BASES DE DONNÉES OBJETS

A - Introduction :

Le modèle relationnel est un modèle simple. Il est composé de tables plates, les attributs des relations sont des types simples etc. Malheureusement, ce modèle peut être parfois insuffisant pour obtenir une modélisation plus fine de la réalité. Par exemple, on pourrait vouloir avoir des attributs dans une relation qui ne soient plus des types simples, mais des types complexes. Les caractéristiques des langages de programmation orientés objets sont intéressantes pour la modélisation de bases de données : système de typage puissant (collections, références), notion de classe, héritage. Il peut donc être utile d'introduire ces notions dans un système de gestion de bases de données. De plus, les analyses et les modèles de systèmes complexes sont de plus en plus faits actuellement via des modèles objets supportant ces notions.

Un langage de programmation orienté objets comme C++, Java ou Smalltalk permet de créer, manipuler et détruire des objets en mémoire virtuelle. La durée de vie d'un objet n'excède pas la durée de vie du programme qui a servi à créer cet objet. Par ailleurs, comme dans tous les langages de programmation, il est possible de sauvegarder les objets dans des fichiers pour prolonger leur durée de vie. Ce type de sauvegarde fournit une première forme de persistance des objets qui assez primitive. En effet :

- la persistance n'est pas transparente, elle doit être gérée explicitement par le ou les programmes qui accèdent aux objets.
- la programmation de ces opérations de sauvegarde est délicate car les adresses des objets sont souvent des adresses virtuelles et il faut donc les convertir en adresses du support physique. De plus, lors de la définition de ce mécanisme d'adressage, il faudra gérer les relations inter-objets : la sauvegarde d'un objet peut amener à sauvegarder en fait tout un ensemble d'objets en relation avec ce dernier.
- l'accès aux objets en mémoire, contrôlé par le mécanisme de mémoire virtuelle, peut avoir de très mauvaises performances, et même au pire nécessiter un accès disque par objet.
- enfin, lorsque les objets sont stockés dans des fichiers physiques par des utilisateurs concurrents, l'accès à ces objets et difficile et nécessite la connaissance de la structure

des fichiers. Cela engendre le problème de la dépendance physique entre programmes et données.

Nous retrouvons ici tous les problèmes engendrés par la gestion des données sur des fichiers classiques.

L'approche base de données objets apporte une solution récente à la gestion transparente de la persistance des objets. Deux approches pour la construction d'un système de gestion de base de données à objets (SGBDO) ont été dégagées. L'approche relationnel-objet, représentée par le standard SQL : 1999 (ou SQL-99) et l'approche orientée objet représentée par le standard de l'ODMG (Object Data Management Group).

Nous verrons également dans le chapitre qu'une solution actuellement utilisée pour la persistance d'objets est l'utilisation d'un mécanisme transparent de mapping vers une base de données relationnelle.

Ce chapitre présente les notions de base de données objets et de système de gestion de bases de données objets (SGBDO).

B - Historique des bases de données orientées objets :

L'approche système dédié désigne un SGBD spécialisé pour une application particulière comme la CAO. En général, un système dédié est un ensemble de d'applications qui utilise un système de gestion de fichiers. Cette approche s'oppose aux SGBD génériques qui visent des domaines d'application plutôt que des applications particulières. Pour les

SGBD génériques, on observe une première étape de développement aboutissant à une première version du SGBD suivie d'une étape d'extension correspondant au réglage pour de meilleures performances et à l'ajout de fonctionnalités pour les versions suivantes.

Il aura fallu en moyenne deux décennies pour chacune des deux premières générations de SGBD (navigationnels et relationnels) arrive à maturité. Un critère de maturité évident est la standardisation du langage (CODASYL et SQL). La période de maturité a été réduite pour les SGBDO qui ont profité des avancées des modèles relationnels et des langages de programmation orientés objets.

C - Pourquoi des bases de données objets ?

La simplicité du modèle relationnel a fait son succès : types simples, ensemble de tables plates, algèbre relationnelle permettant la construction de requêtes avec une base mathématique solide. Malheureusement, le modèle relationnel devient inadapté lorsqu'il est mis en œuvre dans des domaines d'application faisant intervenir des données fortement structurées. Nous présentons dans ce qui suit quelques-unes de ses insuffisances.

I - Nature des applications :

Les applications de gestion classiques sont caractérisées par des types de données simples

(entier, réel, date, . . .), des articles plats, un besoin d'efficacité lors des accès aux données, une gestion de la concurrence, de la répartition et de la sécurité. Les SGBD relationnels satisfont cet ensemble de besoins. Les domaines d'application non traditionnels des bases de données comme l'ingénierie (CAO, FAO, génie logiciel, . . .), la bureautique, l'administration de réseaux et actuellement les données sur le web engendrent des besoins nouveaux. Dans ce cas, le modèle de données doit être beaucoup plus riche avec de nouveaux types de base (graphique, texte, données géométriques, . . .) et doit permettre l'utilisation de structures complexes (hiérarchie ou graphe). De plus, le langage doit permettre l'expression de calculs ou de mises à jour complexes sur des objets de base. Enfin, de nouvelles fonctionnalités sont attendues : objets multimédias, gestion de version ou de configuration, transactions évoluées etc.

II - Gestion des données fortement structurées :

Le modèle relationnel est orienté tuple. Il permet la manipulation de relations plates dont le schéma est décrit par un ensemble d'attributs définis sur des domaines atomiques. Il n'offre pas de solutions satisfaisantes pour gérer des données dont la structure est complexe.

En effet, un objet fortement structuré est décomposé en un ensemble de n-uplets répartis dans différentes relations plates et la notion de hiérarchie sous-jacente à la structure de tels objets se traduit par des références entre n-uplets. La reconstitution de tels objets à travers un modèle relationnel est couteuse car elle impose la réalisation de jointures entre les différentes relations qui les décrivent.

III - Intégration dans un langage de programmation :

Le langage normalisé SQL est incomplet (pas de structures de contrôle complexes par exemple) et exige de recourir à son intégration dans un langage de programmation. Le couplage de SQL avec un langage de programmation offre l'accès aux données relationnelles depuis le langage de programmation grâce à un précompilateur ou à des appels dynamiques. La correspondance entre les variables du programme et les requêtes SQL est assurée par la notion de curseur et ses opérations associées (déplacer, affecter, initialiser,). La programmation avec deux langages, l'un procédural et l'autre déclaratif, est assez délicate. De plus, ce couplage n'est pas homogène : systèmes de types différents, paradigmes de programmation différents (impératif d'un coté, assertionnel de l'autre), styles de programmation différents (élément par élément d'une part, ensembliste d'autre part). Des API comme ODBC ou JDBC permettent ce couplage de façon générique.

IV - Conversion de types ou impedance mismatch :

L'intégration de SQL dans un langage de programmation pose le problème de dysfonctionnement (impedance mismatch) : on encapsule des tables dont les attributs ont des types qui ne sont pas les mêmes que ceux du langage de programmation. La résolution de ce problème implique des conversions de types ainsi que des contrôles de types qui compliquent la programmation.

V - Les interfaces de manipulation :

Les interfaces de manipulation des SGBD sont fondées sur des langages ensemblistes ou prédicatifs simples et puissants parmi lesquels SQL joue le rôle d'un standard. Elles peuvent également être fondées sur une approche « langage naturel » ou sur une approche graphique.

Elles ne sont cependant pas suffisantes lorsque l'utilisateur manipule des applications complexes dans lesquelles le volume de données gérées est très important.

D - Bases de données objets :

Les bases de données objets visent à résoudre les problèmes évoquées dans la section précédente par trois moyens complémentaires :

- réduire, voire éliminer les dysfonctionnements (en particulier l'impedance mismatch) entre langage de programmation et langage de bases de données
- supporter directement les objets arbitrairement complexes grâce à un modèle à objets
- partager le code réutilisable des applications au sein du SGBD.

Bien qu'il y ait consensus sur les fonctions que doit accomplir un SGBDO, il n'y a pas de modèle de base de données objets unique comme c'est le cas pour les bases de données relationnelles. Les modèles de bases de données objets sont issus des langages à objets et des modèles de données sémantiques (inspirés des réseaux sémantiques et des modèles de connaissances issus de l'intelligence artificielle). Ils résultent de l'évolution de modèles comme le modèle relationnel ou le modèle entité/association.

L'objectif commun de tous ces modèles est de pouvoir capturer le plus de sémantique des données en termes de concepts tels que : entité, identité d'entité, association agrégation, généralisation, spécialisation, d'où leur nom de modèle conceptuel. Ces modèles sont ensuite traduits vers des modèles de base de données comme par exemple le modèle relationnel.

I - Base de données objets : Dentition

Une base de données objets est une organisation cohérente d'objets persistants et partagés par des utilisateurs concurrents.

II - Système de gestion de base de données objets :

Un système de gestion de bases de données objets est un SGBD qui fournit les fonctions de base d'un SGBD classique, c'est-à-dire :

- l'intégrité des données : les objets de la base doivent satisfaire les contraintes d'intégrité sémantiques indépendamment des pannes
- la persistance (ou permance) des données : les objets sont stockés indépendamment des programmes qui les créent et doivent être accédées rapidement grâce à des opérations d'accès
- la gestion des transactions : les programmes d'accès à la base de données sont des transactions qui vérifient les propriétés d'ACIDité (Atomicité, Cohérence, Isolation et Durabilité) grâce aux mécanismes de contrôle de concurrence et de fiabilité ;
- l'indépendance physique des données : le schéma conceptuel de la base de données n'impose pas d'implantation particulière des données, celle-ci est décrite par le schéma physique
- la possibilité de requêtes : les requêtes portant sur le schéma conceptuel ou externe doivent pouvoir être exprimées dans un langage déclaratif (comme SQL) qui se prête à l'optimisation.

E - Object Definition Language :

Pour permettre une utilisation directe des types des langage objet l'ODMG (Object Database Managment Group) a choisi de définir un modèle abstrait du définition de base des données objet, mise en œuvre par un langage appelé ODL : Object Definition Language. Ce modèle est ensuite adapté à un langage objet particulier : l'ODMG propose un standard d'intégration en C++, Smalltalk et JAVA.

ODL est un modèle abstrait du définition de base des données objet

ETUDE CONCEPTUELLE

A - Introduction :

Le présent rapport est consacré à la description du dossier conceptuel. Nous l'entamons par décrire l'architecture de notre application. Ensuite, nous présentons l'outil de conception choisi. A la fin de ce rapport, nous consacrons une partie à la conception qui représente une étape importante du processus de réalisation d'un projet informatique. Cette partie s'intéresse essentiellement à l'interaction du système avec son environnement, elle permet de faciliter le travail et de certaines tâches et ce en décomposant le système en des sous-systèmes afin de cerner les difficultés qu'un développeur peut rencontrer.

B - Outils de conception :

I - Présentation :

UML (Unified Modeling Language) est un langage de notation graphique standardisé qui peut être utilisé dans des différents domaines de modélisation (notamment les applications de gestion et les sites web). Il permet de spécifier, de représenter les composantes d'un système informatique et de mettre l'accent sur son utilisation.

II - Vue statique :

1 Diagrammes des cas d'utilisation :

Le digramme des cas d'utilisation simule le fonctionnement global du système et ses interactions avec l'environnement. Ce type de diagramme intervient tout au long du cycle de développement depuis la description du problème en cahier des charges jusqu'à la fin de la réalisation. Il sert donc à définir le produit et son fonctionnement.

1.1 Identification des acteurs :

Un acteur représente un ensemble cohérent de rôles joués par des entités externes (utilisateurs, dispositifs matériels ou autres systèmes) qui interagissent directement avec l'application. De ce fait, le principal acteur de notre application est l'utilisateur.

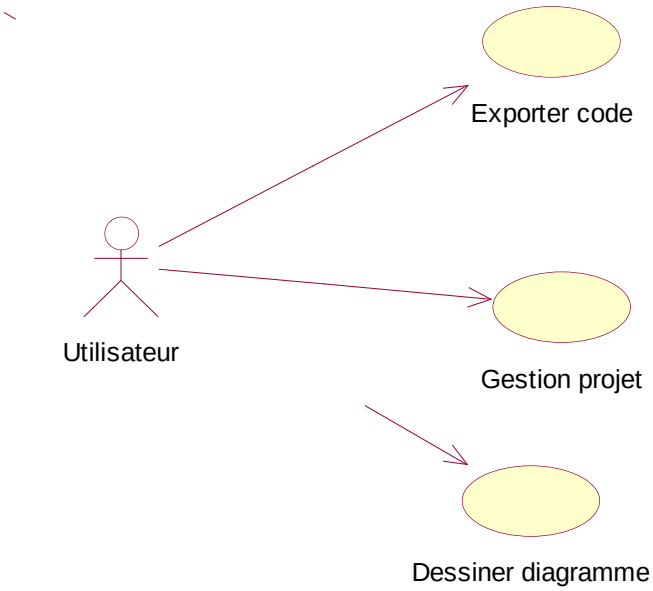
- L'utilisateur : peut dessiner diagramme, gérer projet, exporter code.

1.2 Identification des cas d'utilisation :

Dans cette partie, nous représentons les différentes fonctionnalités de notre application à travers les différents cas d'utilisation.

Un cas d'utilisation est une modélisation d'une fonctionnalité ou d'une classe. Il regroupe une famille de scénarios d'utilisation selon un critère fonctionnel. Les cas d'utilisation se déterminent en observant et en précisant, acteur par acteur, les séquences d'interaction du point de vue de l'utilisateur. Ils se décrivent en termes d'informations échangées et d'étapes dans la manière d'utiliser le système. Ils décrivent les interactions potentielles, sans entrer dans les détails de l'implémentation.

a) Description générale :



-

Figure 1: Diagramme des cas d'utilisation général.

b) Cas d'utilisation Exporter Code:

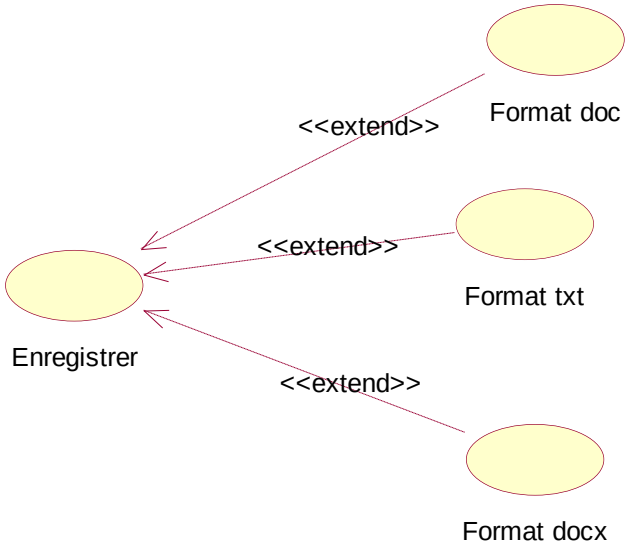


Figure 2: Diagramme des cas d'utilisation Exporter Code.

c) Cas d'utilisation Gestion Projet :

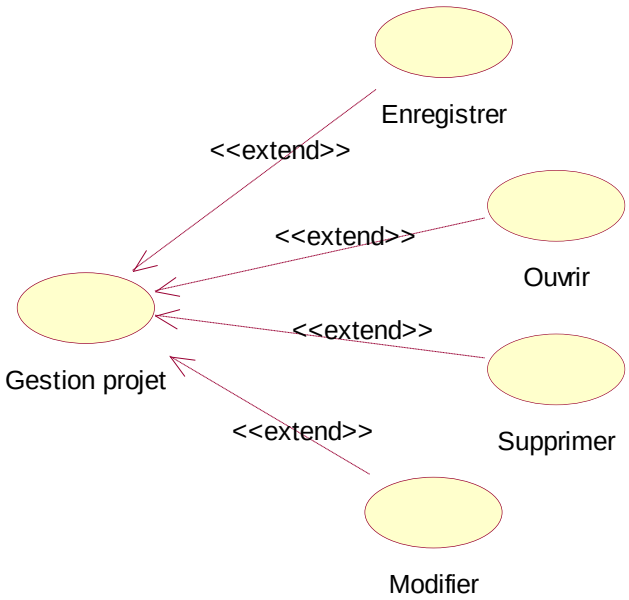


Figure 3 : Diagramme des cas d'utilisation Gestion Projet.

d) Cas d'utilisation Dessiner Diagramme :

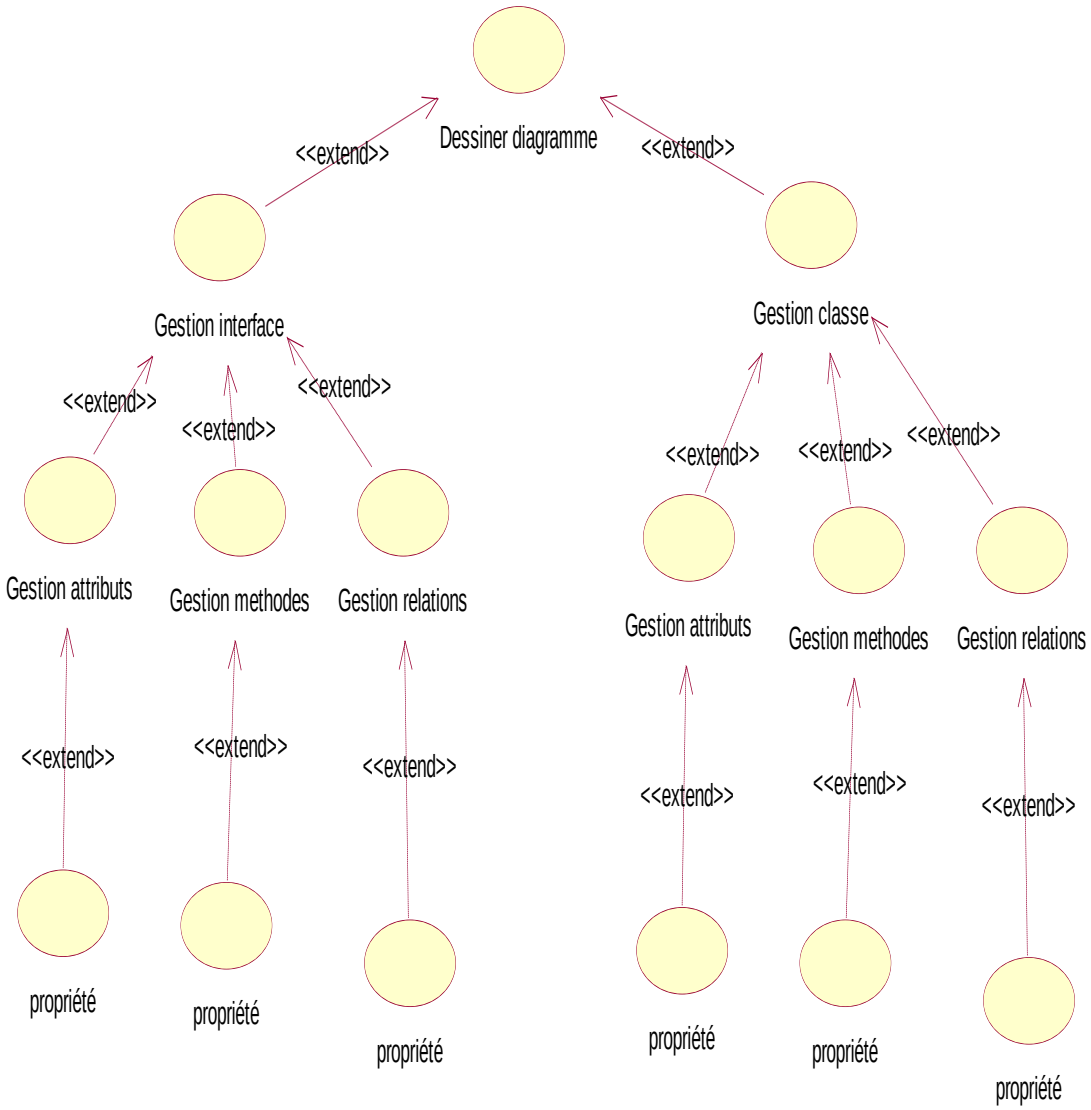


Figure 4: Diagramme des cas d'utilisation Dessiner Diagramme

2 Diagramme de classes :

De point de vue conceptuel, la modélisation d'un ensemble en une classe est justifiée par le fait qu'une classe correspond à une description abstraite d'un ensemble d'objets concrets, de ce fait le digramme de classes nous donne une vue d'objets qui interagissent entre eux pour répondre aux besoins du système.

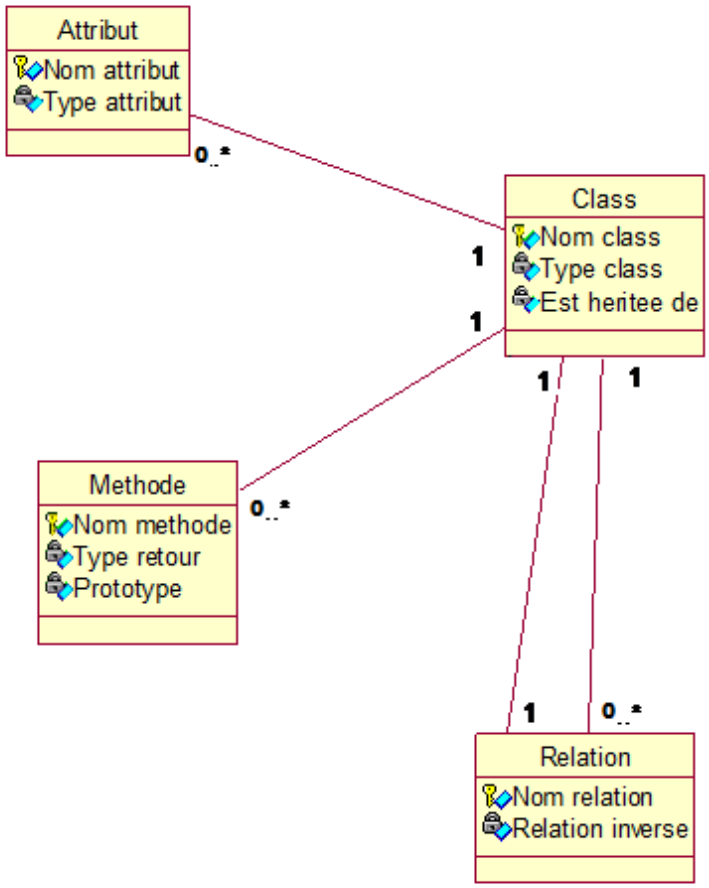


Figure 5: Diagramme de gestion classe

3 Diagramme d'activité :

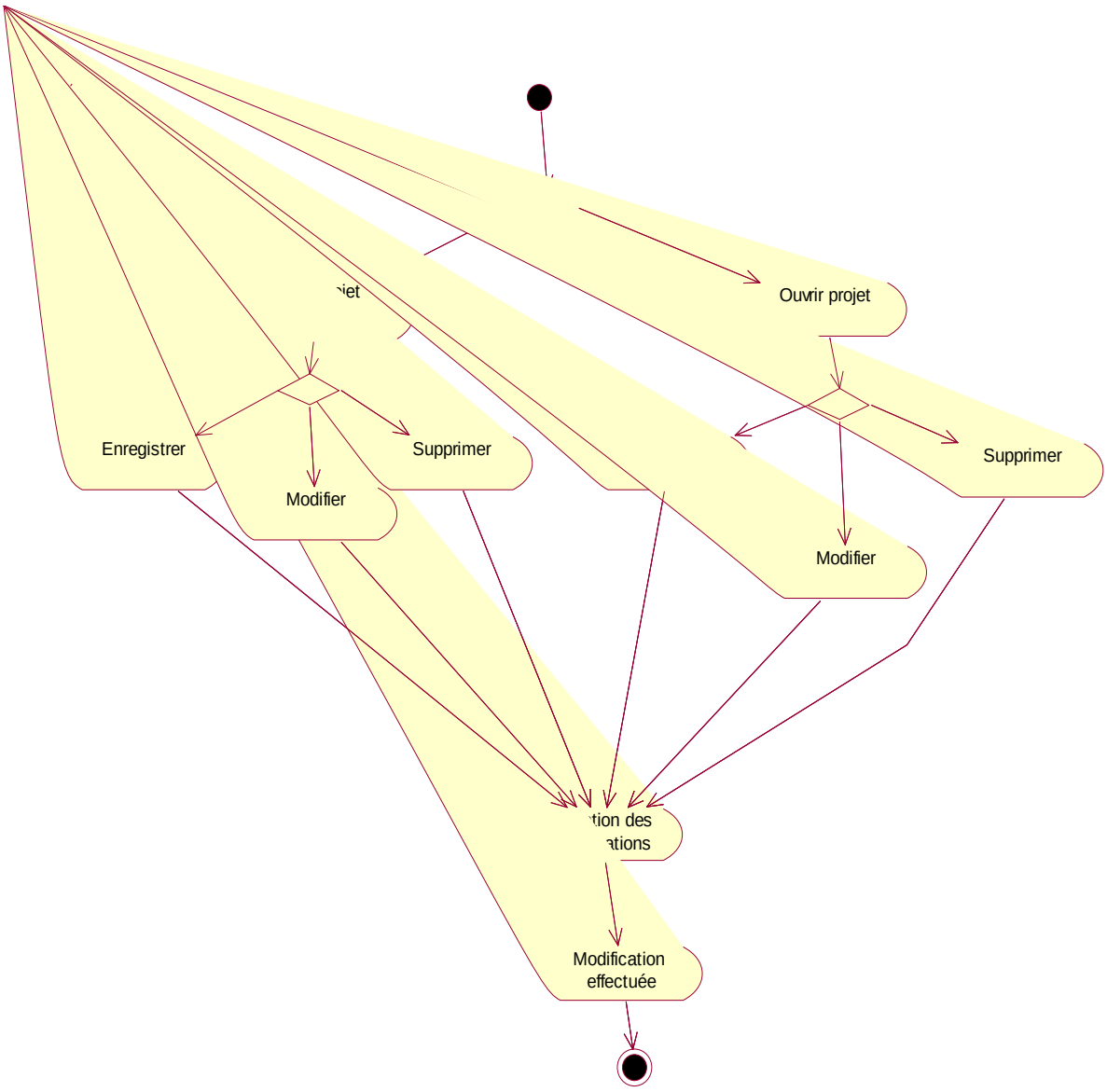


Diagramme d'activité de Gestion projet

III - Vue dynamique :

1 Diagramme de séquence:

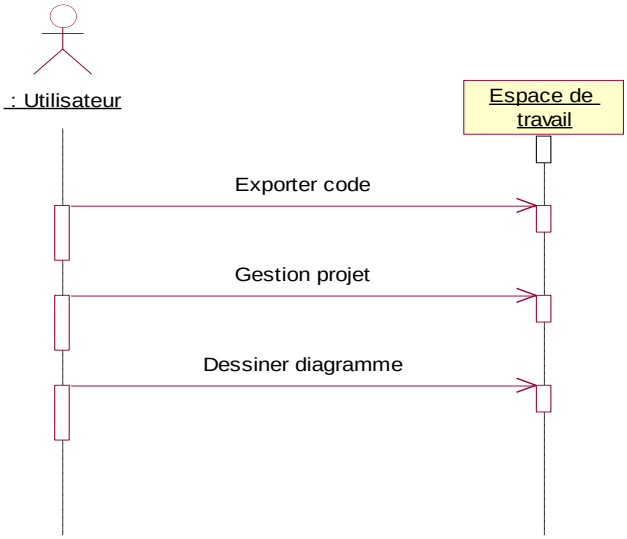


Figure 6: Diagramme de séquence général

Ce diagramme représente le diagramme de séquence général.

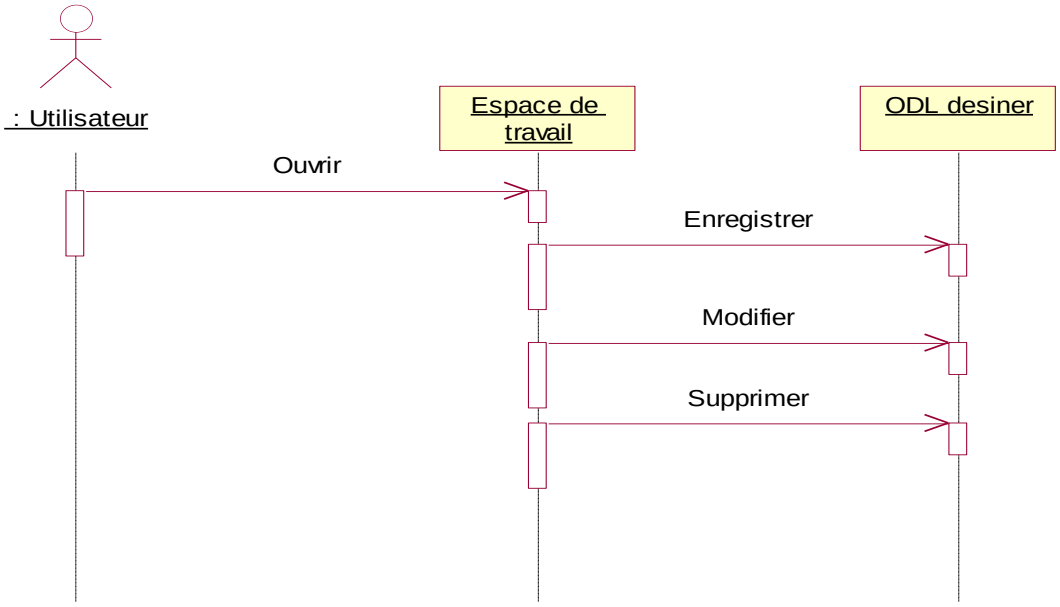


Figure 7: Diagramme de séquence Gestion du projet.

Ce diagramme représente le diagramme de séquence de gestion du projet.

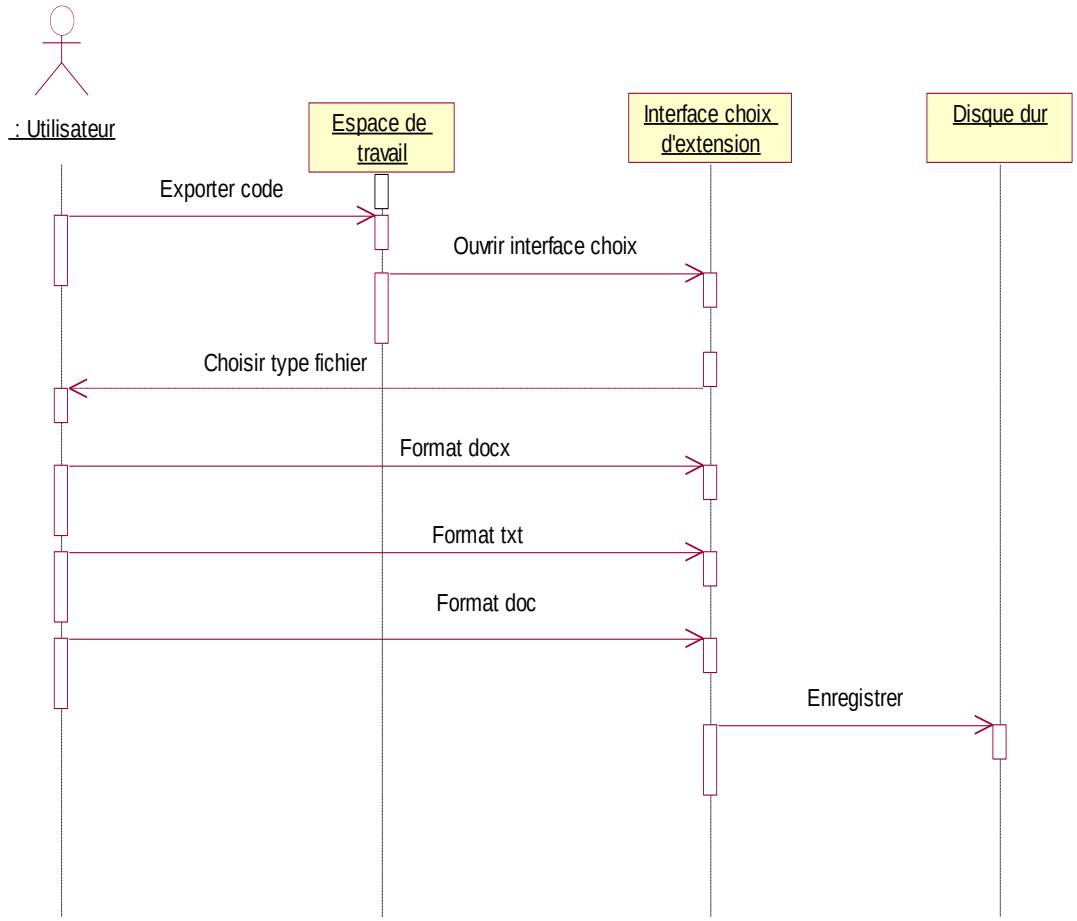


Figure 8 : Diagramme de séquence Exporter Code

Ce diagramme représente le diagramme de séquence Exporter Code.

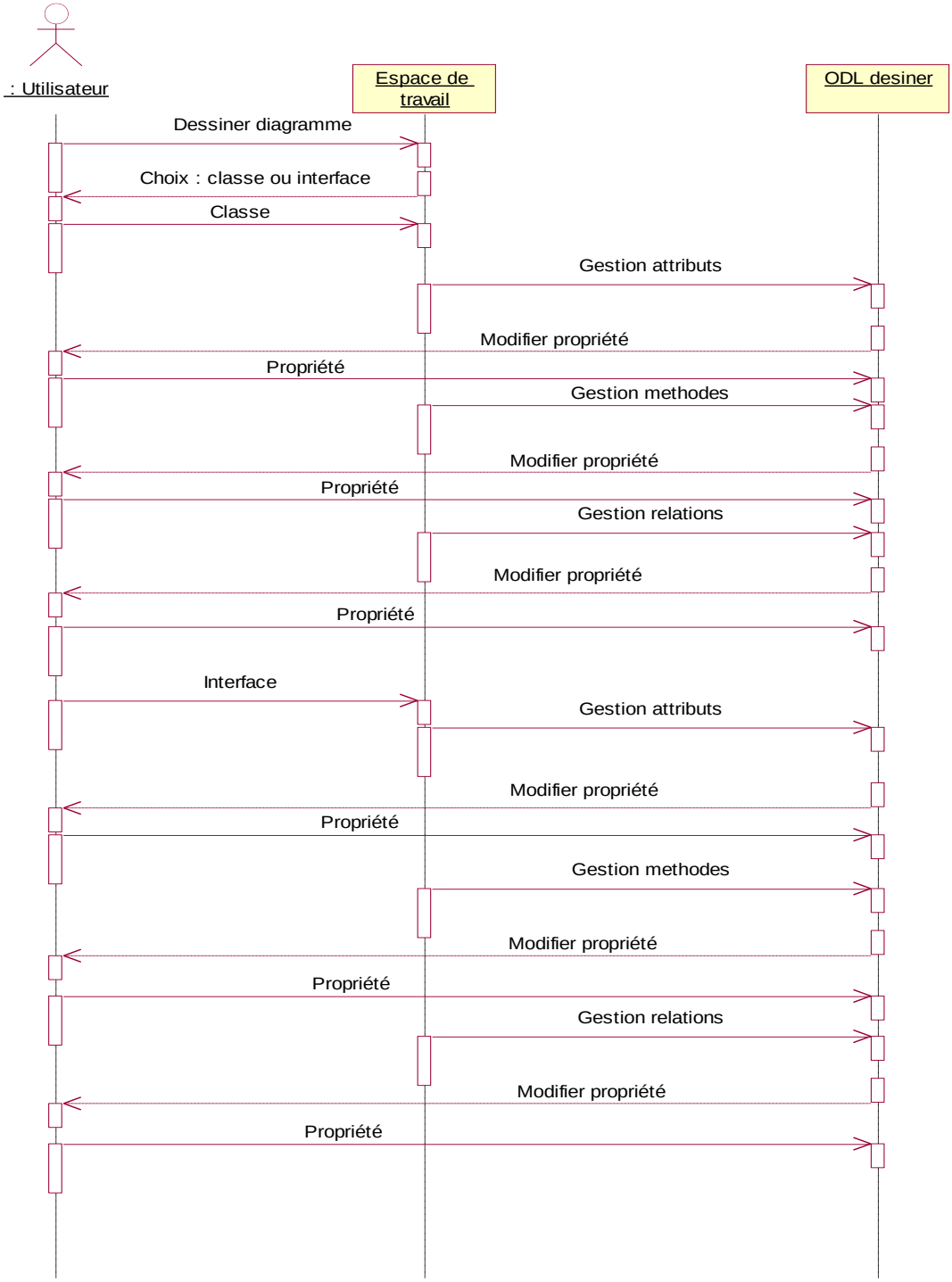


Figure 9: Diagramme de séquence Dessiner Diagramme

Ce diagramme représente le diagramme de séquence Dessiner Diagramme.

C - Conclusion :

Dans ce chapitre, nous avons essayé de présenter la conception de notre site en se basant sur le diagramme de classe (vue statique) et les diagrammes des séquences (vue dynamique).

Nous passons maintenant à décrire les étapes de réalisation de notre application dans le chapitre suivant.

RÉALISATION

Dans ce chapitre on va présenter les différent interfaces de notre Application intitulé « ODL Designer Master Pro ISI » :

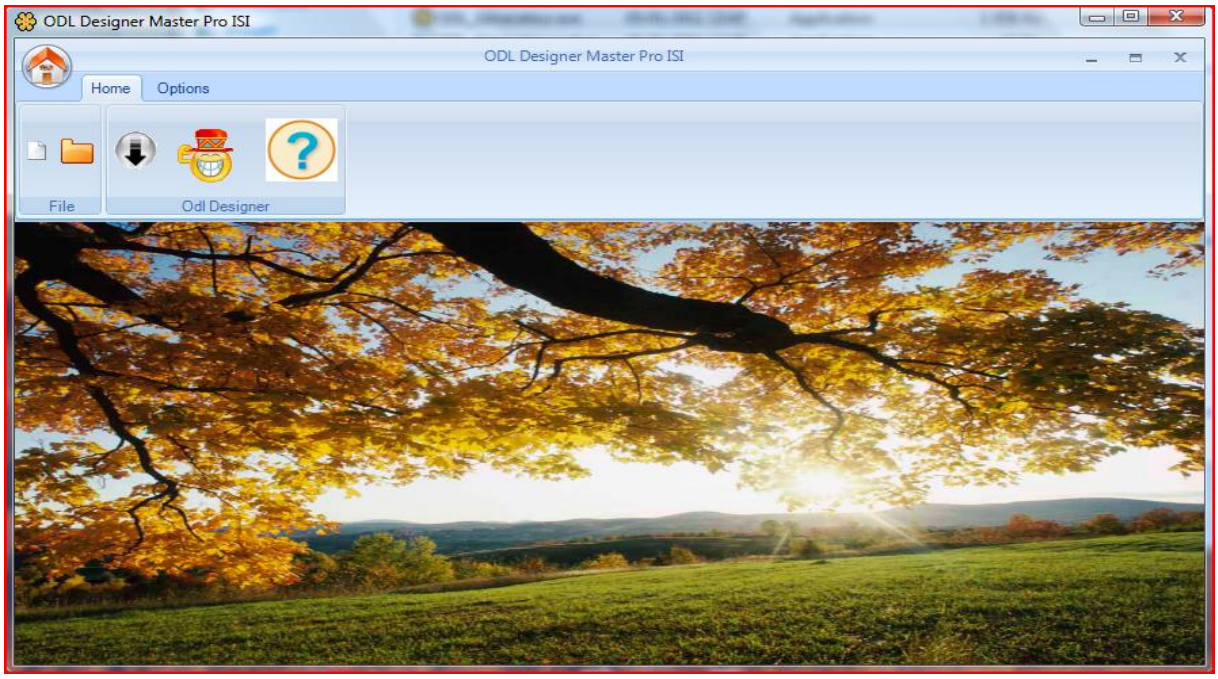


Figure 10: Interface d'Accueil

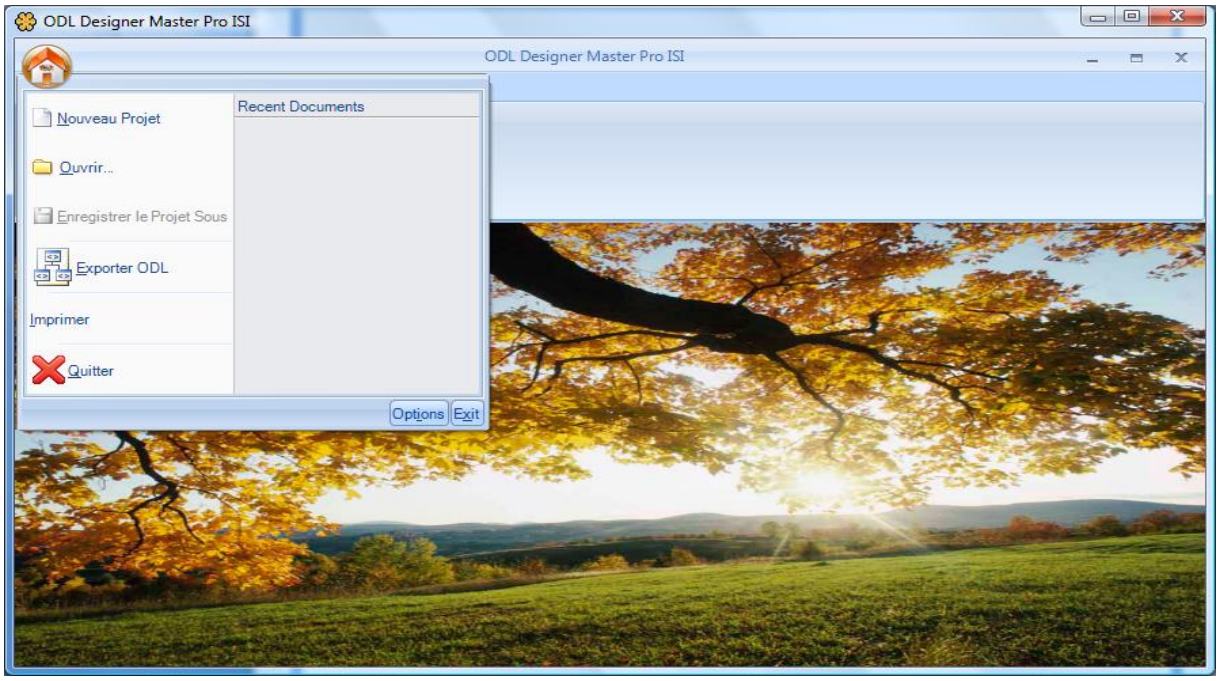


Figure 11 : Menu

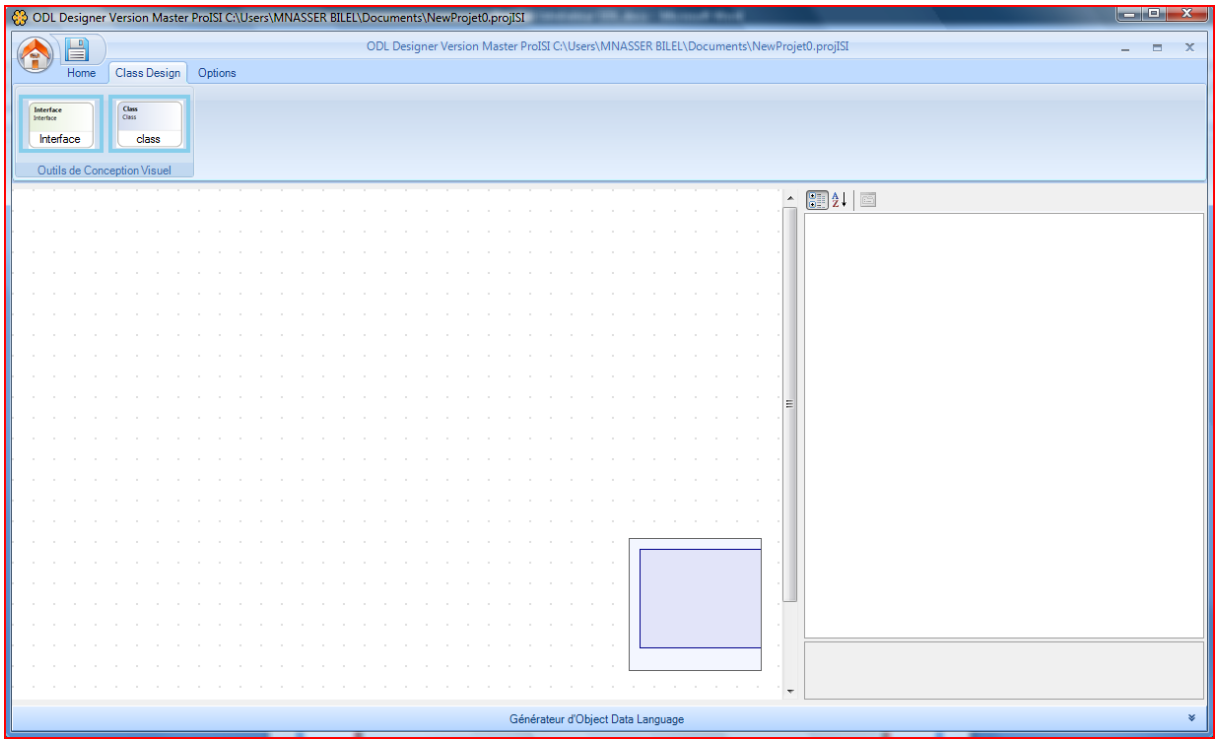


Figure 12 : Interface ODL Designer (projet vide)

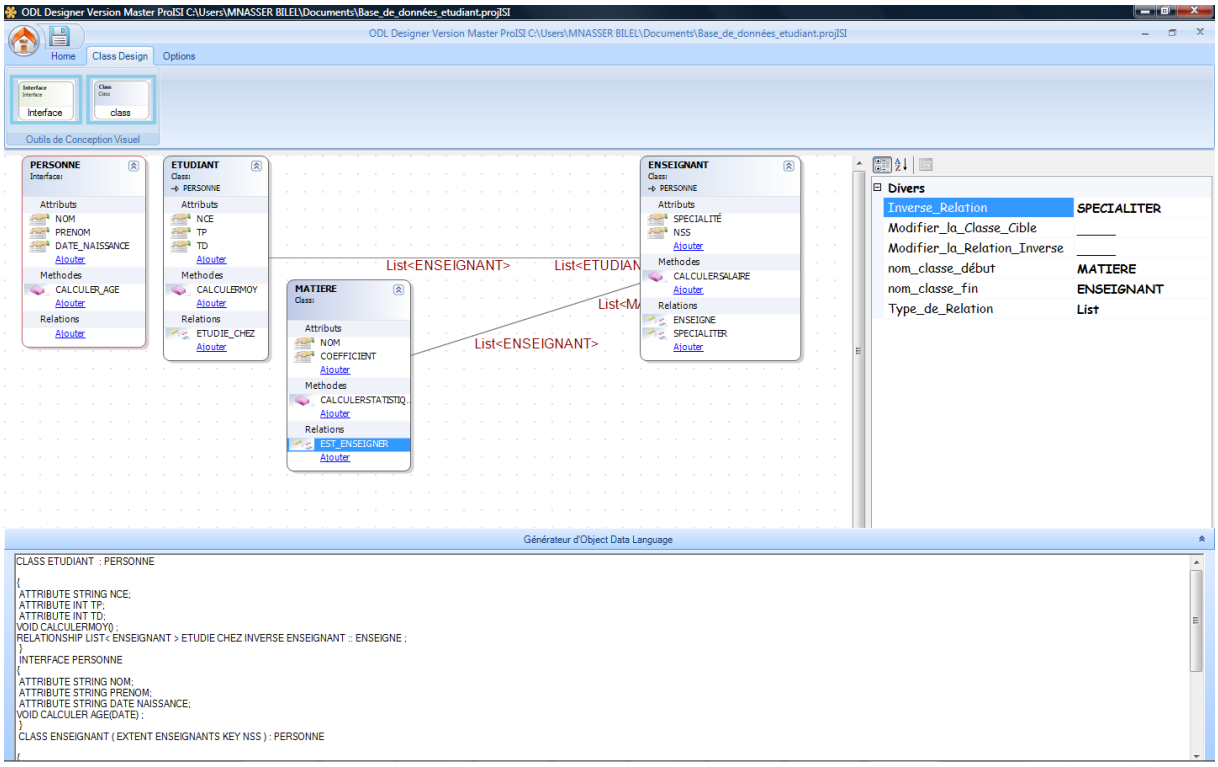


Figure 13 : Interface ODL designer

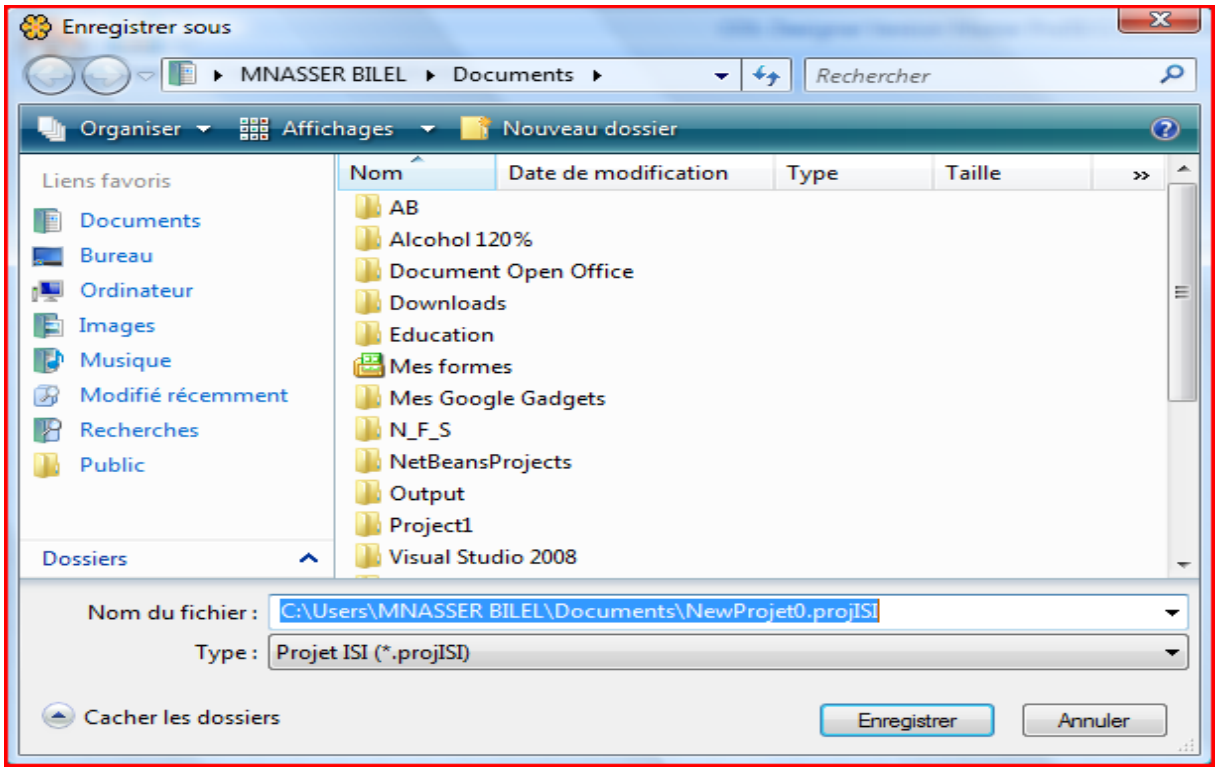


Figure 14 : Interface Enregistrer Sous (espace de travail)

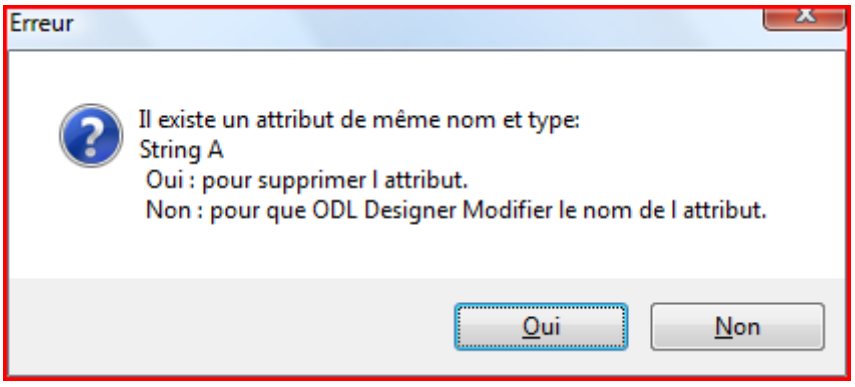


Figure 15 : Interface Erreur (doublons d'attribut)

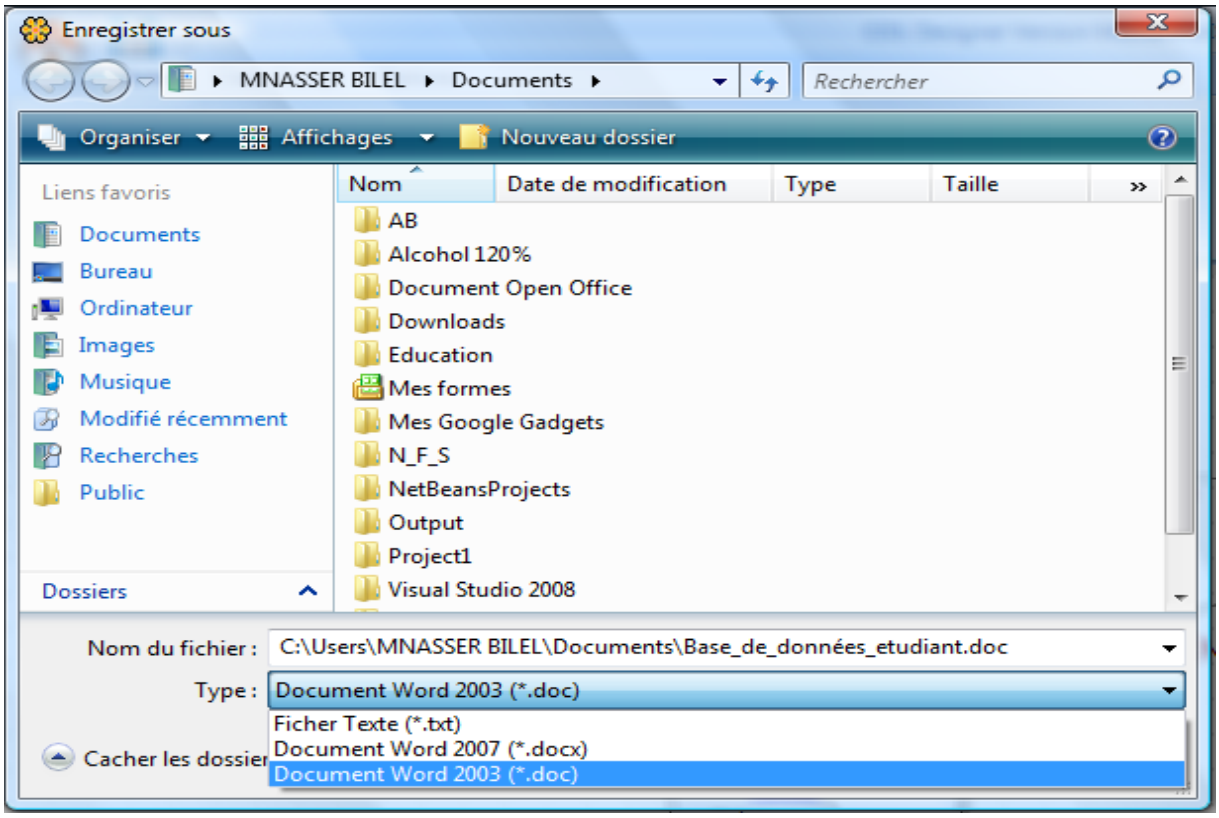


Figure 16 : Interface Enregistrer Sous (code ODL)

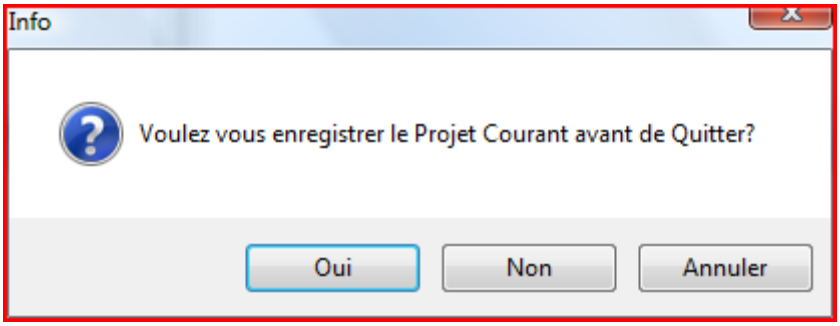


Figure 17 : Interface Avertissement de Sauvegarde

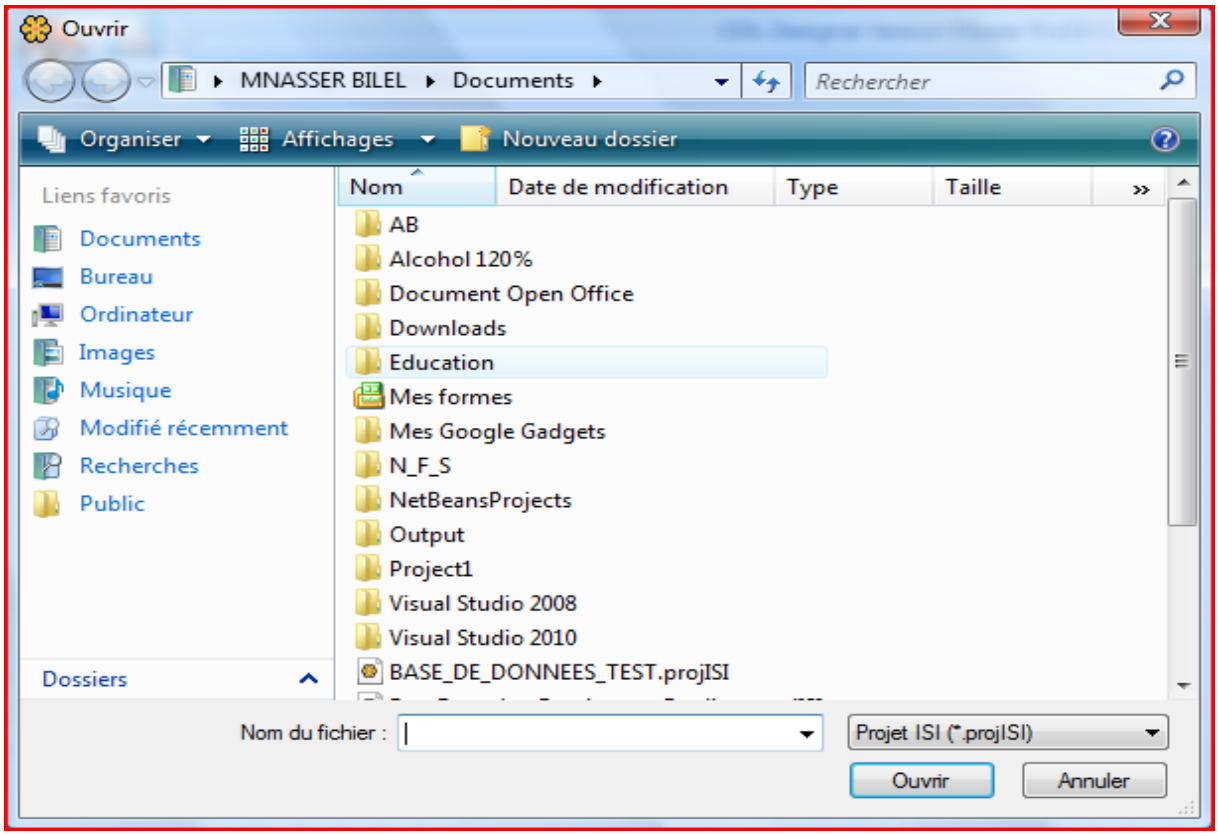


Figure 18 : Interface Ouvrir Projet

Figure 19 : Interface A Propos