République **A**lgérienne **D**émocratique et **P**opulaire
Ministère de l'**E**nseignement **S**upérieur et de la **R**echerche **S**cientifique
Université **A**bderrahmane **M**IRA-**B**ejaia

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Faculté Des Sciences Exactes

Département d'Informatique

Mémoire de fin de cycle

En vue de l'obtention du diplôme de Master en informatique

Spécialité : Génie Logiciel

Présenté par :

CHELOUAH Yanis

BOUSSAA Farouk

Thème

VERS UN ENVIRONNEMENT DE GÉNIE LOGICIEL CENTRÉ PROCÉDÉ BASÉ SUR UML

Devant le jury composé de :

Mr. Mustafa SADI	**M.C.B**	**President**
Mr. Khaled SELLAMI	**M.C**	**Rapporteur**
Mme Nadia HALFOUNE	**M.A.A**	**Examinatrice**

Session normale 2021

Remerciments

Nous tenons à remercier notre encadreur SELLAMI Khaled, enseignant à l'université de Bejaia pour nous avoir fait l'honneur de diriger notre travail.

Nous tenons à remercier aussi Mr SAADi Mustafa et Mme Halfoune Nadia pour avoir accepté d'évaluer notre mémoire

Dédicaces

Je dédie ce travail :

A mes très chers parents, qui m'ont aidé à surmonter toutes les difficultés et qui m'ont soutenu toute au long de ma vie.

A mon précieux oncle, qui a été toujours mon grand soutien.

A mes sœurs Yasmine et Melissa que j'aime énormément.

A la femme de mon oncle, tante Nadia, qui m'a aidée dans les moments difficiles.

Aux enfants de mon oncle Sarah, Ilyas et Arris que j'aime beaucoup.

A ma grande mère, que dieu la garde pour moi.

A toutes les personnes qui m'ont soutenu.

Farouk

Je dédie ce travail :

A mes chers parents.

A mon frère et ma sœur.

A mes oncles et tantes.

A mes très chers grands-parents décédés.

A mon très cher oncle décédé.

A mon ami Hafid Bensadi.

A notre promoteur Khaled Sellami qui nous a grandement aidé.

A mes amis.

Qui m'ont soutenu

Yanis

TABLE DES MATIERES

CHAPITRE I : INTRODUCTION GENERALE 10
I.1. Introduction générale 11
I.2. Problématique 11
I.3. Contexte 11
I.4. Organisation du mémoire 12
CHAPITRE II : ÉTUDE DES ATELIERS GENIE LOGICIEL CENTRÉS PROCÉDÉS 13
II.1. Introduction 14
II.2. Procédés logiciels 14
II.2.1. Modèles de cycle de vie 14
II.2.2. Environnements intégrés de génie logiciel 15
II.2.3. Ateliers de génie logiciel centrés procédés (AGLCP) 16
II.2.4. Ateliers de Génie Logiciel ou Outils CASE : 16
II.2.4.1. Origines 16
II.2.4.2. Définitions 17
II.2.4.3. Objectifs et Avantages des ateliers CASE 17
II.2.4.4. Constituants d'un atelier de génie logiciel 18
II.2.4.5. Intégration d'outils CASE 18
II.2.4.6. Catégories d'AGL 19
II.3. Éléments du modèle de procédés logiciels 19
II.4. Modèle du procédé logiciel 22
II.4.1. Différents formalismes 22
II.4.2. Formalisation des procédés 23
II.4.3. Différents domaines du procédé logiciel 24
II.4.4. Exemple d'Ateliers (Environnements) de Génie Logiciel Centrés Procédés 26
II.5. Conclusion 27
CHAPITRE III : NOTATION UML 28
III.1. Introduction 29
III.2. Notation UML 29
III.2.1. Définition 29
III.2.2. Un peu d'histoire 29
III.2.3. Utilisation 30
III.2.4. Modèle et méthode 30

III.2.5. Points de vue de la modélisation 31
III.2.5.1 Point de vue fonctionnel 31
III.2.5.2 Point de vue statique 31
III.2.5.3 Point de vue dynamique 31
III.2.6. Quelques outils logiciel de modélisation UML 32
III.2.6.1 ArgoUML 32
III.2.6.2 BOUML 33
III.2.6.3 StarUML 33
III.2.6.4 Visual Paradigm 34
III.2.6.5 Eclipse Papyrus 35
III.2.7. Types de diagrammes UML 36
III.2.7.1. Diagramme de cas d'utilisation 36
III.2.7.1.1. Définition 36
III.2.7.1.2. Éléments d'un diagramme de cas d'utilisation 37
III.2.7.1.3. Identifier les acteurs et les cas d'utilisation 43
III.2.7.2. Diagramme de classes 43
III.3 Conclusion 50
CHAPITRE IV : CONCEPTION 51
IV.1. Introduction 52
IV.2. Architecture de l'environnement de développement 52
IV.1.2. Acteurs et entités de l'environnement 54
IV.3. Conceptualisation 54
IV.3.1. Conception de l'outil de modélisation UML 54
IV.3.2 Conception de l'application Web dynamique 56
IV.3.2.1 Modèle relationnel de la base de données 58
IV.4 Conclusion 59
CHAPITRE V : MISE EN ŒUVRE 60
V.1. Introduction 61
V.2. Outils et langages de développement de l'application UML 61
V.2.1. Langage de programmation Java 62
V.2.2. Environnement de développement intégré Netbeans 62
V.2.3. Bibliothèques JGraphx et Swing 63
V.3. Outils et langages de développement de l'application Web 63
V.3.1. Langage HTML 63
V.3.2. Langage CSS 63

V.3.3. Langage JavaScript 64
V.3.4. Langage PHP 64
V.3.5. Langage de programmation XML 64
V.3.6. Le SGBD MySQL 64
V.3.7. Apache 65
V.3.8. Bibliothèques JQuery, AJAX et Bootstrap 65
V.3.9. Environnement de développement intégré Adobe DreamWeaver 2019 66
V.3.9.1 Adobe Dreamweaver 66
V.4. Outil de modélisation UML 66
V.4.1. Structure des objets 66
V.4.1.1. Objet classe 66
V.4.2. Interface utilisateur de l'outil 67
V.4.2.1. Quelques exemples d'utilisation 68
V.4.2.3 Ouverture d'un modèle 74
V.5. Interface Web 74
V.5.1 Inscription 74
V.5.2. Profil concepteur de modèle 76
V.5.2.1 Gestion des modèles 79
V.5.2.2 Gestion des projets 82
V.5.3 Chef de projet 82
V.5.3.1 Inscrire un membre à un projet 82
V.5.4 Membre 82
V.6 Conclusion 87
Conclusion générale et perspectives 88
Bibliographie 89
Webographie 90
Résumé 91
Abstract 91
الملخص 91

TABLE DES FIGURES

Figure II.1 : Modèle en cascade 15
Figure II.2 : Éléments basiques du modèle de procédé 21
Figure II.3 : Domaines du procédé 25
Figure III.1. Les trois axes de modélisation 31
Figure III.2 : Interface graphique de l'outil ArgoUML 32
Figure III.3 : Interface graphique de l'outil BOUML. 33
Figure III.4 : Interface graphique de l'outil StarUML 34
Figure III.5 : Interface graphique de l'outil Visual paradigm. 35
Figure III.6 : Interface graphique de l'outil Eclipse Papyrus. 36
Figure III.7 : Objet graphique d'un cas d'utilisation 38
Figure III.8 : Exemple d'un diagramme avec une note 38
Figure III.9 : Exemple de relation d'association 39
Figure III.10 : Exemple d'une relation de généralisation entre les cas d'utilisation 40
Figure III.11 : Exemple d'une relation de généralisation entre les acteurs 40
Figure III.12 : Exemple de relation d'inclusion entre les cas d'utilisation 41
Figure III.13 : Exemple de relation d'extension entre les cas d'utilisation 42
Figure III.14 : Exemple d'un package 42
Figure III.15 : Exemple d'un diagramme de classes. 43
Figure III.16 : Exemple de représentation graphique d'une classe 45
Figure III.17 : Exemple de relation d'association entre deux classes 45
Figure III.18 : Relation d'association avec une multiplicité 46
Figure III.19 : Relation d'agrégation 46
Figure III.20 : Relation de composition 47
Figure III.21 : Relation de généralisation 48
Figure III.22 : Exemple de classe d'association 49
Figure III.23 : Association réflexive 49
Figure III.24 : Regroupement des classes avec un package 50
Figure IV.1. Architecture de l'environnement de développement 53
Figure IV.2. Actions d'un concepteur de modèle 55

Figure IV.3 Accès du responsable informatique 56
Figure IV.4. Accès d'un chef de projet 57
Figure IV.5. Accès membre 58
Figure V.1 : Langages et outils de développement utilisés 61
Figure V.2 : Bibliothèques utilisées 62
Figure V.3 : Code source de génération d'un élément graphique (classe) avec JGraphx. 67
Figure V.4 : Style utilisé pour mettre en forme un élément graphique (classe). 67
Figure V.5 : Fenêtre principale de l'outil UML 68
Figure V.6 : Fenêtre de sélection du type de modèle (diagramme) à créer. 69
Figure V.7 : Fenêtre d'un nouveau modèle. 69
Figure V.8 : Insertion d'un objet dans la zone de dessin. 70
Figure V.9 Exemple de connexion des objets. 70
Figure V.10 : Exemple d'un modèle de procédé logiciel 72
Figure V.11 : Code source de la fonction d'enregistrement d'un modèle. 73
Figure V.12 : Code source d'ouverture d'un modèle. 74
Figure V.13. Fenêtre d'inscription des utilisateurs 75
Figure V.14 : Fenêtre de connexion d'un utilisateur. 77
Figure V.15 : Page de profil d'un concepteur de modèle. 78
Figure V.16 : Fenêtre de gestion des modèles. 79
Figure V.17 : Boite d'authentification d'un utilisateur dans l'outil de modélisation UML 80
Figure V.18 : Mise à jour du numéro de version à partir de l'outil de modélisation. 81
Figure V.19 : Page de gestion des projets. 83
Figure V.20 : Page pour un chef de projet. 84
Figure V.21 : Page de gestion des membres 85
Figure V.22 : Formulaire d'inscription d'un membre. 86
Figure V.23 : Page Accès membre 86

CHAPITRE I : INTRODUCTION GENERALE

I.1. Introduction générale

Alors que le matériel informatique a fait des progrès très rapides, le logiciel, l'autre ingrédient de l'informatique, traverse toujours une crise qui dure depuis la fin des années 60 du siècle passé. Cette crise peut se percevoir à travers des symptômes tels que : le coût de développement d'un logiciel qui est généralement très élevé avec un délai de livraison rarement respecté ; la qualité du produit livré ne satisfait pas souvent les besoins de l'utilisateur ; la maintenance du logiciel est difficile, coûteuse et souvent à l'origine de nouvelles erreurs …etc. La raison de fond de la crise du logiciel réside dans le fait qu'il est beaucoup plus difficile de créer des logiciels que le suggère notre intuition.

Pour maîtriser la complexité des systèmes logiciels, il convient de procéder selon une démarche bien définie, de se baser sur des principes et méthodes, et d'utiliser des outils performants. Pour réaliser un produit, il faut suivre un procédé qui décrit la suite des actions et opérations à entreprendre pour le développement et la maintenance du logiciel. Les actions et opérations utilisent des méthodes, techniques et outils. Pour évaluer un procédé, ou un produit, on effectue des mesures.

A partir des années 90 à nos jours, les ateliers de génie logiciel (AGL) sont apparus comme un moyen plus efficace pour apporter une solution réelle à certains problèmes du génie logiciel et contribuent nettement à l'amélioration de la productivité et de la qualité du logiciel, notamment en faisant le suivi des différentes phases du processus logiciel et en offrant un cadre cohérent et uniforme de production.

I.2. Problématique

Actuellement les procédés logiciels mis en œuvre sont de plus en plus complexes et difficiles à gérer ; la raison principale est l'évolution rapide des activités sociales et économiques pendant ces dernières décennies, et à la diffusion de la technologie de l'information dans beaucoup de domaines [10]. En conséquence, la conception, l'exécution et la maintenance des modèles pour ces procédés sont de plus en plus complexes. Nous nous intéressons plus particulièrement aux procédés logiciels et à leur modélisation pour faire face à la complexité des procédés.

La méthode utilisée dans la conception d'un modèle de procédé logiciel, est considérée comme son axe principal, car Les activités qui s'en suivent sont liées directement à cette étape de conception. Donc le problème de développement se résume à la difficulté d'adopter une méthode d'analyse et de conception globale qui permet de concilier les différents aspects qu'implique tout projet informatique de grand d'envergure.

Dans le domaine des procédés logiciels plusieurs méthodes ou approches ont été utilisée, pour la modélisation de ces procédés logiciels. Chacune se porte bien pour un type de problème spécifique, parmi ces méthodes on peut citer : UML, les automates d'états finis, les réseaux de pétri, Etc.

I.3. Contexte

Nous avons choisi l'UML comme un outil (méthode) de modélisation, pour la conception de ce modèle de procédé logiciel. Parceque elle permet de vulgariser les aspects liés à la conception et à l'architecture, propres au logiciel, au client. Aussi, elle apporte une compréhension rapide du programme à d'autres développeurs externes en cas de reprise du logiciel et facilite sa maintenance.

UML est outillé par des éditeurs logiciels dans des ateliers de génie logiciel (AGL) qui permettent, en plus de la modélisation, de générer le squelette du code source de certaines parties du système informatique, ce qui ajoute de l'intérêt à UML. Certains de ces ateliers permettent aussi d'effectuer la rétroconception d'un système déjà réalisé : à partir du code, construction du modèle UML. Enfin, UML permet la documentation du système. Cette documentation est utilisée pour faciliter les échanges entre les différents intervenants dans toutes les phases du processus de développement et de maintenance du système informatique.

Notre travaille a pour objectif :

- La conception d'un modèle de procédé logiciel avec le langage UML, puis la réalisation d'un outil logiciel qui supporte cette conception.

- Le développement d'une interface web permettant d'accéder aux modèles de procédés logiciel à des fins de portabilité et modification et d'amélioration (via l'outil UML), ainsi que de gestion et de control de projet.

I.4. Organisation du mémoire

Ce mémoire est structuré comme suit :

- **Chapitre I :** Une introduction qui fait un parcours général du contexte dans lequel s'inscrit notre travail.

- **Chapitre II :** Dans un premier temps il présente les ateliers de génie logiciel en citant l'état de l'art de quelques approches de développement des ateliers de génie logiciel et l'assistance au processus logiciel. Ensuite, décrit d'une manière générale les procédés logiciels et leurs modèles.

- **Chapitre III :** Présente une vue d'ensemble sur la notation UML. Avec quelque outil de modélisation simulant l'UML.

- **Chapitre IV :** Dans ce chapitre on décrit notre approche conceptuelle.

- **Chapitre V :** Décrit la mise en œuvre de notre outil graphique ainsi que de l'interface web dynamique.

On termine notre travail avec une conclusion générale qui résume le contenu de notre mémoire et des perspectives à réaliser à l'avenir.

CHAPITRE II : ÉTUDE DES ATELIERS GENIE LOGICIEL CENTRÉS PROCÉDÉS

II.1. Introduction

Avec l'avènement des outils et d'environnements de génie logiciel, les technologies du procédé logiciel ont connu des bouleversements remarquables ces dernières décennies.

Bien avant, le phénomène : crise du logiciel, précipita les choses en introduisant la discipline du génie logiciel [1], l'objectif était d'améliorer la fabrication du logiciel que ce soit en termes de coût, de qualité, de délai, de productivité et de tous les paramètres entrant en jeu dans la chaîne de production des logiciels, ce qui conduit l'apparition du concept environnement de génie logiciel.

(Sommerville 88) définit un environnement de génie logiciel comme étant un ensemble intégré d'outils et de mécanismes permettant de supporter toutes les phases de développement du logiciel (analyse, conception, écriture de code, test…etc.) [1].

Afin de pourvoir à une demande toujours croissante, la recherche se rapportant au domaine des environnements de développement constitue un axe important, et une multitude de produits et de techniques sont apparus dans les dernières années.

Parmi ces travaux on trouve les modèles de cycle de vie du logiciel, les environnements intégrés de génie logiciel, et les ateliers (environnements) centrés procédés logiciel.

II.2. Procédés logiciels

Un procédé logiciel ou processus logiciel est une méthode qui permet de développer un logiciel, cette méthode se focalise sur l'organisation, le contrôle et les mesures à prendre pour transformer les besoins de l'utilisateur en un logiciel fonctionnel y compris les activités nécessaires à la maintenance de ce logiciel [2]. On parle aussi des procédés par analogie avec des procédés de fabrication.

Un procédé implique des personnes, des technologies et un ensemble d'artefacts (activités, étapes, tâches, acteurs participants, outils) dont la sémantique est propre au domaine.

D'après Osterweil (1987), l'idée de processus (ou procédé) est liée au fait que l'humain résout les problèmes auxquels il est confronté en créant des descriptions de processus appelés modèles de processus. Il décline, pour chaque problème particulier un processus particulier à partir d'une description qui se veut générique : il s'agit d'une instance. Puis, chaque instance peut être exécutée [3].

II.2.1. Modèles de cycle de vie

Pour développer un logiciel, des étapes sont nécessaires pour s'organiser, en partant de la *conception* à la *maintenance* c'est ce que l'on appelle un « modèle de cycle de vie ».

Tous les modèles de cycle de vie ne sont pas les mêmes, mais l'idée est toujours la même: le logiciel est développé en phases discrètes, chacune ayant un résultat défini et un critère de terminaison défini, et chaque phase est achevée avant que ne commence la suivante. Le modèle classique de cycle de vie comprend les phases suivantes : analyse, conception, implémentation, test, et éventuellement phase d'installation. D'autres noms sont parfois donnés à ces phases et un découpage plus fin est souvent utile.

Ils existent plusieurs modèles qui ont été proposés, ces modèles ont permis d'identifier les activités d'un projet et de décrire leurs interactions, parmi les modèles, on peut citer :

1. Modèle en cascade.
2. Modèle en spirale.
3. Modèle en V.
4. Modèle itératif et incrémental.

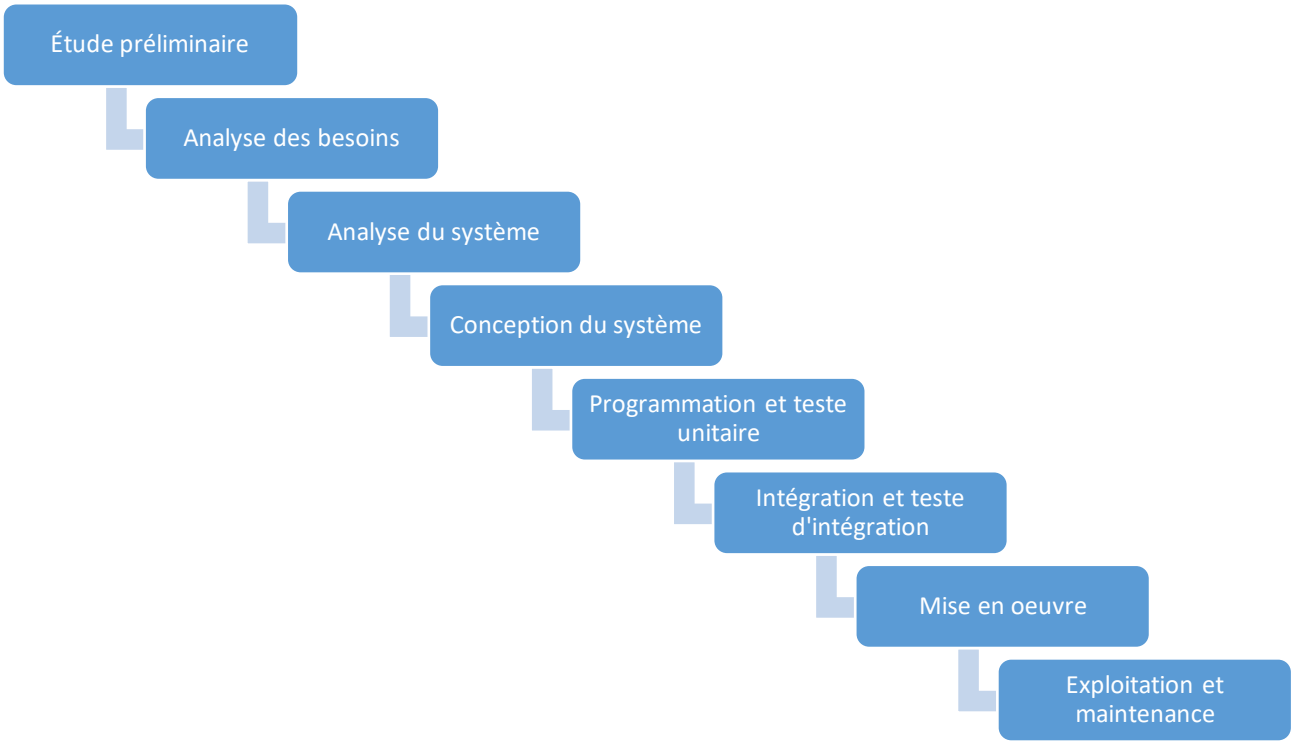


Figure II.1 : Modèle en cascade

Le modèle représenté dans la figure ci-dessus a un principe très simple. Chaque phase du modèle ne se termine que par la production de certains documents ou logiciels. Les résultats d'une phase sont obtenus sur la base des interactions entre ses étapes, ils sont soumis à une revue approfondie et on ne passe à la phase suivante que lorsqu'ils sont jugés satisfaisants. Le modèle original ne comportait pas la possibilité de retour en arrière. Celle-ci a été rajoutée ultérieurement sur la base qu'une étape ne remet en cause que l'étape précédente, cependant, il s'est avéré que ce retour reste insuffisant dans la pratique.

II.2.2. Environnements intégrés de génie logiciel

Pour développer un logiciel, il nous faut définir un cadre de production dans lequel le logiciel ne serait plus conçu de façon artisanale mais bien de manière industrielle. La fabrication du logiciel doit être de plus en plus automatisée ainsi que de nombreux outils d'aide à la réalisation d'une tâche particulière du cycle de vie du logiciel ou spécifique à un domaine

existant, tels que les outils d'aide à l'analyse, à la conception ...etc. En plus, d'autres outils non liés spécifiquement à une phase de projet, comme les outils de gestion de projet ou d'aide à la documentation et à la validation, doivent être aussi automatisés. Parmi ces outils, on peut citer (un compilateur, un éditeur de liens, un gestionnaire de fichiers, …).

Les approches présentés dans la section précédente révèlent une limitation majeure: soit ils proposent une méthodologie rigide (modèles de cycle de vie), soit ils intègrent des outils, des technologies logicielles sans méthodologie précise.

L'aspect intégré permettait d'utiliser cette collection d'outils dans la perspective de produire des logiciels avec davantage de facilités. Pour remédier aux limitations des environnements intégrés et dans le but de prendre en compte le processus logiciel, la deuxième génération d'environnements sera présentée dans la section consacrée aux Environnements de Génie Logiciel Centrés Processus (AGLCP).

II.2.3. Ateliers de génie logiciel centrés procédés (AGLCP)

Pour remédier aux limitations des environnements intégrés, et dans le but de prendre en compte le processus logiciel, de nombreux travaux sont consacrés au développement d'une deuxième génération d'environnements, connus sous le nom Ateliers (Environnements) de Génie Logiciel Centrés Procédés (AGLCP).

L'un des objectifs majeurs des AGLCP est justement de fournir aux utilisateurs à la fois certaines des technologies de production du logiciel tout en intégrant une partie de la gestion de cette dernière dans un environnement relativement intuitif. La plupart des AGLCP développés offre un langage de modélisation pour la définition du processus (que l'on appelle Langage de Modélisation de Processus) et un environnement d'exécution pour interpréter les modèles définis à partir de ce langage. Ce langage offre des concepts pour décrire les rôles, les humains, les activités, les produits manipulés, les contraintes, etc.

Ces environnements constituent la voie privilégiée de la recherche sur les procédés du génie logiciel ces dernières décennies, de nombreux travaux se sont focalisés sur les AGLCP : Marvel, Oz, Peace, Peace+ , SPADE Tempo, APPE, etc.

II.2.4. Ateliers de Génie Logiciel ou Outils CASE :

II.2.4.1. Origines

Le développement d'applications logicielles à travers un environnement supportant tout leur cycle de vie (spécification des besoins, conception, mise au point, test et maintenance) était le rêve des informaticiens depuis longue durée, et au cours des années 80, les Ateliers de Génie Logiciel (AGL) sont apparus comme solution à cet effet.

Les premiers outils AGL visant les étapes de mis en œuvre et de test logiciel ont fait la tradition depuis le début, tandis que ceux visant les premières phases de spécification des besoins et de conception ne font émerger qu'à partir la fin des années 80.

Un autre acronyme des AGL est CASE : « Computer Aided Software Engineering », support pour les phases d'analyse et de conception (uniquement).

Il est clair que le but visé par les AGL est : l'amélioration de la productivité, du suivi, et celui de la qualité : fiabilité, maintenance, évolutivité.

Exemple d'outils de première génération : L'environnement "Unix programmer workbench" composé d'outils d'aide aux activités de programmation (yacc, lex, compilateurs, etc.). L'intégration entre ces différents outils est accomplie par le système de chaînage d'entrée sortie propre au système Unix ("pipe").

Vision actuelle :

Outils interdépendants

a. Supporter toutes les étapes du cycle de vie (outils verticaux)
   - Analyse et de conception,
   - Outils de génération de code,
   - Outils de test
b. Outils qui offrent des fonctionnalités nécessaires aux étapes de cycle de vie tels que les outils de gestion de projet, de gestion de configuration et de documentation (outils horizontaux).

II.2.4.2. Définitions

Le terme Atelier de Génie Logiciel (AGL) ou atelier CASE (Computer Aided Software Engineering) désigne un logiciel aidant à la réalisation de logiciels.

1. C'est un Outil informatique aidant à la production d'un logiciel. Il assiste la démarche de Génie Logiciel poursuivie.

2. C'est un ensemble intégré d'outils qui permet aux développeurs de logiciel de documenter et modéliser un système d'information dès la spécification initiale des besoins jusqu'au projet et son implantation ; en passant par l'application de tests de cohérence, complétude et conformité aux spécifications proposées" [4] .

Remarque :

- Les AGLs sont seulement des aides et ne permettent pas de donner une solution totale à tous les problèmes de développement du logiciel.

- Ils sont des outils de gestion pour le développement de logiciel [5].

II.2.4.3. Objectifs et Avantages des ateliers CASE

Les outils CASE ont été développés dans le but d'aider les développeurs dans la production de systèmes logiciels et de produits de haute qualité.

L'objectif principal des AGL est l'automatisation maximale de tout le processus ou d'une partie du processus de développement du logiciel. Cet objectif, limité par la réalité du terrain, demande d'impliquer une assistance aux différentes phases du cycle de vie du logiciel.

Les outils CASE constituent un moyen indispensable pour résoudre les problèmes du développement d'application et de l'entretien des logiciels. Ils changent de manière significative le temps pris par chaque phase et la distribution du coût dans le cycle de vie de logiciel. Les concepteurs du logiciel s'intéressent plus sur l'analyse et la conception. Une grande partie du code peut être générée automatiquement avec le développement des spécifications détaillées. Ces améliorations rendues possibles par l'utilisation des outils CASE montrent des réductions importantes et très bénéfiques en coûts de développement et de maintenance.

La puissance des outils CASE se situe dans leur répertoire central qui contient des descriptions de tous les composants centraux du système. Ces descriptions sont employées à toutes les étapes du cycle : la création des concepts d'entrée-sortie, la génération automatique de code …etc. Les tâches suivantes continuent à s'ajouter pour construire ce répertoire de sorte que dans la conclusion du projet il contient une description complète du système entier. C'est un dispositif puissant qui n'était pas faisable avant l'introduction des outils CASE.

En résumé, les outils CASE servent pour :

1 - Améliorer la qualité du processus en :

- Augmentant la productivité des équipes.
- Favorisant la standardisation de la production.
- Accroissant la prédictibilité des développements.
- Améliorant la visibilité des projets.
- Augmentant le confort et la créativité des développeurs.

2 - Augmenter la conformité des produits en :

- Aidant à appliquer les plans et les normes d'assurance qualité,
- Permettant de mener à bien des projets complexes et importants en volume et en taille d'équipe,
- Améliorant le travail coopératif,
- Obtenant et mesurant un niveau de qualité défini.
- Aident le développeur à créer les principaux modèles d'un système d'information.
- Vérifient que les modèles sont complets et compatibles avec d'autres modèles.
- Permettent de générer le code à partir des modèles.

II.2.4.4. Constituants d'un atelier de génie logiciel

Un atelier efficace est constitué d'un ensemble cohérent de méthodes, une base de données (Repository) qui gère l'ensemble des objets crée au cours de chacune des étapes de production, un ensemble d'outils nécessaire au développement du logiciel et un ensemble intégré de mécanismes d'interface entre chacune des phases du cycle de vie. Cependant, le doublet (méthodes, outils) est considéré comme l'ensemble des ressources nécessaires à tout AGL.

II.2.4.5. Intégration d'outils CASE

Un AGL intègre différents outils CASE, de manière à les faire coopérer de façon uniforme. Cette intégration peut (devrait) s'effectuer à trois niveaux :

- **Intégration des données :** Les outils CASE manipulent (génèrent, utilisent, transforment, ...) des données: spécification, modèle conceptuel des données, jeux de test, code, manuel utilisateur, ... Différents outils sont amenés à partager une même donnée: les tables générées par un éditeur de diagrammes sont utilisées par un SGBD, le code généré par un éditeur de texte est compilé par un compilateur, à partir d'une spécification algébrique on peut générer des jeux de test, ...

  Un AGL doit prendre en charge la communication de ces données entre les différents outils. Cette intégration peut être simplement physique: tous les outils de l'AGL utilisent un seul format de représentation des données, par exemple des fichiers UNIX, sur une même machine. Cette approche implique que tous les outils de l'AGL connaissent la structure logique (l'organisation) des fichiers qu'ils sont amenés à utiliser: il est nécessaire de normaliser la structure logique des fichiers. L'intégration des données peut se faire également au niveau logique en utilisant un système de gestion des objets qui gère automatiquement les différentes entités et leurs interrelations (cette approche nécessite la définition des différents types de données manipulées).

  Un AGL devrait également gérer la cohérence entre les différentes versions de ces données (gestion de configuration).

- **Intégration de l'interface utilisateur :** tous les outils intégrés dans l'AGL communiquent avec l'utilisateur selon un schéma uniforme, ce qui facilite leur utilisation (voir par exemple l'interface du Macintosh, X11 sous Unix ou Windows95 sous DOS).

- **Intégration des activités :** un AGL peut gérer le séquencement des appels aux différents outils intégrés, et assurer ainsi un enchainement cohérent des différentes phases du processus logiciel. Cet aspect implique que l'on dispose d'un modèle du processus de développement bien accepté.

II.2.4.6. Catégories d'AGL

Les AGL peuvent être classés selon plusieurs aspects :

- **Richesse du support :** ensemble d'outils, outils intégrés, aide à la démarche.
- **Type de problèmes :** logiciels embarqués, temps réel, "business applications", applications métiers …
- **Type de projet d'ingénierie logicielle :** développement logiciel (cf. cycle de vie), intégration de systèmes, système à base de connaissance.
- **Ampleur du projet :** complexité, nombres de participants, durée ...
- **Gestion des ressources du projet :** les considérations managériales des ressources mises en œuvre dans le projet sont-elles prises en compte ? (Planification, ordonnancement, …).
- **Phase du cycle de développement prises en compte :** conception et/ou développement.

II.3. Éléments du modèle de procédés logiciels

Divers éléments de procédé peuvent être représentés dans un modèle de procédé, nous proposons la classification d'éléments de procédé suivante :

• **Éléments primaires :** ce sont les éléments principaux qui reflètent l'essence des procédés sans prendre en compte les aspects planification et exécution de procédé. Ce sont :

- **Activité :** une activité est un ensemble d'actions à réaliser pour accomplir un objectif de développement. Une activité peut être décomposée en d'autres activités, formant ainsi plusieurs niveaux d'abstraction. Une activité peut être concurrente et coopérative, déterministe ou non déterministe.
- **Produit :** un produit est un artéfact utilisé ou élaboré par les activités durant le développement.
- **Rôle :** un rôle est un concept abstrait regroupant un ensemble de responsabilités et de compétences nécessaires pour réaliser certaines activités de développement.
- **Outil :** Les outils sont des systèmes qui assistent la production de logiciel. Il existe deux sortes d'outils : les outils interactifs (éditeurs textuels, outils graphiques …), et les outils simplement exécutables sans interaction (compilateurs, analyseur grammatical…).

• **Éléments secondaires :** ce sont les éléments qui fournissent des informations supplémentaires pour mettre en œuvre un procédé. Ce sont :

- **Agents :** un agent est une personne participant au développement. Un agent n'est pas un rôle, il peut jouer plusieurs rôles en même temps. Dans le sens inverse, un rôle n'est pas forcement associé à un agent, mais peut être associé à un groupe d'agents. Le concept d'agent est nécessaire pour la gestion de processus.
- **Ressources :** une ressource est un élément qui facilite l'exécution d'une activité, par exemple un outil, un guidage, etc. De telles informations sont utiles pour l'assistance au cours d'un processus.
- **Informations qualitatives :** elles permettent d'évaluer la performance et la qualité de procédés, par exemple des métriques, les résultats de révision ou test, etc.
- **Informations organisationnelles :** elles facilitent l'exécution d'un procédé pour un projet spécifique. Généralement, elles concernent le contexte de travail, l'organisation de ressources, de coopérations et de communications.

Les modèles de procédé ne décrivent pas seulement des éléments de procédé, mais aussi des relations entre eux. Les relations principales sont les relations entre activités et produits, entre rôles et activités et entre rôles et produits. La Figure II-2 montre les éléments de procédé et les relations les plus importants.

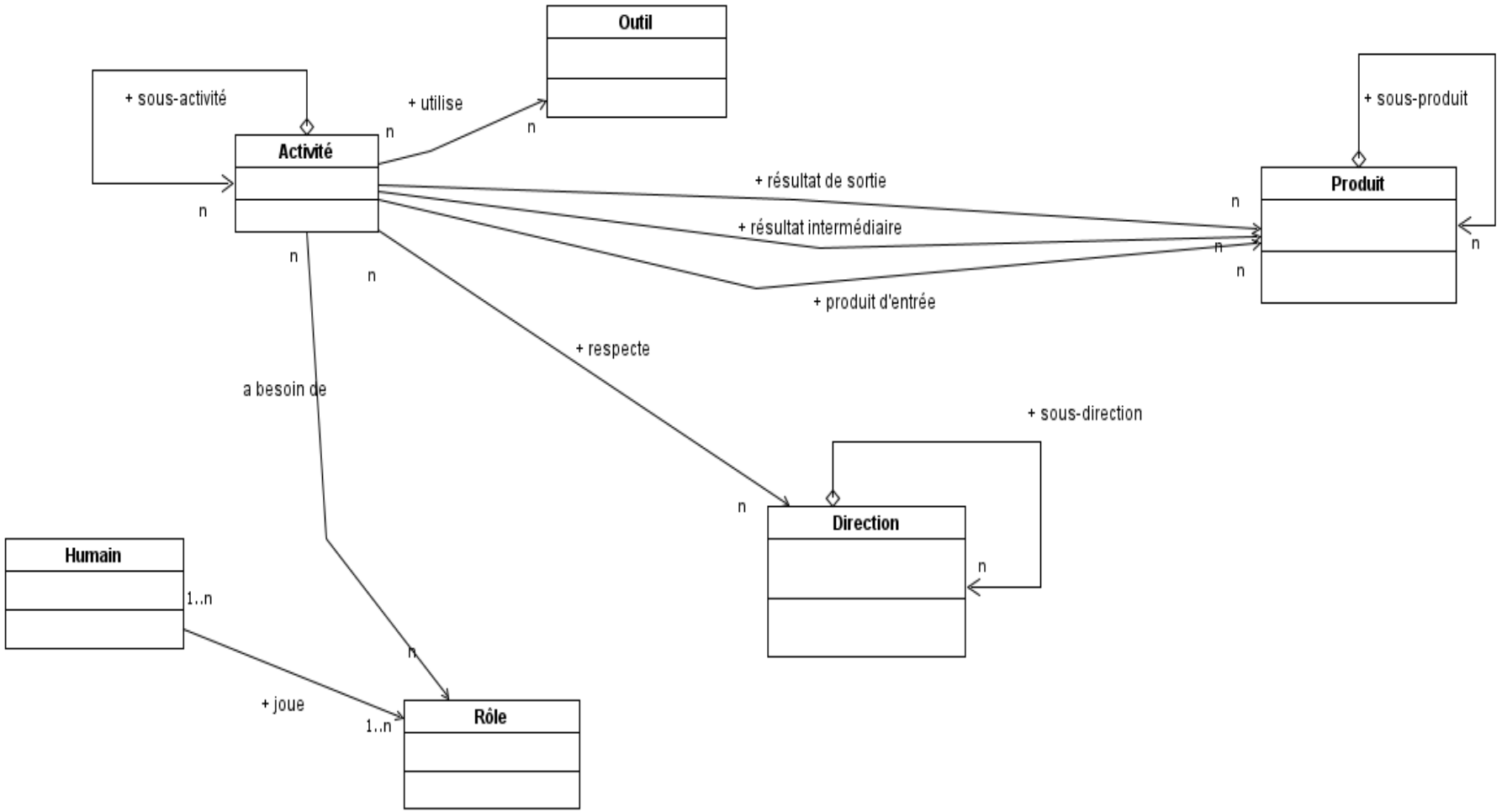


Figure II.2 : Éléments basiques du modèle de procédé

II.4. Modèle du procédé logiciel

Un modèle de procédé est une abstraction du procédé logiciel décrit de manière formelle ou semi-formelle. Il comprend :

- La description des ressources (outils, acteurs, etc.) que requiert le procédé ;
- Les activités et les tâches pour lesquelles le procédé est défini et structuré ;
- L'enchaînement (ou ordonnancement) de ces activités ou tâches ;
- Les informations nécessaires à la définition du procédé.

Le modèle de procédé fixe les fondements nécessaires à la coopération et à la communication entre les différentes entités participant au procédé.

Plusieurs sous-modèles ont été identifiés [2], qui ne font pas tous l'objet de consensus :

- **Un modèle d'activité :** exprimant les activités simples et complexes (agrégation d'activités simples) décrit à l'aide d'un formalisme particulier ;
- **Un modèle de produit :** exprimant les données qui sont manipulées par les activités ;
- **Un modèle d'outils :** pour décrire les outils utilisés dans le cadre du procédé et leurs architectures. Cela peut être réalisé également par le modèle d'activités, en considérant l'outil comme l'enveloppe d'une activité ;
- **Un modèle organisationnel :** qui décrit la structure et contrôle les activités ainsi que leurs ressources ;
- **Un modèle utilisateur :** exprimant la manière dont les différents acteurs tirent parti de l'assistance fournie au travers du support technologique du procédé, leurs responsabilités dans le contexte du procédé, etc.

II.4.1. Différents formalismes

Plusieurs approches ont été à l'origine de formalismes différents. Entre autres nous notons les langages de programmation classiques, les formalismes de modélisation comme les réseaux de Pétri, etc. Parmi les approches et les langages de modélisation de procédé qui en découlent on peut noter [6] :

Approche procédurale

L'idée fondamentale est de représenter le modèle de procédé sous la forme d'un programme. Ce programme décrit de manière détaillée comment le procédé logiciel doit être réalisé.

Approche déclarative

L'approche déclarative utilise des déclarations logiques (règles) pour décrire les procédés. Ceux-ci sont décrits en termes de résultats attendus par l'utilisateur sans détailler la manière dont ces résultats sont obtenus (de façon algorithmique).

Approche fonctionnelle

L'approche fonctionnelle définit le procédé logiciel à travers un ensemble de fonctions mathématiques. Chaque fonction est décrite en termes de relations entre les données d'entrée et les données de sortie.

Approche basée sur les graphes ou les réseaux transitionnels

Cette approche décrit le procédé logiciel à travers un réseau ou un graphe qui compose des nœuds et des arcs. Les réseaux de Petri sont souvent utilisés dans cette approche. Des nœuds représentent des activités, des arcs représentent les transitions entre les activités du procédé [7].

Approche basée sur UML

Cette approche utilise des diagrammes d'UML pour représenter les concepts de procédé et renforce la sémantique de ces diagrammes avec un autre langage formel pour rendre les modèles de procédé à la fois compréhensifs et exécutables.

II.4.2. Formalisation des procédés

On distingue trois phases principales dans le cycle de vie d'un procédé logiciel :

- **Phase de spécification des besoins :** qui consiste à identifier et/ou documenter les besoins du procédé. On pourra, par exemple, définir la performance attendue, les objectifs, etc.
- **Phase de conception, de modélisation :** qui doit définir le procédé pour répondre aux besoins exprimés lors de la phase précédente ;
- **Phase d'implantation :** du procédé tel qu'il a été définit.

Afin de répondre aux objectifs du procédé et de chacune de ces trois phases et ainsi, de couvrir le cycle de vie, des langages ont été proposés selon trois catégories que l'on retrouve dans [8] :

- **Langage de spécification :** répondant aux besoins de la phase de spécification, dont les concepts doivent avoir pour but de décrire l'organisation, les objectifs généraux, etc. ;
- **Langage de modélisation :** pour modéliser le procédé : c'est ce langage qui devra couvrir, pour l'essentiel, les concepts de base du procédé logiciel ;
- **Langage d'implantation :** dont l'objectif sera de décrire la façon dont doit être exécuté le procédé.

Il est acquis que les formalismes de haut niveau sont à privilégier afin de fournir des moyens adaptés (simples) pour la description du procédé.

En effet, le but du domaine n'est pas d'alourdir la compréhension du procédé ainsi décrit mais bien de la simplifier et de permettre aux différents acteurs (managers, développeurs, etc.) d'avoir une vision uniforme du procédé.

Idéalement, un formalisme de modélisation doit satisfaire les besoins suivants :

- Doit être exécutable.
- Doit permettre de décrire et de supporter l'ensemble du cycle de vie du procédé ainsi que tous ses niveaux d'abstraction,
- Doit prendre en compte la description dynamique de l'ordonnancement des activités du procédé,
- Doit permettre de supporter et de modéliser l'évolution des procédés et de leurs modèles,
- Doit permettre de modéliser et de gérer l'incertitude et l'incomplétude de certaines informations,
- Doit permettre d'exprimer et de supporter la communication, la coordination, la négociation et la coopération entre les divers intervenants dans le procédé.

Sachant que le but est donc d'assister le concepteur et le développeur dans la compréhension et dans la modélisation du procédé, des langages graphiques, semi-formels et de haut niveau ont été développés. C'est ainsi que les langages de modélisation tels que **OOA**/**OOD** [9, 10, 11, 12], etc. ont largement inspiré certains formalismes de description de procédé (**APPEL**, **Process Weaver**).

Ces langages viennent en complément de langages formels, indispensables pour décrire de façon rigoureuse les modèles de procédé.

II.4.3. Différents domaines du procédé logiciel

Les procédés ont ainsi pu, grâce aux AGLCP, bénéficier de supports pour leur modélisation, et leur exécution. Même si ces environnements proposent des approches différentes, l'objectif reste le même à savoir supporter des procédés logiciels. Le procédé logiciel est structuré en trois domaines distincts (voir **Figure II.3**) : le domaine de définition de modèles de procédé, le domaine d'exécution de modèles de procédés et le domaine de la mise en œuvre du procédé. Cette distinction permet de mieux comprendre le procédé logiciel.

Domaine de la définition

Contient des descriptions de procédé ou fragments de procédé exprimés dans une certaine notation définissant ce qui pourrait ou bien devrait être mis en œuvre. Une telle description du procédé est appelée modèle du procédé.

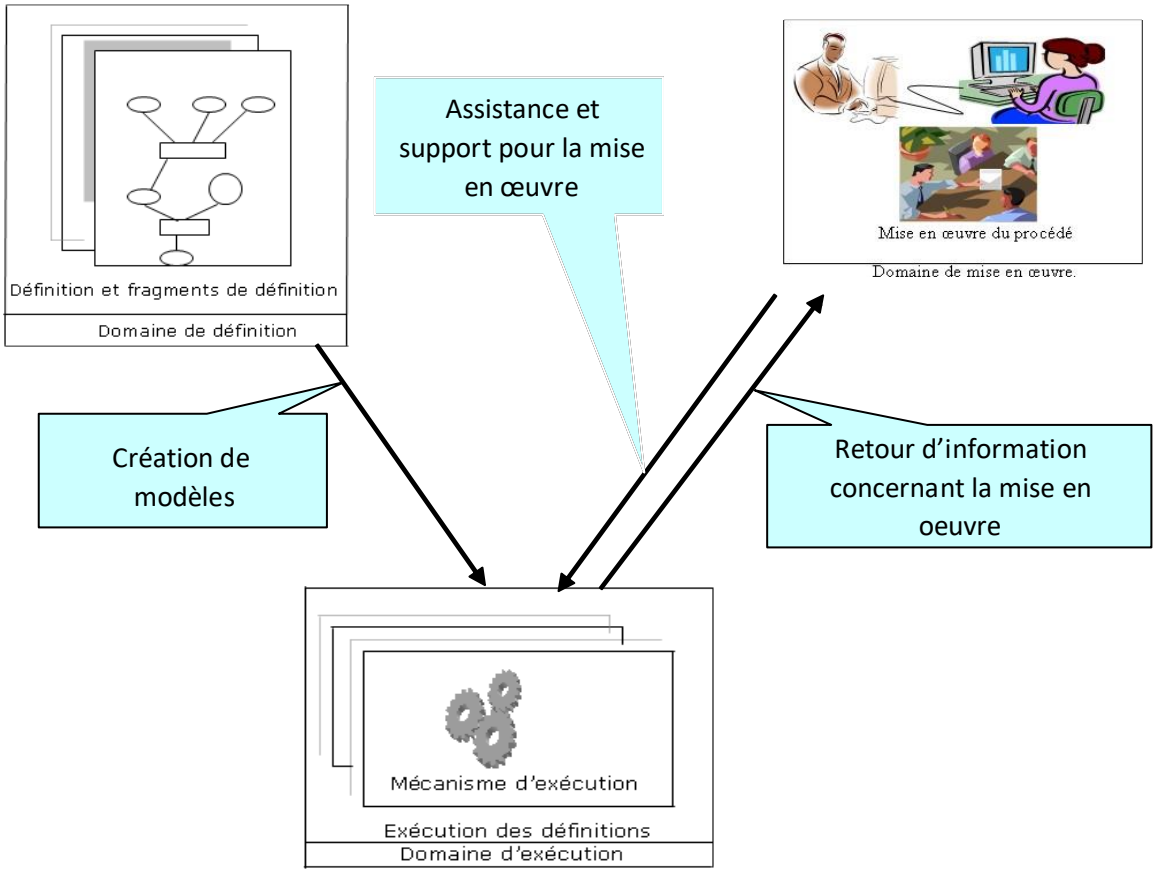


Figure II.3 : Domaines du procédé.

Dans le cas des AGLCP cette notation est "exécutable", c'est-à-dire elle peut être interprétée par un mécanisme d'exécution.

Domaine de la mise en œuvre

Englobe les activités effectives ainsi que les actions prises en charge par tous les participants, concernant aussi bien les activités de développement que celles de gestion du procédé.

Une partie de ces activités seront conduites utilisant des outils logiciels, d'autres seront complètement en dehors du support, comme par exemple des interactions entre les participants au procédé.

Domaine d'exécution

Est concerné par ce qui se passe dans un AGLCP pour supporter le procédé mis en œuvre, admettant que ce dernier est gouverné par la définition qui a été donnée ; il interprète le modèle du procédé à partir d'une représentation exécutable et englobe les mécanismes et les ressources nécessaires de l'environnement pour supporter les domaines de définition et de mise en œuvre.

Le mécanisme d'exécution utilise les définitions afin de déterminer ses interactions avec les participants impliqués dans la mise en œuvre ainsi qu'avec d'autres composants de l'environnement, le but final étant d'assurer l'aide, le guidage ou bien le contrôle de la mise en œuvre en vue d'assurer sa cohérence avec la définition du procédé qui a été donnée.

En d'autres termes, tandis que le procédé est mis en œuvre dans le monde réel (domaine de la mise en œuvre), il est exécuté par la machine dans le monde abstrait du modèle de procédé (domaine de l'exécution).

II.4.4. Exemple d'Ateliers (Environnements) de Génie Logiciel Centrés Procédés

Leu

LEU est un environnement basé sur les réseaux FUNSOFT [13]. Le procédé peut être exécuté par plusieurs moteurs de procédé. Dans ce cas, chacun des moteurs va exécuter des fragments de modèles (ou bien sous-modèles). En ce sens, le procédé est donc lui-même distribué sur plusieurs moteurs de procédé formant ainsi un environnement global.

LEU est un environnement homogène dans le sens où l'ensemble des moteurs partage les concepts et le formalisme décrivant les procédés. Il est possible d'ajouter des moteurs de procédés à l'environnement qui va pouvoir exécuter des modèles (ou fragments de modèles).

Provence

Provence [14] est un AGLCP reposant sur une approche client/serveur selon laquelle les clients peuvent s'abonner aux évènements qui sont produits pendant l'exécution du procédé. Le but de cet environnement est de reposer sur des outils existants. Il peut être vu, dans une certaine mesure, comme une généralisation de Marvel en termes d'interopérabilité et d'intégration d'outils. Cette dernière est assurée par un composant de l'architecture: le gestionnaire de mise en œuvre et son rôle consiste à faire inter opérer les outils logiciels participants à l'environnement.

Provence a été implémenté avec les composants suivants :

Marvel [15] qui gère la cohérence, la disponibilité des objets manipulés au sein de l'environnement, gère la coordination et la coopération entre les différents acteurs (personnes au sein du même processus logiciel) et permet la définition du modèle de processus en termes de règles ;

YEAST assure la liaison avec les clients et déclenche les actions correspondantes aux requêtes des utilisateurs ;

3D FILE SYSTEM fournit un mécanisme permettant aux utilisateurs de créer des vues dynamiques du logiciel et d'effectuer des changements relatifs à cette vue sans affecter la base du logiciel ;

DOTY, éditeur graphique, qui permet de créer et de manipuler les graphes de façon interactive en utilisant le multifenêtrage. L'interprétation des graphes se fait par **DOTY** et le langage qui lui est associé.

Provence pourrait être vu comme une fédération d'outils dans le sens où il repose sur une volonté de proposer un environnement de "composants inter-opérables". En ce sens, **PROVENCE** offre une architecture relativement indépendante des outils qui le composent,

contrairement à la plupart des environnements existants où les outils sont fortement imbriqués, ne pouvant être remplacés. Pourtant si les outils sont bien identifiés et ne sont pas fortement intégrés dans l'environnement, le phénomène d'ajout ou de retrait d'outils implique une forte modification de l'interface entre les composants et doit tenir compte des spécificités liées aux composants d'origine. En effet, certaines hypothèses ont été faites selon les possibilités de ces outils. Par exemple, **MARVEL** doit connaître les chemins des différents fichiers composant le projet et doit savoir quels outils utilisés sur quels objets.

D'autre part, l'architecture proposée s'appuie sur les possibilités intrinsèques de **MARVEL, YEAST, DOTY, 3D FILE SYSTEM**. Le remplacement de ces outils par d'autres ne devrait se faire qu'à la condition que les nouveaux outils disposent des mêmes fonctionnalités et répondent aux mêmes caractéristiques d'implémentation que ceux utilisés. Ces contraintes n'étant pas décrites dans l'environnement (aucune description des contraintes et des caractéristiques des composants), le respect de ces dernières, reste au bon vouloir du concepteur de l'environnement. On peut également noter que l'environnement, tel qu'il a été conçu, ne peut pas fonctionner si l'un ou l'autre des outils est absent.

II.5. Conclusion

L’art de fabriquer des logiciels n’est pas un métier facile. Pour maitriser la complexité des systèmes logiciels il convient de se baser sur des principes et méthodes et utiliser des outils performants pour supporter les activités du processus de développement. Dans ce chapitre, nous avons présenté les AGLs appelés encore outils CASE comme un moyen très efficace pour encadrer les développeurs dans tout le cycle de vie du logiciel. Nous avons abordé les AGLs en précisant d’abord leurs objectifs, leurs types et leur apport pour les développeurs pour produire un logiciel de qualité.

Et comme nous l’avons vu dans la suite de ce chapitre, le domaine des procédés logiciels est vaste et les procédés logiciels sont intrinsèquement complexes. De nombreuses recherches poursuivent leurs efforts pour caractériser et pour mieux cerner les procédés : des approches ont été proposées pour les modéliser et des environnements ont été développés pour les supporter.

CHAPITRE III : NOTATION UML

III.1. Introduction

Dans le chapitre précédant, nous avons vu les notions de bases sur le procédé logiciel et les cycles de vie d'un logiciel et les environnements intégrés du génie logiciel autrement dit les ateliers génie logiciel (AGL) et les AGL centré procédé.

Dans ce chapitre, nous allons voir la notation UML, nous présenterons ce qu'est le langage UML ainsi que certains de ses diagrammes, à savoir diagramme de cas d'utilisation et le diagramme de classes, nous allons détailler certains éléments de ces diagrammes.

Une image vaut mieux qu'un long discours. Ce proverbe résume l'origine de la modélisation, l'expression des besoins est une étape de recueil d'informations pour la création d'un logiciel, ça commence par interroger un client sur des questions comme : Que voulez-vous que le logiciel fasse pour vous ? Dans quel domaine travaillez-vous ? Comment travaillez-vous ? ...etc.

Après avoir recueilli toutes les informations du client, il faudrait transformer le discours avec le client en un ou plusieurs schémas afin de mieux comprendre et de faciliter la conception, nous avons pour cela besoin d'un langage de modélisation à l'exemple du langage UML.

III.2. Notation UML

III.2.1. Définition

UML (*Unified Modeling Language* en anglais) ou langage unifié pour la modélisation, est une notation graphique destinée à créer des modèles orientés objets[1] (diagrammes) en vue d'analyse et de modélisation de logiciels orientés objets. Ce n'est pas un langage de programmation, c'est un ensemble d'outils permettant de la modélisation de la future application informatique.

III.2.2. Un peu d'histoire

Au début des années 90, une cinquantaine de méthodes objets ont vu le jour. Ce qui est un signe de l'intérêt du sujet, et également de confusion. Toutes ces méthodes utilisaient à peu près les mêmes concepts de classes d'associations, de partitions en sous-systèmes .

En octobre 1994, *Grady Booch*et *James Rumbaugh* fondèrent la Rational Software Corporation pour unir leurs efforts en vue de créer une norme industrielle unique à partir de leurs méthodes. C'est ainsi qu'est né la méthode unifié (*Unified Method)* en octobre 1995. Ils sont ensuite rejoints par *Ivar Jacobson*, l'inventeur des cas d'utilisation (*Use cases)*, et ont publié UML 0.9 en juin 1996, puis UML 1.0 en janvier 1997.

La version 1.1, mise au point avec d'autres partenaires, devient un standard en novembre 1997 lorsqu'elle est accepté par l'OMG[2]. La poursuite du développement d'UML fut ensuite

[1]Concept basé sur une collection d'objets.
[2]Object Management Group est une organisation de standardisation international à but non lucratif, son but est de standardiser les modèles objets.

intégralement remise aux mains de l'OMG. En juillet 1998, l'OMG publiait UML 1.2, puis UML 1.3 en juin 1999. En mai 2002, UML 1.4, qui présentait de petites améliorations et quelques extensions, voyait le jour. UML 1.5 a été publié en mars 2003 avec également quelques correctifs.

UML 2.0, publié en 2005, présente quant à lui de profondes modifications et de véritables extensions par rapport aux versions 1.X, comme par exemple le diagramme d'activité et le diagramme de séquences [16].

III.2.3. Utilisation

UML facilite la conception des documents nécessaires au développement d'un logiciel orienté objet, comme standard de modélisation de l'architecture logicielle. Les différents éléments représentables sont :

- Activité d'un objet/logiciel
- Acteurs
- Processus
- Schéma de base de données
- Composants logiciels
- Réutilisation de composants.

Il est également possible de générer automatiquement tout ou partie du code, par exemple en langage Java, à partir des modèles réalisés.

III.2.4. Modèle et méthode

Un modèle est une représentation simplifiée de la réalité. Il permet de capturer les éléments les plus importants pour répondre à un objectif défini.

Dans le cas du logiciel, UML est de plus en plus le langage le plus utilisé pour la modélisation à l'heure actuelle. Il possède une sémantique propre et une syntaxe composée de graphique et de texte et peut prendre plusieurs formes (diagrammes).

Une **méthode** de développement défini à la fois un langage de modélisation et l'étape à suivre lors de la conception. Le langage UML propose uniquement une notation dont l'interprétation est définie par un standard, mais pas une méthodologie complète.

III.2.5. Points de vue de la modélisation

Il existe trois points de vue de modélisation présentés comme suite :

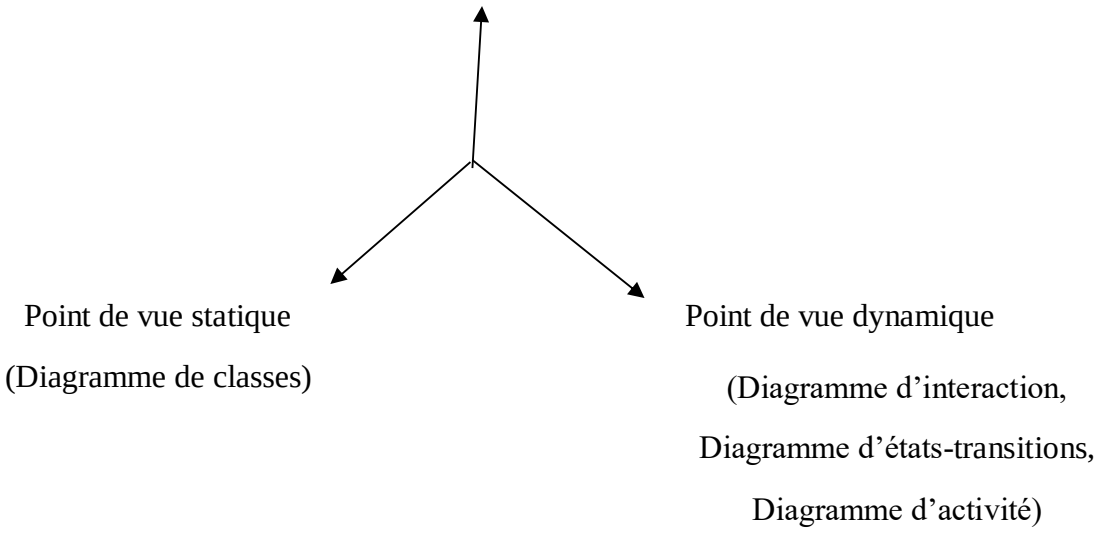


Figure III.1. Les trois axes de modélisation

III.2.5.1 Point de vue fonctionnel

Le point de vue fonctionnel modélise ce que le système doit faire pour les acteurs (les utilisateurs humains ou les autres systèmes). En d'autres termes, il permet de dire précisément ce qui entre et ce qui sort du système, mais en revanche, il ne montre pas comment réaliser cette interface avec l'extérieur. Le diagramme UML utilisé lors de cette phase est le **diagramme de cas d'utilisation** car il recueille, analyse et organise les besoins.

III.2.5.2 Point de vue statique

Le point de vue statique se fait grâce au **diagramme de classe.** Ce diagramme est considéré comme le plus important de la modélisation objet. Il montre la structure interne d'un système. Il contient principalement des classes. Une classe contient des attributs et des opérations (*partie qui sera détaillée dans la partie « Diagramme de classes »).*

III.2.5.3 Point de vue dynamique

Le point de vue dynamique est le pont entre le point de vue fonctionnel et statique, il est établi par les **diagrammes d'interaction :** Ils montrent comment les instances au cœur du système communiquent pour réaliser une certaine fonctionnalité. Les interactions sont variées et nombreuses. Il faut un langage riche pour les exprimer. UML propose plusieurs diagrammes :

diagramme de séquence, diagramme de communication et diagramme de timing.[3]Ils apportent un **aspect dynamique** à la modélisation.

III.2.6. Quelques outils logiciel de modélisation UML

Il existe plusieurs outils logiciel basées modélisations UML, voici quelques-uns :

III.2.6.1 ArgoUML

ArgoUML est un logiciel libre (gratuit) développé en langage Java[4]. Il supporte sept types de diagrammes : cas d'utilisation, classes, séquence, état, collaboration, activité et déploiement. Il est multilingue et multiplateforme. Il supporte la génération de code et l'ingénierie inverse[5].

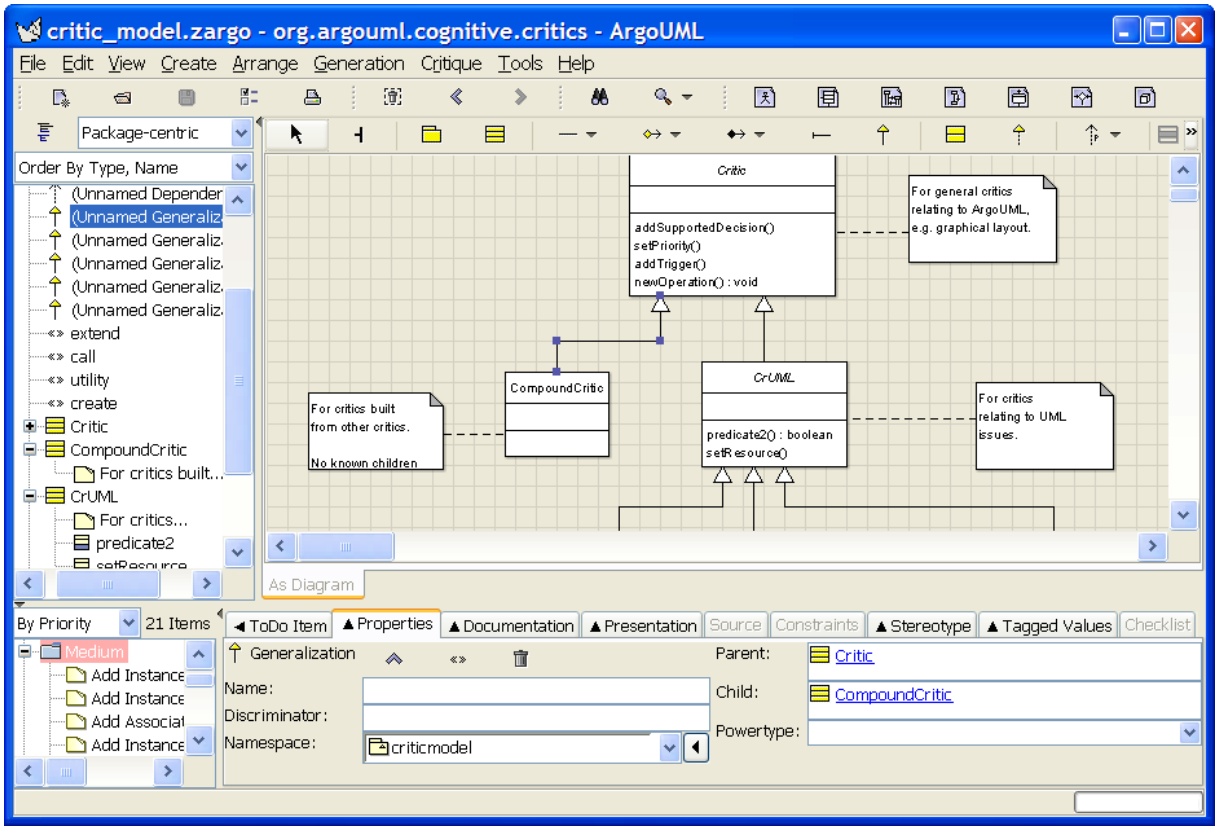


Figure III.2 : Interface graphique de l'outil ArgoUML

[3] Les trois diagrammes ci-dessous ne seront pas étudiés tout au long de ce mémoire.
[4] Java est un langage de programmation orienté objet le plus populaire dans le monde, développé initialement pas la société Sun Microsystems dans les années 1990.
[5]Étudier un objet pour en déterminer le fonctionnement interne.

III.2.6.2 BOUML

BOUML est un logiciel de modélisation UML gratuit développé en langage C++ permettant de créer des diagrammes, la génération de code et l'ingénierie inverse. Il est rapide et extensible.

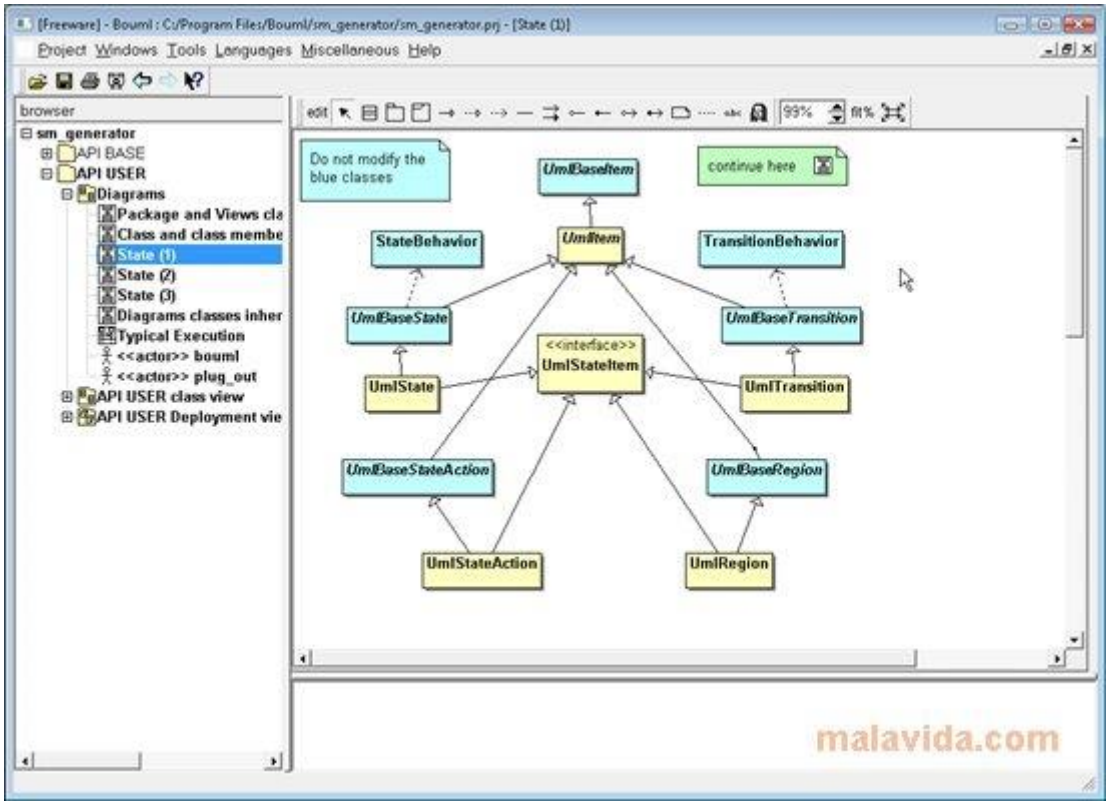


Figure III.3 : Interface graphique de l'outil BOUML.

III.2.6.3 StarUML

StarUML est un langage de modélisation UML en licence propriétaire développé en pascal orienté objet avec Delphi. Il permet de gérer la plupart des diagrammes spécifiés dans la norme UML 2.0.

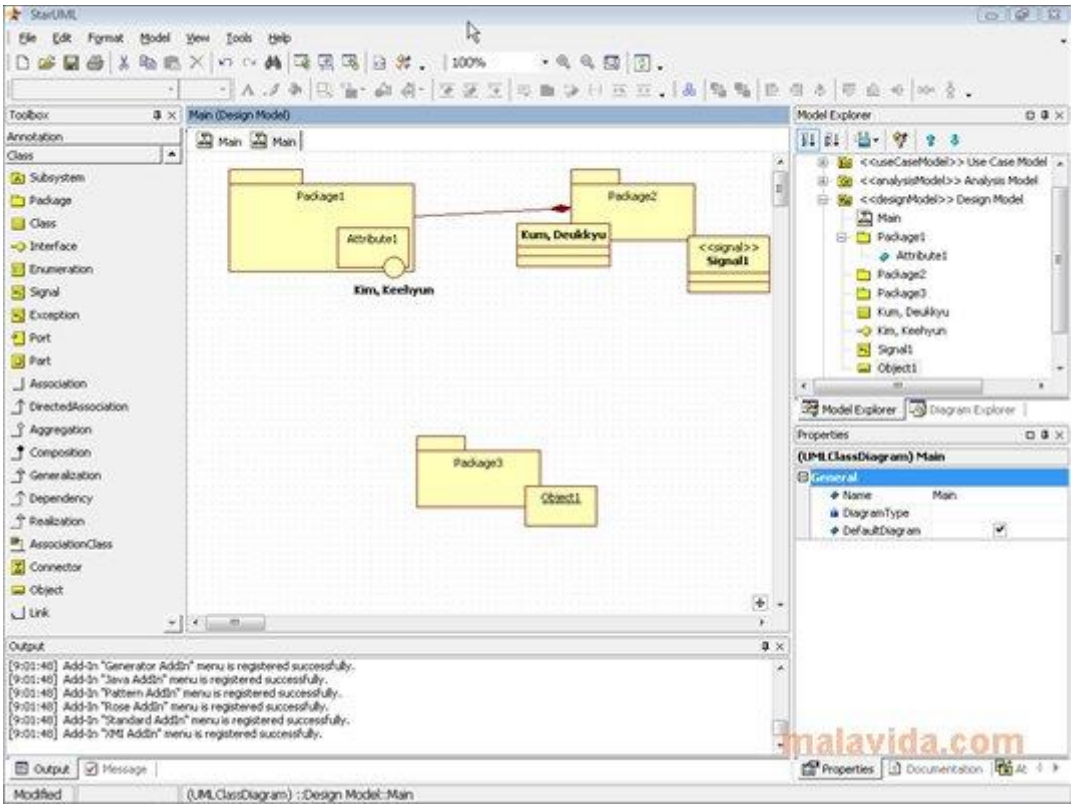


Figure III.4 : Interface graphique de l'outil StarUML

III.2.6.4 Visual Paradigm

Visual Paradigm est un logiciel de modélisation de diagrammes, il supporte plusieurs diagrammes autres que les diagrammes UML, il est très riche en fonctionnalité. Il est payant mais gratuit pour la version communauté. Développé en Java par Visual Paradigm International.

Visual Paradigm peut gérer tous les 14 diagrammes UML. L'édition communauté peut gérer les 14 diagrammes UML.

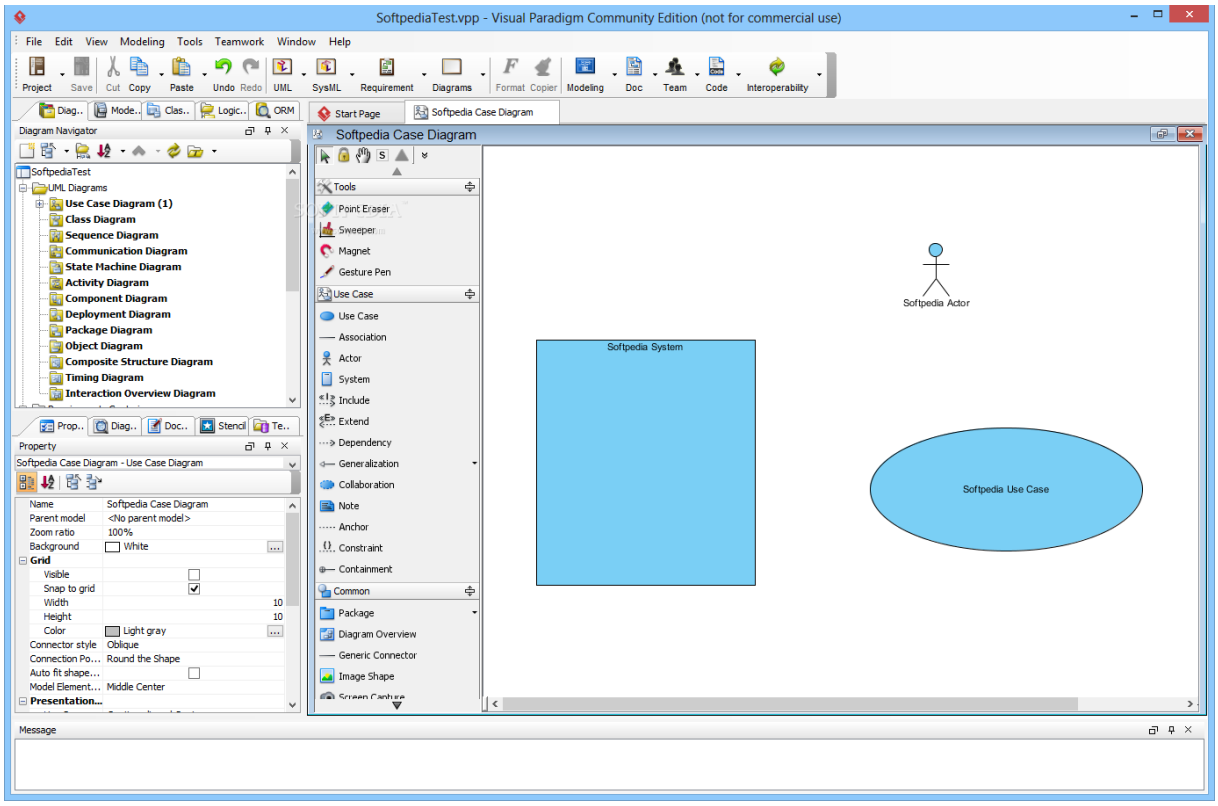


Figure III.5 : Interface graphique de l'outil Visual paradigm.

III.2.6.5 Eclipse Papyrus

Eclipse[6] Papyrus est l'un des outils (extension) de modélisation UML pour L'IDE Eclipse. Avec Papyrus, il est possible de créer des diagrammes UML et générer directement lecode. L'outil Papyrus est gratuit, en anglais et développée en Java par Eclipse Fondation, il est uniquement compatible avec le système d'exploitation Microsoft Windows.

[6] Eclipse est un environnement de développement intégré IDE ou EDI en français développé en Java par Eclipse Fondation et qui s'appuie principalement sur Java, il permet grâce à ses nombreux outils de développement de créer des logiciels informatiques.

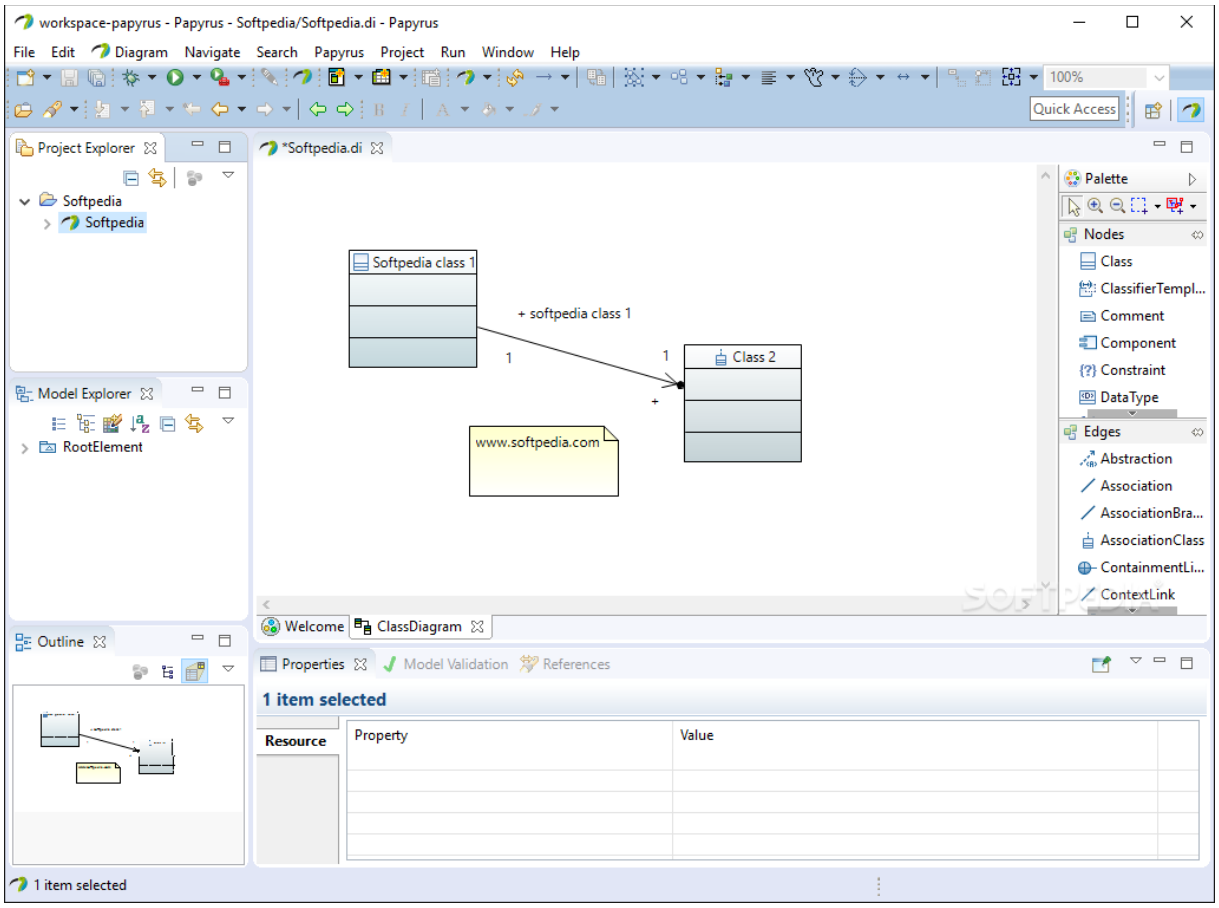


Figure III.6 : Interface graphique de l'outil Eclipse Papyrus.

III.2.7. Types de diagrammes UML

UML possède 14 diagrammes au total, nous allons en citer uniquement deux parmi eux et qui sont l'objet de notre étude, le **diagramme de cas d'utilisation** et le **diagramme de classes.**

III.2.7.1. Diagramme de cas d'utilisation

III.2.7.1.1. Définition

Le diagramme de cas d'utilisation forme la première étape d'analyse d'un système[7], il modélise l'aspect fonctionnel d'un système, c'est-à-dire ce que le système doit faire sans dire comment il va le faire. Le digramme de cas d'utilisation sert aussi à exprimer les besoins d'un utilisateur en les représentants sous forme d'un modèle récapitulatif.

[7] Fait référence à une partie d'un logiciel ou du logiciel lui-même.

III.2.7.1.2. Éléments d'un diagramme de cas d'utilisation

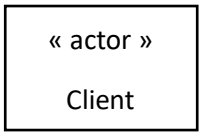
Acteur

Un acteur est l'utilisateur (humain ou non) d'un système, il doit avoir un nom qui le défini en tant que rôle dans le système, par exemple : comptable, client, caissier, agent….

En UML, l'acteur est représenté graphiquement sous trois formes :

1. Sous forme d'une icône (*stick man*) avec un nom qui représente son rôle.

2. Sous forme d'un rectangle avec nom et un mot clé « actor »



3. Sous une forme intermédiaire (un rectangle avec une icône (*stick man)*)

En règle générale, un acteur humain est représenté sous la forme 1 (voir ci-dessous) et un non humain est représenté sous forme d'un rectangle (forme 2 ou 3).

Il faut savoir aussi qu'il existe deux types d'acteurs :

1. **Acteur principal** est celui pour lequel le système produit un résultat observable.
2. **Acteur secondaire** est généralement sollicité pour les informations complémentaires.

Cas d'utilisation

Un cas d'utilisation est une unité qui représente une fonctionnalité d'un système. Il réalise un service de bout en bout, avec un déclenchement, un déroulement et une fin, pour l'acteur qui l'initie.

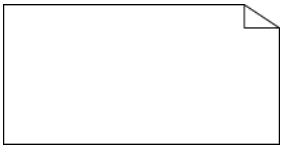
Un cas d'utilisation est représenté graphiquement en UML par une ellipse et du texte (décrivant la fonctionnalité) à l'intérieur ou à l'extérieur de cette ellipse.

Figure III.7 : Objet graphique d'un cas d'utilisation

Note

Une note est une information textuelle supplémentaire pour un élément d'un diagramme de cas d'utilisation, elle permet de commenter, insérer une contrainte ou un corps de méthode (fonction).

Les notes sont représentées par un rectangle avec le coin supérieur droit replié sur lui-même. On peut relier une note à un élément en utilisant une ligne pointillée.



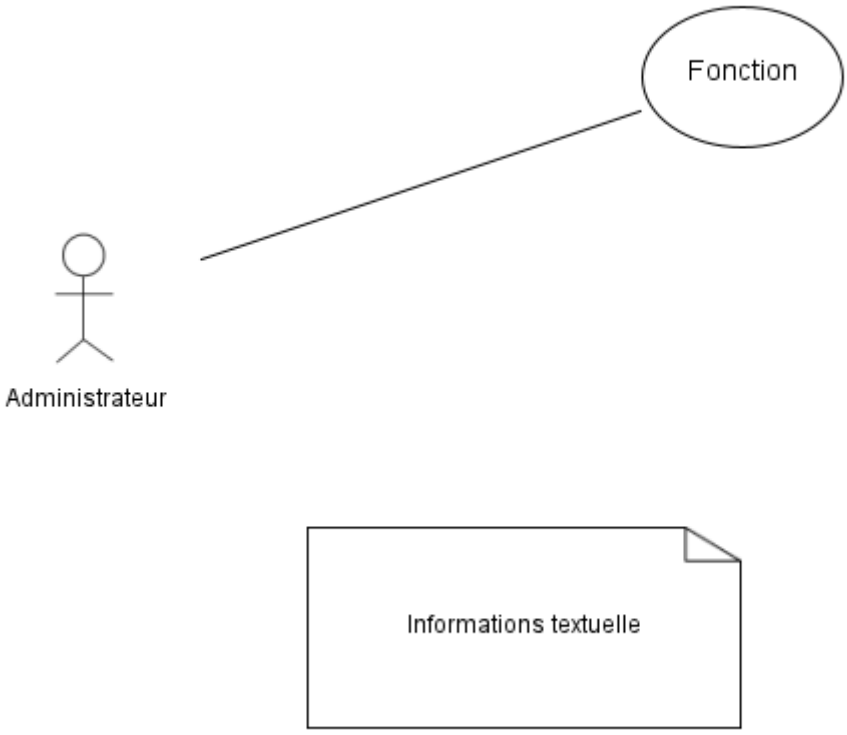


Figure III.8 : Exemple d'un diagramme avec une note

Relations

Une relation est un lien qui relie l'acteur et un cas d'utilisation, elle indique ce que l'acteur fait ou doit faire dans un système. Une relation est représentée par une ligne physique.

Une relation relie aussi les cas d'utilisation, cette relation est constituée de deux parties : « Inclusion » et « Extension » (Les mots « Inclusion » et « Extension » sont des stéréotypes[8].

Relations entre les acteurs et les cas d'utilisation

1. **Association :** Une association est un lien de communication entre l'acteur et les cas d'utilisation, elle est représentée graphiquement par un trait avec ou sans une flèche.

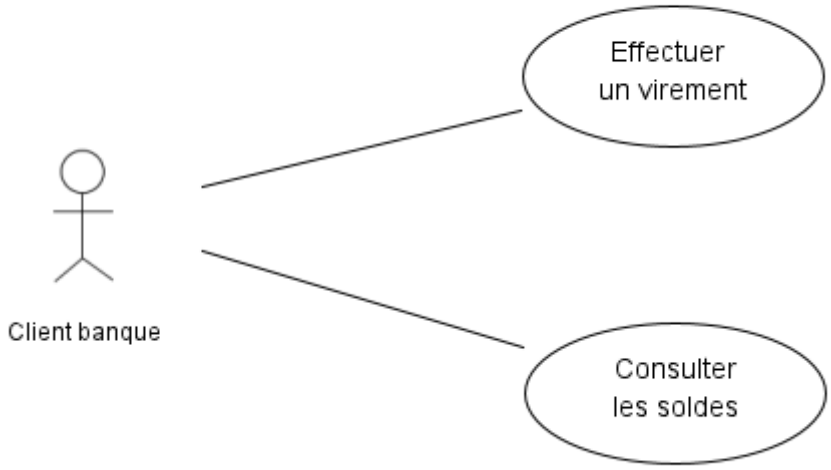


Figure III.9 : Exemple de relation d'association

2. **Généralisation ou héritage**

La relation de généralisation ou héritage ou encore spécialisation permet de montrer qu'un cas d'utilisation est un type spécial d'un autre, ça s'applique aussi pour les acteurs. Le cas d'utilisation le plus spécialisé diffère quelque peu de l'original. C'est-à-dire qu'ils n'ont pas les mêmes fonctionnalités mais l'un hérite de l'autre. Par exemple un cas d'utilisation « Payer par chèque » hérite du cas d'utilisation « Payer » qui est un cas global d'un mode de paiement.

La relation de généralisation est représentée avec un trait continue avec une flèche sous forme d'un triangle vide.

[8] Syntaxe qui permet d'ajouter de la sémantique à un objet.

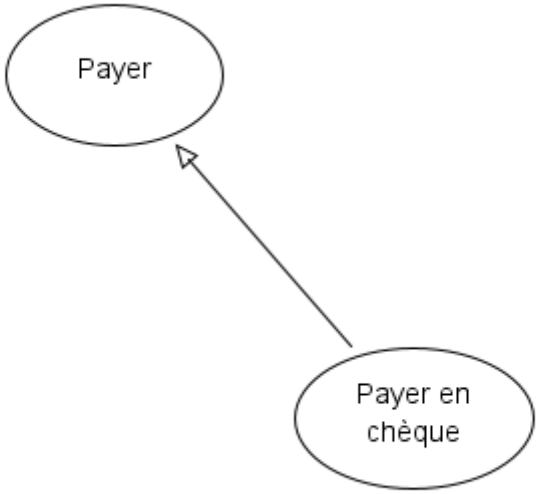


Figure III.10 : Exemple d'une relation de généralisation entre les cas d'utilisation

Il est aussi possible de relier entre les acteurs avec une relation de généralisation (même principe entre les cas d'utilisation).

Figure III.11 : Exemple d'une relation de généralisation entre les acteurs

Dans la figure précédente, on peut dire qu'un super administrateur est un administrateur avec plus de fonctions d'administrations.

Relations entre les cas d'utilisation

1. **Inclusion :** Une relation d'inclusion permet d'inclure un cas d'utilisation par un autre cette relation permet d'indiquer un cas d'utilisation A est un élément primaire dans un cas d'utilisation B en d'autres termes, A dépend de B. La relation d'inclusion est représentée par un trait discontinue avec une flèche orientée et un stéréotype « Include ».

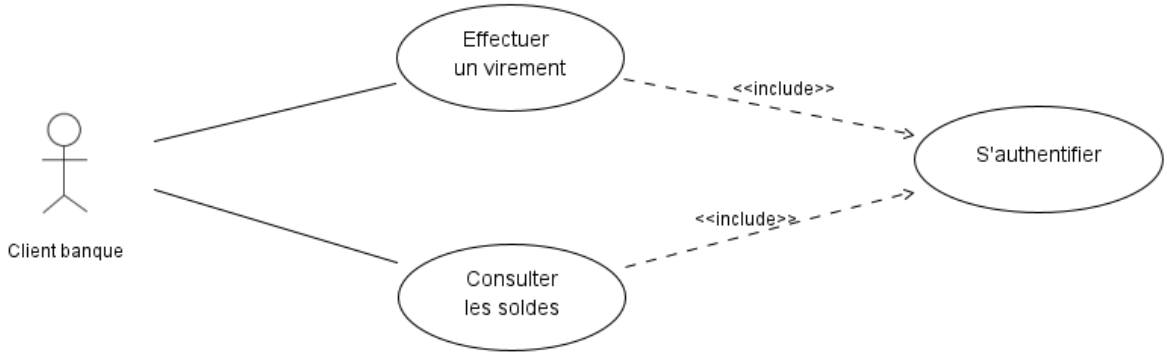


Figure III.12 : Exemple de relation d'inclusion entre les cas d'utilisation

Dans la figure ci-dessus, on peut lire par exemple pour effectuer un virement, il faut s'authentifier. Ceci en va de même pour consulter les soldes.

2. **Extension :** Contrairement à une relation d'inclusion, une relation d'extension permet d'étendre un cas d'utilisation A à un cas d'utilisation B en d'autres termes, le A peut-être appeler à l'exécution de B. La relation d'extension est représentée par un trait discontinue avec une flèche orientée et un stéréotype « Extend ».

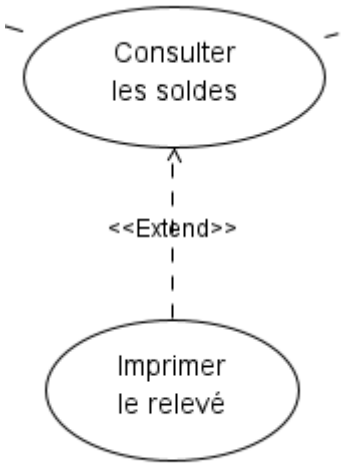


Figure III.13 : Exemple de relation d'extension entre les cas d'utilisation

On peut lire dans la figure ci-dessus qu'on peut imprimer le relevé de compte après avoir consulté les soldes.

Package des cas d'utilisation

Il est possible de regrouper les cas d'utilisation en groupe appeler « package », ce regroupement peut se faire par acteur (tous les cas d'utilisation mettant en scène un acteur), par domaine fonctionnel (tous les cas d'utilisation associés à une fonction donnée.

Un package est représenté en UML graphiquement par deux rectangles disposés verticalement, le premier rectangle contient le nom du package, et le second les éléments d'un diagramme de cas d'utilisations.

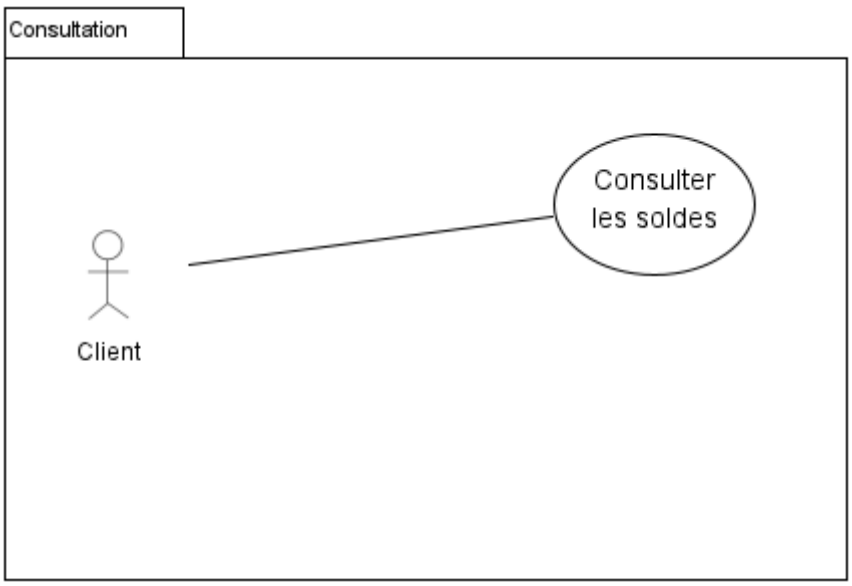


Figure III.14 : Exemple d'un package

III.2.7.1.3. Identifier les acteurs et les cas d'utilisation

Pour **identifier les acteurs,** il faut commencer par identifier les utilisateurs du système, sans oublier les personnes responsables de son exploitation et de sa maintenance.

Attention ! Lorsque plusieurs utilisateurs possèdent le même rôle, on les représente comme un seul acteur. Mais si un utilisateur a plusieurs rôles, il y a plusieurs acteurs.

Pour **identifier les cas d'utilisation,** il faut déterminer pour chaque acteur les différentes intensions pour lesquelles il utilise le système et déterminer dans le cahier des charges[9] les services fonctionnels du système.

III.2.7.2. Diagramme de classes

Définition

Le diagramme de classes a toujours été le diagramme le plus importants dans toutes les méthodes orientées objets. Alors que le diagramme de cas d'utilisation montre le point de vue du système, le diagramme de classe montre la structure interne du système. Il fournit une représentation abstraite des objets d'un système en les modélisant avec des classes et des attributs, qui vont interagir pour réaliser les cas d'utilisation.

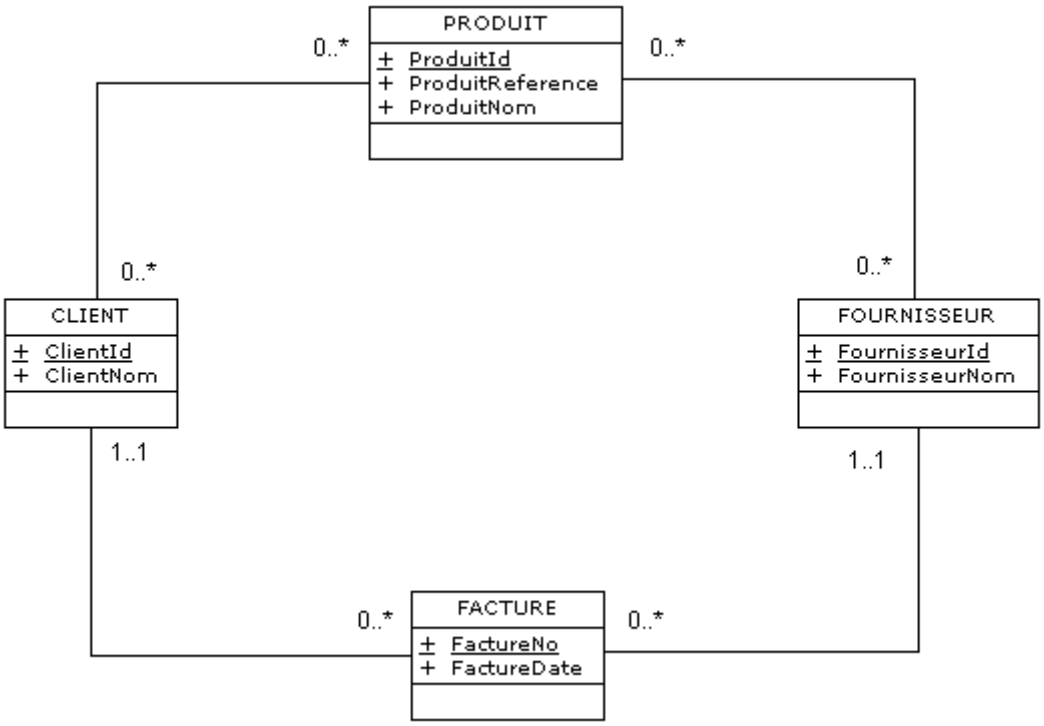


Figure III.15 : Exemple d'un diagramme de classes.

[9] Document utilisé pour écrire les besoins d'un utilisateur lors d'un entretien.

Éléments d'un diagramme de classes

1. Classe

Une classe fournie une représentation abstraite d'un ensemble d'objets ayant les mêmes caractéristiques. Exemple Voiture est une classe, Personne est une classe.

Une classe est composé d'**attributs** et de **méthodes** (opérations) ces deux derniers ont un attribut important dans la programmation orientée objet appelé : type d'***encapsulation***.

Attribut

Un attribut est un **champ** de la classe, autrement dit un type d'information contenu dans la classe. Exemple : Nom est un attribut de la classe Personne, Marque est un attribut de la classe Voiture.

Méthode

Une méthode ou opération est un comportement contenu dans la classe. Exemple : accélérer est une méthode de la classe Voiture.

Encapsulation

Pour protéger l'accès aux attributs et méthodes d'une classe à partir d'une autre classe, on utilise un attribut spécial qui s'appelle « type d'encapsulation ». Ce dernier permet de limiter la portée d'un attribut ou d'une méthode. Il existe 3 types d'encapsulation :

Type d'encapsulation	Description
Privé - signe «-»	Un attribut ou une méthode avec un type privé est uniquement accessible dans la classe où il a été déclaré
Protégé – signe «~»	Un attribut ou une méthode avec un type protégé est accessible dans la classe où il a été déclaré et dans une classe fille (principe d'héritage[10]).
Publique – signe «+»	Un attribut ou une méthode avec un type publique est accessible dans n'importe quelle classe.

Tableau II.1 Types d'encapsulation d'un attribut ou méthode d'une classe.

Une classe est représenté graphiquement en UML par un rectangle avec trois rectangles appelés compartiments disposés verticalement à l'intérieur.

a) **1er compartiment :** Il contient le nom de la classe, le nom d'une classe commence par une majuscule. Il ne doit pas contenir d'espaces.

b) **2e compartiment :** Il contient l'ensemble des attributs de la classe, le nom d'un attribut commence par une minuscule. Il ne doit pas contenir d'espaces.

c) **3e compartiment :** Il contient l'ensemble des méthodes de la classe, le nom d'une méthode commence aussi par une minuscule. Il ne doit pas contenir d'espaces.

[10] Dans la programmation orientée objet, le principe d'héritage sert à réutiliser les attributs ou méthodes d'une classe mère sans à les redéclarer dans une classe fille. Lorsqu'une classe B hérite de classe A, on dit que la classe B est une classe fille et la classe A est la classe mère.

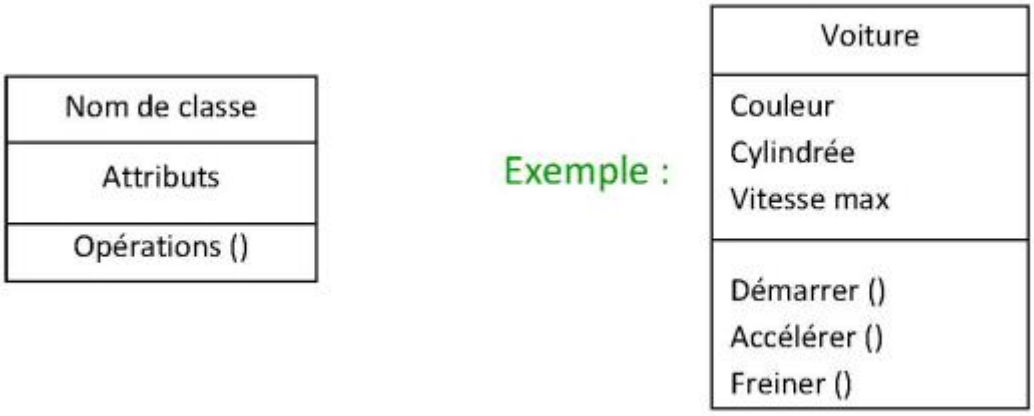


Figure III.16 : Exemple de représentation graphique d'une classe

2. Objet

Un objet est une instance d'une classe.

Exemple : Amir est une instance de Personne, Renault Laguna est une instance de Voiture.

3. Associations

Une association est une relation sémantique entre deux classes, elle est représentée graphiquement par un trait continue. Il est possible d'ajouter un texte descriptif pour la relation.

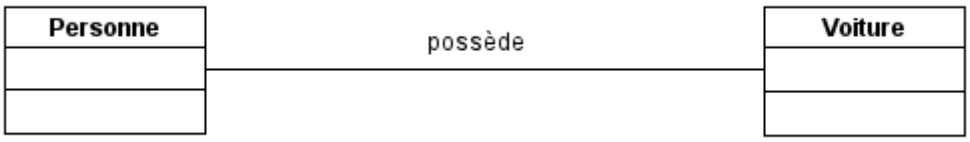


Figure III.17 : Exemple de relation d'association entre deux classes

Pour chaque association, on doit ajouter une indication de **multiplicité,** Une multiplicité apparaît à chaque extrémité d'une relation, elle indique le nombre d'objets d'une classe apparaissant à une extrémité pouvant s'associer à un seul objet de la classe dans l'autre extrémité.

Les principales multiplicités sont :

Multiplicité	Représentée par le symbole
Plusieurs	*
N fois	1,2,3, 4….n
Au minimum n	n…*
Entre n et m	n…m

Tableau II.2 : Types de multiplicités.

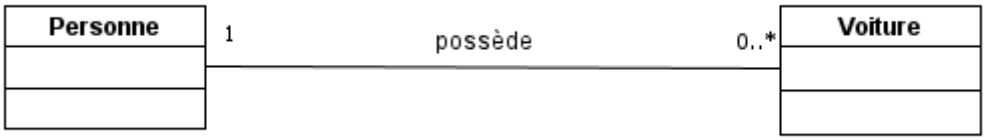


Figure III.18 : Relation d'association avec une multiplicité

4. Agrégation

Une agrégation une forme particulière d'association, elle est non symétrique, c'est-à-dire qu'elle n'est pas bidirectionnelle contrairement à une association, elle permet d'exprimer une relation d'appartenance, elle n'a pas besoin d'être nommée car elle se lit implicitement « est composé de » ou « contient ».

La représentation graphique d'une relation d'agrégation est un trait continue avec un losange vide à une extrémité (du côté de l'agrégat).

Figure III.19 : Relation d'agrégation

Nous pouvons lire dans la figure ci-dessus qu'une salle de cours peut avoir plusieurs chaises et une chaise peut être déplacé dans plusieurs salles de cours.

5. Composition

La composition également appelée « Agrégation composite » est comme une agrégation avec deux contraintes supplémentaires :

- Un composant n'appartient qu'à un seul objet composé.

- La destruction d'un objet composé entraine la destruction de ses composants.

La représentation graphique d'une relation de composition est un trait continue avec un losange plein à une extrémité (du côté de l'agrégat). La multiplicité coté agrégat ne peut prendre que 0 ou 1, par défaut elle vaut 1.

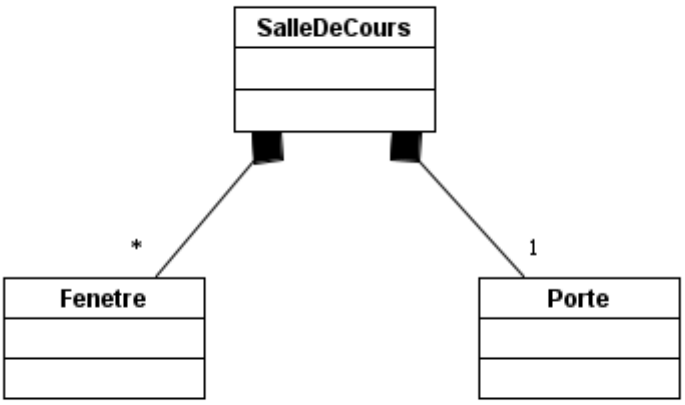


Figure III.20 : Relation de composition

Dans la figure ci-dessus, on peut lire comme ceci : une salle de cours est composée de plusieurs fenêtres et une seule porte.

6. Généralisation

La généralisation ou héritage est une forme de relation très important dans la modélisation objet car elle simplifie la représentation en utilisant l'**héritage.** C'est exactement le même principe comme nous l'avions vu dans les cas d'utilisation, sauf qu'ici c'est une relation entre les classes. En effet la classe générale (classe mère) va être utilisée pour contenir tout ce qui est en commun pour un ensemble de classes (attributs et méthodes), puis un ensemble de sous-classes « classes filles » vont contenir chacune la description propre de la classe mère.

Une classe fille hérite toutes les propriétés de la classe mère (attributs et méthodes) en plus de ses propriétés.

La représentation graphique d'une relation de généralisation en UML est un trait continue avec un triangle vide à une seul extrémité.

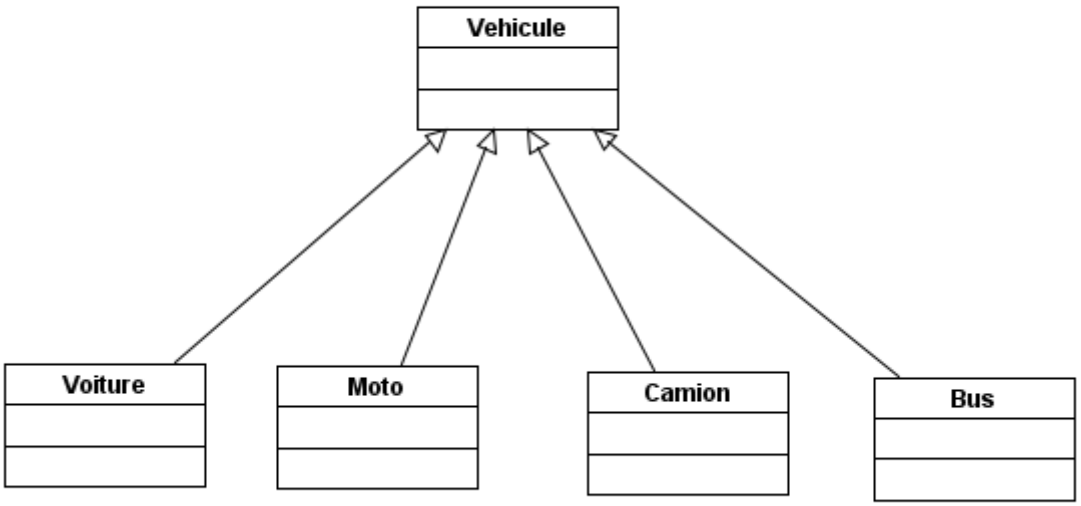


Figure III.21 : Relation de généralisation

Dans la figure ci-dessus, nous pouvons interprété la figure comme suite : une voiture, moto, camion et bus sont des véhicules.

7. Classe d'association

Une association peut avoir ses propres attributs qui ne sont disponible dans aucune des classes qu'elle lit. Cette association est appelée **classe d'association**. Elle fonctionne exactement comme les autres classes du modèle.

Les attributs d'une classe d'association ne peuvent pas être affecter à aucune autre classe.

La représentation graphique d'une classe d'association en UML est une classe avec un trait discontinue relier à une autre association (relation).

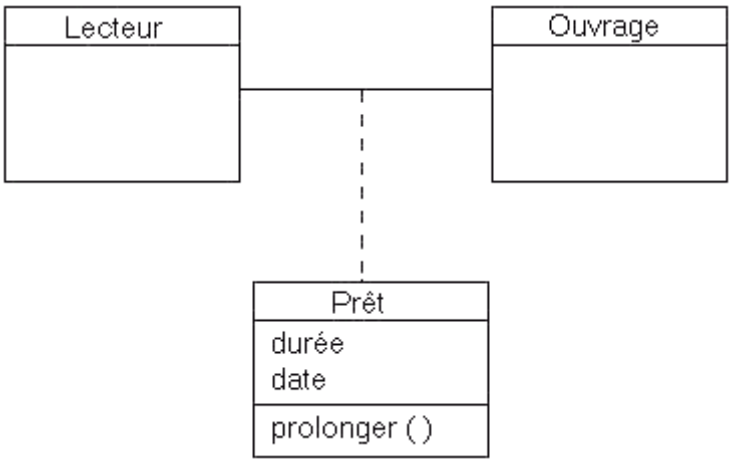


Figure III.22 : Exemple de classe d'association

Dans la figure ci-dessus, on peut interpréter la figure de la façon suivante : un lecteur peut prêter plusieurs livres pour une durée et une date déterminée avec possibilité de prolongation.

8. Association réflexive

Une association est dite réflexive, si ses deux extrémités pointent vers la même classe. Une association réflexive est symétrique (elle peut se lire dans chaque sens)

La représentation graphique en UML est un trait orthogonal.

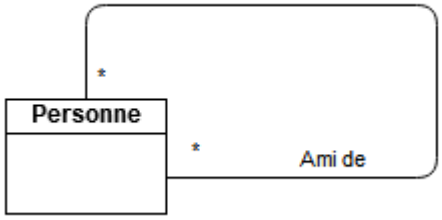


Figure III.23 : Association réflexive

Dans la figure ci-dessus, on peut dire qu'une personne X est ami d'une personne Y, l'inverse est aussi correct.

9. Package

Tout comme dans un diagramme de cas d'utilisation, le diagramme de classes a lui aussi l'élément « Package », il a exactement la même fonction dans un diagramme de cas d'utilisation sauf qu'ici au lieu de regrouper les cas d'utilisation, on regroupe les classes.

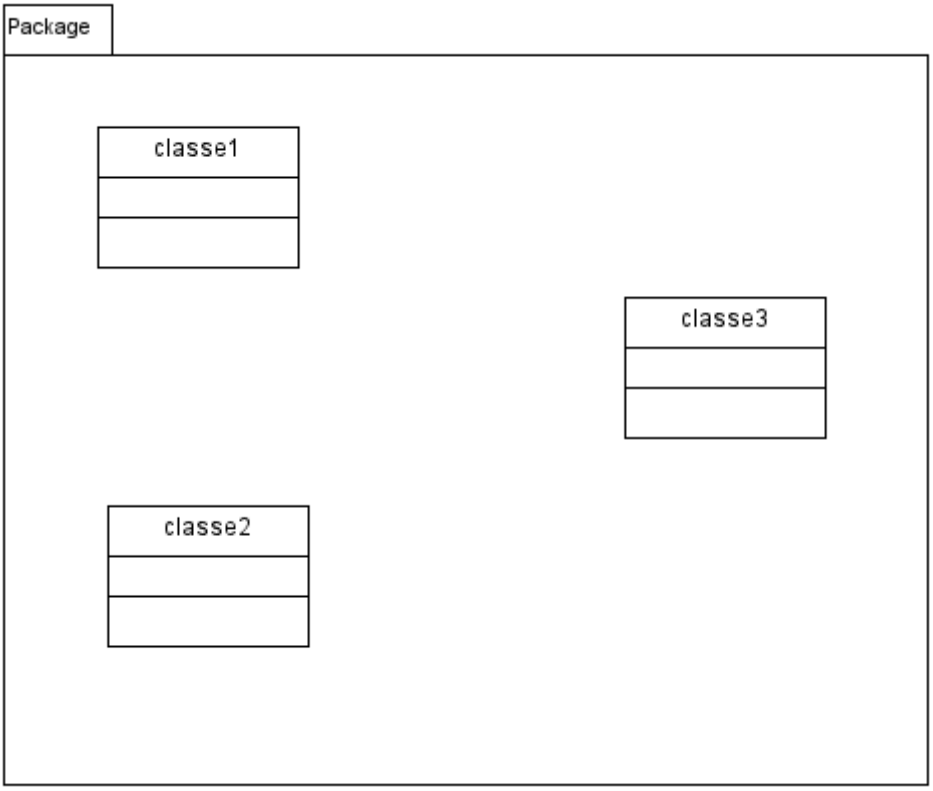


Figure III.24 : Regroupement des classes avec un package

III.3 Conclusion

Dans ce chapitre, le langage UML est abordé avec l'étude de deux de ses diagrammes (diagramme de cas d'utilisation et diagramme de classes) ainsi que leurs éléments.

Le langage UML est un langage de modélisation très riche et puissant, c'est un outil d'aide au développement logiciel grâce à ses diagrammes, on peut ainsi modéliser les parties d'un système afin de passer facilement au mode réalisation.

Dans le chapitre prochain, nous allons concevoir notre approche de modélisation de procédé logiciel utilisant UML.

CHAPITRE IV : CONCEPTION

IV.1. Introduction

Ce chapitre sera consacré à la partie conception de notre outil UML et une plateforme Web pour gestion des projets des développeurs.

Ce chapitre est divisé en deux parties :

- La première partie détaillera la conception de notre outil UML ou on peut l'appeler aussi un environnement de modélisation UML.
- La seconde partie détaillera la conception d'une plateforme Web dynamique pour l'accès à, des modèles de procédés logiciel suivi par des projets, ou enregistrer dans la base de données à des fins de portabilité, de gestion et d'amélioration, les modèles UML définis à l'aide de notre outil peuvent être enregistrés dans la base de données.

IV.2. Architecture de l'environnement de développement

L'architecture de notre environnement montre les [11]différents outils et modules intervenants dans le processus de développement (modélisation, instanciation, exécution).

Cette architecture est base sur l'outil graphique de modélisation UML, d'une base de données (pour stocker les modèles, projet et membre) et d'une interface Web dynamique pour la gestion.

La figure ci-dessus montre cette architecture.

[11] Est une interface dont le contenu (les données) change en fonction des opérations des utilisateurs.

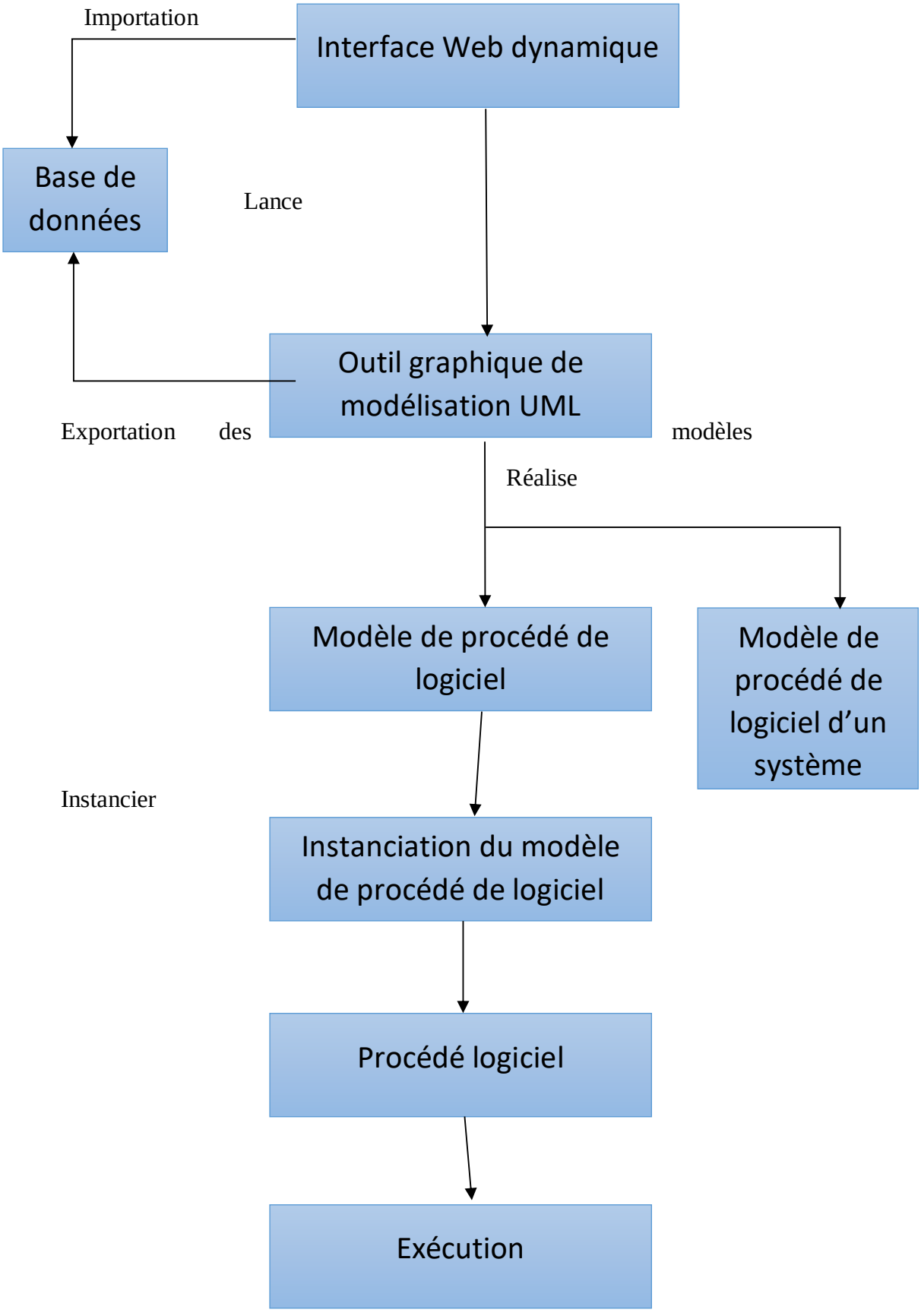


Figure IV.1. Architecture de l'environnement de développement

IV.1.2. Acteurs et entités de l'environnement

Concepteur de modèle (Responsable informatique) : Il supervise la définition, la politique et la stratégie globale de développement des logiciels. Cette politique est décrite sous la forme d'un modèle de procédé logiciel.

Modèle de procédé logiciel : Un modèle de procédé logiciel est la base de toute définition d'un procédé logiciel. Il offre une description globale de types d'activités, de ressources (matérielle, logicielle, humaine…), des relations entre ces différents types.

Chef de projet : Il a la responsabilité de définir un procédé logiciel à partir d'un modèle de procédé logiciel, c'est ce qu'on appelle instancier un modèle. Pour cela, il devra contrôler, définir la nature de chacune des activités de son projet.

Membre : Un membre est un agent affecté pour un projet pour un rôle précis (développer, tester, créer un design…), son travail est donc d'exécuter des tâches définit par son chef de projet.

Procédé logiciel : Un procédé logiciel est le résultat d'une instanciation d'un modèle de procédé logiciel, il est aussi un ensemble cohérent d'activités qui fait intervenir un certain nombre de personnes, d'outils, de produits et de techniques afin d'assurer le développement d'un logiciel.

Base de données : La base de données a pour rôle de stocker toutes les informations définies au cours du processus de définition, d'instanciation et d'exécution d'un modèle.

IV.3. Conceptualisation

L'objectif de la conceptualisation est de comprendre la structure et les fonctionnalités de notre application (outil de modélisation + application Web), qui va nous permettre de :

- Définir le conteur du système à modéliser.
- D'identifier les fonctionnalités principales.

Nous utiliserons dans ce cas le diagramme de cas d'utilisation que nous avons défini dans le chapitre précédent. Ce diagramme va modéliser la relation entre un acteur et les fonctionnalités qui lui sont propres.

IV.3.1. Conception de l'outil de modélisation UML

La figure ci-dessus décrit les actions d'un concepteur de modèle qui pourra effectuer dans notre outil UML.

Dans notre outil UML, un modèle est soit un diagramme de classes ou un diagramme de cas d'utilisation.

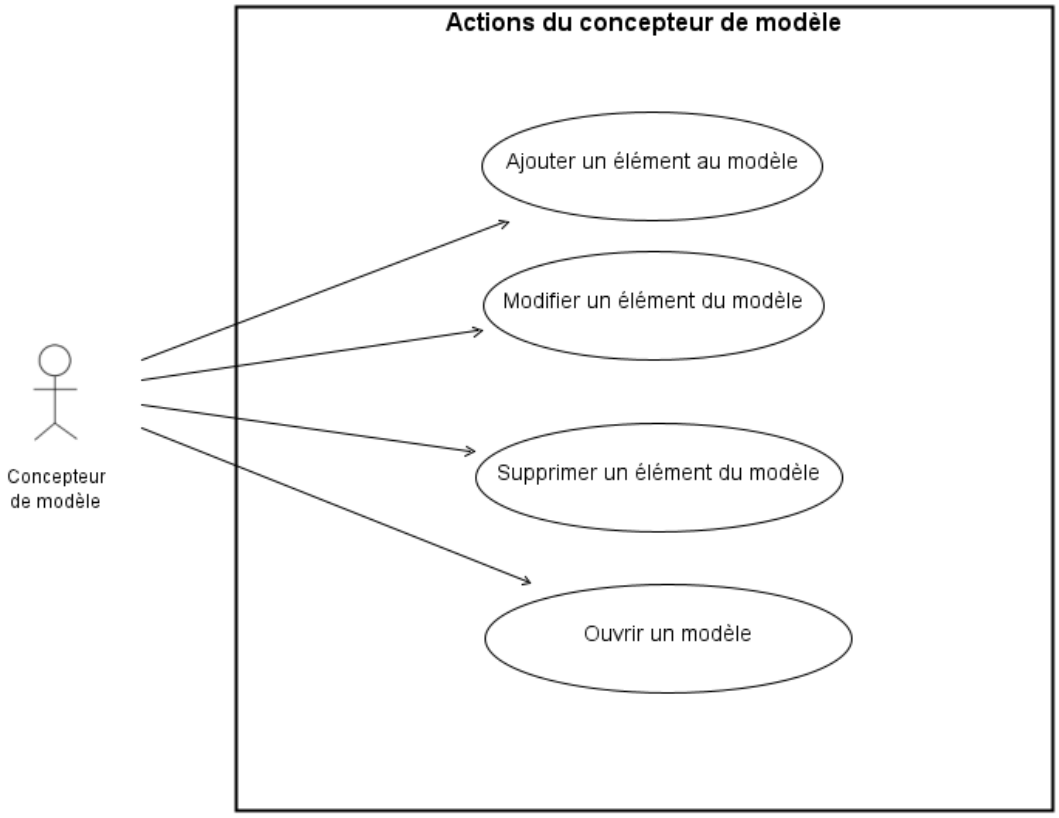


Figure IV.2. Actions d'un concepteur de modèle

Un élément du modèle est :

- Dans un Diagramme de cas d'utilisation
  - Un acteur.
  - Un cas d'utilisation.
  - Une relation (lien).
  - Une note.
- Dans un diagramme de classes
  - Une classe.
  - Une relation.
  - Une note.

Une règle à respecter dans la création d'un diagramme est qu'on ne doit pas dupliquer le nom d'une classe ou d'un cas d'utilisation ou d'un acteur.

IV.3.2 Conception de l'application Web dynamique

Dans cette partie, nous présenterons l'application Web dynamique qui joue un rôle, d'un côté comme une plateforme de communication entre les membres (Concepteur, chef de projet, membres), et dans un autre coté l'accès aux projets en cours, ainsi que la modélisation des procédés logiciels correspondants, en utilisant notre outil de modélisation UML défini auparavant.

Pour accéder aux projets, nous définissons les types de profils suivants :

- **Responsable informatique** : Appelé également concepteur de modèle, qui est en charge de la conception et de la gestion du modèle de procédé et des projets et des instances de procédé.

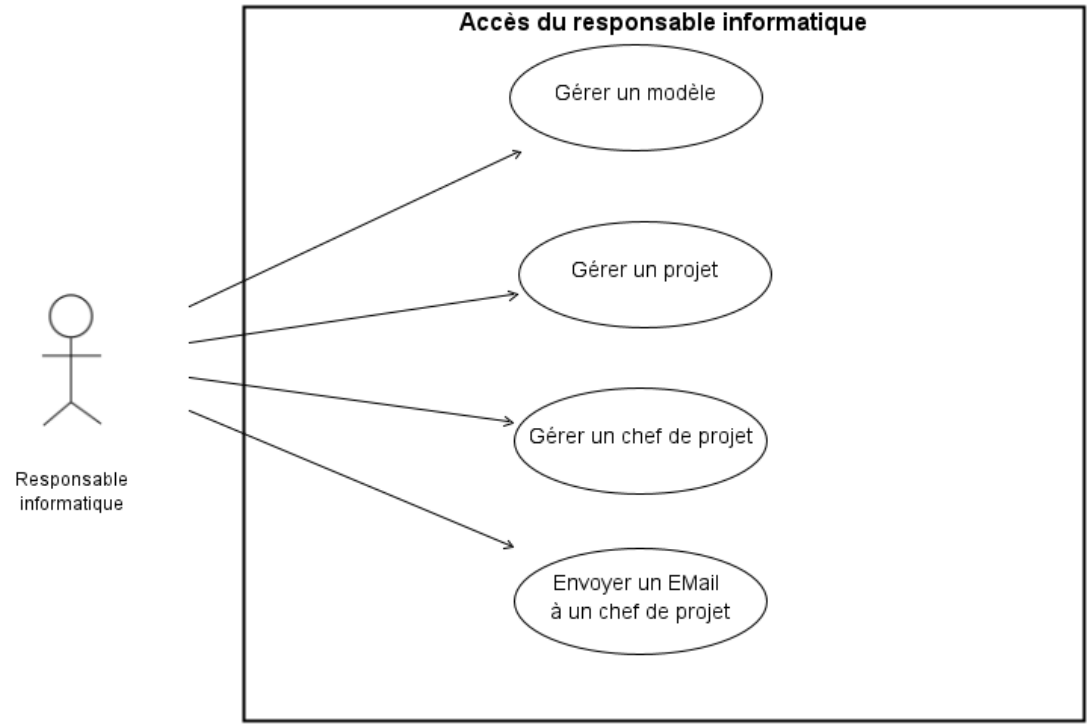


Figure IV.3 Accès du responsable informatique

- **Chef de projet** : qui est en charge de gérer les membres d'un projet. Il est en charge aussi d'instanciation d'un modèle, et son exécution.

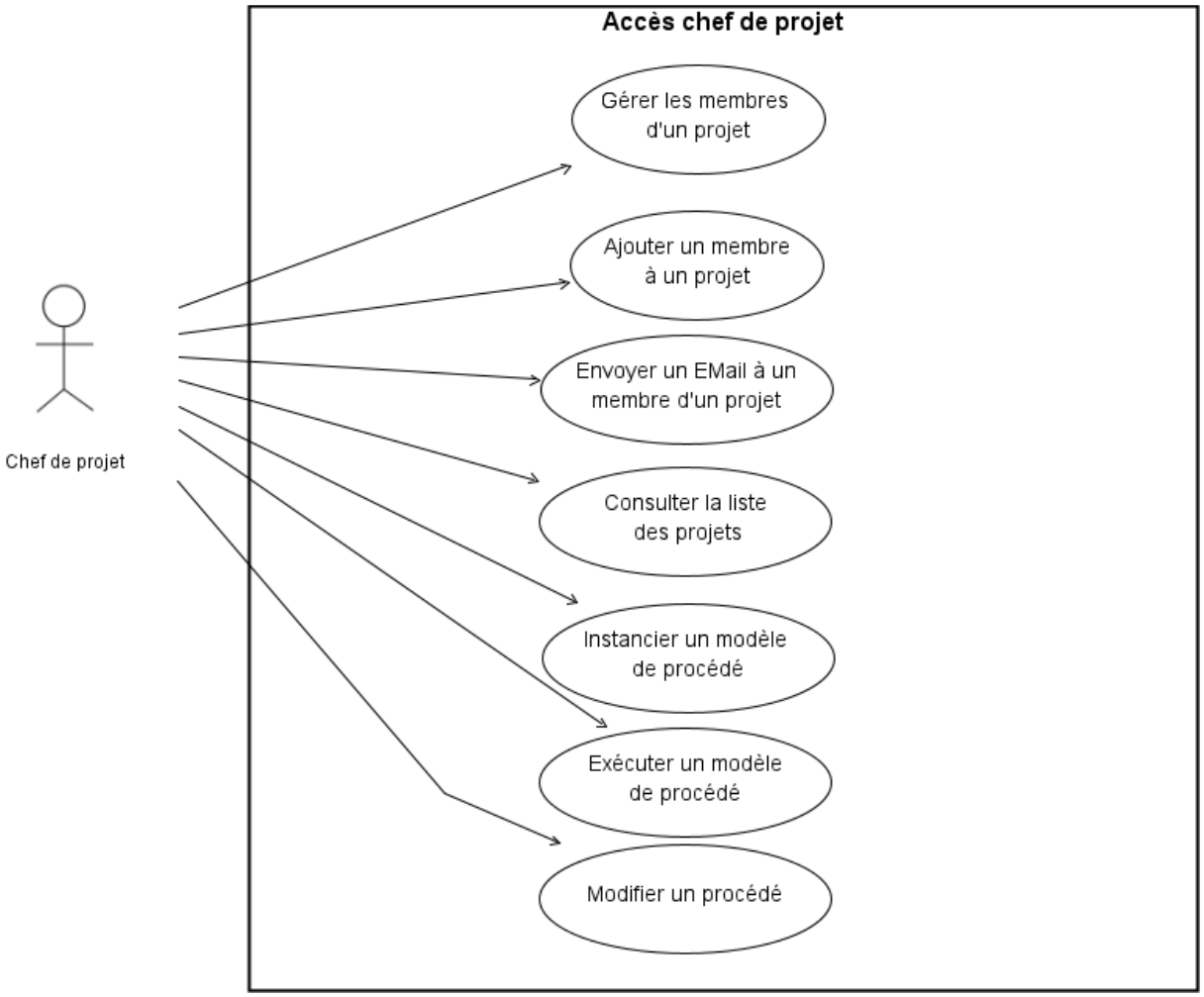


Figure IV.4. Accès d'un chef de projet

- **Membre** : Son rôle est de participer à la réalisation d'un ou plusieurs projets, comme un agent développeur, un infographiste, un technicien, etc. Ses droits de gestion et d'accès sont limités seulement à son rôle dans ses projets.

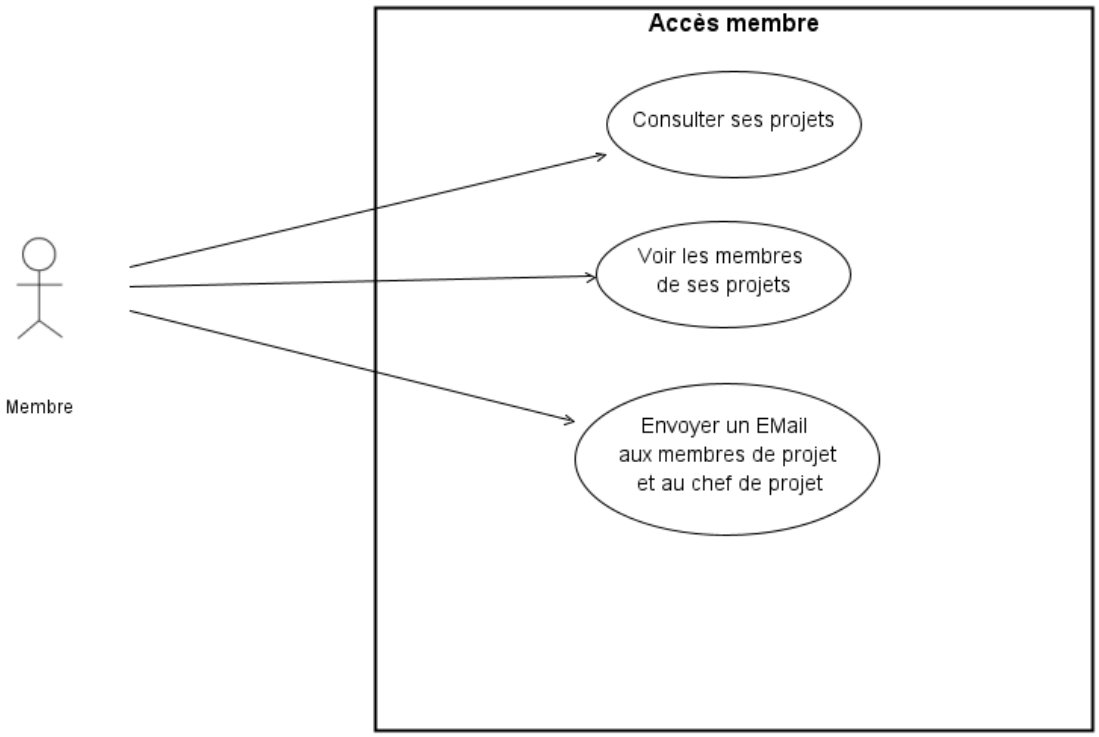


Figure IV.5. Accès membre

Pour stocker les informations de l'interface Web telles que (les profils, les projets, les modèles…), nous avons besoin d'une base de données. Nous détaillerons son modèle relationnel (structure relationnel) dans la partie qui va suivre.

IV.3.2.1 Modèle relationnel de la base de données

Le modèle relationnel a vu le jour en 1970, il a été l'un des travaux de Edgar Frank Codd. La principale publication est « A Relational Model for Large Shared Data Banks », Communication of the ACM, vol.13, N° 6, 1970.

Ce modèle est construit avec les concepts suivants :

- **Relation :** Structure de données qui décrit la table qui sera créée à l'aide du langage SQL.
- **Attribut :** Représentation d'une information atomique qui décrit une colonne d'une table.
- **Clé primaire :** formée d'un ou plusieurs champs d'une base de données qui identifient de manière unique un enregistrement dans une table.
- **Clé étrangère :** définit une relation entre deux tables, et d'assurer la cohérence des données.

Pour pouvoir implémenter une base de données, il faut traduire le modèle conceptuel en modèle logique. Ce qui signifie qu'il faut traduire un modèle UML en modèle relationnel.

En appliquant les règles de passage[12], nous obtenons les relations suivantes :

- **Concepteur** (id_concepteur, utilisateur, mot_de_passe,email).
- **Chef** (id_chef, utilisateur, mot_de_passe, email, #id_concepteur).
- **Membre** (id_membre, utilisateur, mot_de_passe, email, role).
- **Modele** (id_modele, date_modele, version, chemin_fichier, #id_concepteur).
- **Projet** (id_projet, date_debut, date_fin, cloture, #id_chef, #num_modele).
- **Inscrit_membre** (#id_chef, #id_membre).
- **Inscription** (numero, role, date_inscr, #num_projet, #id_membre).

IV.4 Conclusion

Dans ce chapitre, nous avons détaillé l'architecture de conception de notre outil de modélisation UML et aussi celle de notre application web.

Nous avons vu aussi comment la base de données est structurée avec un modèle logique de données. Dans le chapitre suivant, nous détaillerons la réalisation des deux applications (application UML et application web) afin de savoir quels sont les différents langages de programmation utilisés et les environnements de développement utilisés ainsi que les bibliothèques de développements.

[12] Modélisation avancée en UML et en relationnel, Stéphane Crozat

CHAPITRE V : MISE EN ŒUVRE

V.1. Introduction

Dans cette section nous allons présenter le prototype permettant la mise en œuvre de l'environnement de modélisation de procédés logiciels.

Nous avons utilisé JAVA comme langage de programmation pour la mise en œuvre de notre outil de modélisation, et PHP pour l 'interface Web.

V.2. Outils et langages de développement de l'application UML

Dans cette partie, nous définirons l'ensemble des outils et langages de développement utilisés pour réaliser notre outil de modélisation UML.

Figure V.1 : Langages et outils de développement utilisés

Figure V.2 : Bibliothèques utilisées

V.2.1. Langage de programmation Java

Java est un langage de programmation orientée objet et une plate-forme informatique qui ont été créés par Sun Microsystems en 1995. Beaucoup d'applications et de sites Web ne fonctionnent pas si Java n'est pas installé et leur nombre ne cesse de croître chaque jour. Java est rapide, sécurisé et fiable. Des ordinateurs portables aux centres de données, des consoles de jeux aux superordinateurs scientifiques, des téléphones portables à Internet. [16]

Notre choix s'est porté sur JAVA pour plusieurs raisons :

- La haute sécurité
- On peut utiliser le même code sur différentes plates-formes
- Langage orienté objet
- Langage simple
- Il est interprété
- Performances élevées ...etc

V.2.2. Environnement de développement intégré Netbeans

IDE est un acronyme pour « Integrated Development Environment ». Il s'agit donc d'un logiciel qui supporte le développeur dans son travail en lui offrant de nombreux outils : éditeur de code, compilateur, débogueur, etc.

Chaque langage de programmation possède souvent son ou ses IDE propres. C'est le cas de Java pour lequel on peut utiliser de nombreux IDE, par exemple JBuilder, Eclipse, WebSphere et NetBeans qui est celui que nous utiliserons.

NetBeans est un environnement de développement intégré gratuit et à code source ouvert destiné au développement d'applications sous Windows, Mac, Linux et Solaris.

L'environnement IDE simplifie le développement d'applications Web, d'entreprise, de bureau et mobiles utilisant les plates-formes Java et HTML5. Il offre également une assistance pour le développement d'applications PHP et C/C++.

Nous avons choisi Netbeans comme environnement de développement car :

- IL permet de développer et déployer rapidement et gratuitement des applications graphiques Swing, des Applets...etc., dans un environnement fortement personnalisable
- L'EDI Netbeans repose sur un noyau robuste et un système de plugins performant...etc.

V.2.3. Bibliothèques JGraphx et Swing

JGraphx est une bibliothèque Java qui permet de dessiner des graphes en 2D dans une application.

JGraphX est basé sur le principe Modèle/Vue, c'est-à -dire que vous voyez la représentation graphique d'un modèle. Ce modèle est basé sur une structure de données de type arbre.

Swing est une bibliothèque graphique pour le langage de programmation Java, faisant partie du package Java Foundation Classes (JFC), inclus dans J2SE. Swing constitue l'une des principales évolutions apportées par Java 2 par rapport aux versions antérieures.

Swing offre la possibilité de créer des interfaces graphiques identiques quel que soit le système d'exploitation sous-jacent.

V.3. Outils et langages de développement de l'application Web

V.3.1. Langage HTML

HTML signifie « HyperText Markup Language » qu'on peut traduire par « langage de balises pour l'hypertexte ». Il est utilisé afin de créer et de représenter le contenu d'une page web et sa structure. D'autres technologies sont utilisées avec HTML pour décrire la présentation d'une page (CSS) et/ou ses fonctionnalités interactives (JavaScript).

L'hypertexte désigne les liens qui relient les pages web entre elles, que ce soit au sein d'un même site web ou entre différents sites web. Les liens sont un aspect fondamental du Web. Ce sont eux qui forment cette « toile » (ce mot est traduit par web en anglais). En téléchargeant du contenu sur l'Internet et en le reliant à des pages créées par d'autres personnes, vous devenez un participant actif du World Wide Web.

V.3.2. Langage CSS

Les feuilles de styles (en anglais "Cascading Style Sheets", abrégé CSS) sont un langage qui permet de gérer la présentation d'une page Web. Le langage CSS est une recommandation du World Wide Web Consortium (W3C), au même titre que HTML ou XML.

Les styles permettent de définir des règles appliquées à un ou plusieurs documents HTML. Ces règles portent sur le positionnement des éléments, l'alignement, les polices de caractères, les couleurs, les marges et espacements, les bordures, les images de fond, etc.

Le but de CSS est de séparer la structure d'un document HTML et sa présentation. En effet, avec HTML, on peut définir à la fois la structure (le contenu et la hiérarchie entre les différentes parties d'un document) et la présentation. Mais cela pose quelques problèmes. Avec le couple HTML/CSS, on peut créer des pages web où la structure du document se trouve dans le fichier HTML tandis que la présentation se situe dans un fichier CSS.

V.3.3. Langage JavaScript

JavaScript (souvent abrégé en « JS ») est un langage de script léger, orienté objet, principalement connu comme le langage de script des pages web. Mais il est aussi utilisé dans de nombreux environnements extérieurs aux navigateurs web tels que Node.js, Apache CouchDB voire Adobe Acrobat. Le code JavaScript est interprété ou compilé à la volée (JIT). C'est un langage à objets utilisant le concept de prototype, disposant d'un typage faible et dynamique qui permet de programmer suVant plusieurs paradigmes de programmation : fonctionnelle, impératVe et orientée objet.

V.3.4. Langage PHP

PHP (officiellement, ce sigle est un acronyme récursif pour Hypertext Preprocessor) est un langage de scripts généraliste et Open Source, spécialement conçu pour le développement d'applications web. Il peut être intégré facilement au HTML.

En plus de ça, il apporte aussi :

- La collection de données
- La génération, dynamique des pages
- L'envoi et réception de cookies
- L'interfaçage avec un grand nombre de base de données (dbase, Mysql, Access, Ets)

V.3.5. Langage de programmation XML

XML (Extensible Markup Language), est un ensemble de règles (format ou langage standard) pour encoder des documents sous une forme lisible par une machine. Il est défini dans la spécification XML de sa première version (1.0) produite par le W3C en 1998, et d'autres spécifications connexes, toutes les normes ouvertes gratuites. Il est essentiellement destiné pour la publication et surtout l'échange de documents sur le Web. Son origine vient de HTML, héritier de SGML. XML est aussi général que SGML, mais simplifié.

L'objectif de l'utilisation de XML est de mettre en pratique la simplicité, la généralité et la convVialité des données Web. Ceci à travers un format de données textuel avec un support puissant.

XML est exploité dans des applications de commerce électronique et pour la recherche d'information à travers les moteurs de recherche généralisés.

Il faut noter que le langage XML est utilisé aussi dans l'outil de modélisation UML afin de structurer les modèles de procédés logiciels.

V.3.6. Le SGBD MySQL

MySQL a été lancé à l'origine en 1995. Depuis, il a connu quelques changements de propriétaire et de gestion, avant de se retrouver chez Oracle Corporation en 2010. Alors qu'Oracle est en charge maintenant, MySQL est toujours un logiciel open source, ce qui signifie que vous pouvez l'utiliser et le modifier librement.

Notre choix c'est porté sur le SGBD MySQL comme système de gestion de base de données (SGBD), car il répond largement à nos attentes notamment pour ses caractéristiques de performance, de fiabilité et de simplicité d'utilisation.

V.3.7. Apache

Apache est un logiciel de serveur web gratuit et open-source qui alimente environ 46% des sites web à travers le monde. Le nom officiel est Serveur Apache HTTP et il est maintenu et développé par Apache Software Foundation.

Il permet aux propriétaires de sites web de servir du contenu sur le web – d'où le nom « serveur web » -. C'est l'un des serveurs web les plus anciens et les plus fiables avec une première version sortie il y a plus de 20 ans, en 1995.

V.3.8. Bibliothèques JQuery, AJAX et Bootstrap

JQuery, est une bibliothèque JavaScript . Compatible avec l'ensemble des navigateurs Web, elle a été conçue et développée en 2006 pour faciliter l'écriture de scripts. Il s'agit du Framework JavaScript le plus connu et le plus utilisé. Il permet d'agir sur les codes HTML, CSS, JavaScript et AJAX et s'exécute essentiellement côté client.

JQuery permet, entre autres, de gagner en rapidité dans l'interaction avec le code HTML d'une page Web. Elle propose comme principales fonctionnalités :

- La manipulation du Document Object Model (DOM)
- La gestion des événements (mouvements de souris, clics, etc.) et de l'AJAX (architecture informatique)
- La création d'effets d'animation
- La manipulation des feuilles de style en cascade

AJAX (Asynchronous JavaScript + XML) n'est pas une technologie en soi, mais elle désigne une « nouvelle » approche utilisant un ensemble de technologies existantes, dont : HTML ou XHTML, les feuilles de styles CSS, JavaScript, le modèle objet de document (DOM), XML, XSLT, et l'objet XMLHttpRequest. Lorsque ces technologies sont combinées dans le modèle AJAX, les applications Web sont capables de réaliser des mises à jour rapides et incrémentielles de l'interface utilisateur sans devoir recharger la page entière du navigateur (d'une façon asynchrone). Les applications fonctionnent plus rapidement et sont plus réactives aux actions de l'utilisateur.

L'utilisation d'**AJAX** fonctionne sur tous les navigateurs Web courants : Google Chrome, Safari, Mozilla Firefox, Internet Explorer, Opera, etc.

Bootstrap est un framework développé par l'équipe du réseau social Twitter. Il utilise les principaux langages de développement web (HTML, CSS & Javascript), fournit aux développeurs des outils pour créer un site facilement.

Bootstrap est créé afin de permettre de développer des sites web qui peuvent s'adapter à toutes les résolutions d'écran (ordinateur, smartphone ou sur tablette). Il contient des formulaires, boutons, outils de navigation et autres éléments interactifs, aussi c'est un ensemble qui a des extensions JavaScript en option. On appelle ce type de framework un "Front-End Framework".

V.3.9. Environnement de développement intégré Adobe DreamWeaver 2019

V.3.9.1 Adobe Dreamweaver

(Anciennement Macromedia Dreamweaver) est un éditeur de site web de type WYSIWYG (What You See Is What You Get : ce que vous voyez est ce que vous obtiendrez).

Dreamweaver fut l'un des premiers éditeurs HTML de type tel écrit tel écran, mais également l'un des premiers à intégrer un gestionnaire de site. Ces innovations le propulsèrent rapidement comme l'un des principaux éditeurs de site web, aussi bien utilisable par l'apprenti que par le professionnel.

L'utilisateur peut choisir entre deux modes de conception :

- Le mode création qui permet d'effectuer la mise en page directement à partir d'outils simples.
- L'autre mode offre la possibilité d'afficher et de modifier directement le code (HTML ou autre) qui compose la page.
- On peut utiliser les deux modes d'affichage dans un affichage mixte.

Avec l'évolution des technologies de l'internet, Il offre aujourd'hui la possibilité de concevoir des feuilles de style.

Dreamweaver est édité par la société Adobe Systems et fait partie de la suite de développement Studio 8 de l'éditeur, qui comprend Macromedia Flash, Macromedia Fireworks (édition graphique) et Macromedia Coldfusion (serveur). Macromedia, qui éditait Dreamweaver auparavant, a été racheté par Adobe en décembre 2005.

Dans la suite, nous présentons notre outil de modélisation UML, nous détaillerons sa structure générale et la structure de ses objets.

V.4. Outil de modélisation UML

V.4.1. Structure des objets

Dans cette partie, nous présentons la structure générale des objets UML.

V.4.1.1. Objet classe

L'objet classe dans l'outil de modélisation UML est structuré en deux objets (Classe avec compartiments, Classe sans compartiment). Cet objet est généré grâce à la bibliothèque vue précédemment *Jgraphx,* avec l'ajout d'un style pour l'objet afin de lui donner un aspect d'une classe

comme celle du diagramme de classe, ce style utilise le même modèle du langage feuille de style en cascade (CSS) c'est-à-dire un ensemble de règles de mise en forme.

```
mxCell cell_parent = (mxCell) graph.insertVertex(parent, null, "classe" + nb_add_classe,
        arg0.getX(), arg0.getY(), 100, 65, "classe");


mxCell a = (mxCell) graph.insertVertex(cell_parent, "attribut", "", 0, 0, cell_parent.getGeometry().getWidth(), 25,
        "contenu_classe");


mxCell b = (mxCell) graph.insertVertex(cell_parent, "methode", "", 0, 45, cell_parent.getGeometry().getWidth(), 25,
        "contenu_classe");
```

Figure V.3 : Code source de génération d'un élément graphique (classe) avec JGraphx.

```
public void creer_style_classe() {
    graph.getModel().beginUpdate();

    mxStylesheet classestyle = new mxStylesheet();
    Map<String, Object> classe = new HashMap<String, Object>();

    classe.put(mxConstants.STYLE_SHAPE, mxConstants.SHAPE_SWIMLANE);
    classe.put(mxConstants.STYLE_OPACITY, 100);
    classe.put(mxConstants.STYLE_FOLDABLE, mxConstants.NONE);
    classe.put(mxConstants.STYLE_FONTSTYLE, mxConstants.FONT_BOLD);
    classe.put(mxConstants.STYLE_FONTSIZE, 12);
    classe.put(mxConstants.STYLE_STARTSIZE, 25);
    classe.put(mxConstants.STYLE_VERTICAL_ALIGN, mxConstants.ALIGN_MIDDLE);
    classe.put(mxConstants.STYLE_ALIGN,mxConstants.ALIGN_CENTER);
    classe.put(mxConstants.STYLE_FONTCOLOR, "#000000");
    classe.put(mxConstants.STYLE_STROKECOLOR, "#000000");
    classe.put(mxConstants.STYLE_RESIZABLE, true);
    classe.put(mxConstants.STYLE_FILLCOLOR, "#ffffff");
    classe.put(mxConstants.STYLE_PORT_CONSTRAINT, 0.5);
    classe.put(mxConstants.STYLE_ALIGN, mxConstants.ALIGN_CENTER);
    graph.setStylesheet(classestyle);
    style = graph.getStylesheet();
    style.putCellStyle("classe", classe);
    graph.getModel().endUpdate();
}
```

Figure V.4 : Style utilisé pour mettre en forme un élément graphique (classe).

Remarque

Tous les autres objets (lien, classe association, package, acteur, cas d'utilisation, …) ont une structure comme celle d'une classe. La différence de leur structure est dans les paramètres de celle-ci.

V.4.2. Interface utilisateur de l'outil

Dans cette partie, nous présentons l'interface utilisateur de l'outil de modélisation UML que nous avons développés.

Voici la fenêtre principale de notre outil UML :

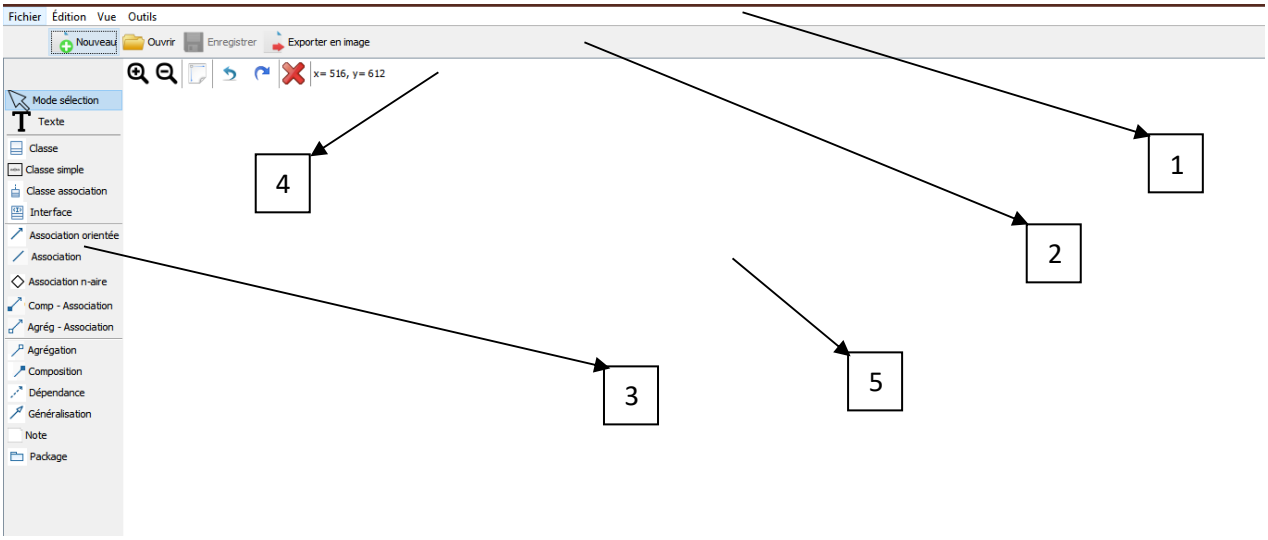


Figure V.5 : Fenêtre principale de l'outil UML

La **Figure V.5** montre la fenêtre principale de notre outil UML qui est divisé en plusieurs parties :

- **Barre de menu :** La barre de menu (1) affiche l'ensemble des fonctionnalités de l'outil UML, chaque menu contient des actions.
- **Barre d'outils :** La barre d'outils (2) affiche les fonctions générales de gestion d'un modèle (Création, Ouverture, enregistrement, Exportation).
- **Palette des éléments UML :** Cette palette (3) contient les éléments d'un diagramme à modéliser.

L'outil de modélisation UML a deux palettes, l'une contient les éléments d'un diagramme de classe et l'autre pour n diagramme de cas d'utilisation.

- **Barre d'outils de modélisation :** Cette barre (4) contient les fonctions tels que : zoomer, ajuster la taille du modèle à la fenêtre, supprimer un objet, annuler ou répéter une action et les coordonnés du curseur de la souris.
- **Zone de modélisation :** Cette zone (5) affiche le modèle (diagramme) généré ou en cours de génération. Cette zone est interactive.

V.4.2.1. Quelques exemples d'utilisation

Créer un nouveau modèle

Pour créer un modèle de procédé logiciel, il faut cliquer sur le menu **Fichier** ensuite **Nouveau,** une fenêtre s'affiche demandant qu'elle type de modèle à créer.

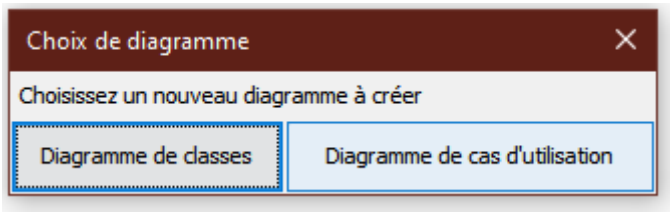


Figure V.6 : Fenêtre de sélection du type de modèle (diagramme) à créer.

Après avoir choisi le modèle à créer, sa palette correspondante va s'afficher à gauche de l'écran ainsi que les outils de dessin.

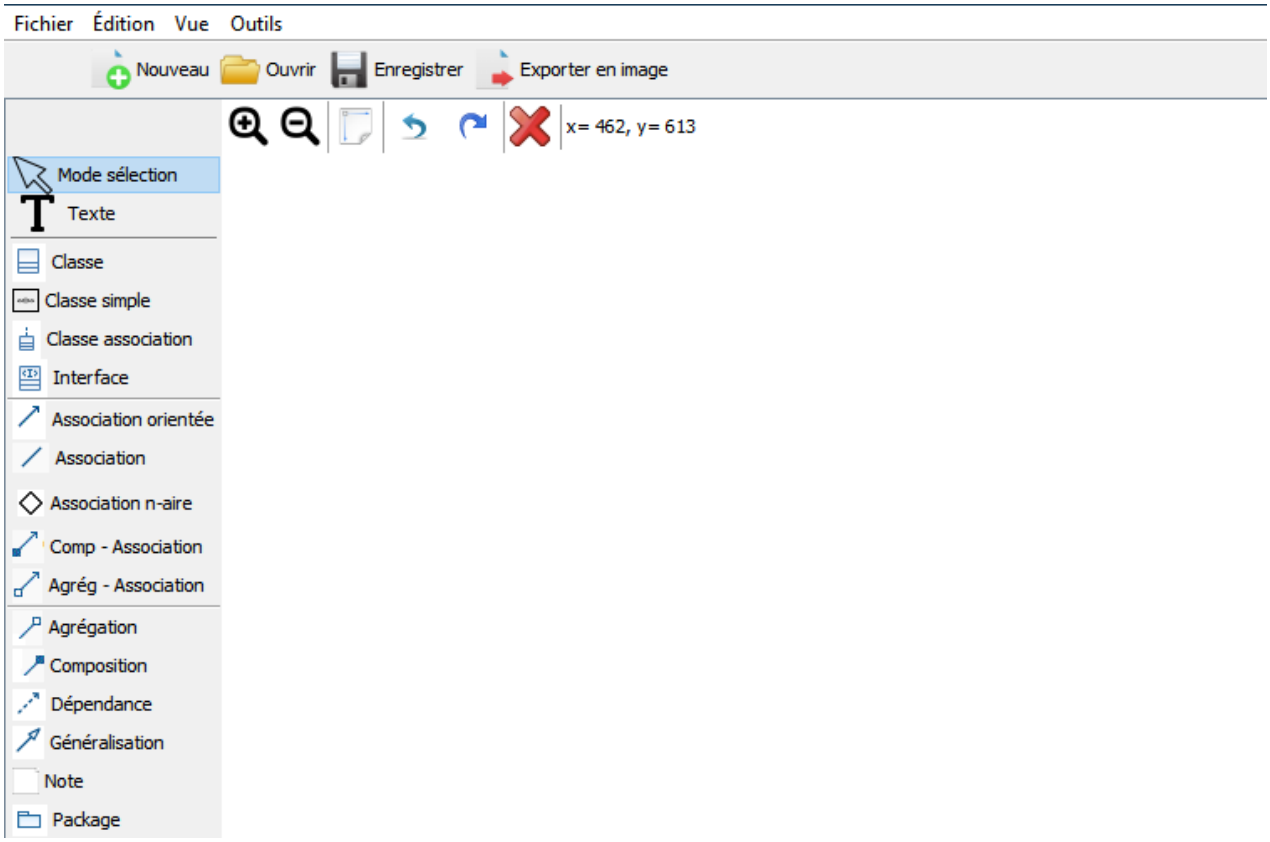


Figure V.7 : Fenêtre d'un nouveau modèle.

Insertion d'un objet

Pour insérer un objet dans la zone de dessin, il suffit de cliquer sur un objet dans une palette ensuite cliquer sur la zone de dessin pour l'insérer.

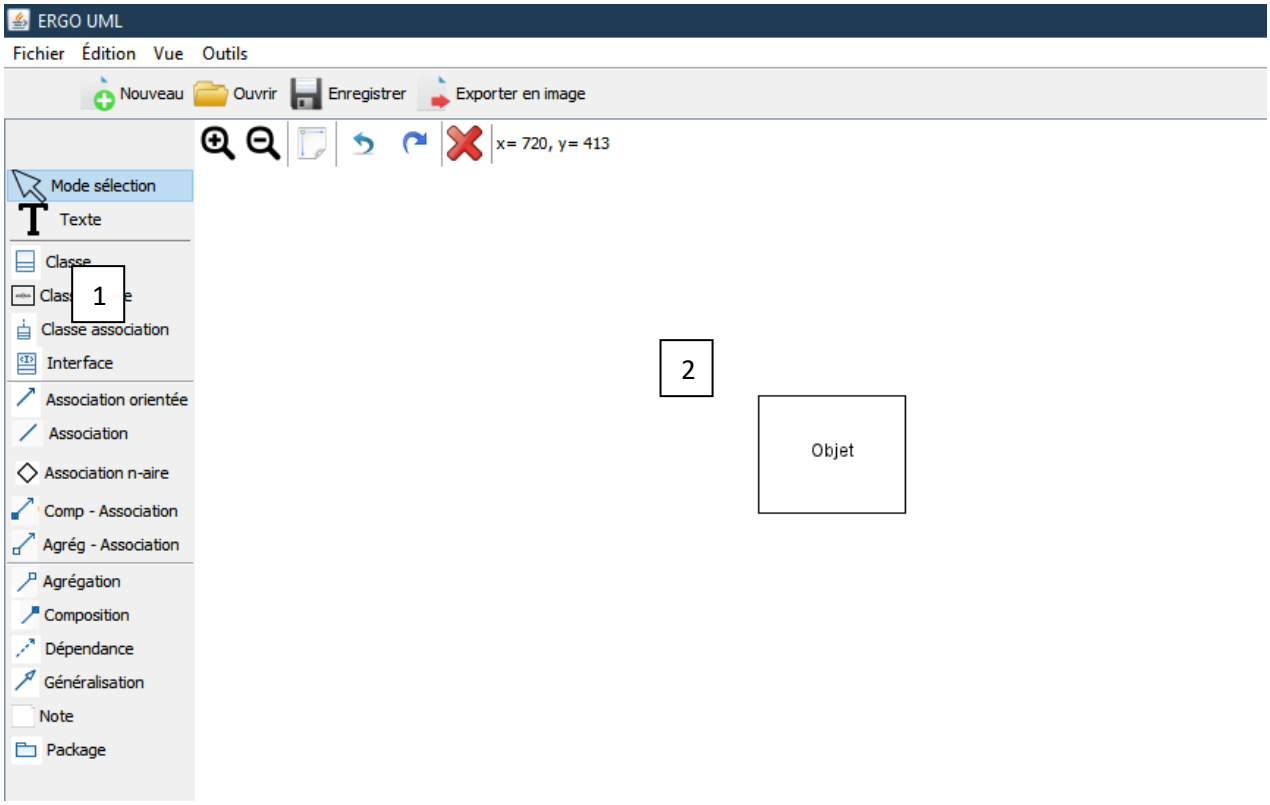


Figure V.8 : Insertion d'un objet dans la zone de dessin.

Insérer un lien

Pour insérer un lien (ligne), il suffit de sélectionner le type de lien dans la palette ensuite cliquer et maintenir le bouton gauche de la souris au centre d'un objet et glisser jusqu'à autre objet, une fois que le second objet est colorié en vert, relâché le bouton pour les connecter.

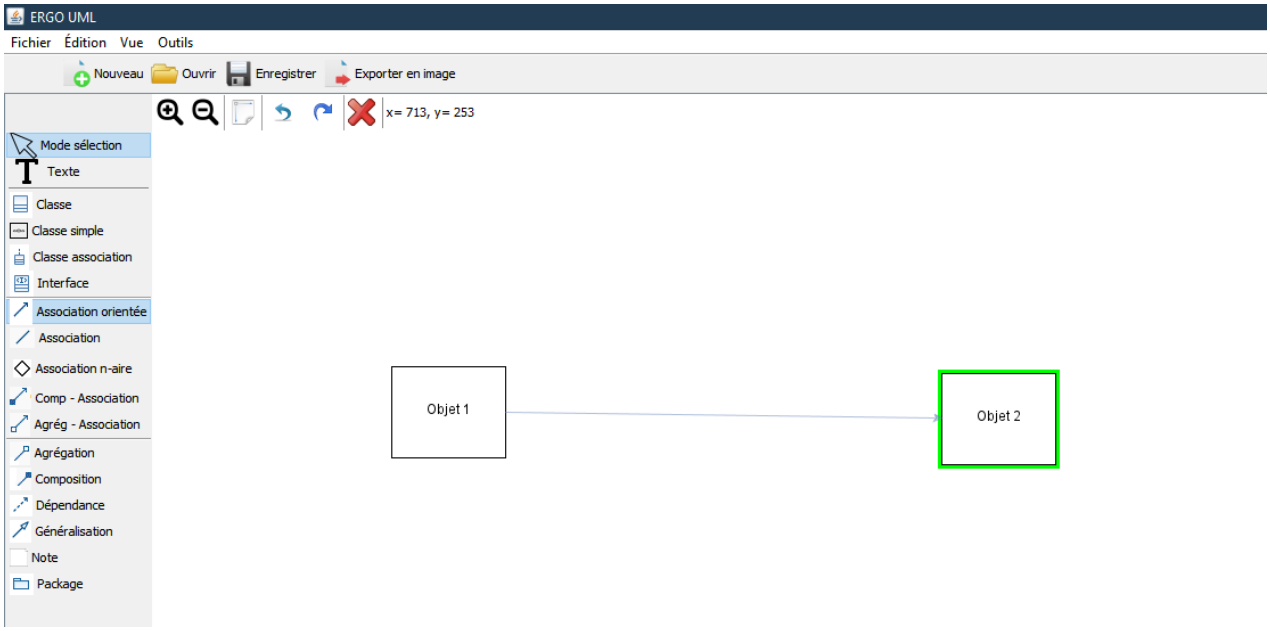


Figure V.9 Exemple de connexion des objets.

Supprimer un objet

Pour supprimer un objet dans la zone de dessin, il suffit de sélectionner un objet puis faire un clic droit et ensuite cliquer sur *Supprimer*, un message de confirmation s'affiche, il suffit de répondre sur *Oui* pour supprimer l'objet.

Notre outil de modélisation permet donc de dessiner des diagrammes (diagramme de classes et diagramme de cas d'utilisation), et on peut aussi dessiner un modèle de procédé logiciel. Voici un exemple concret de création d'un modèle de procédé logiciel grâce à la palette d'un diagramme de classe.

Enregistrement d'un modèle

L'enregistrement d'un modèle se fait grâce à la bibliothèque *JGraphx* qui permet de générée un fichier qui contient la structure du modèle écrite en langage XML. Chaque objet du modèle est enregistré sous forme de balise XML.

L'outil de modélisation UML enregistre deux extensions de fichiers (*.diagc* pour un modèle qui contient un diagramme de classe, *.dcu* pour un modèle qui contient un diagramme de cas d'utilisation).

Le modèle est aussi référencé dans la base de données de l'application web en tant que ligne dans la vue (table) des modèles (voir dans la section application web plus bas), cette référence est un chemin complet du fichier dans l'ordinateur afin qu'un concepteur ou chef de projet puisse ouvrir le modèle directement à partir de l'application web.

La figure V.11 affiche le code source de la fonction de l'enregistrement d'un modèle.

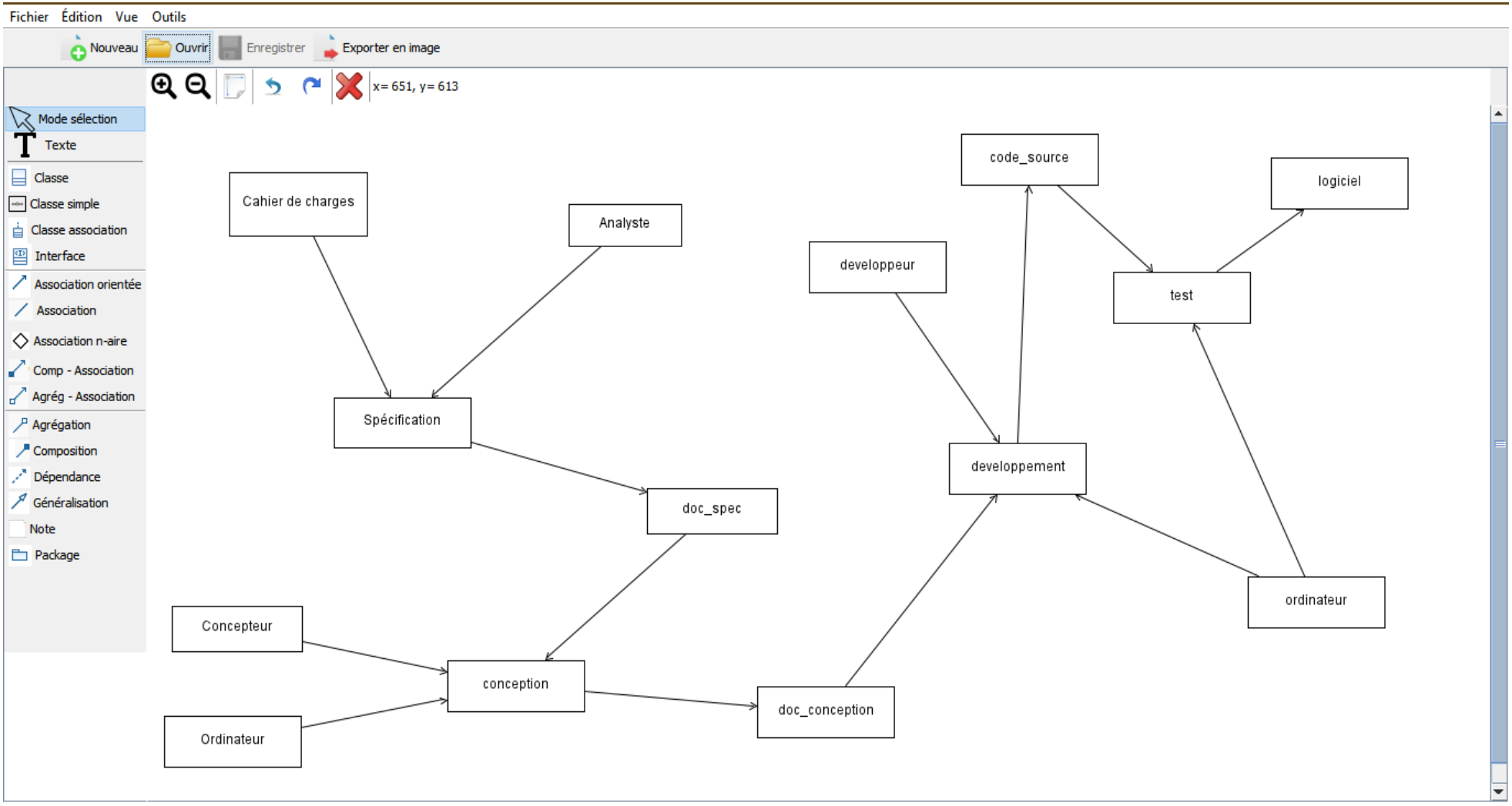


Figure V.10 : Exemple d'un modèle de procédé logiciel

```
private void saveDiagram()
{
    mxCodec codec = new mxCodec();
            JFileChooser fileName = new JFileChooser();
            fileName.addChoosableFileFilter(new FileNameExtensionFilter("Diagrammes (*.diagc, *.dcu)", "diagc", "dcu"));
            fileName.setAcceptAllFileFilterUsed(false);
            if(fichier_ouvert.length() == 0)
            {
                fileName.setDialogTitle("Enregistrer");
                fileName.showSaveDialog(graphComponent);
            }
            String xml = mxXmlUtils.getXml(codec.encode(graph.getModel()));
            try {
                if(palette_dc.isVisible())
                    mxUtils.writeFile(xml, (fichier_ouvert.isEmpty())?fileName.getSelectedFile().getAbsolutePath()+".diagc" : fichier_ouvert);
                else
                    mxUtils.writeFile(xml, (fichier_ouvert.isEmpty())?fileName.getSelectedFile().getAbsolutePath()+".dcu" : fichier_ouvert);
                btnNewButton_5.setEnabled(false);
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
                btnNewButton_5.setEnabled(true);
            }


}
```

Figure V.11 : Code source de la fonction d'enregistrement d'un modèle.

V.4.2.3 Ouverture d'un modèle

L'ouverture d'un modèle se fait grâce à la bibliothèque *JGraphx* qui comme dans l'enregistrement utilise le langage XML pour interpréter le modèle enregistré. Lorsque l'outil de modélisation UML ouvre un fichier, il lit le contenu du fichier du modèle et interprète la structure XML et dessine le modèle dans la zone correspondante.

```
String fichier = modele;
fichier_ouvert = fichier;
if(fichier.isEmpty())return;
loadStyles();

Document document = mxXmlUtils.parseXml( mxUtils.readFile(fichier));
mxCodec codec = new mxCodec( document);
codec.decode( document.getDocumentElement(), graph.getModel());

palette_dc.setVisible(fichier.endsWith(".diagc"));
palette_dcu.setVisible(fichier.endsWith(".dcu"));
if(palette_dc.isVisible())
    palette_dc.add(dependance_btn);
else
    palette_dcu.add(dependance_btn);
```

Figure V.12 : Code source d'ouverture d'un modèle.

V.5. Interface Web

Nous passons à présent à une autre interface, il s'agit de l'interface de l'application web qui joue un rôle crucial dans la gestion des projets et la collaboration des équipes. En ce qui concerne la gestion d'un projet, on doit définir un modèle de procédé logiciel à une entreprise. Ce processus passe par plusieurs étapes mais dans ce travail, on s'intéresse uniquement à la modélisation.

V.5.1 Inscription

Pour pouvoir modéliser des modèles de procédés logiciels à distance, il faut identifier les utilisateurs, chaque utilisateur est identifié par un profil. Et chaque profil a ses droits d'accès :

- **Concepteur de modèle :** appelé aussi l'administrateur, peut créer un projet et lui affecter un chef de projet.
- **Chef de projet :** Un chef de projet prend en charge un projet et inscrit des membres à un projet.
- **Membre :** Exécute ou instancie un modèle d'un projet une fois inscrit dans ce dernier.

Voici la fenêtre d'inscription des utilisateurs pour pouvoir accéder aux données :

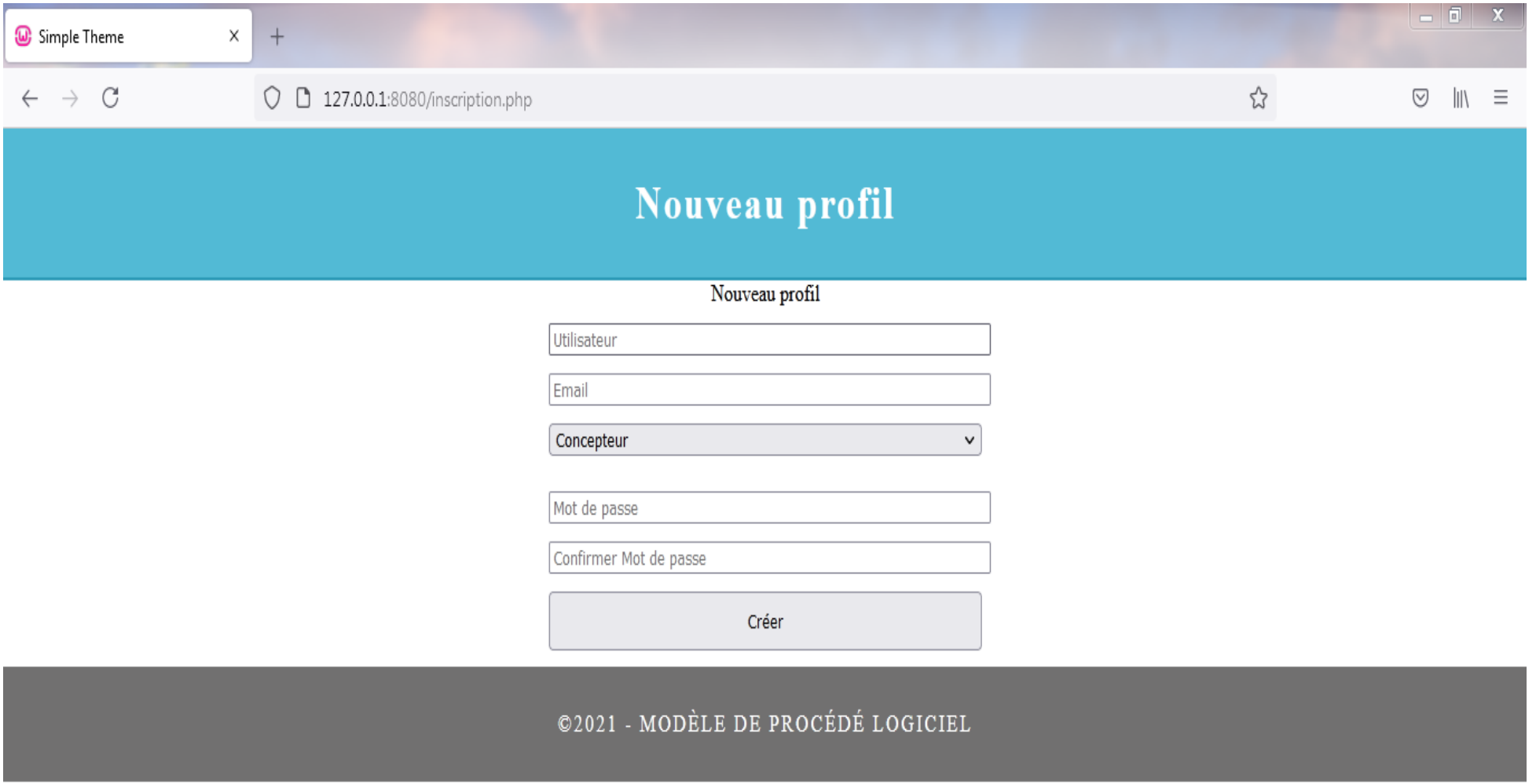


Figure V.13. Fenêtre d'inscription des utilisateurs

Le formulaire d'inscription est très simple, il suffit de saisir le nom de l'utilisateur (nom, prénom), son email, et son mot de passe pour s'inscrire et bien évidemment le type de profil. Une fois les champs saisis et en validant le formulaire, l'utilisateur sera redirigé vers sa page en fonction de son profil.

Si un utilisateur est déjà inscrit, il peut directement utiliser la page de connexion afin d'accéder à sa page de profil.

V.5.2. Profil concepteur de modèle

La page de profil d'un concepteur de modèle regroupe l'ensemble de ses fonctions, parmi ses fonctions : Gestion des modèles, Gestion des projets, Gestion des chefs de projets, envoyer un email à un chef de projet. La page de profil du concepteur de modèle contient un menu d'accès à la liste des modèles, à la liste des projets, à la liste des chefs de projets. Ainsi que 4 compteurs affichant le nombre de chaque élément disponible dans la base de données.

La figure V.14 affiche un aperçu du profil d'un concepteur de modèle.

Figure V.14 : Fenêtre de connexion d'un utilisateur.

Figure V.15 : Page de profil d'un concepteur de modèle.

V.5.2.1 Gestion des modèles

La fenêtre de modèles comprend l'ensemble des fonctionnalités de gestion des modèles, par exemple la création d'un modèle ou sa suppression. Une vue (table) des modèles existants s'affiche pour le concepteur afin de voir l'intégralité des modèles crées. Chaque ligne correspond à un modèle avec ses informations tels que son numéro de version.

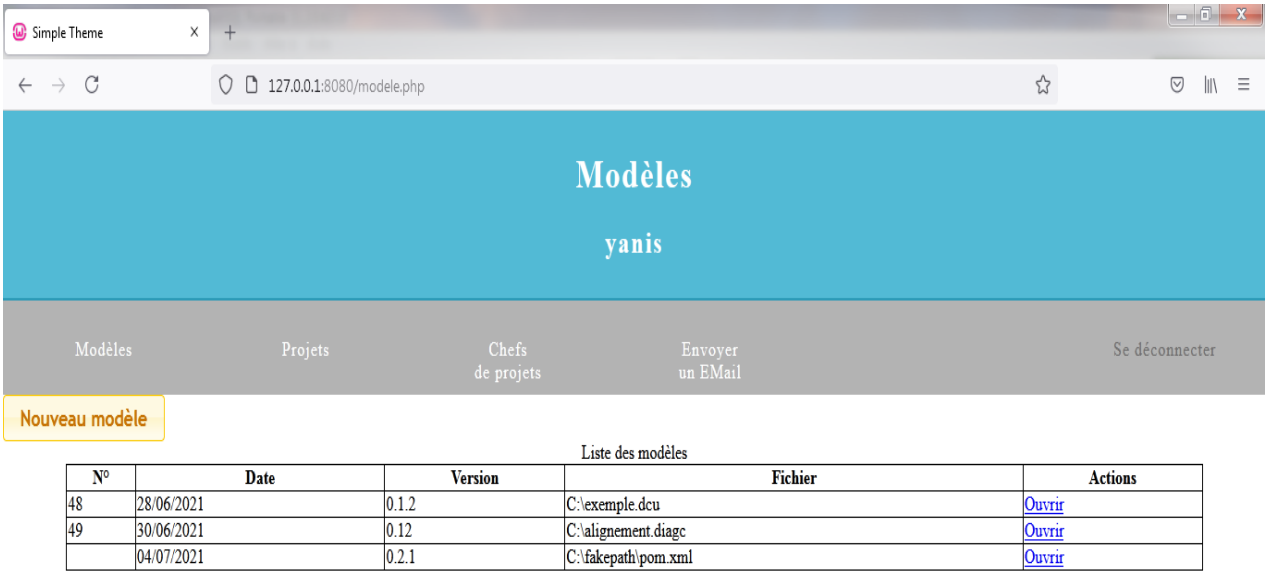


Figure V.16 : Fenêtre de gestion des modèles.

Pour créer un modèle, il faut commencer par créer un nouveau fichier à partir de l'outil de modélisation UML ensuite l'enregistrer, lorsqu'on clique sur le bouton *Enregistrer* dans l'outil de modélisation UML, celui-ci invite l'utilisateur à s'authentifier en tant que concepteur de modèle afin que ce dernier associe le modèle à son profil.

Si on ouvre un modèle de procédé logiciel avec l'outil de modélisation UML, et qu'on opère à des modifications ensuite on enregistre les modifications, l'outil demande de s'authentifier à nouveau pour des raisons de sécurité, une fois authentifier, une boite de dialogue s'affiche invitant à saisir le nouveau numéro de version.

La figure V.18 montre un aperçu de demande de modification du numéro de version.

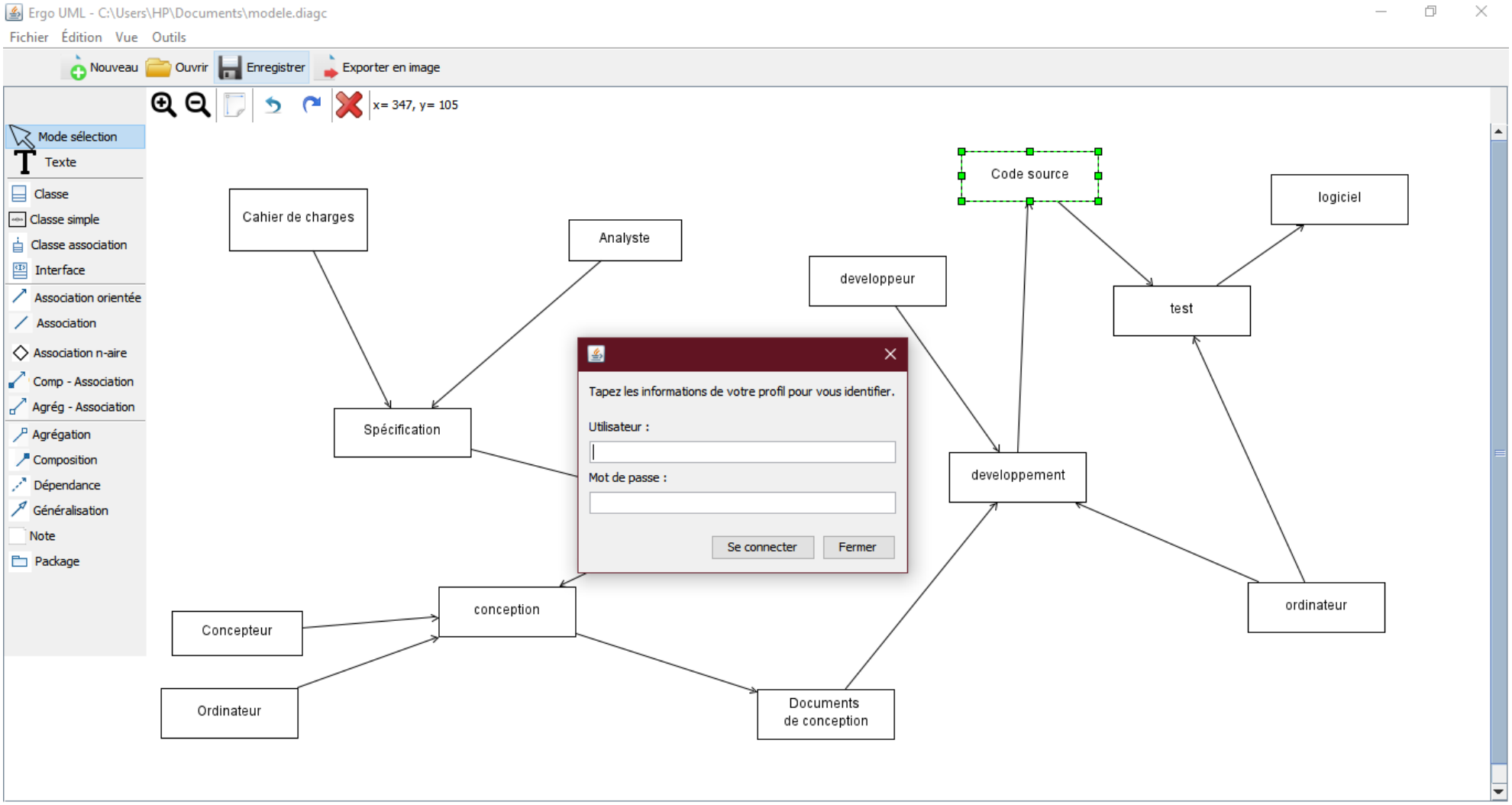


Figure V.17 : Boite d'authentification d'un utilisateur dans l'outil de modélisation UML

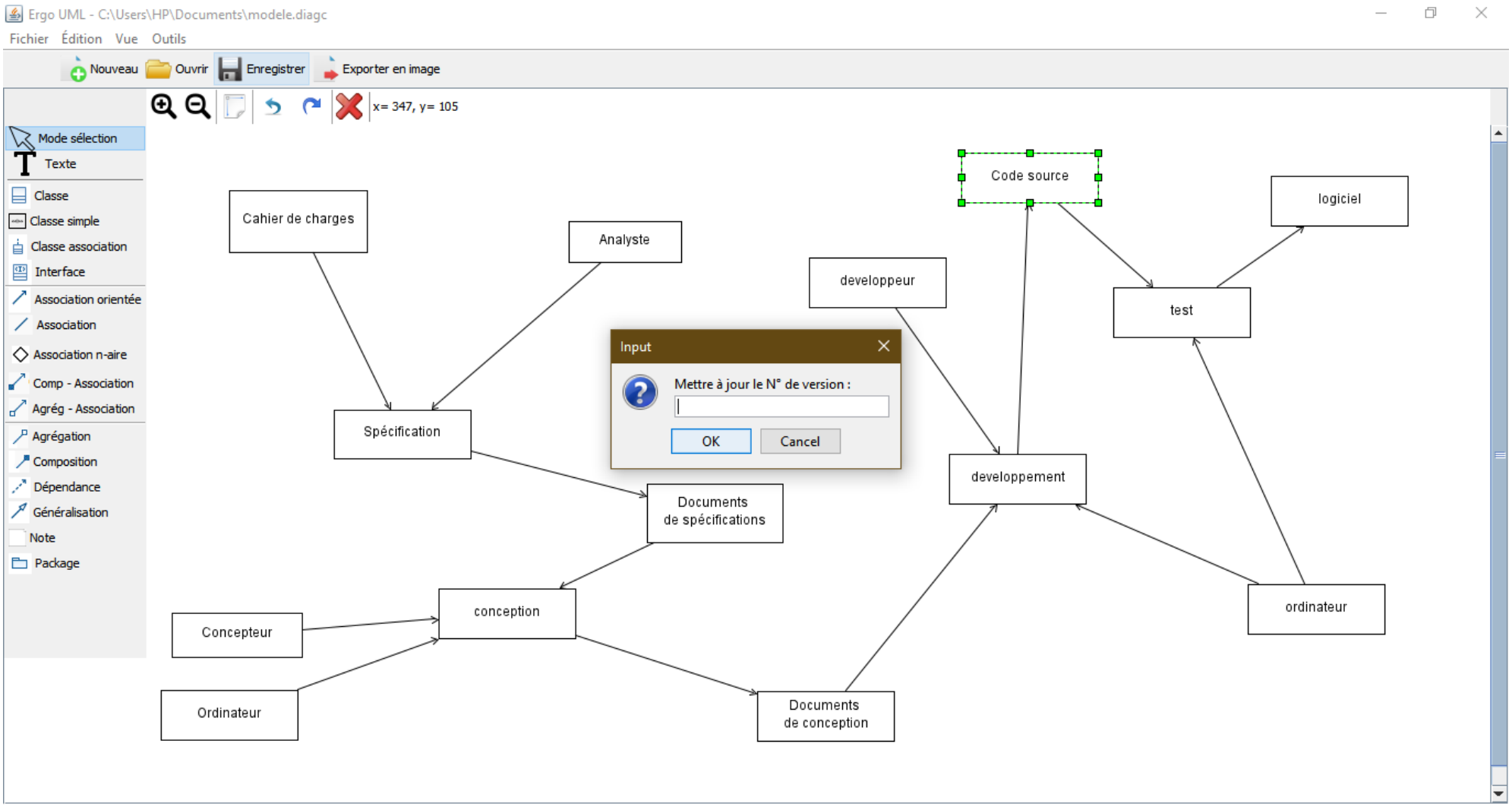


Figure V.18 : Mise à jour du numéro de version à partir de l'outil de modélisation.

V.5.2.2 Gestion des projets

La page « Gestion des projets » affiche l'ensemble des projets crées par un concepteur de modèle.

Dans cette page, le concepteur peut créer, modifier, supprimer un projet, associer un chef de projet à un projet et ouvrir un projet (pour afficher ses détails).

La figure V.19 montre l'aperçu de la page de gestion des projets.

V.5.3 Chef de projet

Un chef de projet inscrit des membres à un projet, ce dernier est désigné par un concepteur de modèle pour gérer un projet.

V.5.3.1 Inscrire un membre à un projet

Pour inscrire un membre à un projet, il faut enregistrer ses informations (nom, prénom, email, mot de passe, …) ensuite lui affecté un numéro (ID) d'un projet, une fois le formulaire remplit, il suffit de le valider afin d'enregistrer ses informations.

La figure V.21 affiche l'interface de gestion des membres.

V.5.4 Membre

Un membre est une personne qui fait partie de l'équipe qui exécute un modèle, il est inscrit par un chef de projet et lui désigne à projet à instancier et exécuter.

C'est le seul profil a disposé de droits d'accès restreints.

Il dispose de sa propre page pour gérer les projets ou il est inscrit.

La figure V.23 montre l'interface du profil membre.

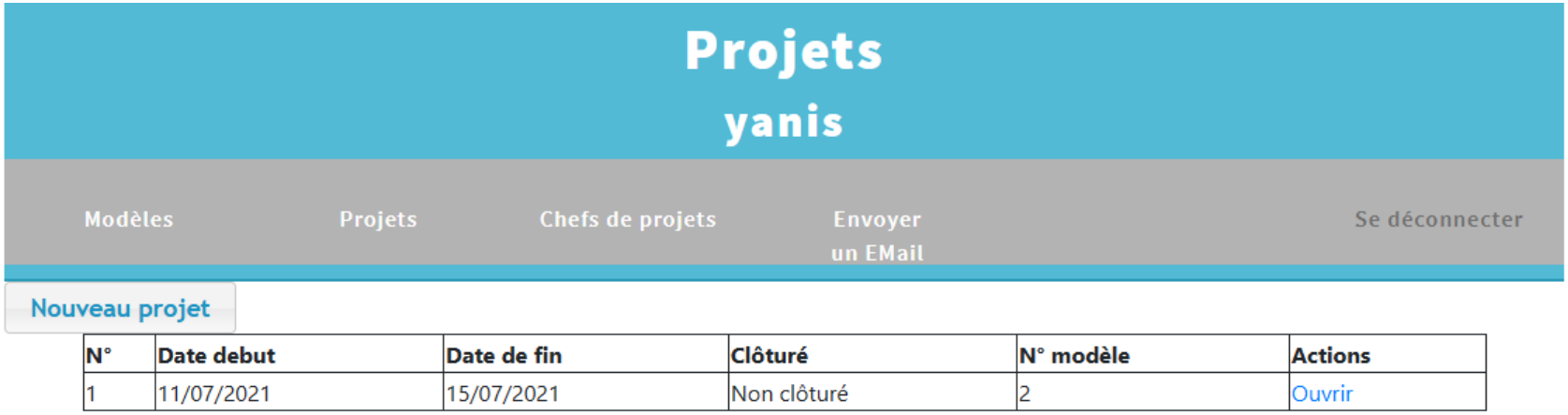


Figure V.19 : Page de gestion des projets.

Figure V.20 : Page pour un chef de projet.

Figure V.21 : Page de gestion des membres

La page suivante affiche le formulaire d'inscription d'un nouveau membre.

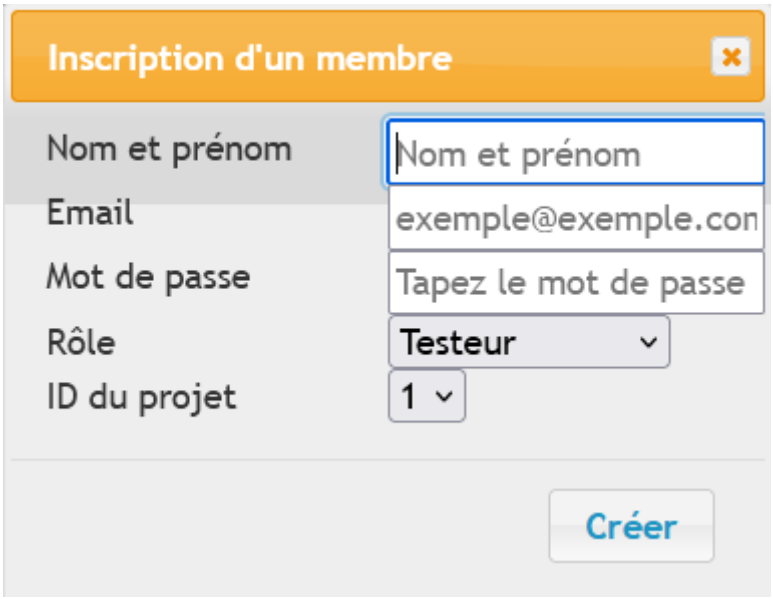


Figure V.22 : Formulaire d'inscription d'un membre.

N°	Date debut	Date de fin	Clôturé	N° modèle	Actions
1	11/07/2021	15/07/2021	Non clôturé	2	Ouvrir

Figure V.23 : Page Accès membre

V.6 Conclusion

Nous concluons ce chapitre avec un résumé de ce que nous avons vu tout au long de ce chapitre.

Dans la première partie du chapitre, nous avons défini les différents langages de programmation et différents outils et les bibliothèques de développement utilisés afin de développer notre outil de modélisation UML ainsi que la plateforme Web de gestion des projets.

Dans la seconde partie, nous avons décrit les différents objets (formes) générés par notre outil de développement pour réaliser un modèle de procédé logiciel.

Dans la dernière partie, nous avons présenté l'interface graphique de notre outil de développement UML et ses principales fonctionnalités et aussi l'interface graphique de notre plateforme Web

Conclusion générale et perspectives

Depuis la fameuse crise de logiciel des années 70 qui a fait naitre la discipline de Génie Logiciel, cette dernière n'a cessé de progresser et récemment la notion d'industrie du logiciel, séparée du matériel ou de son contexte système recouvre une grande variété d'activités répétitives : conception / développement de produit logiciel clés en main, regroupés sous le nom de procédés logiciel.

Notre travail s'inscrivait dans ce contexte d'élaboration d'un système de gestion des projets des procédés logiciels en utilisant une application web, et un outil de modélisation de procédé logiciel.

Nous avons dans ce sens, procédé à l'étude des différents concepts de modélisations, de dresser un état de l'art sur les environnements de développements, et les environnements génie logiciel centrés procédés. Nous avons aussi étudié la notation UML en expliquant le langage de modélisation UML et ses différents diagrammes, nous avons étudié en détail deux de ses diagrammes les plus utilisés (diagramme de classes, diagramme de cas d'utilisation)

Nous avons choisi le langage UML comme langage de modélisation car il est très utilisé dans le domaine de développement de logiciel, il est aussi normalisé dans le monde, sa notation graphique permet d'exprimer visuellement une solution objet, ce qui facilite la comparaison et l'évaluation de solutions.

L'objectif de notre travail a été de concevoir un outil de modélisation de modèle de procédé logiciel basé sur le langage UML, l'interface de notre outil de modélisation reprend le concept des autres outils de modélisations afin de simplifier la création des modèles de procédés logiciel.

Notre outil de modélisation UML comprend plusieurs fonctionnalités (Création d'un modèle, Mise à jour du numéro de version d'un modèle dans la base de données de l'application web, ouverture d'un modèle, enregistrement, Exporter un modèle au format image…).

Une deuxième interface a été aussi conçue pour cette fin, cette interface est une application web dynamique, elle permet de gérer les projets, les acteurs (responsable informatique appelé aussi concepteur de modèle, chefs de projets et des membres qui peuvent être : développeur, testeur, infographiste…).

Ce travail a été réalisé dans le cadre de notre projet de fin de cycle Master en Génie Logiciel, et il peut être exploité par des boites de développement (SSII), ou par des développeurs en équipe de produits logiciel

Des perspectives ultérieures sont prévues afin d'enrichir notre travail, il serait intéressant :

- De personnaliser les objets d'un modèle (jeu de couleurs, mise en forme du texte…).
- Attribuer des permissions d'utilisation des modèles (aspects sécurité).
- Ajout de service d'exécution, d'instanciation, d'évolution des modèles de procédé logiciel.
- Intégrer une Gestion Electronique des Documents (GED).

Bibliographie

[1] I. Sommerville, Le Génie Logiciel et ses applications, (3rd edition) ", Addison-Wesley, Wokingham, 1988.

[2] Khaled, Sellami. Vers un modèle de support de l'interopérabilité des fédérations de composants logiciels. Diss. Université de Béjaia-Abderrahmane Mira, 2005.

[3] Osterweil (L), software processes are software too. In: proceedings ofthe Int'l Conf. onS oft. Eng, PP.2-13-Monterey 1987, 2nd edition .

[4] Noran. R. J., "Working together to integrate case". IEEE Software, vol. 12, (1999), pp. 12–16.

[5] Waman S. Jawadekar, « Software Engineering, principles and practice ». The Mac Graw-Hill companies. Forth edition (2006).

[6] M. Amiour, Vers une fédération de composants interopérables pour les environnements centrés procédés logiciels", Thèse de doctorat, Université Joseph Fourier – Grenoble I, juin 1999.

[7] Sellami, Khaled, Nouredine Ait-Ikhlef, and Arezki Azzi. Modélisation de procédés logiciels à base de réseau de Petri : conception et réalisation d'un modèle de procédé logiciel pour le développement de logiciels de qualité. Éditions Universitaires Européennes, 2011.

[8] V. Ambriola, R. Conradi, A. Fuggetta, Assessing Process-Centred Software Engineering Environments" ACM transactions on Software Engineering and Methodology, Vol. 6, No 3, juillet 1997, pages 283-328.

[9] P. Coad, E. Yourdon, "Object Oriented Analysis", Second Edition, Yourdon Press, Prentice Hall, 1991.

[10] P. Coad, E. Yourdon, "Object Oriented Design", Yourdon Press, Prentice Hall, 1991.

[11] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object- Oriented Modeling and Design", Prentice Hall, 1991.

[12] P.A. Muller, "Modélisation Objet avec UML", Editions Eyrolles, 1996.

[13] G. Graw, V. Gruhn, "Process Management in-the-many", 4th European Workshop, EWSPT'95, avril 1995.

[14] N.S. Barghouti, B. Krishnamurthy, "An Open Environment for Process Modelling and Enactment", In [Schäfer 1993], pages 33-36.

[15] I.Z. Ben-Shaul, G.E. Kaiser, "Process evolution in the Marvel environment", dans [Schäfer 1993].

[16] S.Graine, UML 2 Modèle orienté objets. Les éditions l'Abeille — 2009.

Webographie

[17] www.java.com

[18] https://developer.mozilla.org/fr/

[19] www.oracle.com

Résumé

Ce document a été rédigé dans le but d'obtenir le diplôme en Master en informatique spécialité Génie logiciel. La problématique traitée est la modélisation des procédés logiciels basée sur le langage UML. Le domaine des procédés logiciels constitue l'une des préoccupations majeures en génie logiciel afin d'industrialiser le développement de logiciels complexes. Dans la littérature, plusieurs approches de modélisation ont alors été proposées.

Notre travail s'inscrit dans ce contexte, de conception d'un environnement de modélisation graphique des procédés logiciels utilisant l'approche du langage UML en raison de sa normalisation, sa simplicité et son interopérabilité. Une application web permettant de gérer ces modèles (création, modification, suppression, visualisation) et de les mettre à disposition d'une équipe de développement afin de collaborer entre eux, est associée à ce travail.

Mots-clés : *Modélisation des systèmes, Procédé logiciel, langage UML.*

Abstract

This document has been written with the aim of obtaining the Master's degree in Computer Science with Software Engineering specialty. The problem addressed is the modeling of software processes based on the UML language. The field of software processes is one of the major concerns in software engineering in order to industrialize the development of complex software. Several modelling approaches were then proposed.

Our work falls within this context, of designing a graphical modeling environment of software processes using the UML language approach because of its standardization, simplicity and interoperability. A web application to manage these models (creation, modification, deletion, visualization) and to make them available to a development team in order to collaborate with each other, is associated with this work.

Keywords: *Systems modeling, Software process, UML language.*

الملخص

تمت كتابة هذه المذكرة بهدف الحصول على دبلوم ماستير في الاعلام الالي تخصص في هندسة البرمجيات. الاشكالية التي تم التعامل معها هي نمذجة عمليات البرامج على أساس لغة UML. يعد مجال عمليات البرمجيات أحد الاهتمامات الرئيسية في هندسة البرمجيات من أجل تصنيع و تطوير البرمجيات المعقدة، تم بعد ذلك اقتراح العديد من مناهج النمذجة.

يقع عملنا في هذا السياق، لتصميم بيئة للنمذجة الرسومية لعمليات البرامج باستخدام نهج لغة UML نظرًا لتوحيدها وبساطتها وإمكانية التشغيل البيني. يرتبط بهذا العمل تطبيق ويب يجعل من الممكن إدارة هذه النماذج (الإنشاء والتعديل والحذف والتصور) وإتاحتها لفريق التطوير من أجل التعاون فيما بينها.

الكلمات المفتاحية: *نمذجة الأنظمة، عملية البرمجيات، لغة UML.*