour les opérations privées, inaccessibles à tout objet hors de la classe.

Il n'y a pas de visibilité par défaut en UML ; l'absence de visibilité n'indique ni un attribut public ni privé, mais seulement que cette information n'est pas fournie. Le type est en général un type de base (entier, flottant, booléen, caractères, tableaux...), compte tenu de ce qui a été dit plus haut. La valeur par défaut est affectée à l'attribut à la création des instances de la classe, à moins qu'une autre valeur ne soit spécifiée lors de cette création. En analyse, on se contente souvent d'indiquer le nom des attributs. On note en les soulignant les attributs de classe, c'est-à-dire les attributs qui sont partagés par toutes les instances de la classe. Cette notion, représentée par le mot clef static en C++, se traduit par le fait que les instances n'auront pas dans la zone mémoire qui leur est allouée leur propre champ correspondant à l'attribut, mais iront chercher sa valeur directement dans la définition de la classe.

4.2.2 Les opérations

Une opération, pour une classe donnée, est avant tout un travail qu'une classe doit mener à bien, un contrat qu'elle s'engage à tenir si une autre classe y fait appel. Sous l'angle de la programmation, il s'agit d'une méthode de la classe. La notation complète pour les opérations est la suivante :
<visibilité> <nomOpération> (listeParamètres) : <typeRetour> {propriété}

La propriété, notée en français, ou sous forme d'équation logique, permet d'indiquer un pré-requis ou un invariant que doit satisfaire l'opération.

La liste des paramètres est de la forme <nom> : <type> = <valeur par défaut>

Il est souhaitable de distinguer deux familles d'opérations, celles susceptibles de changer l'état de l'objet (ou un de ses attributs) et celles qui se contentent d'y accéder et de le visualiser sans l'altérer. On parle de modifiants, ou mutateurs et de requêtes ou accesseurs .On parle également d'opérations d'accès (qui se contentent de renvoyer la valeur d'un attribut) ou d'opérations de mise à jour qui se cantonnent à mettre à jour la valeur d'un attribut.

Notons qu'une opération ne se traduit pas toujours par une unique méthode. Une opération est invoquée sur un objet (un appel de procédure) alors qu'une méthode est le corps de cette même procédure. En cas de polymorphisme, quand un super-type a plusieurs sous-types, une même opération correspond à autant de méthodes qu'il y a de sous-types qui ont redéfini cette opération (+1).

4.3 Les interfaces

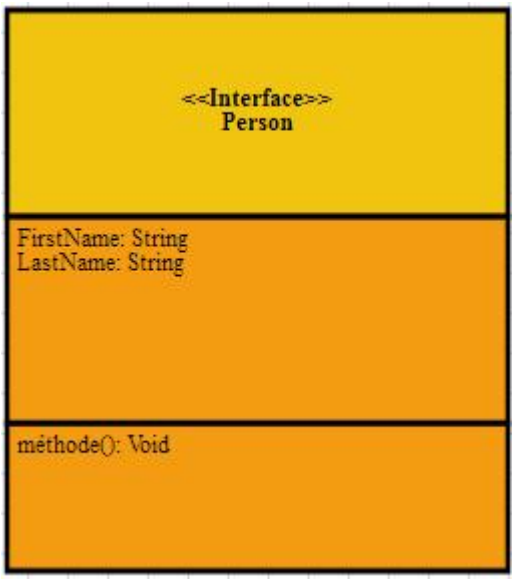

FIG.2.6. interfaces

La notion d'interface est légèrement variable d'un langage de programmation à l'autre. En C++, on conçoit plutôt l'interface comme la totalité des éléments d'un objet visibles de l'extérieur de cet objet. En pratique, il s'agit des méthodes et attributs publics, c'est-à-dire accessibles par toutes les autres classes de l'application. En JAVA, l'interface est plutôt vue comme une spécification externe des interactions qu'une classe peut avoir avec les autres classes de l'application. Distincte de la liste des méthodes publiques, l'interface est plutôt un « contrat » de la spécification de l'ensemble des traitements que la classe s'engage à effectuer si

elle veut prétendre disposer d'une certaine interface. Une classe qui dispose d'une interface doit implémenter toutes les opérations décrites par celle-ci. La notion d'interface proposée par UML, bien que théoriquement indépendante de tout langage, semble davantage se rapprocher du sens donné à cette notion en JAVA.

4.4 Les associations

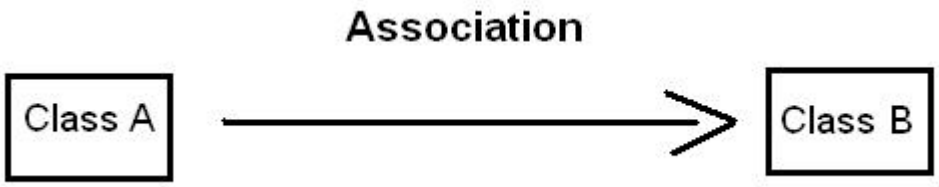

FIG.2.7. Association

Les associations représentent des relations entre objets, c'est-à-dire entre des instances de classes.

En général, une association est nommée. Par essence, elle a deux rôles, selon le sens dans lequel on la regarde. Le rapport entre un client et ses demandes n'a rien à voir avec celui qui unit une demande à son client, ne serait-ce que dans le sens où un client peut avoir un nombre quelconque de demandes alors qu'une demande n'a en général qu'un client propriétaire.

On cherchera, pour plus de lisibilité et pour faciliter les phases suivantes de la conception, à expliciter sur le diagramme le nom d'un rôle. En l'absence de nom explicite, il est d'usage de baptiser le rôle du nom de la classe cible.

À un rôle peut être ajoutée une indication de navigabilité, qui exprime une obligation

de la part de l'objet source à identifier le ou les objets cibles, c'est-à-dire en connaître la ou les références.

Quand une navigabilité n'existe que dans un seul sens de l'association, l'association est dite monodirectionnelle. On place généralement une flèche sur le lien qui la représente graphiquement, mais ce lien peut être omis quand il n'y a pas d'ambiguïté.

Une association est bidirectionnelle quand il y a navigabilité dans les deux sens, et induit la contrainte supplémentaire que les deux rôles sont inverses l'un de l'autre.

Agrégation et composition

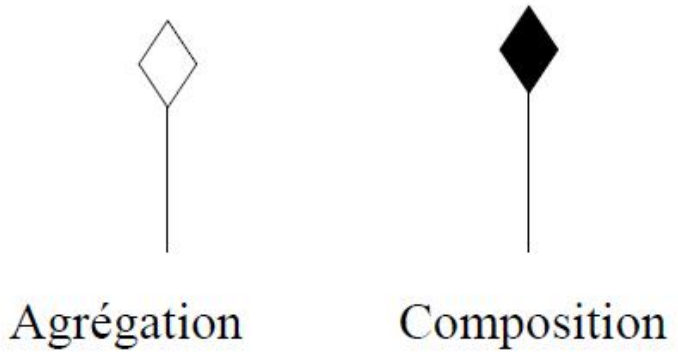

FIG.2.8. Agrégation et composition

Il est des cas particuliers d'associations qui posent souvent problème, ce sont les relations de la forme « partie de », pour lesquels plusieurs définitions existent et donc plusieurs modèles et manières de faire. Aussi faut-il s'entendre entre concepteurs sur les définitions qui vont suivre.

La **composition**, représentée par un losange noir, indique que l'objet « partie de » ne peut appartenir qu'à un seul tout. On considère en général que les parties d'une composition naissent et meurent avec l'objet propriétaire. Par exemple, les chambres d'un hôtel entretiennent une relation de composition avec l'hôtel. Si on rase l'hôtel, on détruit les chambres.

À l'inverse, on parle d'agrégation quand les objets « partie de » sont juste référencés par l'objet, qui peut y accéder, mais n'en est pas propriétaire. Cette relation est notée par un losange blanc. Par exemple, un train est constitué d'une série de wagons, mais ces wagons peuvent être employés pour former d'autres trains. Si le train est démantelé, les wagons existent toujours.

La confusion entre composition et agrégation illustre parfaitement la relative perméabilité entre l'analyse et la conception. Devant la difficulté à décider de la nature d'une relation, décision qui relève pourtant de l'analyse, on s'appuie généralement sur la conception pour fixer son choix. En pratique, on se demande si l'objet « partie de » peut ou doit être détruit lorsqu'on détruit l'objet qui le contient et, si la réponse est affirmative, on choisit une relation de composition.

4.4.1 Les cardinalités (ou multiplicités)

Un rôle est doté d'une multiplicité qui fournit une indication sur le nombre d'objets d'une même classe participant à l'association. La notation est la suivante :

1	:	Obligatoire (un et un seul)
0..1	:	Optionnel (0 ou 1)
0..* ou *	:	Quelconque
n..*	:	Au moins n
$n..m$	:	Entre n et m
l,n,m	:	l, n, ou m

FIG.2.9. Cardinalités

Les termes que l'on retrouve le plus souvent sont : 1,*, 1..* et 0..1 (figure 2.9).

Mais on peut imaginer d'autres cardinalités comme 2, pour les extrémités d'une arête d'un graphe. Il faut noter que les cardinalités se lisent en UML dans le sens inverse du sens utilisé dans MERISE. Ici, la multiplicité qualifie la classe auprès de laquelle elle est notée.

5 Les diagrammes de séquences (vue fonctionnelle)

Les diagrammes de séquences mettent en valeur les échanges de messages (déclenchant des événements) entre acteurs et objets (ou entre objets et objets) de manière chronologique, l'évolution du temps se lisant de haut en bas. Chaque colonne correspond à un objet (décrit dans le diagramme des classes), ou éventuellement à un acteur, introduit dans le diagramme des cas. La ligne de vie de l'objet représente la durée de son interaction avec les autres objets du diagramme.

Un diagramme de séquences est un moyen semi-formel de capturer le comportement de tous les objets et acteurs impliqués dans un cas d'utilisation. On peut indiquer un type de message particulier : les retours de fonction qui, bien entendu, ne concernent aucun message mais signifient la fin de l'appel de l'objet appelé. Ils permettent d'indiquer la libération de l'objet appelant (ou de l'acteur). Un emploi abusif de retours de fonction peut alourdir considérablement le diagramme, aussi un usage parcimonieux est-il conseillé.

On peut faire apparaître de nombreuses informations de contrôle le long de la ligne

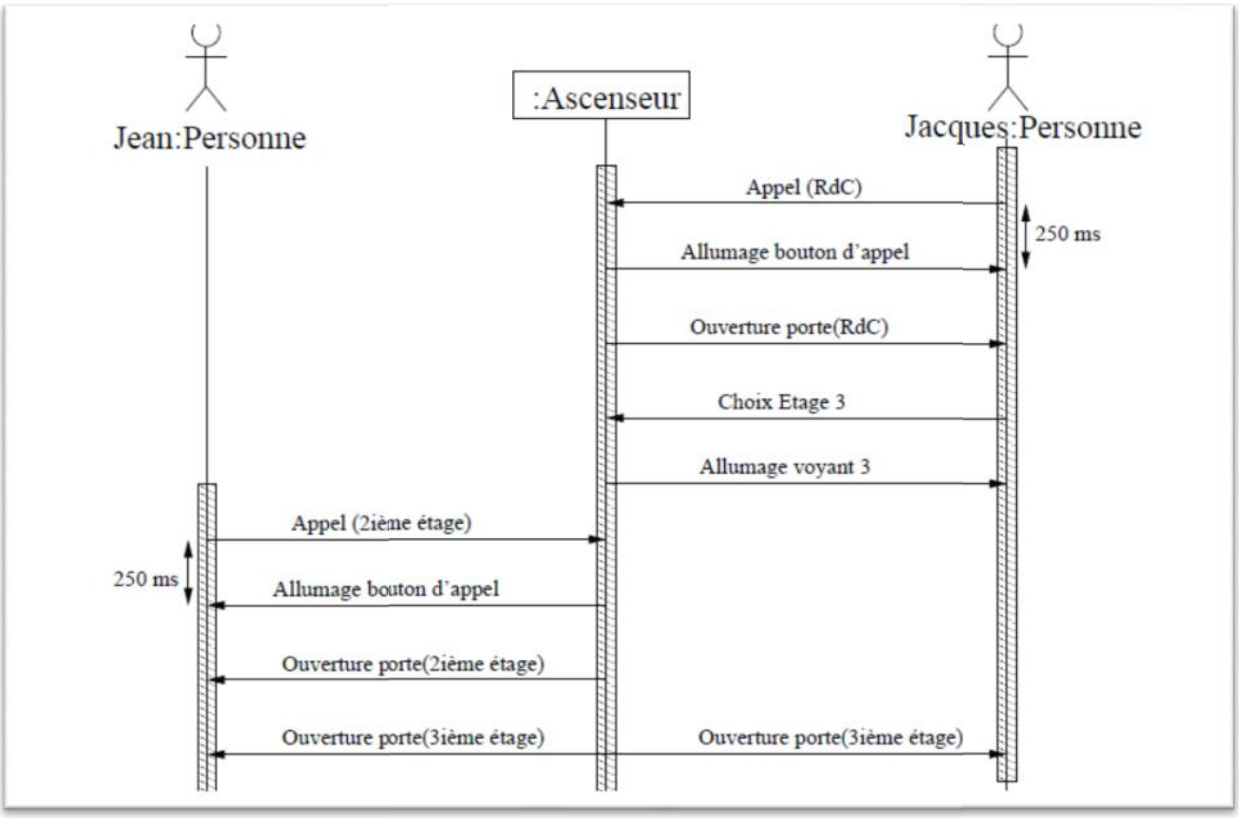

FIG.2.10. Exemple de diagramme de Séquences

de vie d'un objet. Par exemple, sur la figure (....), on a fait apparaître le délai de 250 millisecondes entre le moment où l'utilisateur appuie sur un bouton et le moment où le voyant correspondant s'allume. Deux notions, liées au contrôle des interactions s'avèrent utiles :

– la première est la condition qui indique quand un message doit être envoyé. Le message ne sera transmis que si la condition est vérifiée. On indique les conditions entre crochets au-dessus de l'arc du message .

– la seconde est la façon de marquer la répétitivité d'un envoi de message. Par exemple, si l'on doit répéter un appel pour toute une collection d'objets (pour tous les éléments de la liste des demandes), on fera précéder le dénominateur du message par un « * ».

Un diagramme des séquences permet de vérifier que tous les acteurs, les classes, les associations et les opérations ont bien été identifiés dans les diagrammes de cas et de classes. Il constitue par ailleurs une spécification utile pour le codage d'un algorithme ou la conception d'un automate. Le diagramme de séquence de conception ci-dessous permet de voir un exemple dans lequel la signature des méthodes est à peu près formalisée.

6 Les diagrammes d'états (vue dynamique)

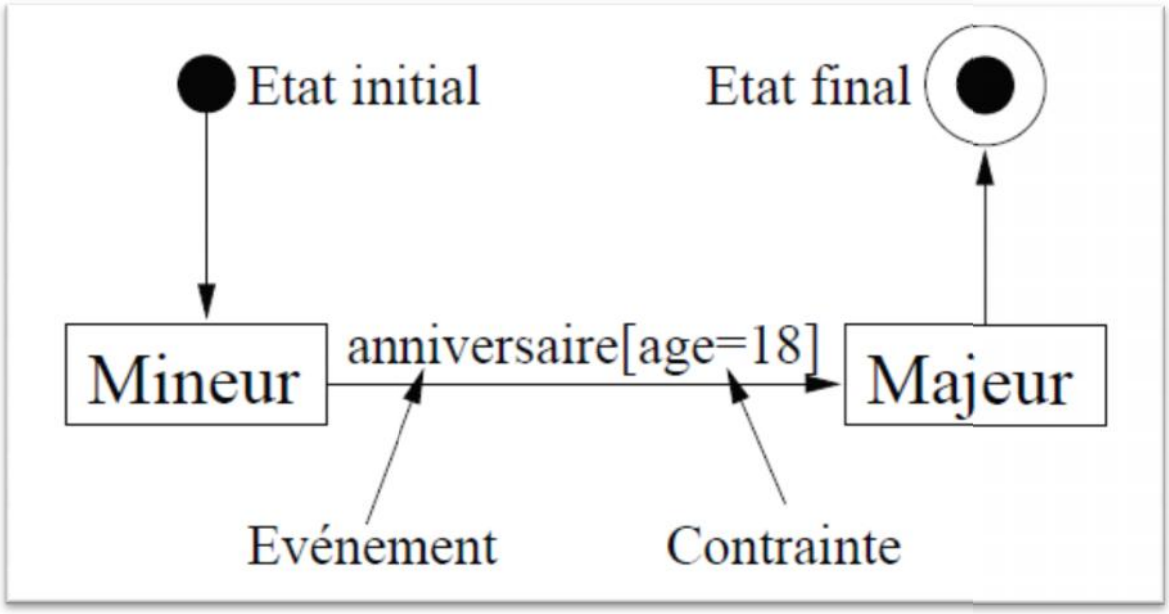

FIG.2.11. Exemple de diagramme d'états

Les diagrammes d'états décrivent tous les états possibles d'un objet (vu comme une machine à états). Ils indiquent en quoi ses changements d'états sont induits par des événements. Les modèles orientés objets s'appuient la plupart du temps sur les Statecharts de David Harel [18]. C'est aussi le cas d'UML.

Si les diagrammes de séquences regroupaient tous les objets impliqués dans un unique cas d'utilisation, les diagrammes d'états indiquent tous les changements d'états d'un seul objet à travers l'ensemble des cas d'utilisation dans lequel il est impliqué. C'est donc une vue synthétique du fonctionnement dynamique d'un objet.

Les diagrammes d'états identifient pour une classe donnée le comportement d'un objet tout au long de son cycle de vie (de la naissance ou état initial, symbolisée par le disque plein noir, à la mort ou état final, disque noir couronné de blanc).

Conclusion

Il faut retenir de ce chapitre qu'UML fournit un moyen visuel standard pour spécifier, concevoir et documenter les applications orientées objets, en collectant ce qui se faisait de mieux dans les démarches méthodologiques préexistantes. En fin de compte, l'intérêt de la normalisation d'un langage de modélisation tel que UML réside dans sa stabilité et son indépendance vis-à-vis de tout fournisseur d'outil logiciel.

Quand au langage lui-même, il a été conçu pour couvrir tous les aspects de l'analyse et de la conception d'un logiciel, en favorisant une démarche souple fondée sur les interactions entre les différentes vues que l'on peut avoir d'une application. Il permet enfin de fournir directement une bonne partie de la documentation de réalisation, dans le cours même des processus d'analyse et de conception.

Finalement, les bénéfices que l'on retire d'UML sont multiples.

– D'une part, le langage, tel qu'il a été conçu, incite l'adoption d'une démarche de modélisation souple et itérative qui permet de converger efficacement vers une bonne analyse et une bonne conception.

– D'autre part, parce qu'il est formalisé (c'est-à-dire que sa sémantique est clairement spécifiée et décrite, on parle de langage semi-formel) et standard, le langage est supporté par différents outils, qui peuvent rendre des services dans la transition des différentes vues au code et du code aux différentes vues. Dans le premier cas, les outils deviennent capables de générer des squelettes de code. Dans le second cas, il s'agit d'engendrer des vues UML à partir du code, c'est le reverse engineering 5.

– Enfin, on peut imaginer à termes des outils de détection automatiques d'incohérence entre les différentes vues. Certains de ces aspects sont opérationnels (la détection d'interdépendances entre packages, la propagation des modification d'une classe sur différentes vues, ...).

Chapitre III

Implémentation et Résultats expérimentaux

I : Introduction

Dans ce chapitre nous montrons l'implémentation d'un éditeur web UML avec multicouche de conception. L'approche se distingue par l'utilisation d'un diagramme de classe.

I-1- Implémentation du système :

L'objectif de cette partie est d'offrir l'implémentation de notre approche en basant sur les algorithmes que nous avons utiliser et cela dons le but de valider les résultats **.**

Ce chapitre commence par décrire l'environnement de développement en particulier le sous RAPID PHP, puis offre une présentation du langage de programmation JAVA SCRIPT , HTML , CSS.

I-1-1- Environnement :

L'environnement de travail regroupe l'environnement matériel et celui de développement.

I-1-1-1- Environnement matériels :

Nous avons utilisé un ordinateur qui possède les caractéristiques suivantes :

- Processeur : Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz 2.30 GHz
- Disque dur : 1000 Go
- RAM : 6 Go
- Carte graphique : AMD Radeon HD 8600M Series
  Intel(R) HD Graphics Family

I-1-1-2- Environnement de développement RAPID PHP :

Est un éditeur PHP qui intègre de nombreuses fonctions telles que Auto complète, vérificateur de syntaxe, débogueur et de nombreux autres outils pour le développement rapide de PHP.

RAPID PHP Editor contient également d'autres outils de développement pour aider sur HTML, CSS, JAVASCRIPT et beaucoup d'autres langues.

Fait partie d'une famille de produits couvrant la plupart des aspects du développement web moderne intégrant ainsi de nombreuses autres fonctionnalités utilisées par les développeurs.

Certaines fonctionnalités:

- (X) HTML vers HTML5
- CSS vers CSS3
- Intelligence de code
- Recherche et remplacement puissants
- Prise en charge de plusieurs cadres
- Code d'embellissement
- FTP Explorer (FTP / SFTP / FTPS)
- Explorateur de fichiers
- Explorateur de base de données
- Extraits de code
- Validateurs et débogueurs
- RAPIDE, réel, rapide
- Beaucoup d'autres outils disponibles

***Figure 3.1* :** fenêtre de démarrage de RAPID PHP

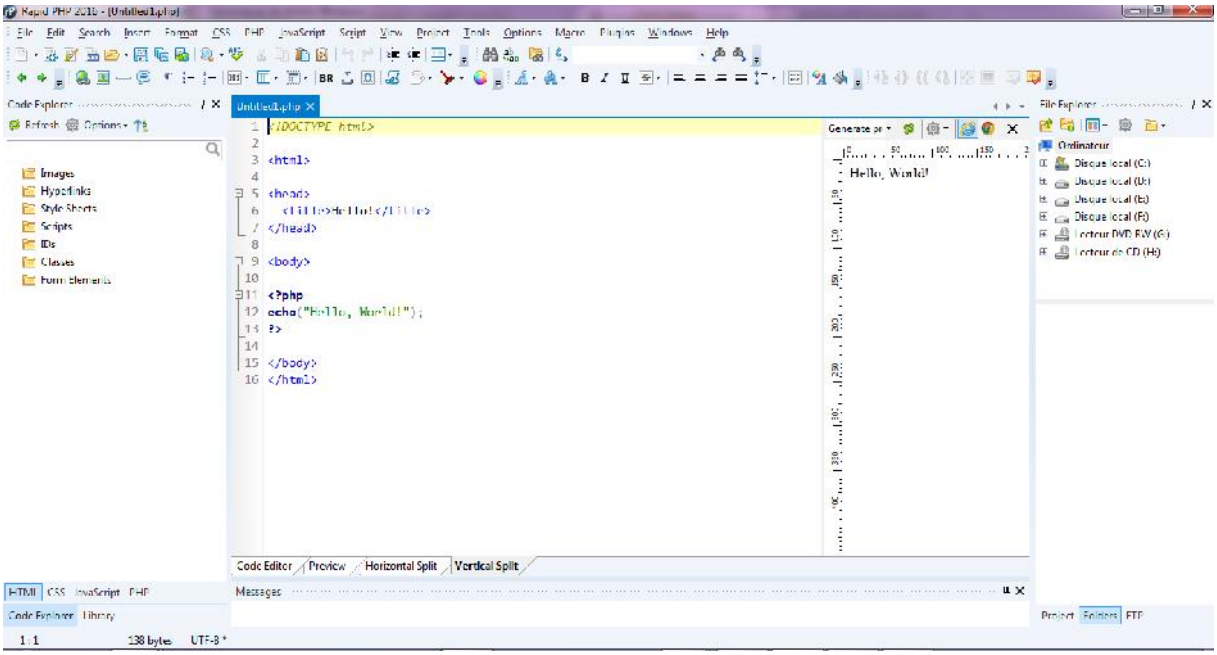

Figure 3.2 : Représente fenêtre RAPID PHP plateforme.

I.2.RAPID PHP comme éditeur HTML

Rapid PHP est un éditeur complet HTML et XHTML avec toutes les fonctions essentielles, telles que HTML Auto Complete, Inspector et la documentation intégrée

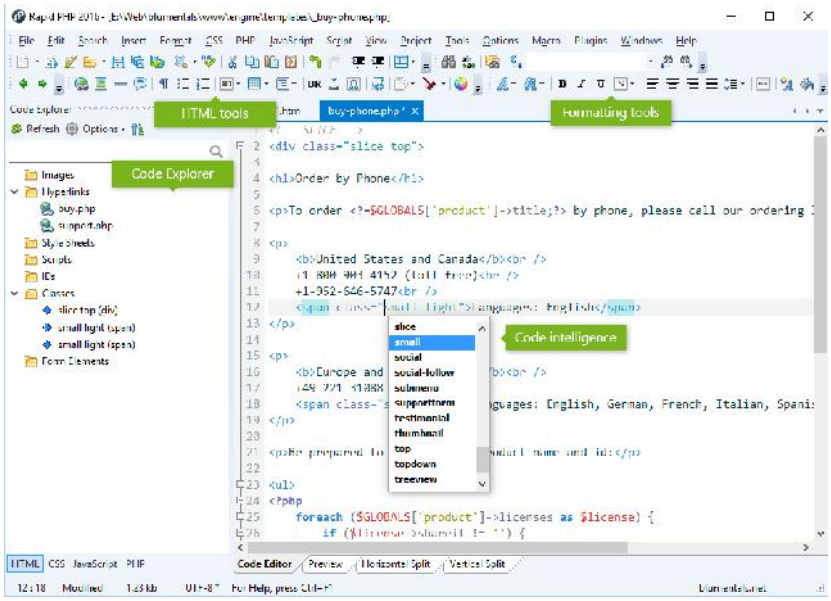

Figure 3.3 : Représente fenêtre RAPID PHP comme éditeur HTML.

I.3.RAPID PHP comme éditeur CSS

RAPID PHP est également un éditeur CSS complet avec toutes les fonctions essentielles telles que CSS Auto Complete, Inspecter et la documentation intégrée.

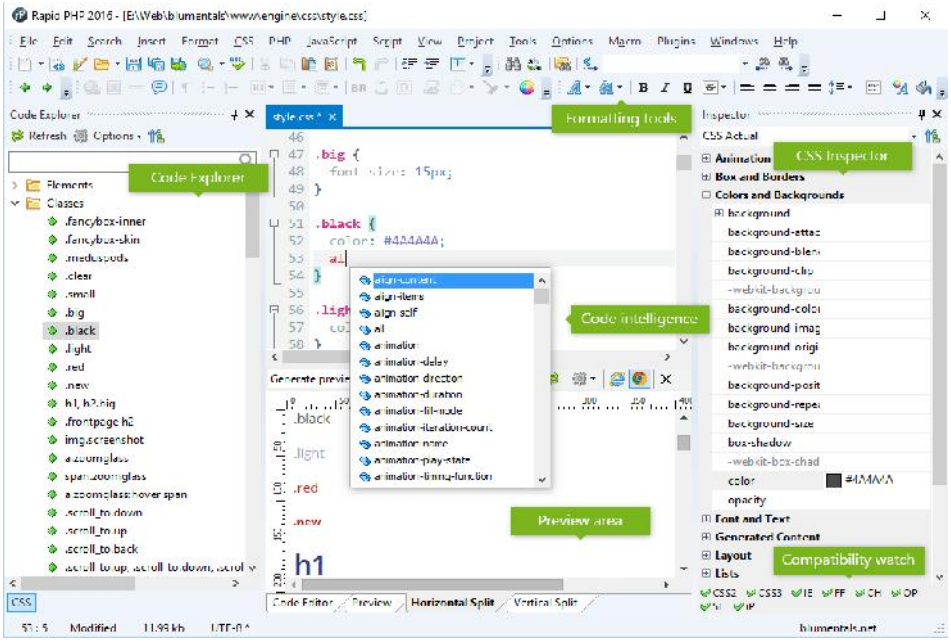

Figure 3.4 : Représente fenêtre RAPID PHP comme éditeur CSS.

I.4.RAPID PHP comme éditeur JAVA SCRIPT

RAPID PHP comprend un éditeur de JavaScript à grande échelle avec des outils essentiels, tels que Auto Complete et Code Explorer.

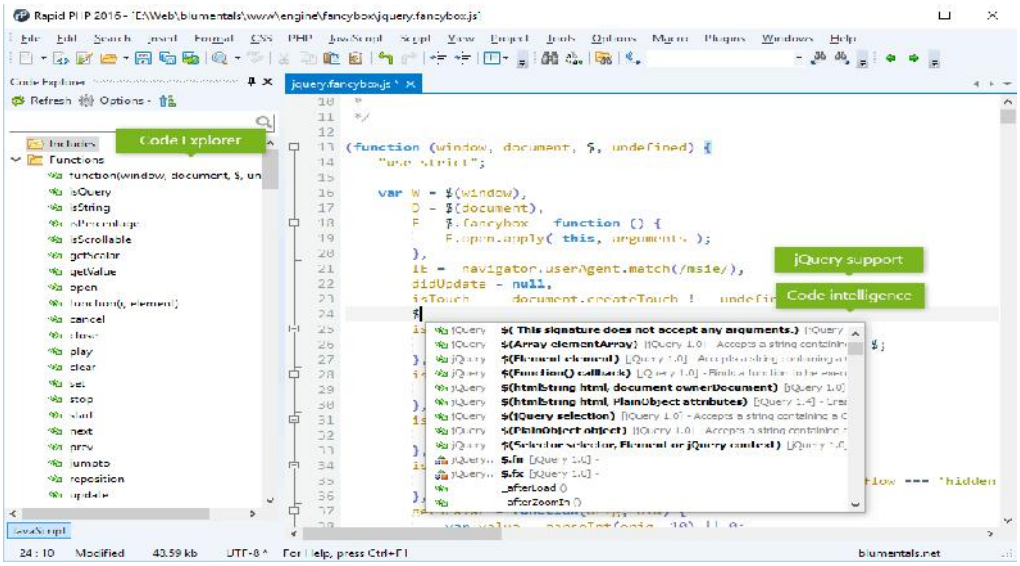

Figure 3.5 : Représente fenêtre RAPID PHP comme éditeur JAVASCRIPT.

I-4-1- Présentation de JAVA SCRIPT

JAVA SCRIPT est un langage de programmation développé par Netscape en 1995 sous le nom de Live Script c'est un langage orienté objet .

***Les caractéristiques du JavaScript**

- C'est un script.

- C'est un langage orienté objet, comme je vous l'ai déjà dit.

- Le code n'est pas compilé :

il est donc plus rapide à produire (pas besoin de compilateur, un seul fichier, ...),

mais il est moins puissant qu'un programme en C, par exemple, et relativement limité : il se limite plus ou moins à la page web sur laquelle il se trouve. Il ne permet donc pas de faire des choses comme manipuler des fichiers sur votre disque dur seulement des choses assez simples.

Il est déterminé par une norme, nommée ECMA-262 ou ECMAScript. De la même manière que le W3C se charge de définir clairement le (x)HTML, le JS possède une norme qui fixe également des lois et des limites pour celui-ci, rendant ainsi ce langage plus "officiel". Ainsi, le code est plus facile à écrire, car il y a beaucoup moins de problèmes de compatibilité. Il y a cependant pour le JS quelques différences d'un navigateur à l'autre (des fonctions de l'un qui ne marchent pas sur l'autre, ...), mais cela n'a pas beaucoup de conséquences.

I-4-2- Présentation de HTML(HyperText Markup Language)

L'HyperText Markup Language, généralement abrégé HTML, est le format de données conçu pour représenter les pages web. C'est un langage de balisage permettant d'écrire de l'hypertexte, d'où son nom. HTML permet également de structurer sémantiquement et logiquement et de mettre en forme le contenu des pages, d'inclure des ressources multimédias dont des images, des formulaires de saisie, et des programmes informatiques. Il permet de créer des documents interopérables avec des équipements très variés de manière conforme aux exigences de l'accessibilité du web. Il est souvent utilisé conjointement avec le langage de programmation JavaScript et des feuilles de style en cascade (CSS). HTML est initialement dérivé du Standard Generalized Markup Language (SGML).

I-4-3- Présentation de CSS(Cascading Style Sheets)

Les feuilles de style en cascade, généralement appelées CSS de l'anglais Cascading Style Sheets, forment un langage informatique qui décrit la présentation des documents HTML et XML. Les standards définissant CSS sont publiés par le World Wide Web Consortium (W3C). Introduit au milieu des années 1990, CSS devient couramment utilisé dans la conception de sites web et bien pris en charge par les navigateurs web dans les années 2000.

I-5- Structure du système :

Nous détaillons ici la structure de données fondamentales permettant de réaliser notre système. Cette structure doit couvrir les éléments mentionnés dans notre modélisation qui sont :

I-5-1-La Bibliothèque JointJS :

La bibliothèque de diagramme de JointJS vous permet de créer des outils de diagramme entièrement interactifs pour tous les navigateurs modernes. JointJS n'est pas seulement une bibliothèque de dessin, mais son architecture MVC (plus MV) sépare les graphiques, les éléments et les modèles de liens de leur rendu. Cela facilite la connexion de JointJS à votre application .

I-5-2-La Bibliothèque AngularJS :

AngularJS est un framework JavaScript libre et open-source développé par Google.

AngularJS est fondé sur l'extension du langage HTML par de nouvelles balises et attributs pour aboutir à une définition déclarative des pages web, par opposition à l'utilisation systématique de l'élément div et à la définition des éléments de présentation en JavaScript.

Le code HTML étendu représente alors la partie « vue » du patron d'architecture MVC (modèle-vue-contrôleur) auquel AngularJS correspond, via des modèles en couche appelés « scopes » et des contrôleurs permettant de prototyper des actions en code JavaScript natif. AngularJS utilise une boucle de dirty-checking (qui consiste à surveiller et à détecter des modifications sur un objet JavaScript) pour réaliser un data-binding bidirectionnel permettant la synchronisation automatique des modèles et des vues.

AngularJS embarque un sous-ensemble de la bibliothèque open source jQuery appelé jQLite, mais peut aussi utiliser jQuery si elle est chargée.

I-5-3-Interface graphique :

Contient le menu avec Barre d'outils dotés des options et pour naviguer avec les fonctionnalités du projet

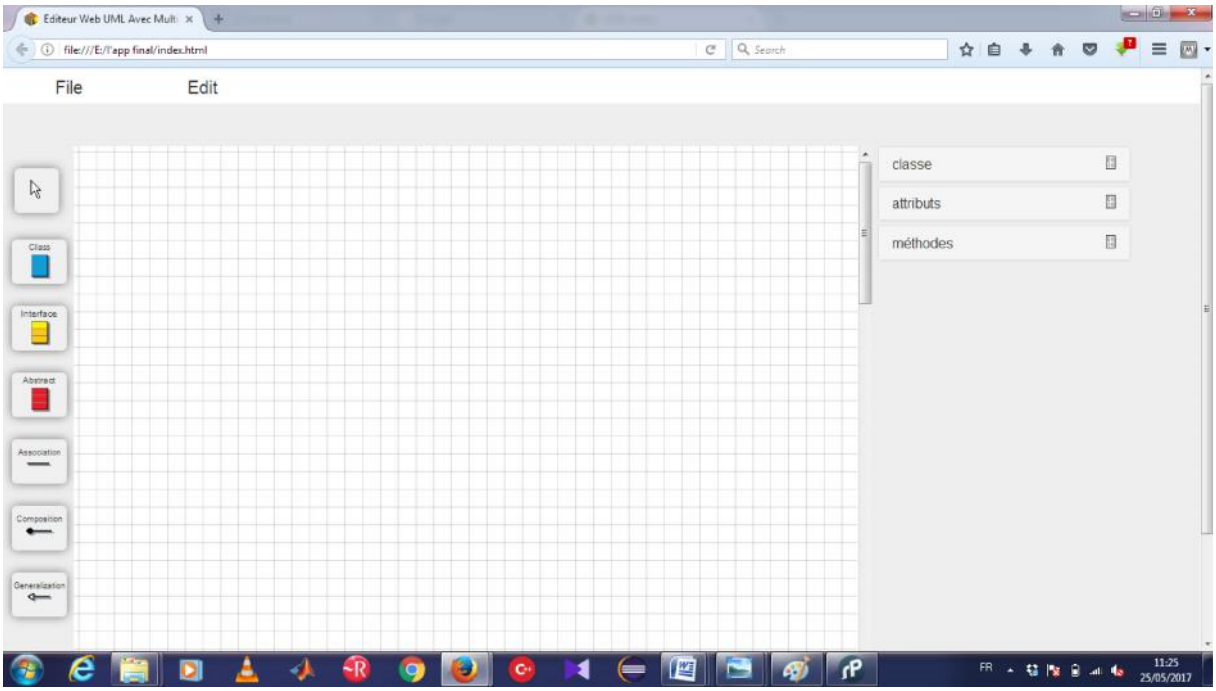

Figure 3.6 : **Interface graphique du projet**.

I-5-4-Barre d'outils :

la barre d'outils est un élément de base de l'interfaces graphiques qui regroupe plusieurs boutons. Il s'agit donc d'une rangée d'icônes regroupées en un bloc qui nous aider a retirées ou ajoutées de l'interface graphique.

Figure 3.7 : **Barre d'outils.**

I-5-5-l'accordions :

L'accordions est un élément de contrôle graphique comprenant une liste d'éléments verticalement empilés, il comprend trois liste:

classe : nommée les classes, contrôles la hauteur et la largeur des classes

attributs: ajouter ou supprimé des attributs

méthodes : ajouter ou supprimé des méthodes

Figure 3.8 : **l'accordions.**

I-5-6-Paper:

La zone d'affichage ou on peut dessiner les diagramme , c'est une zone zoomable l'orsque on utiliser le 3éme bouton de la sourie (Mouse Wheel) on obtient une zone qui contient le code source.

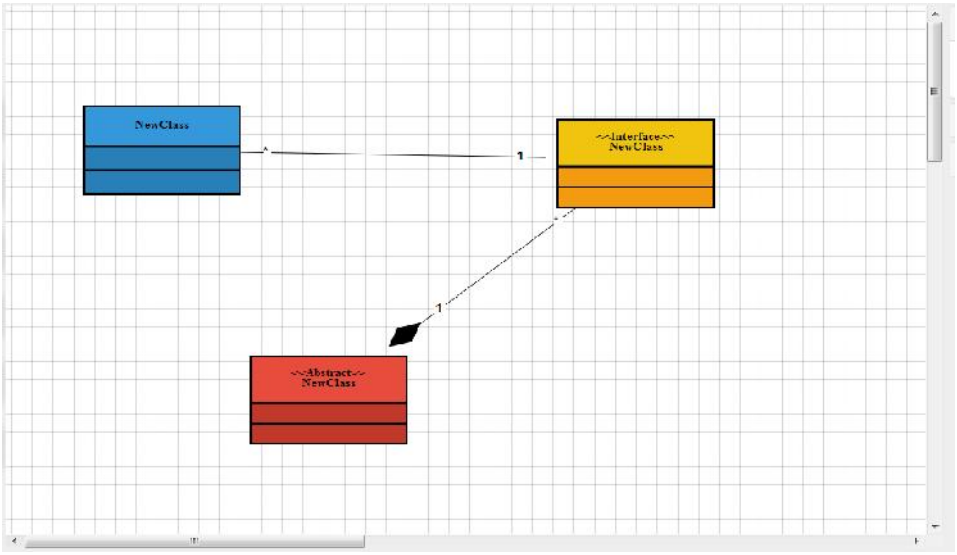

Figure 3.9 : **Paper (Zone D'affichage).**

I-5-7-Diagramme de classe:

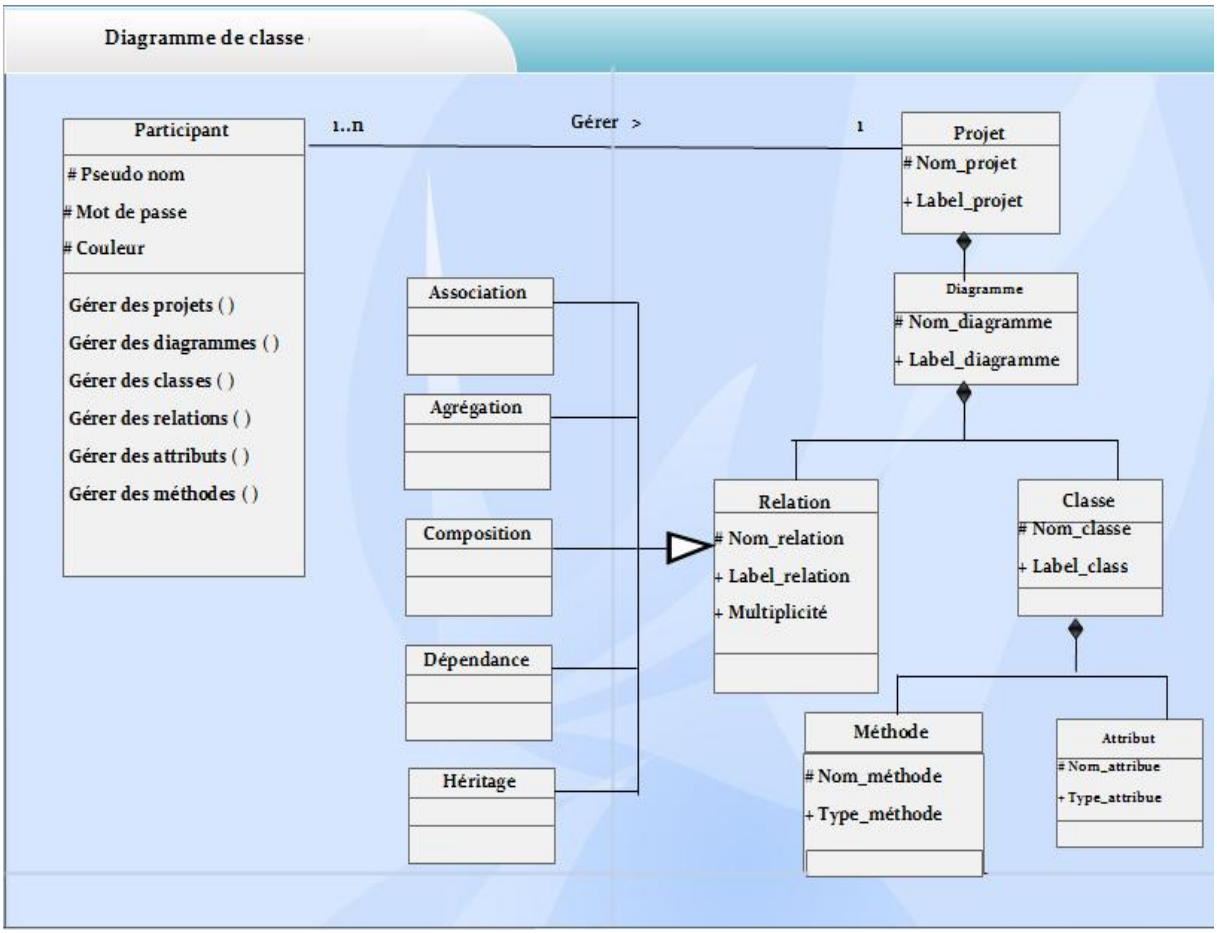

Figure 3.10 : **Diagramme de classe.**

Conclusion :

L’objectif de notre projet de fin d’étude était de concevoir et implémenter une application d’un éditeur web uml avec une multicouches de conception.

Le point de départ de la réalisation de ce projet nous avons rappelé les bases du Web et a ensuite parlé du concept des éditeurs web et de l'importance de la multicouches et une signification particulière le zoom.

Par la suite, nous nous sommes intéressés à fait une étude détaillée sur l’UML .

Le dernier volet de notre projet était la partie réalisation qui a été consacrée à la présentation des outils du travail et les interfaces les plus significatives de notre application. L’apport de ce travail a été d’une importance très considérable, en effet, il nous a permis : de suivre une méthodologie de travail bien étudié, d’approfondir nos connaissances dans le monde de développement des applications et de nous bien nous exercer sur l'uml et sa conception multicouche.

Pour conclure, on peut dire que nous avons obtenu notre objectif visé au début, qui est la réalisation d'un éditeur web UML pour les diagrammes de classe.

Notre travail restera toujours ouvert aux critiques et suggestions pouvant le réajuster, surtout aux améliorations pouvant lui être amenées par des études ultérieures.

Notre souhait ultime est que notre travail apportera une aide aux utilisateurs de ce domaine et que cette étude soit un modèle pour d'autres.

[1] Peter Bright. "Hands-on with Windows 8: A PC operating system for the tablet age". Ars Technica.

[2] Sketchpad: A man-machine graphical communication system

[3] Dataland: the MIT's '70s media room concept that influenced the Mac

[4] Piccolo (formerly Jazz): ZUI toolkit for Java and C# (no longer actively maintained)

[5] Piccolo2D: Piccolo's successor.

[6] "Sigma lenses: focus-context transitions combining space, time and translucence", Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, 2008

[7] Popular Mechanics 2007. Retrieved November 11, 2011. Glen Derene. Wii 2.0: Loop remote lets you click by gesture.

[8]http://www.apple.com/ios/ios7/

[9] : Object Solutions - Managing the Object-Oriented Project. G.Booch. Pearson Education 1996

[10]: Introduction au Rational Unified Process, Ph.Krachten . Eyrolles 2000

[11]: Use Case Driven Object Modeling with UML. D.Rosenberg, K.Scott. Wesley Longman

[12]: The Unified Software Development Process. G.Booch, J.Rumbaugh,.Jacobson. Addison Wesley 1999

[13]: Concevoir des applications WEB avec UML. J.Conallen. Eyrolles 2000

[14]: Ben Shneiderman. – Direct manipulation: a step beyond programming languages. IEEE Computer, aout 1983, pp. 57–69.

[15]: Alain Karsenty. – GroupDesign : un collecticiel synchrone pour l'edition partagee de documents. – The`se, LRI, Universite´ Paris-Sud, Orsay,

1994.

[16]: Danny Goodman. – The Complete HyperCard Handbook. – Bantam,

1987.

[17]: Eric A. Bier, Maureen C. Stone, Ken Fishkin, William Buxton et

Thomas Baudel. – A taxonomy of see-through tools. Proceedings of ACM

CHI'94 Conference on Human Factors in Computing Systems, pp. 358–364.

– 1994.

url1 - http ://www.csioo.com/cetusfr/software.html: carrefour Cetus : Orienté Objets ; excellent, plus de 8000 liens.

[18] url3 - http ://www.rational.com/UML/resources.html: site de RATIONAL (principal acteur de UML)

[19] J. Rumbaugh. Modélisation et conception orientées objets. Masson,

France, 1996.

[20] G. Booch. Object-Oriented Software Analysis and Design with Applications. Benjamin Cummings, 1994.

[21] I. Jacobson,M. Christerson, P. Jonsson, and G. Overgaard. Object-Oriented Software Engineering : A Use case Driven Approach. Addison Wesley, 1992.

[22] O. Sigaud. Introduction à la conduite de projet reposant sur un langage orienté objets. support du cours « Génie logiciel et programmation orientée objets » de l'ENSTA, 2005.

[23] I. Jacobson, G. Booch, and J. Rumbaugh. The Objectory Software Development Process. AddisonWesley, 1997.

[24]. D. Harel. Statecharts : A visual formalism for complex systems. Science of Computer Programming, 8, 1987.