CENTRE UNIVERSITAIRE - SALHI AHMED - DE NAAMA
INSTITUT DES SCIENCES & TECHNOLOGIES
DEPARTEMENT DE MATHEMATIQUES ET INFORMATIQUE

Mémoire de fin d'études

Master en informatique

Spécialité : systèmes informatiques

Thème

Conception et réalisation d'un outil CASE : UML vers JAVA

Présenté par :

- TASFAOUT Aicha
- LARADJI Fatima Zohra Hiba

Encadreur :

- Pr. BOUCHIHA Djelloul

Année universitaire : 2019/2020

Remerciement


Remerciement

C’est avec un grand plaisir qu’on réserve ces quelques lignes en signe de gratitude et de profonde reconnaissance à tous ceux qui, de près ou de loin, ont contribué à la réalisation et l’aboutissement à ce travail.

Tout d’abord, on remercie notre encadreur Monsieur **BOUCHIHA Djelloul** pour son soutient, son sérieux, sa disponibilité, ses précieux conseils et son aide tout au long de l’élaboration de ce travail.

Nous remercions les membres de jury qui ont bien voulu examiner et évaluer ce mémoire.

Nous nous acquittons, enfin, volontiers d’un devoir de gratitude et de remerciements à tous nos enseignants pour la qualité de l’enseignement qu’ils ont bien voulu nous prodiguer durant nos études, afin de nous fournir une formation efficiente.

Dédicace


Dédicace

Avec un énorme plaisir, un cœur ouvert et une immense joie, que je dédie mon travail

A mes chers parents pour tous leurs sacrifices, leur patience, leur soutien et leur encouragement.

A mes frères, mes sœurs et en particulier à **Zahra**.

A toute la famille **ZAOUI** et la famille **TASFAOUT**.

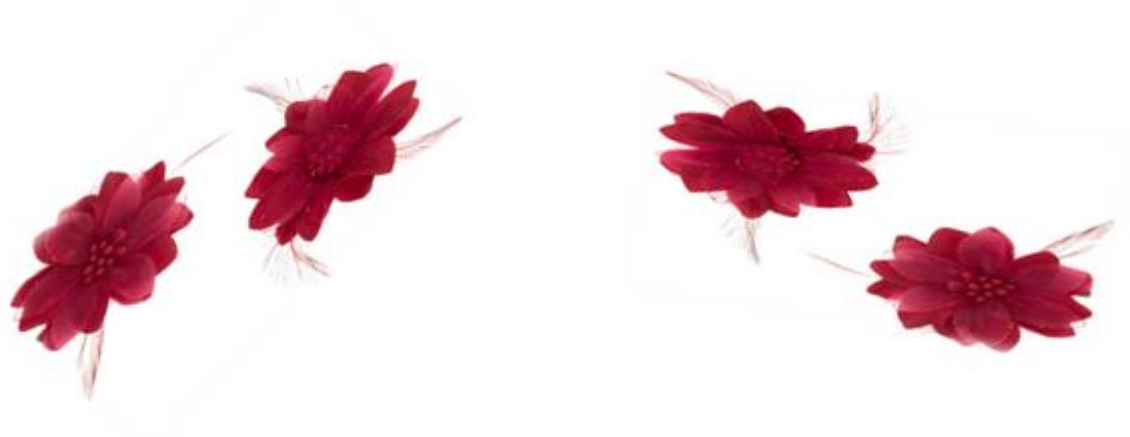
Sans oublier tous les professeurs que ce soit du primaire, du moyen, du secondaire et de l'enseignement supérieur.

A toutes personnes qui m'ont encouragé ou aidé au long de mon parcours universitaire.

Que ce travail soit l'accomplissement de vos vœux tant allégués, et le fruit de votre soutien infaillible.

Que dieu leur accorde santé et prospérité.

TASFAOUT Aicha



Dédicace

Avec un énorme plaisir, un cœur ouvert et une immense joie, que je dédie mon travail

A mes chers parents et en particulier à papa pour tous leurs sacrifices, leur patience, leur soutien et leur encouragement.

A mes frères et sœurs.

A toute la famille **FARES** et la famille **LARADJI**.

Sans oublier tous les professeurs que ce soit du primaire, du moyen, du secondaire et de l'enseignement supérieur.

A toutes personnes qui m'ont encouragé ou aidé au long de mon parcours universitaire.

Que ce travail soit l'accomplissement de vos vœux tant allégués, et le fruit de votre soutien infaillible.

Que dieu leur accorde santé et prospérité.

LARADJI Fatima Zohra Hiba

Résumé

Dans ce projet, un outil CASE a été réalisé pour aller à JAVA à partir d'UML. Pour cela, le langage UML est utilisé (langage de modélisation unifié) dans la partie de conception. L'objectif de ce travail est l'automatisation maximale de tout le processus ou d'une partie du processus de développement du logiciel, et aider les développeurs dans la production de leurs logiciels.

Mots clés : outil CASE, UML, JAVA, conception.

ملخص

في هذا المشروع، تم إنشاء برنامج CASE للانتقال من UML إلى JAVA. إذ تم استعمال اللغة UML في مرحلة التصميم. الهدف من هذا البحث هو جعل جزء من مراحل تطوير البرامج يتم بصفة آلية و مساعدة المبرمجين في إنتاج أنظمة البرمجيات.

الكلمات المفتاحية : برنامج CASE، UML، JAVA، تصميم.

Abstract

In this project, A CASE tool has been made to go to JAVA from UML. Therefore, the UML language is used (unified modeling language) in the design phase. The objective of this work is the maximum automation of the whole process or part of the software development process in order to help developers in the production of their software systems.

Keywords: CASE tool, UML, JAVA, design.

Table des matières

Liste des figures

Liste des tableaux

Acronymes

Notion	Signification
AGL	Atelier de génie logiciel
API	Application Programming Interface
CASE	Computer Aided Software Engineering
IBM	International Business Machine Corporation
JDBC	Java Database Conccectivity
MVC	Model-View-Controller
OMG	Object Management Group
OMT	Object Modeling Technique
OOSE	Object Oriented Software Engineering
PHP	HyperText preprocessor
SGBD	Système de Gestion de Base de Données
SQL	Structured Query Lauguage
UML	Unified Modeling Language

Introduction générale

Les logiciels et systèmes informatiques sont présents aujourd'hui dans tous les domaines de l'activité humaine (industrie, construction, communication…).

Génie logiciel c'est L'ensemble des activités de conception et de mise en œuvre des produits et des procédures tendant à rationnaliser la production du logiciel et son suivi.

Génie logiciel à des activité principales : L'analyse des besoins, La spécification global, La conception architecturale et détaillé, La programmation, L'implémentation, La validation et la vérification, le suivi et la maintenance.

Pour faciliter la réalisation de ces étapes et assurer la cohérence, on a qu'on appelle les outils CASE

Un outil CASE 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". Il faut noter ici que les outils CASE sont seulement des aides et ne permettent pas de donner une solution totale à touts les problèmes de développement du logiciel. Ils sont des outils de gestion pour le développement de logiciel (site 1, 2020)

Notre objectif est de réaliser un outil CASE UML2JAVA, Mais la question qui se pose, et qui trouveras sa réponse dans ce mémoire, est comment aller au java à partir d'UML?

La suite de ce mémoire est organisée en quatre chapitres, plus la conclusion :

Dans le premier chapitre intitulé « **Etat de l'art** », nous présenterons un état des connaissances : il s'agit d'une étude de l'existant plus choix des outils de développement.

Dans le deuxième chapitre intitulé « **Background** », nous présenterons les notions, outils et méthodes utilisés pour élaborer le projet et mettre en œuvre notre système.

Dans le troisième chapitre intitulé « **Conception du système**», nous présenterons la conception de notre système.

Le quatrième chapitre intitulé « **Implémentation** » introduit un guide d'utilisation avec un exemple illustratif pour faciliter la tâche aux utilisateurs de notre système. Il s'agit de concevoir une application JAVA qui facilitera le passage d'UML vers JAVA.

Enfin, nous clôturons ce mémoire par une conclusion dans laquelle nous synthétisons notre travail, et nous exposons quelques perspectives futures.

Chapitre I:
Etat de l'art

1. Introduction

Dans ce chapitre nous présentons un état des connaissances : il s'agit d'une étude de l'existant plus choix des outils de développement.

2. Les outils CASE

2.1. L'origine de « CASE »

Le terme CASE « Computer Aided Software Engineering » est synonyme de génie logiciel assisté par ordinateur. Cela signifie développement et maintenance de projets logiciels à l'aide de divers outils logiciels automatisés.

Le terme CASE a été d'abord appliqué à des outils qui fournissent un support pour les phases d'analyse et de conception du cycle de développement logiciel et la plupart des premiers outils étaient destinés à automatiser les méthodes structurées qui étaient disponibles.

La vision actuelle de CASE est celui d'un ensemble d'outils interdépendants qui prennent en charge tous les aspects du processus de développement logiciel. Elle inclue des outils qui prennent en charge des phases spécifiques du cycle de vie telles que les outils d'analyse et de conception, outils de génération de code, outils de test et les outils qui fournissent des fonctionnalités durant le cycle de vie tels que les outils de gestion de projet, de gestion de configuration et de documentation. (Kimberly et Dennis, 1992)

2.2. Objectif des outils CASE

Le développement des logiciels complexes fiables et efficaces demande l'effort et la collaboration de beaucoup de gens spécialistes et peut prendre des années pour l'accomplir. Des petites erreurs dans la logique des programmes peuvent avoir des conséquences énormes pour le développeur. Et ainsi 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.

2.3. Avantages des outils CASE

2.4. Fondamentalement, les outils CASE: (site 1, 2020)

- 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.

2.5. Fonctionnalité d'un outil CASE

conception générale du projet, étapes ou phases de réalisation ; composition et organisation de l'équipe projet ; calendrier, charges de travail, moyens et budgets ; conventions de nommage des données et des sous-ensembles de programmes ; structuration des données ; aide à l'édition de programmes dans différents langages ; compilation ; génération de code *optimisé* ; édition de liens ; aide aux tests et suivi des corrections ; bibliothèques de sous-ensembles pouvant être réutilisées dans plusieurs projets documentations ; gestion des versions successives ou des variantes d'un même programme. Les AGL couvrent donc un champ au-delà des environnements de développement intégrés. (site 3, 2020)

3. Quelques outils CASE existants

3.1. IBM Rational software (site 3, 2020)

IBM Rational Software Architect réalise ce qu'on appelle modèles de commentaires, associés au code technique d'attribut. L'outil fournit une API pour spécifier le code processus de génération. Cela signifie que Rational Software Architect est un générateur basé sur API. Génération de code en ligne technique peut également être associée à cet outil car elle prend en charge le langage de programmation C ++, qui a le "define" instruction.

3.2. SPARX Entreprise Architect (site 3, 2020)

Possède le langage intégré pour écrire des modèles. C'est pourquoi cet outil peut être associé à des modèles et à une technique de filtrage ainsi qu'à une technique de génération de code basée sur l'API. Sparx Enterprise Architect prend également en charge C ++ et certains autres langages de programmation, qui sont connectés à la technique de génération de code en ligne. L'outil prend en charge au moins la documentation JavaDoc, ce qui signifie que Sparx Enterprise Architect pourrait être associé à la technique des attributs de code.

3.3. Visual Studio 2010 (site 3, 2020)

donne la possibilité de définir des modèles de texte à l'aide de fragments de code C # . Cela permet d'associer Visual Studio 2010 aux modèles et au filtrage ainsi qu'aux techniques de génération basées sur l'API. La technique de génération de code en ligne peut également être associée à cet outil en raison de la prise en charge du langage de programmation C ++. Le langage de programmation C # a la capacité de générer automatiquement de la documentation (documentation XML), qui est la caractéristique de la technique des attributs de code.

3.4. Eclipse (site 3, 2020)

Eclipse offre également au développeur la possibilité d'écrire des modèles, ce qui lui permet de réaliser les modèles et la technique de génération de code de filtrage. Eclipse possède également ce que l'on appelle le mécanisme de génération de code de workflow, qui est fondamentalement très similaire aux principes de la technique de traitement de trame. L'outil prend en charge certains langages de programmation tels que C ++, ce qui permet de

l'associer à la technique de génération de code en ligne. La technique des attributs de code est également liée à Eclipse en raison de sa capacité à générer automatiquement la documentation du programme, telle que JavaDoc.

Le tableau suivant montre la classification des outils relatifs par rapport aux techniques de génération de code :

Technique de génération de code	Outils			
	IBM Rational software Architect 8.0	Sparx Entreprise Architect 9	MS Visual Studio 2010	Eclipse 3.7.2
Modèles et filtrage		X	X	X
Modèles et méta-modèle				
Processeurs de trame				X
Générateur basé sur l'API	X	X	X	
Génération en ligne	X	X	X	X
Attributs de code	X	X	X	X

Tableau I.1 : classification des outils relative aux techniques de génération de code

Le tableau I.1 représente la classification des outils relative par rapport aux techniques de génération de code. Dans ce tableaux on a six Technique de génération de code et on compare entre quatre outils, les techniques sont modèles et filtrage, Modèles et méta-modèle, Processeurs de trame, Générateur basé sur l'API, Génération en ligne et Attributs de code. Les outils sont IBM Rational software Architect 8.0, Sparx Entreprise Architect 9, MS Visual Studio 2010, Eclipse 3.7.2.

- IBM Rational software Architect 8.0 utilise 3 techniques : Générateur basé sur l'API, Génération en ligne et Attributs de code
- Sparx Entreprise Architect utilise 4 techniques : modèles et filtrage, Générateur basé sur l'API, Génération en ligne et Attributs de code
- MS Visual Studio 2010 utilise 4 techniques : modèles et filtrage, Générateur basé sur l'API, Génération en ligne et Attributs de code
- Eclipse 3.7.2. utilise 4 techniques : modèles et filtrage, Processeurs de trame, Génération en ligne et Attributs de code

Alors ces technique séparé sur 2 approches : Approche basée sur des modèles (template-based approach), Approche basée sur les visiteurs (Visitor-based approach) , les deux figures suivantes montrent la Taxonomie de chaque approche :

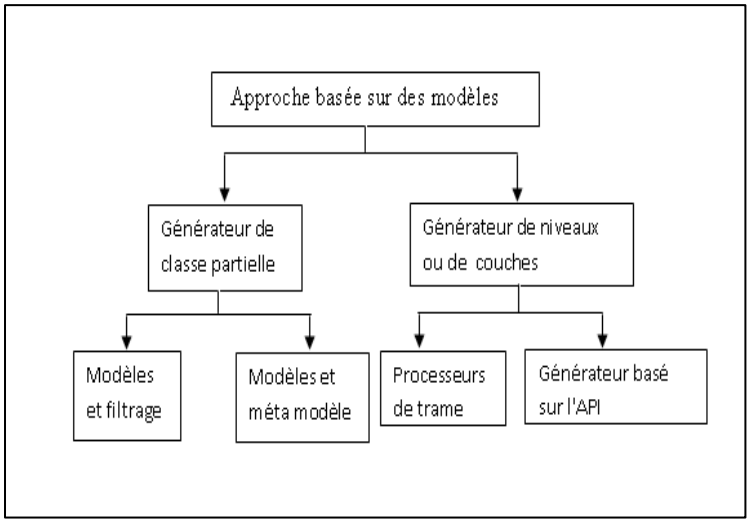


Figure I.1 : Taxonomie d'approche basée sur un modèle

Les modèles et le filtrage ainsi que les modèles et les techniques de méta modèle sont associés au générateur de classes partielles car il fonctionne avec des modèles. Ces techniques sont également les plus simples - elles permettent de générer uniquement la carcasse d'un code système. (Site 3, 2020) Les générateurs de niveaux ou de couches ont un mécanisme plus complexe, c'est pourquoi ils implémentent des processeurs de trame et des générateurs basés sur des API.

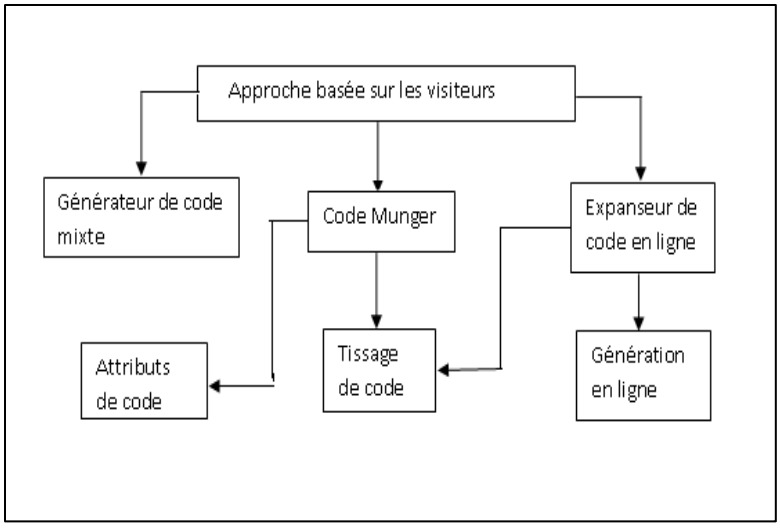


Figure I.2 : Taxonomie de l'approche basée sur les visiteurs

La technique d'attribut de code est associée aux générateurs de type code munger en raison de leur capacité à créer automatiquement une documentation de programme. (Site 3, 2020) La technique du générateur en ligne est implémentés dans les générateurs de type d'extension de code en ligne car ils ne changent pas la forme du code source d'entrée mais y ajoutent de nouveaux fragments.

4. Conclusion

Dans ce chapitre, nous avons vu un état des connaissances, passant maintenant au chapitre suivant concernant le background.

Chapitre II: Background

1. Introduction

Dans ce chapitre nous introduisons deux notions importantes liées à notre système. Comme notre système, déjà nommé UML2JAVA, permet le passage d'UML vers JAVA, alors ce chapitre a été consacré à ces deux notions UML et JAVA.

2. UML (Unified Modeling Language)

2.1. Définition

Selon OMG (Object Management Group), UML est un langage visuel dédié à la spéciation, la construction et la documentation des artefacts d'un système logiciel.

UML, c'est l'acronyme anglais de « Unified Modeling Language ». On le traduit par «Langage de modélisation unifié».

UML est donc un langage visuel constitué d'un ensemble de schémas, appelés des diagrammes, qui donnent chacun une vision différente du projet à traiter. UML nous fournit donc des diagrammes pour représenter le logiciel à développer : son fonctionnement, sa mise en route, les actions susceptibles d'être effectuées par le logiciel, etc.

Encore, UML est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est couramment utilisé en développement logiciel et en conception orientée objet.

L'UML est le résultat de la fusion des précédents langages de modélisation objet : Booch, OMT et OOSE. Principalement issu des travaux de GradyBooch, James Rumbaugh et Ivar Jacobson, UML est à présent un standard adopté par l'Object Management Group (OMG). (Booch et al., 2000).

2.2. Les diagrammes UML2

UML2 est constitué de 13 diagrammes qui représentent chacun un concept du système ou logiciel. Ces diagrammes sont classés en deux catégories : statiques et d'interaction.

2.2.1. Les modèles statiques (structurels)

Les modèles statiques sont :

- **Diagramme de classes :**

Un diagramme de classes est un graphe d'éléments connectés par des relations. Un diagramme de classes est donc une vue graphique de la structure statique d'un système (car on ne tient pas compte du facteur temporel dans le comportement du système), exprimée en termes de classes et de relations entre ces classes.

Une classe décrit un ensemble d'objets, et une association décrit un ensemble de liens; les objets sont des instances des classes, et les liens sont des instances des associations (Piechocki, 2006).

Diagramme de classes = classes + relations
Objet = instance d'une classe

- **Diagramme d'objet :**

Le diagramme d'objets permet d'éclairer un diagramme de classes en l'illustrant par des exemples. Il est, par exemple, utilisé pour vérifier l'adéquation d'un diagramme de classes à différents cas possibles.

Un diagramme d'objets représente des objets (i.e. instances de classes) et leurs liens (i.e. instances de relations) pour donner une vue figée de l'état d'un système à un instant donné. Un diagramme d'objets peut être utilisé pour :

• Illustrer le modèle de classes en montrant un exemple qui explique le modèle.

• Préciser certains aspects du système en mettant en évidence des détails imperceptibles dans le diagramme de classes.

• Exprimer une exception en modélisant des cas particuliers ou des connaissances non généralisables qui ne sont pas modélisés dans le diagramme de classes.

• Prendre une image d'un système à un moment donné.

Le diagramme de classes modélise les règles et le diagramme d'objets modélisent les faits (Audibert, 2009).

- **Diagramme de déploiement :**

Un diagramme de déploiement décrit la disposition physique des ressources matérielles qui composent le système, et montre la répartition des composants sur ces matériels. Chaque ressource étant matérialisée par un nœud. Le diagramme de déploiement précise comment les composants sont répartis sur les nœuds et quelles sont les connexions entre les composants ou les nœuds. Les diagrammes de déploiement existent sous deux formes: spécification et instance.

Le tableau suivant décrit les différents éléments d'un diagramme de déploiement (Galland, 2002) :

Tableau II.1: Eléments d'un diagramme de déploiement

Elément	Définition
Nœud	Chaque ressource matérielle est représentée par un nœud.
Supports de communication	Les supports de communication sont symbolisés par des relations entre les nœuds.

- **Diagramme de structure composite :**

Le diagramme de structure composite permet de décrire des collaborations d'instances (de classes, de composants…) constituant des fonctions particulières du système à développer.

- **Diagramme de composants :**

Ce diagramme décrit les composants et leurs dépendances dans l'environnement de réalisation. Il s'agit d'une vue statique de l'implémentation du système illustrant les choix de réalisation. Le tableau suivant décrit les différents éléments d'un diagramme de composants (Galland, 2002) :

Tableau II.2: Eléments d'un diagramme de composants

Elément	Définition
Composant	Elément physique représentant une partie de l'implémentation du système.
Module	Représente une unité pour le regroupement et la manipulation des composants.
Dépendance	Représente les relations de dépendances entre composants/modules.

- **Diagramme de paquetages :**

Le diagramme de paquetages est un diagramme structurel (statique) d'UML qui représente les paquetages (ou espaces de noms) composant un système, ainsi que les relations qui lient ces différents paquetages.

Ce diagramme est utilisé pour séparer le modèle en conteneurs logiques, et décrire leurs interactions à un haut niveau (Henocque, 2008).

2.2.2. Les modèles d'interactions ou dynamiques (de comportement)

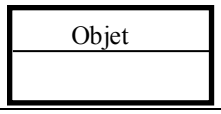
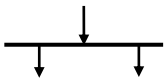
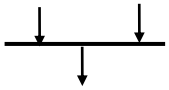
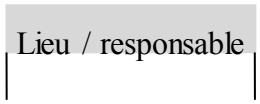
Les modèles dynamiques sont :

- **Diagramme d'activité :**

Le diagramme d'activité n'est autre que la transcription dans UML de la représentation du processus telle qu'elle a été élaborée lors du travail qui a préparé la modélisation : il montre l'enchaînement des activités qui concourent au processus.

Tableau II.3: Notations pour le diagramme d'activité

Elément/Notation	Description
Début	Point de départ
Action/Activité	Une activité est quelque chose qui se passe dans le

	processus (dans le workflow). C'est une action, un événement… effectuée par une personne, un ordinateur…
Transition 	Une transition représente le passage d'une activité à une autre.
Nœud d'objet : 	Objet produit ou utilisé par des actions/activités. Permet de modéliser les flots de données ou les flots d'objets.
Débranchement : 	Une transition entrante et plusieurs transitions parallèles sortantes et/ou des flots d'objets.
Jointure : 	Plusieurs transitions entrantes et/ou flots d'objets; une transition sortante. La poursuite des activités n'a lieu que lorsque tous les flots entrants ont atteint la jointure.
Swimlanes 	Dans les diagrammes d'activités, il est fréquent de montrer qui ou quoi est responsable de l'exécution d'une activité. C'est pour cette raison que le champ de responsabilité est partitionné, et que les activités sont placées dans les différentes partitions.
Alternative 	Le diagramme d'activité introduit un symbole pour la décision (ou l'alternative : if/switch).
Etat terminal : 	fin du processus

Remarque :

Un diagramme d'activité a toujours un et un seul point/état de départ/initial mais peut avoir plusieurs points/états finaux/terminaux (Audibert, 2009).

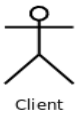
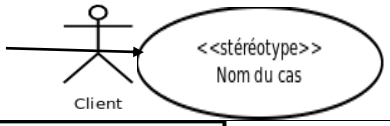
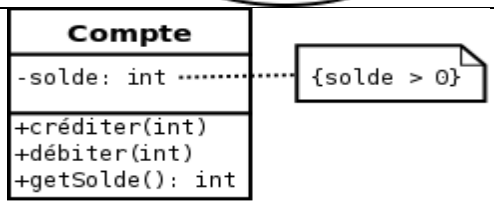
- **Diagramme de cas d'utilisations :**

Un diagramme de cas d'utilisation capture le comportement d'un système, d'un sous-système, d'une classe ou d'un composant tel qu'un utilisateur extérieur le voit. Il scinde la fonctionnalité du système en unités cohérentes, les cas d'utilisation, ayant un sens pour les acteurs. Les cas d'utilisation permettent d'exprimer le besoin des utilisateurs d'un système ; ils sont donc une vision orientée utilisateur de ce besoin au contraire d'une vision informatique. Il ne faut pas négliger cette première étape pour produire un logiciel conforme aux attentes des

utilisateurs. Pour élaborer les cas d'utilisation, il faut se fonder sur des entretiens avec les utilisateurs.

Le diagramme de cas d'utilisation représente la structure des grandes fonctionnalités nécessaires aux utilisateurs du système. C'est le premier diagramme du modèle UML, celui où s'assure la relation entre l'utilisateur et les objets que le système met en œuvre (Audibert, 2009).

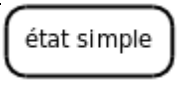
Tableau II.4: Notations pour le diagramme de cas d'utilisation

Elément/Notation	Description
	Un acteur est l'idéalisation d'un rôle joué par une personne externe, un processus ou une chose qui interagit avec un système.
	Un cas d'utilisation est une unité cohérente représentant une fonctionnalité visible de l'extérieur.
	Une association entre un acteur et un cas d'utilisation est représentée par une ligne pleine.
	Une note contient une information textuelle comme un commentaire, un corps de méthode ou une contrainte.

- **Diagramme d'états-transitions :**

Le diagramme d'états-transitions représente la façon dont évoluent (*i.e.* cycle de vie) les objets appartenant à une même classe (Audibert, 2009). La modélisation du cycle de vie est essentielle pour représenter et mettre en forme la dynamique du système.

Tableau II.5: Notations pour le diagramme d'états-transitions (Audibert, 2009)

Elément/Notation	Description
	Abstraction d'un moment de la vie d'une entité pendant lequel elle satisfait un ensemble de conditions.
	Changement d'état.
	État initial : Initialisation du système, exécution du constructeur de l'objet.
	État final : Fin de vie du système, destruction de l'objet

- **Diagramme de communication :**

Diagramme de communication (nommé diagramme de collaboration dans UML1) - utilise trois types de concepts :

- Des instances de classes (objets du système étudié) qui interviennent lors du scénario.
- Des acteurs qui reçoivent ou émettent des événements lors du scénario.
- Les appels aux méthodes ou flux d'information vers un acteur externe.

Un diagramme de communication contient uniquement (Johnen, 2017) :

- Des instances de classe (objets) qui sont dans le diagramme de classes.
- Des instances d'acteurs qui sont dans le diagramme de contexte statique.
- Un événement (flux) reçu par un objet d'une classe correspond à l'appel d'une opération/méthode de cette classe (elle porte le même nom que l'événement ou flux).

Tableau II.6: Notations pour le diagramme de communication (Johnen, 2017)

Elément/Notation	Description
Objet	Des instances de classes.
	Un lien est une connexion entre deux objets, qui indique qu'une forme de navigation et de visibilité entre eux est possible. Autrement dit, un lien permet d'acheminer des messages dans un sens ou dans l'autre.
	Chaque message entre objets est représenté par une expression, une flèche indiquant sa direction, et un numéro indiquant sa place dans la séquence.

- **Diagramme de séquence :**

Le diagramme de séquence représente la succession chronologique des opérations réalisées par un acteur. Il indique les objets que l'acteur va manipuler et les opérations qui font passer d'un objet à l'autre.

Exemple :

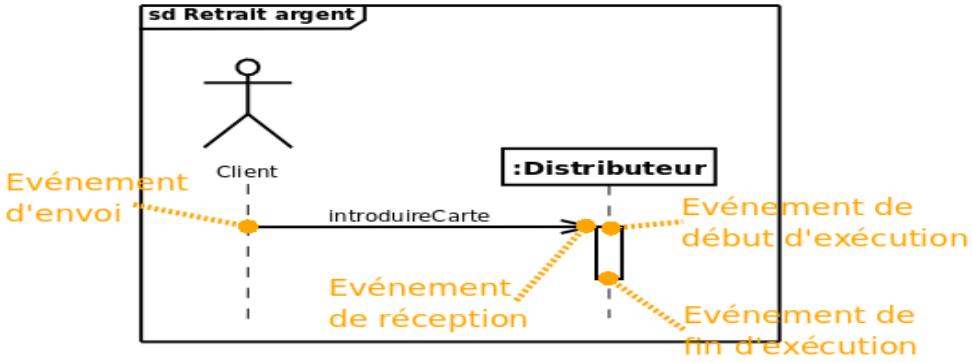


Figure II.1 : Différents évènements dans un diagramme de séquence

Le diagramme de séquence est une variante du diagramme de collaboration. Par opposition aux diagrammes de collaboration, les diagrammes de séquence possèdent intrinsèquement une dimension temporelle mais ne représente pas explicitement les liens entre les objets (Steffe, 2005).

- **Diagramme global d'interaction :**

Le diagramme global d'interaction permet de représenter une vue générale des interactions décrites dans le diagramme de séquence et des flots de contrôle décrits dans le diagramme d'activité.

Le diagramme global d'interaction privilégie la vue générale des flux de contrôle dans lesquels les nœuds sont des interactions ou des utilisations d'interactions.

Autrement dit, le diagramme global d'interaction est un diagramme d'activité dans lequel on représente des fragments d'interaction ou des utilisations d'interactions (Gabay & Gabay, 2008). Ainsi, il est possible de représenter :

- Des choix de fragments d'interactions (fusion).
- Des déroulements parallèles de fragments d'interactions (débranchement et jonction).
- Des boucles de fragments d'interaction.

- **Diagramme de temps :**

Le diagramme de temps permet de représenter les états et les interactions d'objets dans un contexte où le temps a une forte influence sur le comportement du système à gérer.

Autrement dit, le diagramme de temps permet de mieux représenter des changements d'états et des interactions entre objets liés à des contraintes de temps.

Pour cela, le diagramme de temps utilise en plus des lignes de vie, les concepts suivants :

- Des états ou des lignes de temps conditionnés avec deux représentations graphiques possibles.

- Des représentations propres aux aspects temporels : échelle de temps, contrainte de durée, événements…

Le diagramme de temps utilise trois concepts de base :

- Ligne de vie : elle représente l'objet que l'on veut décrire. Elle se dessine de manière horizontale. Plusieurs lignes de vie peuvent figurer dans un diagramme de temps.

- État ou ligne de temps conditionnée : les différents états que peut prendre l'objet d'étude sont listés en colonne permettant ainsi de suivre le comportement de l'objet ligne par ligne (une ligne pour un état).

- États linéaires : il s'agit du même concept que le précédent, mais la représentation de la succession des états est faite de manière linéaire à l'aide d'un graphisme particulier (Gabay & Gabay, 2008).

3. Les types de relations :

UML définit quatre types de relation (site 12, 2020) : relation structurelle, de spécialisation/généralisation, de réalisation et de dépendance.

3.1. Relation structurelle (ou association)

Une relation structurelle décrit un ensemble de liens. Un lien est une connexion entre objets.

Exemple : Un contrat concerne un client.

```
Class Contrat {
        Client bénéficiaire
        ….
    }
```

Le bénéficiaire est une des caractéristiques d'un contrat ; la relation est structurelle.

Cette relation est représentée par un **trait plein**, pouvant être orienté. La multiplicité est notée du côté du rôle cible de l'association. Elle spécifie le nombre d'instances pouvant être associées (liées) avec une seule instance source de la relation. Dans la phrase *un contrat concerne un client, contrat* est la source et *client* est la destination.

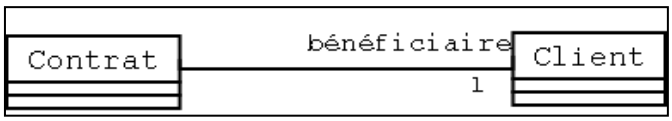


Figure II.2 : exemple d'une relation structurelle

Une multiplicité est exprimée soit par un couple de valeur N..M soit par une seule valeur lorsque N est M sont égaux.

3.2.Relation de spécialisation/généralisation

Relation d'héritage, dans laquelle les objets de l'élément spécialisé (classe enfant) peuvent remplacer les objets de l'élément général (classe parent).

Exemple : Un client professionnel est une sorte de client.

```
Class ClientProfessionnel extends Client {….  }
```

Notation UML : Un **trait plein**, orienté de la classe spécialisée (enfant) vers son modèle (parent) et se terminant par une **flèche fermée**.

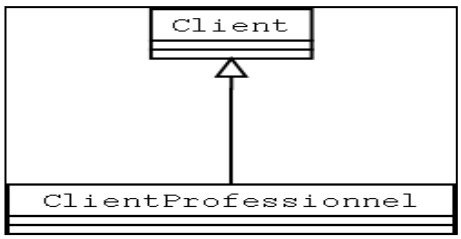


Figure II.3 : exemple d'une relation de spécialisation/généralisation

Attention : Ceci n'est vrai que si un client professionnel **se comporte comme** un client.

3.3.Relation de réalisation

Relation dans laquelle une interface définit un contrat garanti par une classe d'implémentation.

Exemple : Le formulaire SaisirClient est un écouteur d'événements.

```
Class SaisirClient extends JFrame implements ActionListener {
   …
  Public void actionPerformed (ActionEvent e ) {
            // faire quelque chose
     }
}
```

La fenêtre SaisirClient **réalise** le contrat définit par l'interface ActionListener.

Notation UML : Un **trait discontinu** partant de la classe d'implémentation et allant vers l'interface, se terminant par une **pointe de flèche fermée**, la même utilisée par la spécialisation/généralisation.

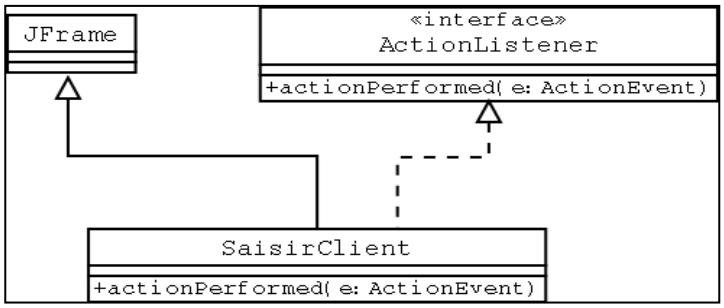


Figure II.4 : exemple d'une relation de réalisation

3.4.Relation de dépendance

Relation entre éléments du modèle ne nécessitant pas forcément un lien entre objets. Lorsque cette relation est réalisée par des liens entre objets, ces derniers sont limités dans le temps, contrairement à d'autres relations plus structurelles (cas d'une association - voir au-dessus).

Un élément A *dépend* d'un élément B, lorsque A utilise des services de B. De ce fait, tout changement dans B peut avoir des répercussions sur A.

Exemple : Un contrat dispose d'un service d'impression (méthode impression), qui utilise une méthode (print), dont la spécification est déclarée par l'interface Printer.

```
Class Contrat {
    …
   Public void impression () {
        Printer imprimante = PrinterFactory.getInstance () ;
```

```
        …
        Imprimante.print (client.getName () ) ;
        …
    }
}
```

On remarquera que le lien entre un objet "contrat" et une "imprimante" est momentané, il ne dure que le temps d'exécution de la méthode impression. En d'autres termes, l'imprimante n'a pas lieu d'être un attribut de la classe Contrat, et de ce fait, ce n'est pas une association, mais une simple dépendance.

Notation UML : Un **trait discontinu** partant de la classe dépendante et pointant vers la classe proposant les services sollicités, se terminant par une **pointe de flèche ouverte** (c'est ce qui la distingue de la relation de réalisation).

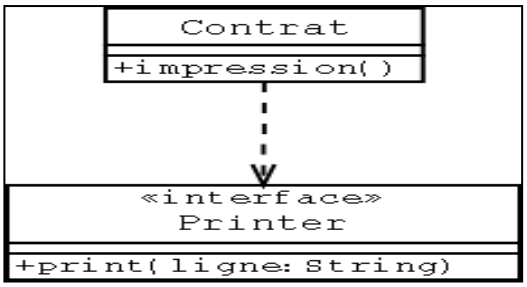


Figure II.5 : exemple d'une relation de dépendance

4. Les règles de passage UML vers schéma relationnelle : (site 13, 2020)

4.1. Transformation des classes : chaque classe du diagramme UML devient une relation, il faut choisir un attribut de la classe pouvant jouer le rôle de clé.

4.2. Transformation des associations : Nous distinguons trois familles d'associations :

- Association 1.. ? : il faut ajouter un attribut de type clé étrangère dans la relation fils de l'association. L'attribut porte le nom de la clé primaire de la relation père de l'association.
- Association *..* et n-aire et classes-association : la classe-association devient une relation. La clé primaire de cette relation est la concaténation des identifiants des classes connectées à l'association.
- Association 1.. 1 : il faut ajouter un attribut de type clé étrangère dans la relation dérivée de la classe ayant la multiplicité minimale égale à un. L'attribut porte le nom de la clé primaire de la relation dérivée de la classe connectée à l'association. Si les deux multiplicités minimales sont à un, il est préférable de fusionner les deux classes en une seule.

5. JAVA

La technologie Java définit à la fois un langage de programmation orienté objet et une plateforme informatique. Créée par l'entreprise Sun Microsystems (souvent juste appelée "Sun") en 1995, et reprise depuis par la société Oracle en 2009, la technologie Java est indissociable du domaine de l'informatique et du Web. On la retrouve donc sur les ordinateurs, mais aussi sur les

téléphones mobiles, les consoles de jeux, etc. L'avènement du Smartphone et la puissance croissante des ordinateurs, ont entraîné un regain d'intérêt pour ce langage de programmation. (site 4, 2020)

6. MYSQL

MySQL est une base de données relationnelle libre qui a vu le jour en 1995 et très employée sur le Web, souvent en association avec PHP (langage) et Apache (serveur web). MySQL fonctionne indifféremment sur tous les systèmes d'exploitation (Windows, Linux, Mac OS notamment). Le principe d'une base de données relationnelle est d'enregistrer les informations dans des tables qui représentent des regroupements de données par sujets (table de clients, table de fournisseurs, table de produits, par exemple). Les tables sont reliées entre elles par des relations**.** (site 5, 2020)

Le langage SQL dans "MySQL" signifie « Structured Query Language » : c'est un langage universellement reconnu par MySQL et les autres systèmes de gestion de bases de données (SGBD), et permet d'interroger et de modifier le contenu d'une base de données. Les autres SGBD utilisées en informatique sont essentiellement Microsoft SQL Server et Oracle.

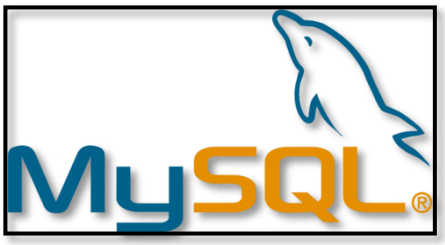


Figure II.6 : Logo de MySQL

7. JDBC

L'API **JDBC** (**J**ava **D**atabase **C**onnectivity) permet une connectivité standard et indépendante entre des applications java qui l'utilise et les serveurs de bases de données relationnelles. L'API JDBC met à la disposition des développeurs un cadre pour que des applications Java puissent se connecter à des bases de données relationnelles et exécuter des requêtes. (site 6, 2020)

8. Eclipse

8.1. Définition

Eclipse est un IDE, Integrated Development Environment (EDI environnement de développement intégré en français), c'est-à-dire un logiciel qui simplifie la programmation en proposant un certain nombre de raccourcis et d'aide à la programmation. Il est développé par IBM, gratuit et disponible pour la plupart des systèmes d'exploitation. (site 7, 2020)

8.2. L'interface d'Eclipse

Lorsque vous lancer Eclipse, cette fenêtre s'affiche :

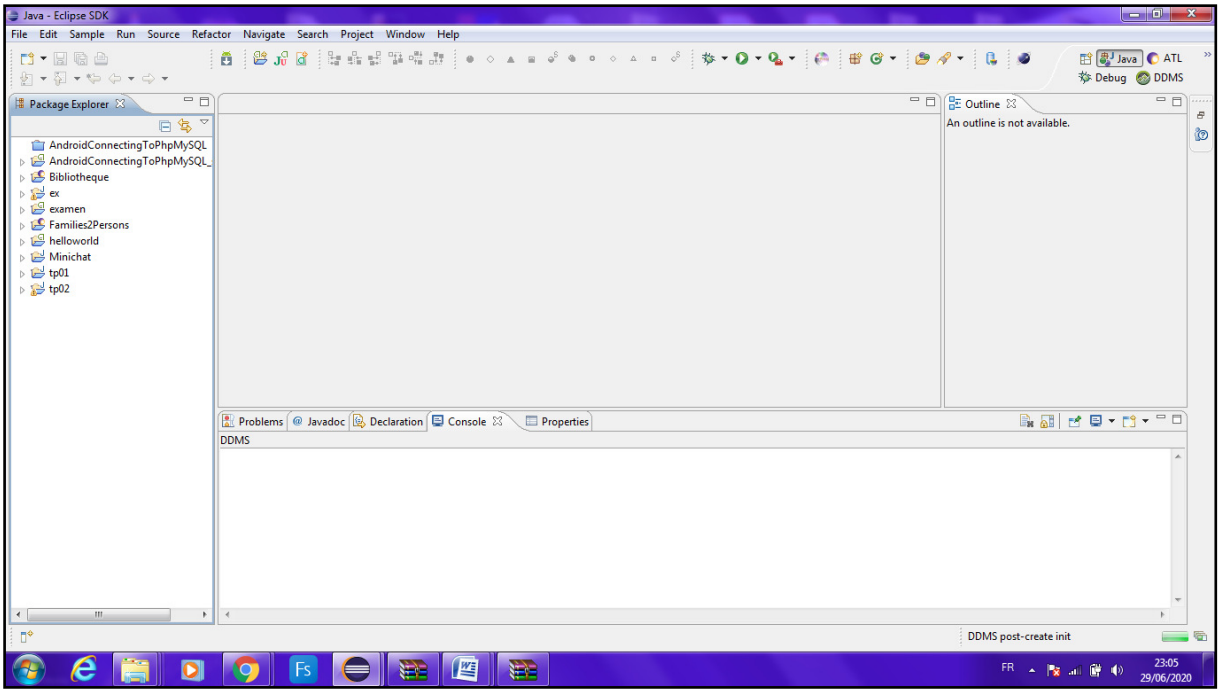


Figure II.7 : L'interface d'Eclipse

9. Les outils techniques

Nous avons réalisé notre application selon la décomposition MVC.

9.1. MVC

MVC (Model-View-Controller) (site 8, 2020) est un modèle de conception architecturale qui encourage une meilleure organisation des applications grâce à une séparation des préoccupations. Il divise une application interactive en trois composants : Modèle, Vue et Contrôleur. Il impose l'isolement des données d'entreprise (modèles) des interfaces utilisateur (vues). Un troisième composant (contrôleurs) gère traditionnellement la logique, les entrées utilisateur et la coordination des modèles et des vues. L'objectif de MVC est d'aider à structurer la séparation des préoccupations d'une application en trois parties :

- Le modèle : est responsable de la gestion des données de l'application. Il reçoit les entrées de l'utilisateur du contrôleur.
- La vue : signifie la présentation du modèle dans un format particulier. Il définit la façon dont les informations seront affichées à l'écran (via des composants par exemple). Il s'agit de l'interface utilisateur.
- Le contrôleur : répond à l'entrée de l'utilisateur et effectue des interactions sur les objets du modèle de données. Le contrôleur reçoit l'entrée, la valide éventuellement, puis transmet l'entrée au modèle. C'est en quelque sorte l'intermédiaire entre le modèle et la vue.

La figure suivante montre l'interaction entre le modèle, la vue et le contrôleur : (site 9, 2020)

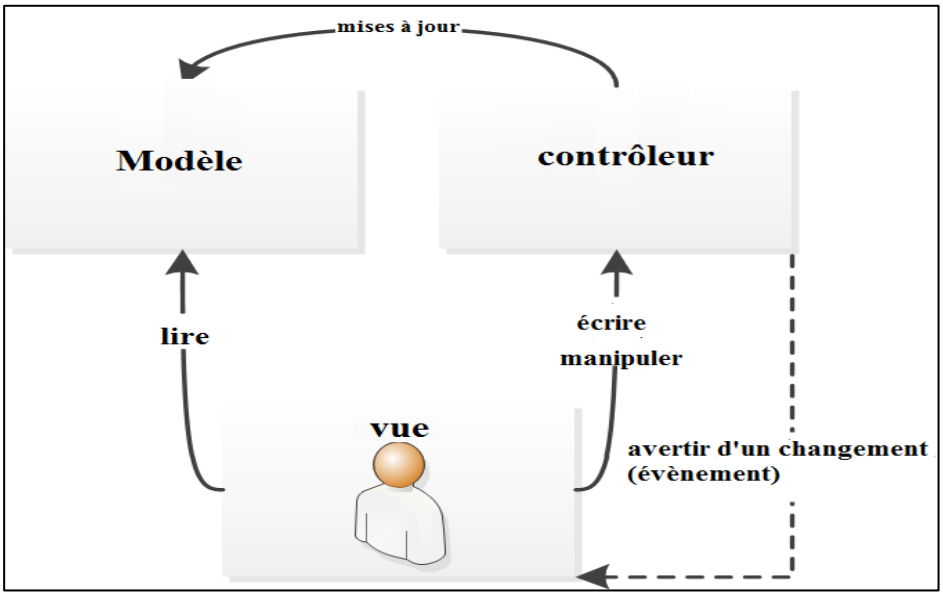


Figure II.8 : l'interaction entre le modèle, la vue et le contrôleur

9.1.1. MVC dans le monde réel

MVC est utile lors de la planification de votre application, car il vous donne un aperçu de la manière dont vos idées doivent être organisées en code réel (site 10, 2020). Par exemple, imaginons que vous créez une application de liste de tâches. Cette application permettra aux utilisateurs de créer des tâches et de les organiser en listes.

Le modèle d'une application de tâches peut définir ce qu'est une «tâche», et qu'une «liste» est un ensemble de tâches.

Le code de vue définira à quoi ressemblent les tâches et les listes visuellement. Les tâches peuvent avoir une grande police ou une certaine couleur.

Enfin, le contrôleur peut définir comment un utilisateur ajoute une tâche ou en marque une autre comme terminée. Le contrôleur connecte le bouton d'ajout de la vue au modèle, de sorte que lorsque vous cliquez sur «ajouter une tâche», le modèle ajoute une nouvelle tâche.

9.1.2. Les avantages et inconvénients

Cette architecture apporte plusieurs avantages lors de la création et de la mise en place d'un projet (site 11, 2020). Tout d'abord, elle facilite la maintenance et les évolutions futures. En effet, étant donné qu'il y a une séparation entre les différentes couches, il sera plus facile de modifier uniquement la partie vue ou encore uniquement le traitement de la requête dans le contrôleur. Cependant, même si beaucoup de Framework MVC existent, cette architecture nécessite directement une plus grosse architecture étant donné qu'il y aura trois fois plus de fichiers. Ainsi, pour les petits projets, cela semble potentiellement inutile.

L'architecture MVC est donc intéressante à utiliser et permet de soigner son architecture logicielle, mais n'est pas forcément adaptée à des petits projets en raison de sa complexité et du nombre de fichiers à créer.

Son utilité peut parfois sembler inexistante au début d'un projet mais son besoin se montre rapidement lorsque celui-ci gagne en taille.

C'est donc une bonne idée de prendre l'habitude de l'utiliser lors du développement de projets.

10. Conclusion

Dans ce chapitre, nous avons vu un background montrant les notions et les outils liés au développement de notre système UML vers JAVA. Le chapitre suivant introduit la conception de notre système.

Chapitre III: Conception du système

1. Introduction

Nous présentons dans ce chapitre la partie conception de notre système qui va permettre de passer d'UML vers Java.

La conception était faite en langage de modélisation UML. Parmi les diagrammes UML, nous avons choisi ceux que nous avons jugés nécessaires pour la mise en œuvre de notre application. On s'est limité aux diagrammes de cas d'utilisation et d'activité.

2. Diagramme de cas d'utilisation

La figure suivante présente le diagramme de cas d'utilisation définissant les exigences fonctionnelles attendues, les acteurs (utilisateurs du système), ainsi que les relations qui unissent acteurs et fonctionnalités :

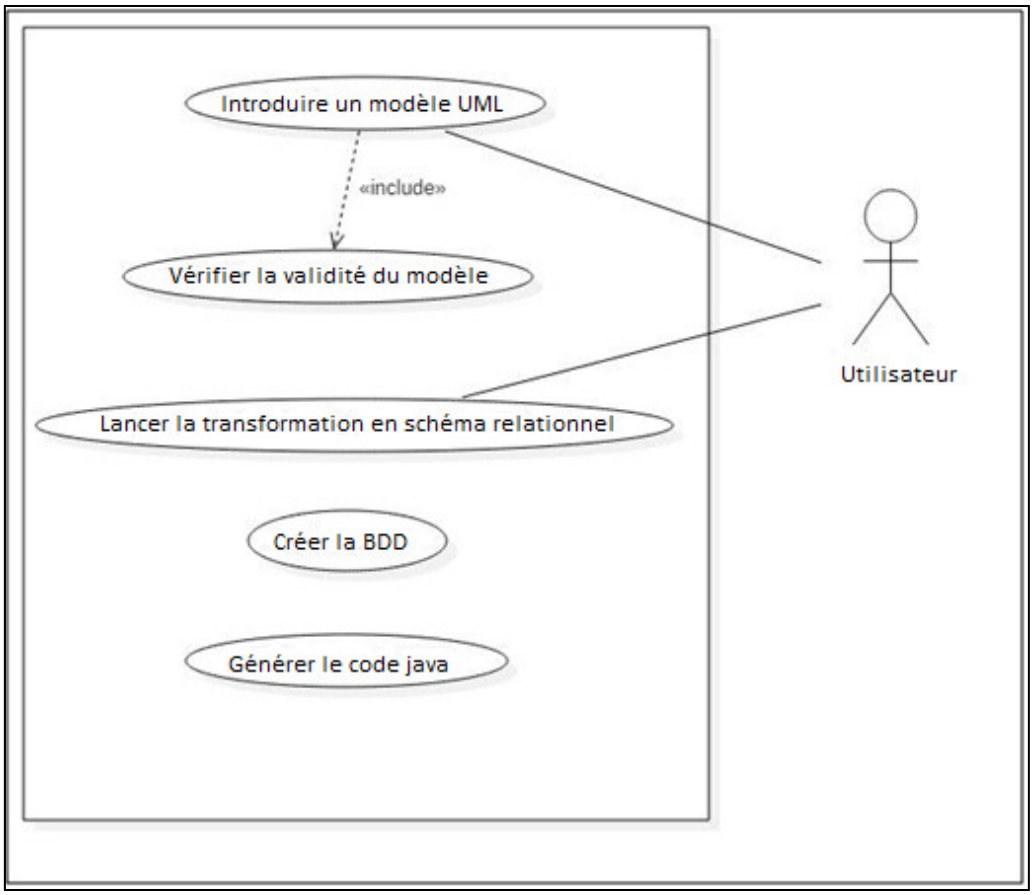


Figure III.1 : Diagramme de cas d'utilisation montrant les différentes fonctionnalités du système

Il est clair que notre système assure principalement les fonctionnalités suivantes :

- Introduire un modèle UML : l'utilisateur, qui est le seul acteur représenté dans le diagramme de cas d'utilisation ci-dessus, doit introduire un diagramme UML (principalement le diagramme de classes) à travers une interface assurant cette

fonctionnalité. Ce diagramme doit être vérifié et validé par le système avant de passer à l'étape suivante.

- Lancer la transformation en schéma relationnel : l'utilisateur lance ensuite la transformation du diagramme de classes déjà introduit en schéma relationnel.
- Créer la BDD : le système crée une BDD correspondant au schéma relationnel déjà généré.
- Générer le code JAVA : le système permet de générer le code Java manipulant la BDD déjà créée (ajout suppression, modification et stockage)

3. Diagramme d'activité

Le suivant est le diagramme d'activité entre l'utilisateur et le système

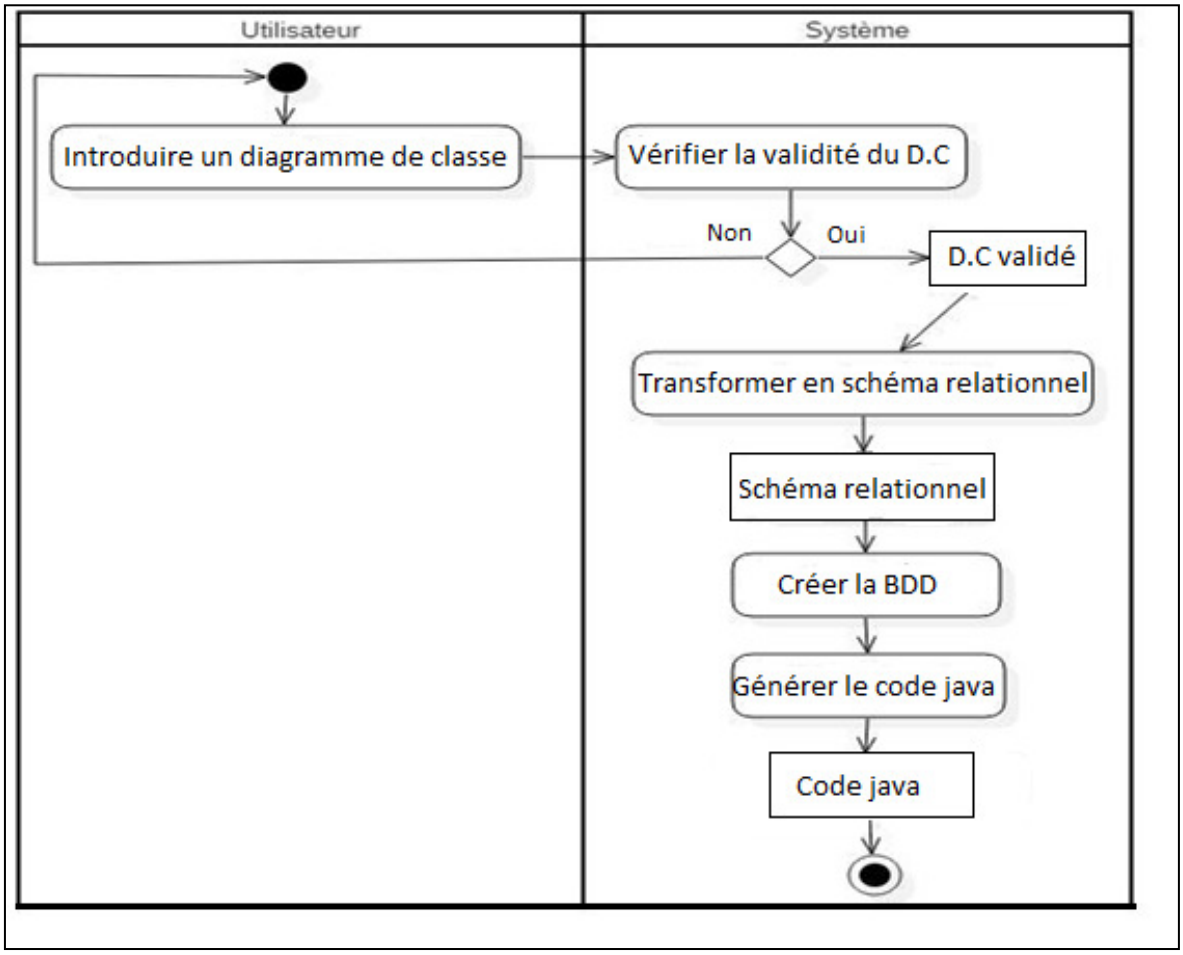


Figure III.2 : Digramme d'activité du processus de transformation UML vers Java

Comme déjà mentionné au niveau du diagramme de cas d'utilisation, un processus de transformation UML vers Java commence par l'introduction d'un diagramme de classe par l'utilisateur. Le système vérifie et valide le diagramme de classes. Si cette opération réussit, le système le transforme en modèle relationnel à partir duquel va être créée une BDD. Enfin, un code Java manipulant cette BDD va être automatiquement généré.

4. Conclusion

Après la conception présentée dans ce chapitre, vient l'implémentation qui est l'étape de concrétisation technique du projet ; c'est la phase de développement pur qui sera détaillée dans le chapitre suivant.

Chapitre IV : Implémentation

1. Introduction

Ce chapitre présente l'architecture et la méthode d'utilisation de notre système, tout en présentant un exemple d'application. Ceci donne un guide aux utilisateurs pour bien utiliser le logiciel. En plus, on décrit les stratégies qu'on a utilisées pour réaliser notre travail.

2. L'architecture de notre système

La figure suivante montre l'architecture de notre système :

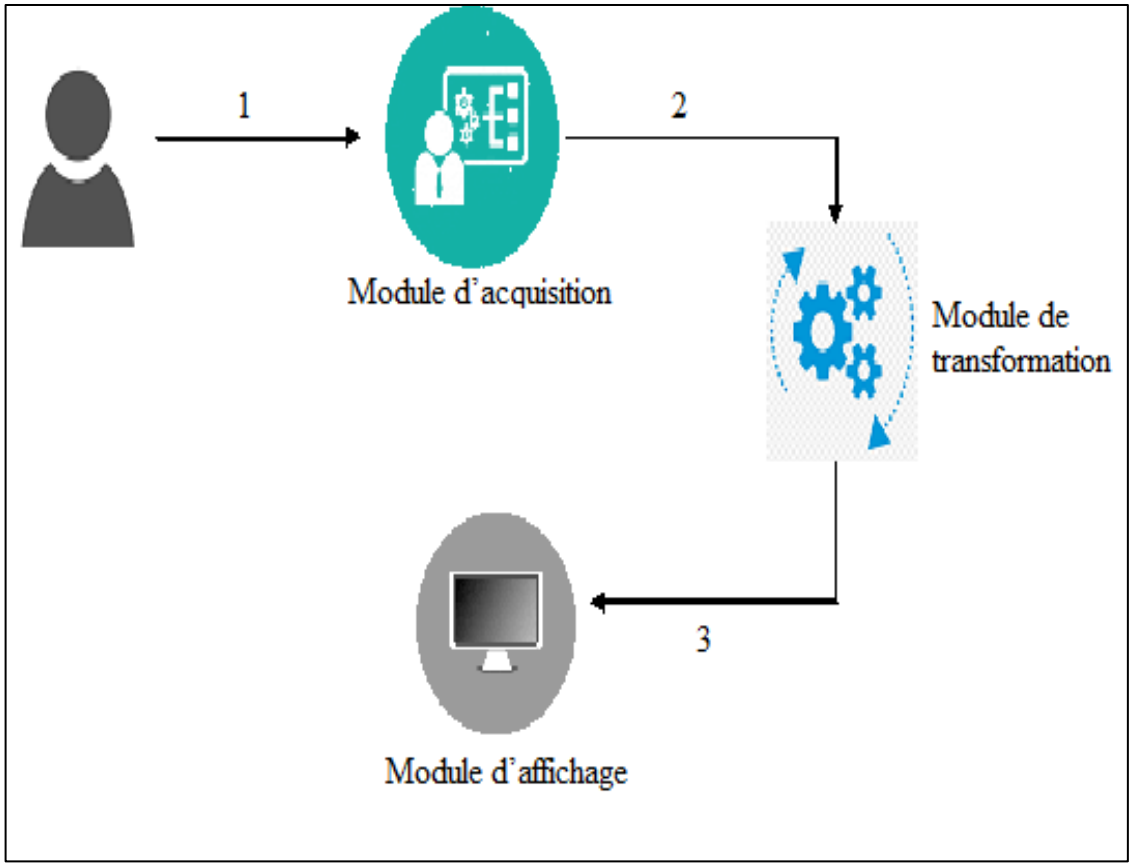


Figure IV.1 : l'architecture de notre système

Le système contient trois grands modules : module d'acquisition, module de transformation et module d'affichage. Dans ce qui suit, on décrit ces trois composantes :

1. Module d'acquisition : l'utilisateur introduit le diagramme de classes.
2. Module de transformation : c'est la transformation du diagramme de classes en modèle relationnel et code Java.
3. Module d'affichage : permet d'afficher le résultat de la transformation (modèle relationnel + code Java).

3. Présentation de l'application

La figure IV.2 représente l'interface principale de notre application. Dans cette fenêtre existe le menu File et le menu Generate.

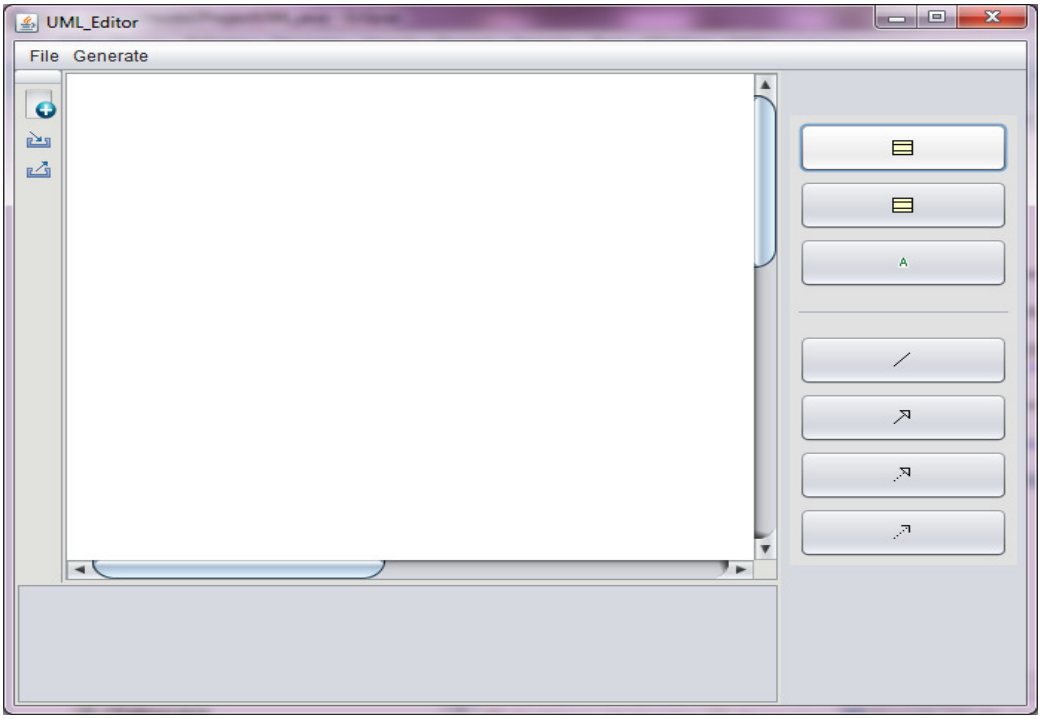


Figure IV.2 : l'interface principale

Dans le menu « File», on trouve «New Project » qui permet de créer un nouveau projet.

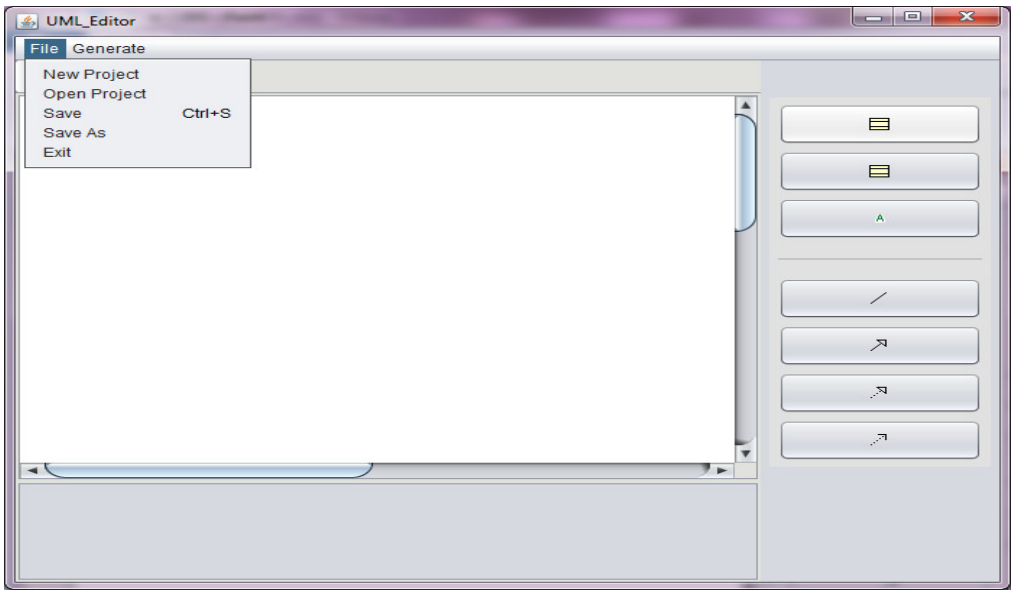


Figure IV.3 : Le menu File

Lorsqu'on clique sur «New Project», la fenêtre suivante affichée :

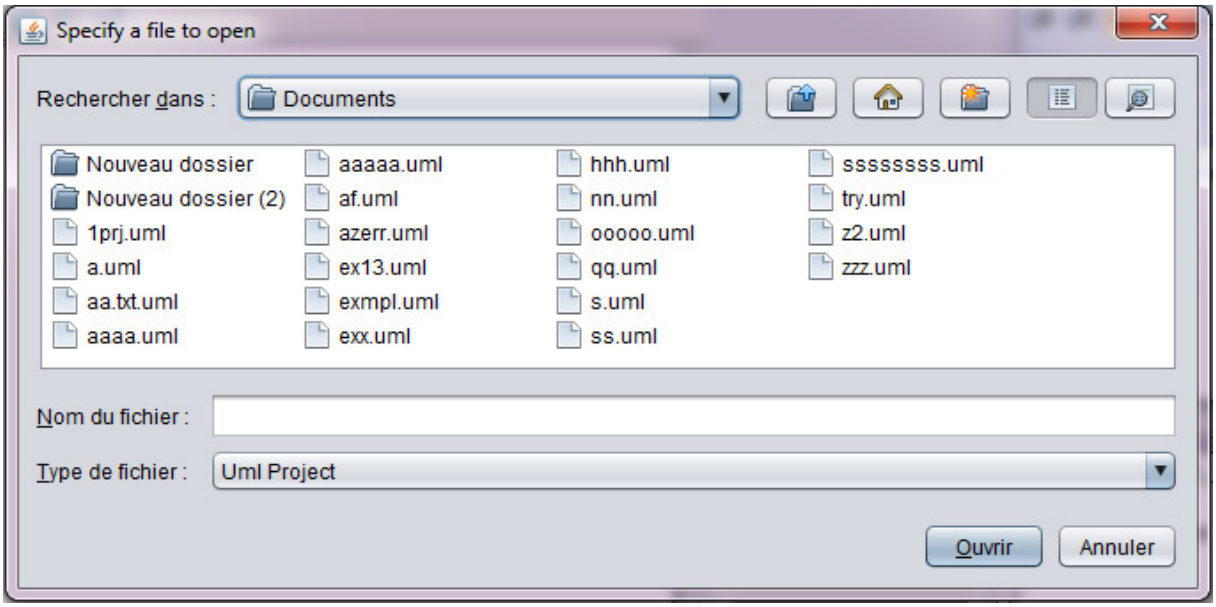


Figure IV.4 : fenêtre d'ouverture d'un nouveau projet

Lorsqu'on clique sur «Save» ou «Save as », la fenêtre suivante s'affiche (figure IV.5). Entrer le nom du projet et cliquer sur « Enregistrer » pour le sauvegarder.

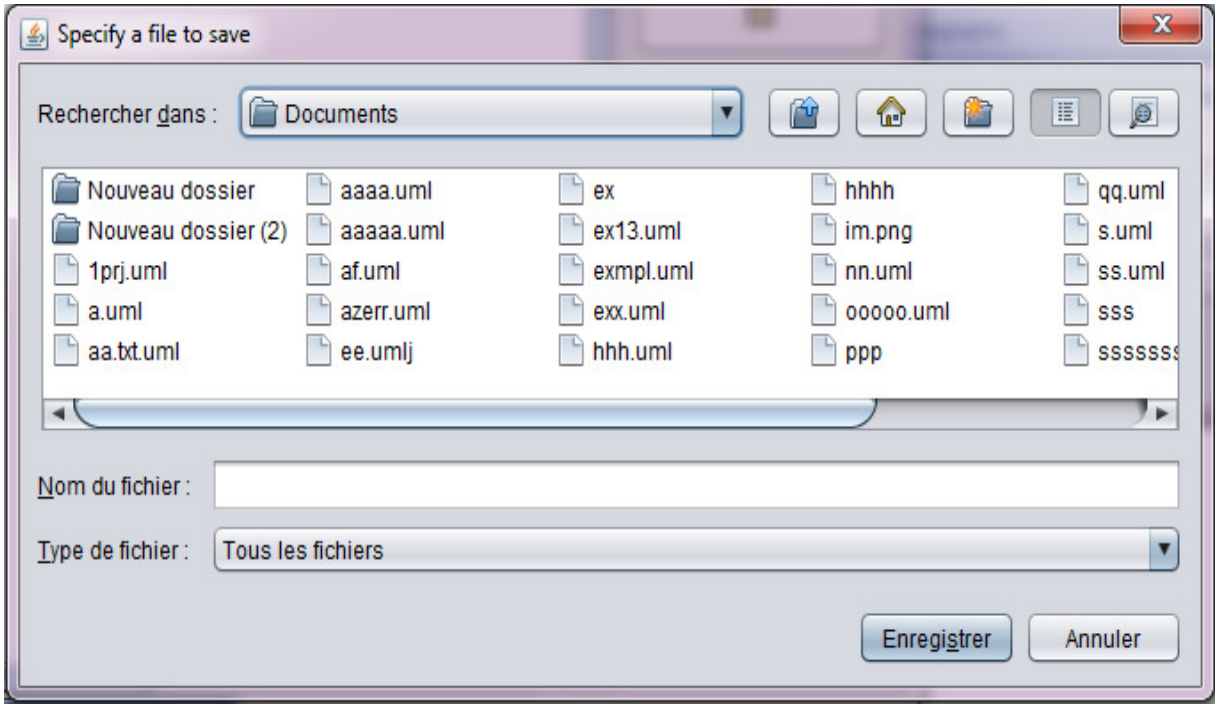


Figure IV.5 : Fenêtre d'enregistrement d'un projet

Pour fermer l'éditeur cliquer sur « Exit ».

Le menu « Generate » contient Relational schema, Data base et Java code.

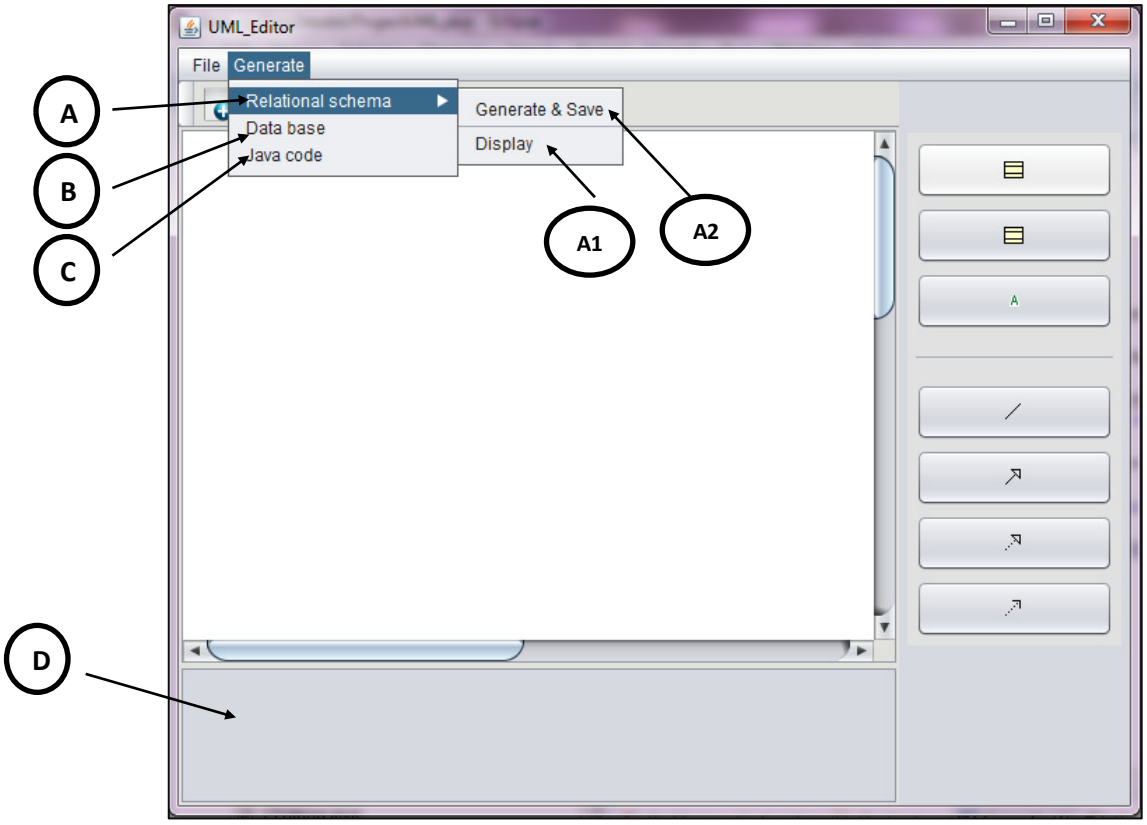


Figure IV.6 : Le menu Generate

(**A**) : le sous_menu « reletional schema ».

(**B**) : pour créer la BDD.

(**C**) : pour générer le code java.

(**A1**) : pour générer le schéma relationnel et le sauvegarde dans un fichier texte.

(**A2**) : pour afficher le schéma relationnel dans la zone (**D**).

La figure IV.7 représente la barre d'outils qui se trouve dans l'interface principale et qui contient trois composants :

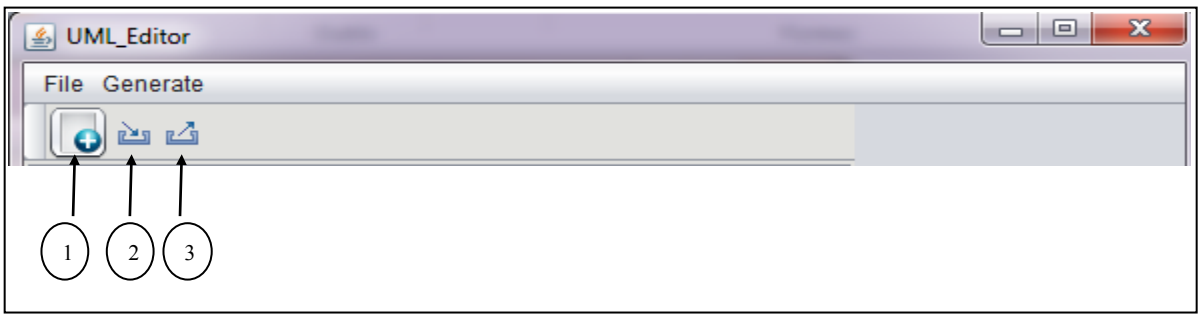


Figure IV.7 : barre d'outils 1

(**1**) : pour ouvrir un nouveau projet.

(**2**) : pour importer un projet.

(**3**) : pour exporter le projet.

Dans l'interface principale, on trouve une deuxième barre d'outils (figure IV.8).

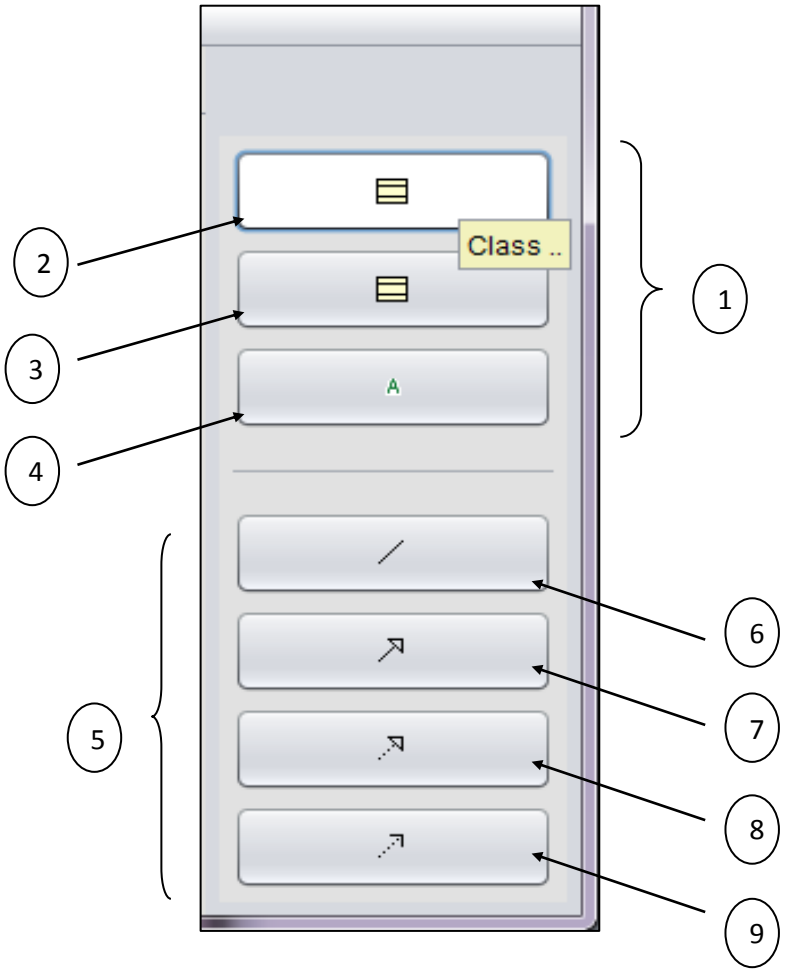


Figure IV.8 : Barre d'outils 2

(**1**) : Type de classe.

(**2**) : Classe.

(**3**) : Classe abstraite.

(**4**) : Interface.

(**5**) : Type de relation entre classe.

(**6**) : Relation structurelle (ou association).

(**7**) : Relation de spécialisation/généralisation.

(**8**) : Relation de réalisation.

(**9**) : Relation de dépendance.

Ainsi, on trouve dans l'interface principale une zone de dessin dans laquelle le diagramme de classes s'affiche.

Si vous cliquez sur (**1**) le résultat s'affiche dans la zone de dessin (figure IV.9).

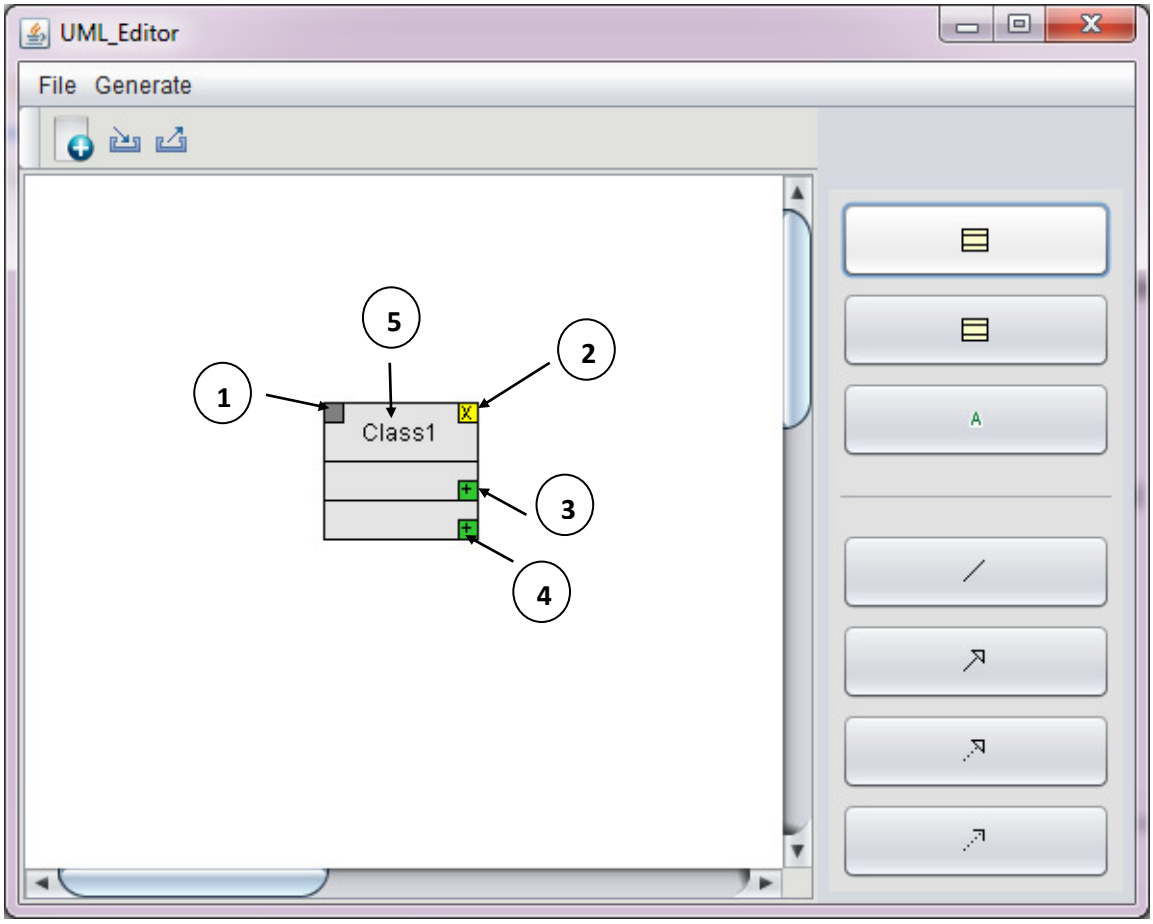


Figure IV.9 : Manipulation d'une classe

(**1**) : Pour déplacer la classe.

(**2**) : Pour supprimer la classe.

(**3**) : pour ajouter un attribut.

(**4**) : Pour ajouter une méthode.

(**5**) : Pour modifier le nom de la classe.

Si vous cliquez sur (**3**) la fenêtre suivante affichée :

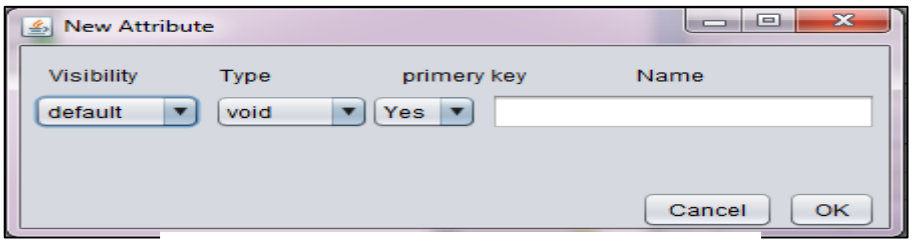


Figure IV.10 : l'ajout d'un attribut

Entrer le nom d'attribut, sélectionner la visibilité et le type de cet attribut. Si l'attribut est un identifiant choisir « yes », sinon sélectionner « no ». Enfin, cliquer sur « OK ».

Si vous cliquez sur (**4**), la fenêtre suivante est affichée :

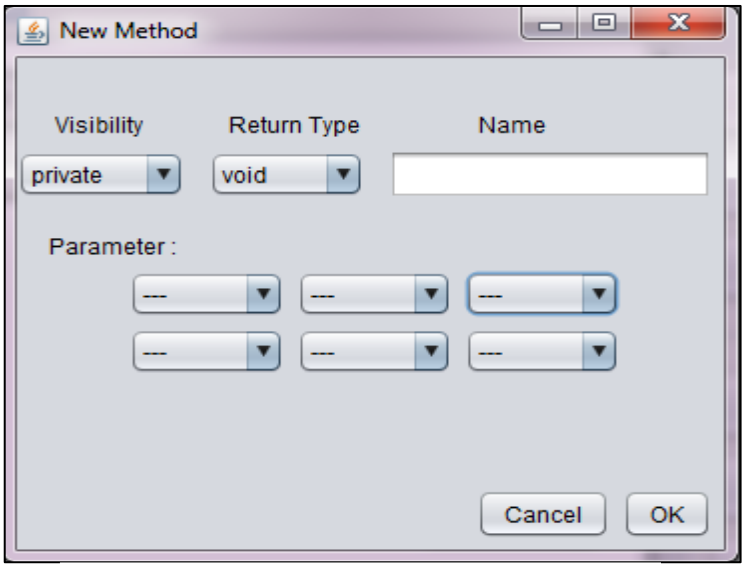


Figure IV.11 : L'ajout d'une méthode

Entrer le nom de la méthode, sélectionner la visibilité, le type de résultat retourné et sélectionner les paramètres de la méthode s'ils existent. Enfin, cliquer sur « OK ».

Si vous cliquez sur (5), la fenêtre suivante apparait :

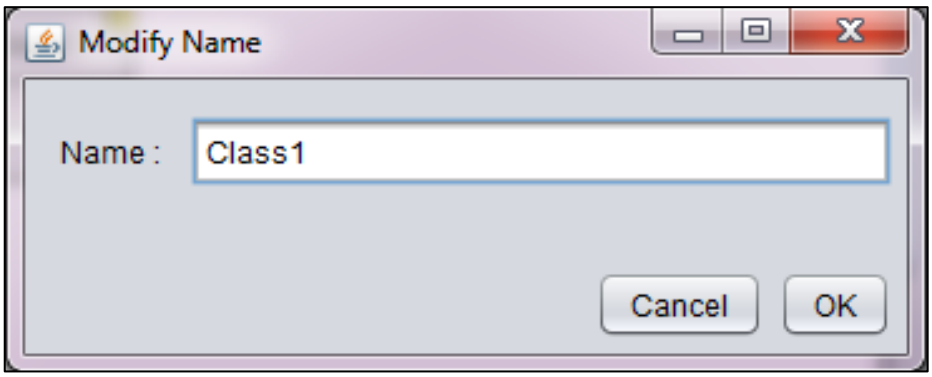


Figure IV.12 : la modification du nom d'une classe

Entrer le nom de la classe et cliquer sur « OK ».

Si vous avez moins de deux classes dans votre digramme et vous appuyez sur un des types de relation entre classes, le message suivant apparait :

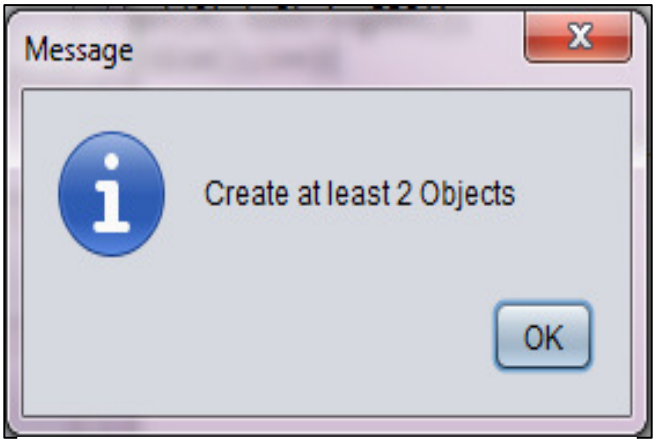


Figure IV.13 : Message d'information de relation

Sinon, la figure IV.14 apparait :

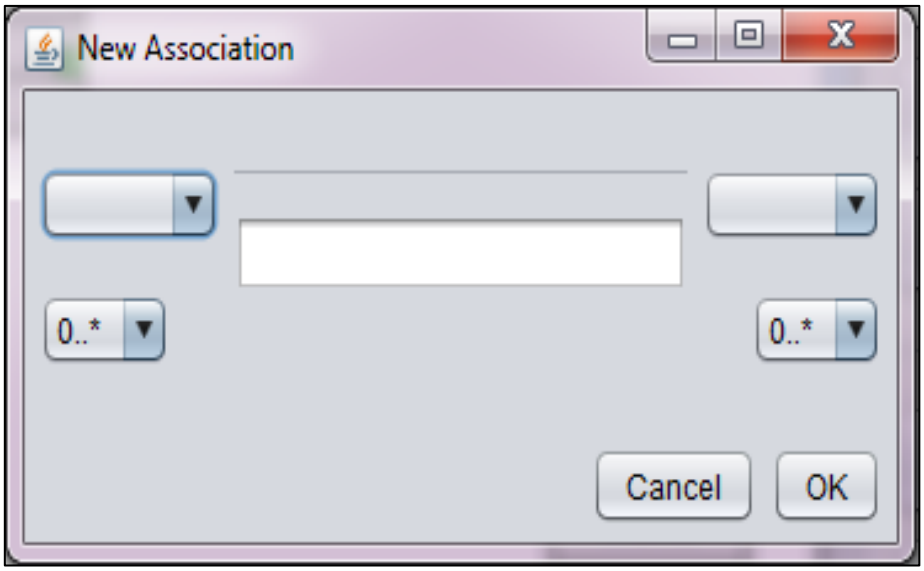


Figure IV.14 : L'ajout d'une relation

Il faut saisir le nom de la relation, sélectionner les cardinalités et la relation.

4. Exemple d'utilisation

Nous avons choisis un diagramme de classe qui contient quatre classes : livre, auteur, édition et libraires. Après d'introduire ce diagramme dans notre application, on aura la figure suivante :

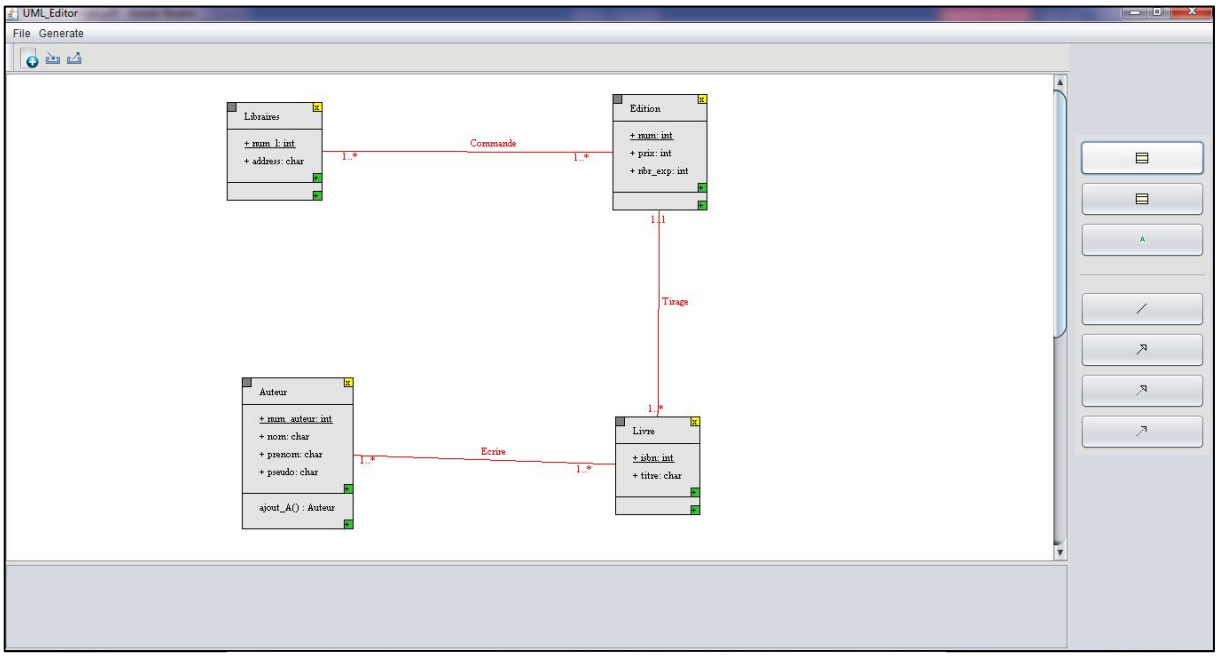


Figure IV.15 : Exemple d'introduire un diagramme de classe

Ensuite, on clique sur « Save » ou « CTRL+S » pour enregistrer notre diagramme.

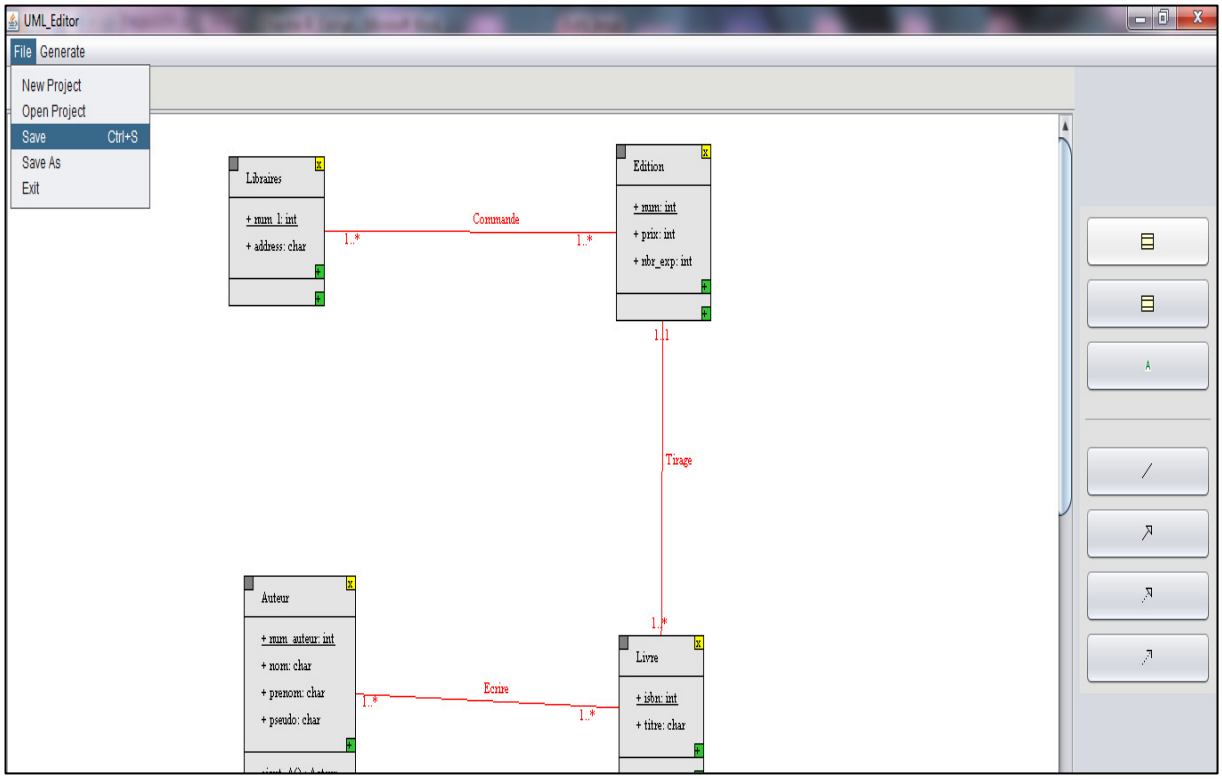


Figure IV.16 : Exemple d'enregistrer le diagramme de classe

On clique sur « relational schema » dans le menu « Generate », on choisis « generate and save » pour générer le schéma relationnel et le sauvegarde dans un fichier texte.

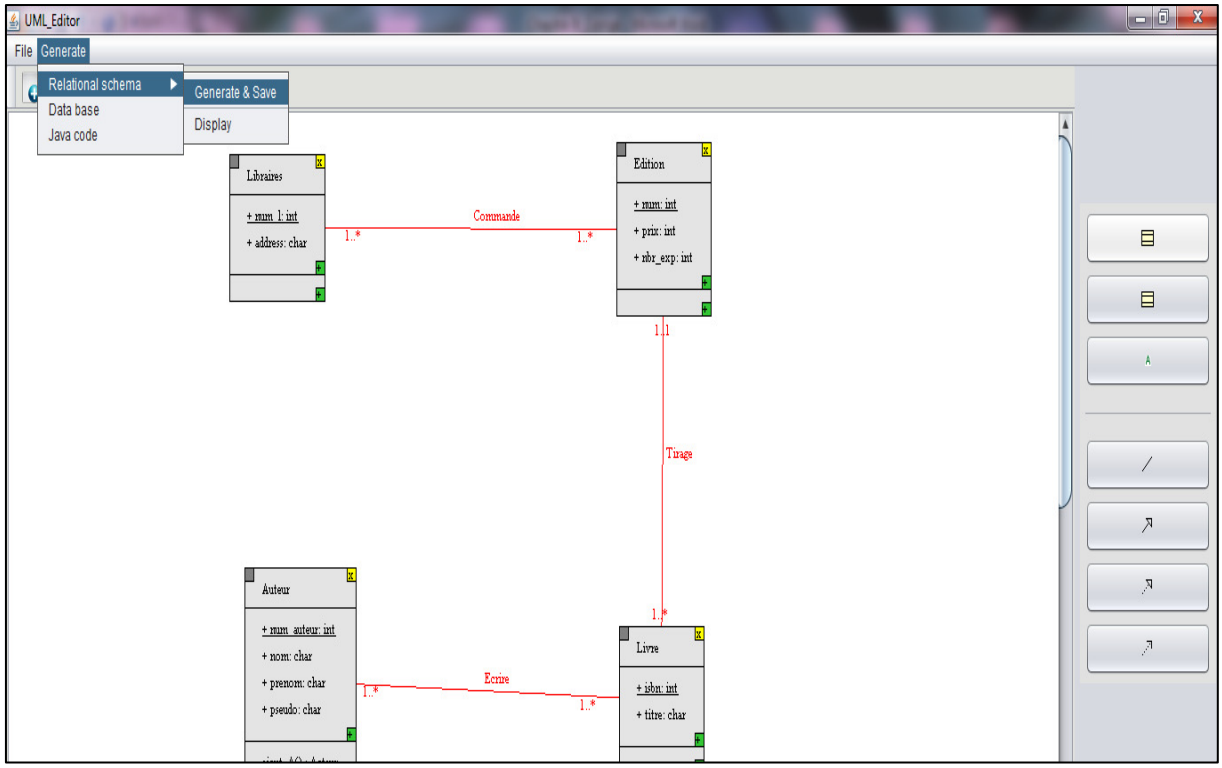


Figure IV.17 : Exemple de générer le schéma relationnel du diagramme (1)

On clique sur « display » pour afficher le schéma relationnel dans la zone en bas de la fenêtre (figure IV.18).

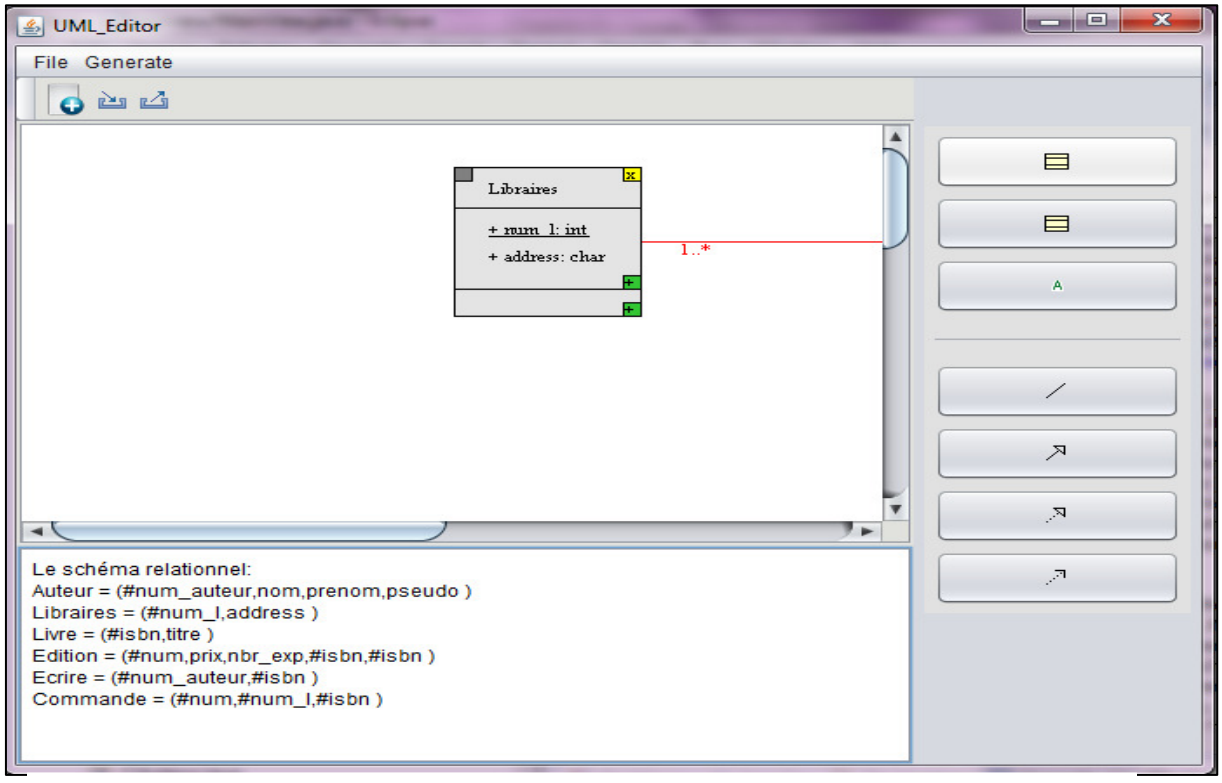


Figure IV.18 : Exemple de générer le schéma relationnel du diagramme (2)

On clique sur « Data base » dans le menu « Generate» pour créer la BDD.

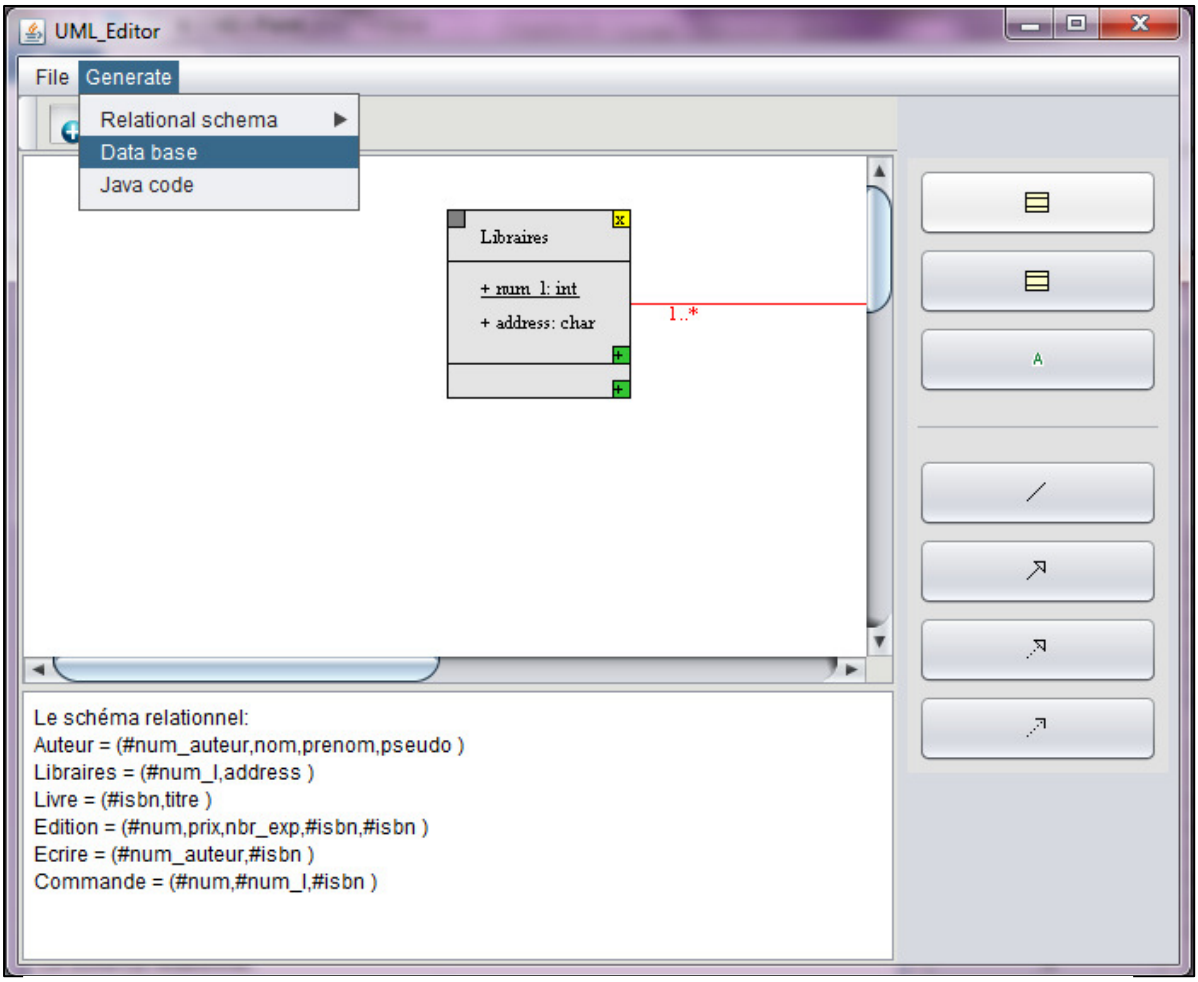


Figure IV.19 : Exemple de générer la BDD (1)

On aura le résultat suivant :

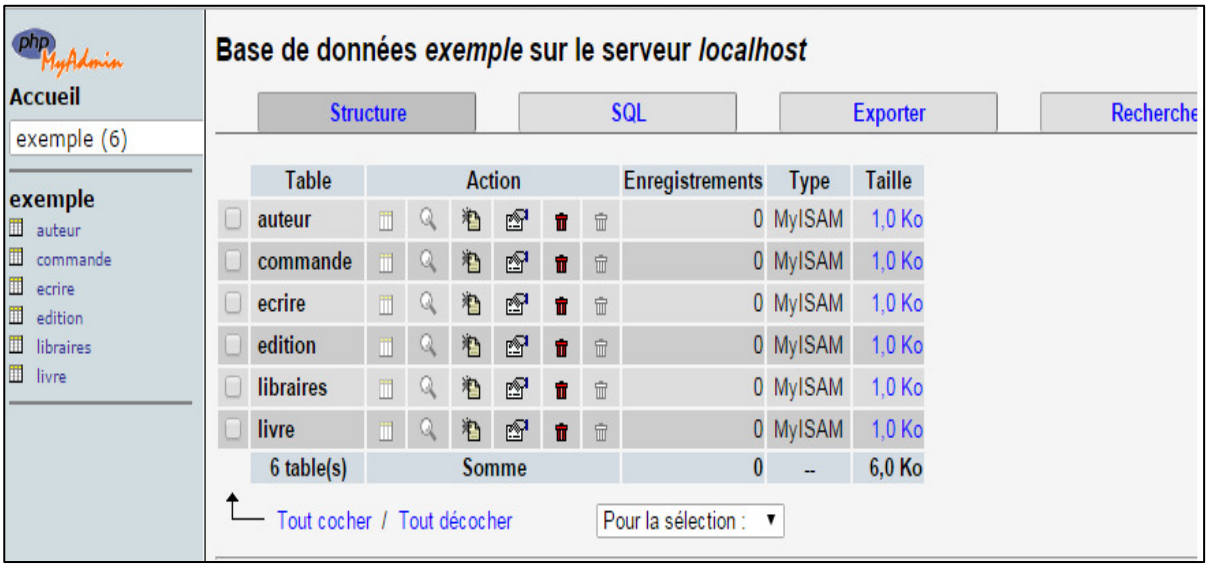


Figure IV.20 : Exemple de générer la BDD (2)

Pour générer le code java, on clique sur « java code » dans le menu « Generate » et on aura le résultat suivant :

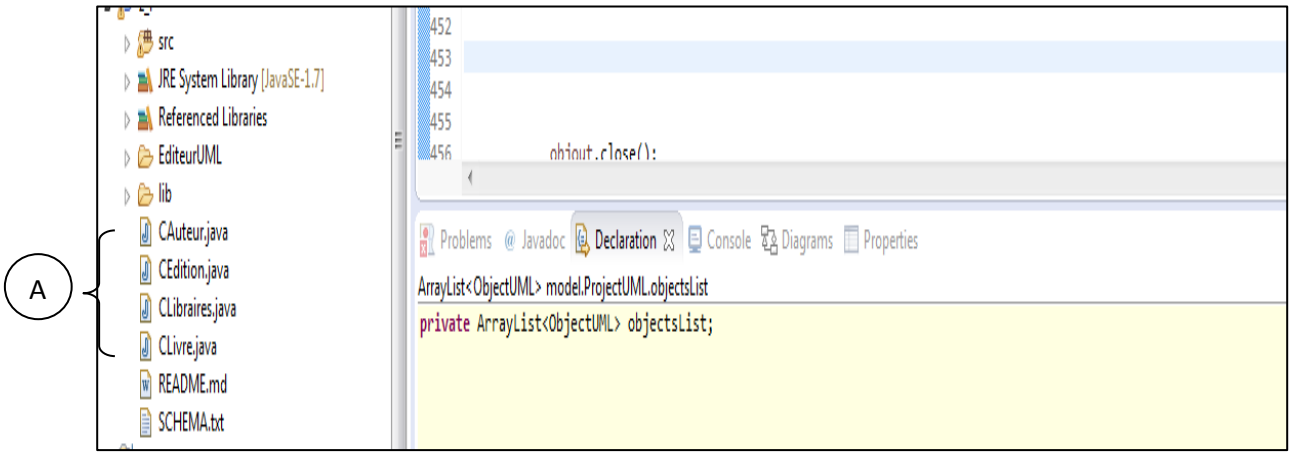


Figure IV.21 : Exemple de générer le code java

(A): les classes java générées.

5. Conclusion

Dans ce chapitre, nous avons présenté les interfaces et un guide d'utilisation de notre logiciel. Dans ce suit, une conclusion générale synthétise notre travail et trace quelques perspectives.

Conclusion Générale

L'art de fabriquer des logiciels n'est pas un métier facile. Pour métriser 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.

Nous avons présenté l'outil CASE UML vers JAVA comme un moyen efficace qui aide les développeurs dans le cycle de vie du logiciel.

Un outil CASE est un ensemble intégré d'outils qui permet aux développeurs de logiciels 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 (site 1, 2020). Il faut noter ici que les outils CASE sont seulement des aides et ne permettent pas de donner une solution totale à touts les problèmes de développement du logiciel. Ils sont des outils de gestion pour le développement des logiciels.

Pour l'implémentation de notre logiciel, nous avons utilisé Eclipse sous Windows. Pour la conception, nous avons utilisé UML.

Notre mémoire comportait un chapitre état de l'art ; il s'agit d'une étude de l'existant plus le choix des outils de développement. Le chapitre background, en second lieu, décrit les différentes notions nécessaires à la compréhension et à la mise en place de notre système. Les deux chapitres qui suivent présentaient la conception de notre système en UML, et la mise en œuvre de l'application.

Notre travail reste une initiative pour une future recherche dans laquelle nous enrichirons notre application par d'autres diagrammes comme le diagramme d'activités avec celui de classes.

Ce projet a fait l'objet d'une expérience intéressante, qui nous a permis d'améliorer nos connaissances dans le domaine de la conception et la programmation des systèmes informatiques. Nous espérons aussi passer à une approche à base de modèle pour le passage d'UML vers Java.

Bibliographie et Webographie

- L. Audibert, UML 2 de l'apprentissage à la pratique. Developpez.com. 12 janvier 2009. (https://laurent-audibert.developpez.com/Cours-UML/). (Vu 21/03/2020)
- G. Booch, J. Rumbaugh et I. Jacobson (2000). Le guide de l'utilisateur UML. Eyrolles. 2000. (https://www.eyrolles.com/Informatique/Livre/le-guide-de-l-utilisateur-uml-9782212091038/). (Vu 2202/2020)
- J. Gabay ET D. Gabay. UML 2 analyse et conception. DUNOD, Paris. 2008.
- S.Galland. Analyse, conception objet, diagrammes de déploiement. SIMMO/ENSM.SE. Octobre 2002. Disponible sur le lien : (http://oneway-it.com/projets/portfolio/docs/UML.Deploiement.4pp.pdf). (vu le 30/04/2020)
- L. Henocque. UML2 Cours n° 10, Diagramme de paquetage. 2008. disponible sur le lien : (http://remy-manu.no-ip.biz/UML/Cours/coursUML10.pdf). (Vu 22/03/2020)
- C.Johnen. UML Cours, Diagramme de communication. IUT de Bordeaux, Ver 2. Disponible sur le lien : (http://www.labri.fr/perso/johnen/pdf/IUT-Bordeaux/UMLCours/diagramme-Communication.pdf). (Vu 30/06/2020)
- S.Kimberly, S.Dennis, "Guide to CASE Adoption", Ed Morris, (1992).
- L.Piechocki. Cours UML. 4 Aout 2006. Disponible sur le lien : (http://dico.developpez.com/html/247-Conception-UML-Unified-Modeling-Language.php). (Vu 15/04/2020)
- J. Steffe. Cours UML13. ENITA de Bordeaux. Mars 2005. Disponible sur le lien : (http://www.anor.fr/fichiers/1.pdf). (Vu 20/03/2020)

- Site1 : https://bu.umc.edu.dz/theses/informatique/ABD5735.pdf. (vu le 24/3/2020)
- Site2 :https://fr.wikipedia.org/wiki/Atelier_de_g%C3%A9nie_logiciel#Fonctionnalit%C3%A9s_d'un_AGL (vu le 25/2/2020)
- Site3 :https://content.sciendo.com/view/journals/acss/14/1/article-p9.xml?language=en (vu le 30/5/2020)
- Site4 : https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203555-javadefinition/#:~:text=La%20technologie%20Java%20d%C3%A9finit%20%C3%A0,l'informatique%20et%20du%20Web. (vu le 12/4/2020)
- Site5 **:**http://www.mosaique-info.fr/glossaire-web-referencement-infographie-multimedia-informatique/m-glossaire-informatique-et-multimedia/448-mysql-definition.html (vu le 30/6/2020)
- Site6 :https://ocamil.com/index.php/javaee/jdbc/jdbc-comprendre-jdbc(vu le 13/3/2020)
- Site7 :https://dept-info.labri.fr/ENSEIGNEMENT/programmation2/intro-eclipse/#:~:text=Eclipse%20est%20un%20IDE%2C%20Integrated,d'aide%20%C3%A0%20la%20programmation. (vu le 27/4/2020)
- Site 8 : https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-model-view-control-mvc/(vu le 24/08/2020)
- Site 9 :https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur#Exemples_d%E2%80%99architecture_MVC(vu le 24/08/2020)
- Site 10 :https://www.codecademy.com/articles/mvc(vu le 24/08/2020)
- Site 11 : https://www.supinfo.com/articles/single/8729-architecture-mvc-qu-est-ce-que-c-est (vu le 24/08/2020)

- Site 12 : http://www.linux-france.org/prj/edu/archinet/DA/fiche-uml-relations/fiche-umlrelations.html?fbclid=IwAR0OoydWZCOqTwBKT6oBxAIY7ECvZxKLOWLXTzQTptgUNHs62SBtAd8zuuA (vu le 25/08/2020)
- Site 13 : https://www.memoireonline.com/08/09/2577/m_Conception-et-realisation-dune-application-de-suivi-de-patients-dans-un-etablissement-hospitalier7.html (vu le 25/08/2020)