EFREI
ECOLE D'INGENIEURS DES TECHNOLOGIES
DE L'INFORMATION ET DU MANAGEMENT

EFREI I3-IR SI2
23 janvier 2006

PROJET DE FIN D'ETUDE

Faure Rémi

Mouret Clément

Nguyen Anh Tai

Table des matières

1 Introduction **1**

1.1 Terminologie et sigles utilisés 2

1.2 Documents de référence 2

2 Analyse des besoins **3**

2.1 Les bases de données dans le monde professionnel 3

2.2 Développement et bases de données 3

2.2.1 Types de logiciels 3

2.2.2 Développement 3

2.2.3 ORM, la solution ? 4

2.3 Conclusion : les bases de données, un développement en pleine extension 5

3 Hornet : Un outil de développement pour Visual Studio .NET **6**

3.1 Object Relational Mapping Development Tool 6

3.2 Bases de données et langages de programmation 7

3.2.1 Systèmes de gestion de base de données : 7

3.2.2 Langages de programmation : 8

3.3 Un outil d'aide au développement 8

3.3.1 L'utilisateur : le développeur 8

3.3.2 Intégration au processus de développement 8

3.3.3 Récapitulatif des avantages d'Hornet 9

4 Spécifications **10**

4.1 Fonctionnalités de l'application 10

4.1.1 Entrées et sorties de l'outil 10

4.1.2 Compatibilité du prototype avec les SGBD et langages 10

4.1.3 Environnement de travail 11

4.2 Description de l'application côté utilisateur 11

4.2.1 Installation 11

4.2.2 Configuration 11

4.2.3 Utilisation 11

4.3 Représentation UML du fonctionnement de l'Add-In 13

4.3.1 Cas d'utilisation 13

4.3.2 Description des cas d'utilisation 14

4.4 Spécifications de la bibliothèque de classes générée 17

4.4.1 Cas d'utilisation 17

4.4.2 Problèmatiques 17

5 Gestion de projet et PDL **18**

5.1 L'équipe du projet Hornet 18

5.2 Logique de déroulement et planification 19

5.2.1 Méthodologie de développement 19

5.2.2 Objectifs des étapes 20

5.2.3 Fournitures 21

5.2.4 Détails du plan d'itération 21
5.2.5 Calendrier 23
5.2.6 Tableaux de bord 28
5.3 Méthodes, techniques et outils 28
5.3.1 Conception & Développement 28
5.3.2 Organisation de la communication 28
5.4 Maîtrise des risques 29
5.5 Assurance qualité 29
5.5.1 Recueil et analyse des problèmes 29
5.5.2 Gestion de la documentation technique 30
5.5.3 Gestion de configuration 30
5.5.4 Respect de normes de développement 31

6 Architecture logicielle 32

6.1 Architecture générale 32
6.2 Interface SGBD : Extraction du schéma de la base de données 34
6.2.1 Concept 34
6.2.2 Fonctionnement 34
6.3 Add-In Visual .NET 2003 35
6.3.1 Visual Studio 35
6.3.2 Architecture 35
6.4 Moteur de template : Génération des fichiers sources Hornet 38
6.4.1 Concept 38
6.4.2 Fonctionnement du template 39
6.5 Bibliothèque de classes Hornet : *Data Persistent Objects Library* 39

6.5.1 Concept et structure de la bibliothèque de classes 39
6.5.2 La couche de base Hornet 39
6.5.3 La couche de paramétrage 41
6.5.4 Encapsulation des données 41

7 Bilan **43**

7.1 Résultat obtenu 43
7.2 Difficultés rencontrées 43
7.2.1 Recherche d'informations 43
7.2.2 Application de la méthodologie de travail 44
7.2.3 Gestion du temps 44
7.3 Evolution 44
7.4 Apports du projet pour ses membres 45
7.4.1 Apport sur le plan technique 45
7.4.2 Méthodologie de travail 45
7.4.3 Un travail valorisant vu de l'extérieur 46

A Tableau de bord **47**

B Normes syntaxiques **48**

B.1 Mise en place de la norme 48
B.2 Conventions de nommage générales 48
B.3 Conventions de nommage spécifiques 50
B.4 Autres recommandations 51
B.5 Documentation du code 52

C Environnement Visual Studio **53**

D Diagrammes annexes **54**

D.1 Bibliothèque de classes Hornet : *Data Persistent Objects Library* 54

D.1.1 *Data Field* 54

D.1.2 *Data Object* 55

D.1.3 *SQL Query Engine* 57

E Critères d'évaluation d'un ORM **58**

E.1 Basic features 58

E.2 Useful extended features 58

E.3 Flexibility 58

E.4 Assistance, ease of use 59

E.5 Optimizations, performance, design 59

E.6 Evolution, compatibility 59

F Licences **61**

F.1 Freeware ou Shareware 61

F.2 Licences Open-Source 66

Chapitre 1

Introduction

L'utilisation de données persistantes est maintenant incontournable dans la logique informatique actuelle. Les données deviennent des ressources stratégiques : l'entreprise capitalise et exploite des données pour améliorer sa productivité, gérer son savoir faire, partager des informations... Les applications en interaction avec un serveur de gestion de base de données sont très nombreuses (SI, ERP, CRM,...). Avec l'explosion de l'informatisation des systèmes d'information, elles représentent un marché de taille pour le monde du logiciel.

Concevoir, puis développer les interfaces d'accès aux données est une étape de plus en plus longue et complexe, la quantité d'information à échanger étant en constante augmentation. Cette étape est coûteuse pour les éditeurs de logiciels. En effet elle monopolise des ingénieurs ou/et des techniciens pendant une période conséquente.

La problématique pour l'entreprise, et le monde du logiciel libre, est la suivante : comment réduire les coûts de développement liés à l'implémentation des interactions avec un serveur de gestion de base de données ? L'une des méthodes consiste à exploiter le concept d'*Object Relational Mapping* (ORM) permettant d'automatiser les liens entre les données dynamiques d'une application chargée en mémoire et les données stockées dans une base de données. Un tel concept permet de s'affranchir de la méthode d'interaction entre les données en mémoire et les données stockées sur un serveur, voir même de la méthode de stockage des données.

Notre projet de fin d'étude, *Hornet Project - Object Relational Mapping Developpment Tool*, consiste à implémenter un tel concept à travers un outil de développement conçu par et pour des développeurs permettant l'automatisation du passage du modèle relationnel des données vers le modèle objet. Dans le cadre du premier prototype, l'outil est réalisé pour l'environnement .NET de Microsoft, de plus en plus utilisé en entreprise, sous la forme d'un Add-In pour Visual Studio .NET 2003.

1.1 Terminologie et sigles utilisés

- PFE : Projet de Fin d'Etude
- PDL : Plan de Développement Logiciel
- CRA : Compte-rendu d'avancement
- .NET : Technologie Microsoft pour le developpement d'application
- SGBD : Serveur de Gestion de Base de Données
- BDD : Base De Données
- SQL : Server Query Language
- ORM : Object Relational Mapping
- CVS : Concurent Version System
- ML : Mailing List (Liste de diffusion)
- IDE : Integrated Development Environment

1.2 Documents de référence

- Cahier des charges V1.0
- Document de spécification V1.0

Chapitre 2

Analyse des besoins

2.1 Les bases de données dans le monde professionnel

Les serveurs de gestion de bases de données (SGBD) sont depuis une dizaine, voir une vingtaine d'années très utilisés dans le monde de l'entreprise tous domaines confondus. Les besoins grandissent d'autant plus que les systèmes d'information informatisés connaissent une croissance fulgurante. Des multinationales aux PME, les entreprises ont besoin de capitaliser puis d'exploiter les données de leurs environnements. Les données deviennent des ressources stratégiques. La gestion des données devient alors une réelle problématique pour l'entreprise.

2.2 Développement et bases de données

2.2.1 Types de logiciels

De nombreuses applications informatiques utilisent le stockage de données à partir d'un serveur de gestion de base de données comme Oracle, SQL Serveur, MySQL...

Voici une liste, non exhaustive, de types de logiciel dont la gestion des données est primordiale et conséquente en terme de développement :

- Système d'information
- Progiciel de gestion intégré (ERP)
- Application de gestion
- Site internet commercial
- Fichiers clients (ou Customer RelationShip Management - CRM)

2.2.2 Développement

Le codage de la partie dialogue avec la base de données d'une solution informatique, d'un système d'information par exemple, est une part importante du développement. Il est difficile de chiffrer un pourcentage de temps de développement par rapport à l'ensemble du projet, mais le constat, par rapport à notre expérience, est le suivant : concevoir puis développer l'interface d'accès aux données est une tâche longue et donc par définition coûteuse puisque qu'elle monopolise des ressources (des

techniciens et/ou des ingénieurs) pendant une période relativement importante.

Le ou les développeurs en charge du développement de la partie base de données de l'application ont besoin d'un solide bagage technique pour pouvoir appréhender toutes les difficultés de développement d'un programme interagissant avec un serveur de gestion de base de données. Une connaissance du langage SQL est évidement nécessaire, même si le niveau de connaissance peut varier selon la difficulté des requêtes à effectuer sur la base de données. De plus, connaître le fonctionnement et les spécificités des SGBD est primordiale.

2.2.3 ORM, la solution ?

L'idée est la suivante : automatiser ou réduire la phase de développement permettant d'accéder aux données de la base de données. En effet, le traitement des données pour les stocker dans une base de données, est une tâche répétitive et très similaire d'un projet à un autre. Il est donc peu avantageux pour l'entreprise de faire les mêmes opérations encore et encore d'un projet à un autre.

Il existe différents types d'outil permettant de réaliser ce type d'automatisation dont les ORM (*Object Relational Mapping*). Ils permettent la génération de classes objets à partir d'une base de données relationnelle. Le développeur peut ainsi accéder à la base de données sans le moindre effort et une connaissance du langage SQL limitée.

Sun Microsystems, avec son langage de programmation Java, a bien compris l'intérêt d'un tel concept : réduire les coûts de développement en aidant le développeur à interagir avec la base de données de façon intuitive et facile d'accès. Sun a ainsi rajouté un degré d'abstraction entre la base de données et le programmeur pour que ce dernier soit libéré des contraintes de sauvegarde de ses données. C'est pourquoi cette notion est incluse dans l'environnement Java. Il faut noter que la solution développée par Sun pour le langage Java est le passage de l'objet à la base de données et non de la base de données à l'objet (ORM). Le but reste identique, mais la mise en oeuvre et l'intégration dans le processus de développement sont différents.

A l'inverse, il n'existe pas, à l'heure actuelle, de solution de ce type développée par Microsoft pour l'environnement de développement Visual Studio .NET 2003 qui est de plus en plus utilisé en entreprise. Certains éditeurs de logiciel (payants ou libres) ont développé des logiciels de type ORM. Les plus connus sont :

- NHibernate
- Data Tier Modeler
- OlyMars
- Object Borker

L'étude des solutions déjà existantes a montré que les solutions ORM actuellement sur le marché ne sont pas sans défauts. On retrouve notamment les défauts suivants :

- Complexité du paramétrage
- Application externe à l'IDE Visual Studio .NET
- Un mapping table par table (on ne peut plus parler d'objet dans ce cas)
- Logiciel payant (problématique pour un developpement non ou semi-professionnel)
- Complexité du code généré

– Contrainte sur les tables de données

2.3 Conclusion : les bases de données, un développement en pleine extension

Avec l'explosion de l'informatisation des systèmes d'information, le marché des bases de données augmente fortement. Le besoin en développement orienté SGBD n'a jamais été aussi fort qu'aujourd'hui et représente un véritable enjeux pour les entreprises : stocker, capitaliser, gérer et exploiter des données pour améliorer leur productivité. Un outil d'aide au développement comme un ORM semble donc répondre à un véritable besoin pour les entreprises.

Chapitre 3

Hornet : Un outil de développement pour Visual Studio .NET

3.1 Object Relational Mapping Development Tool

L'ORM (Object-Relational Mapping) est un concept qui consiste à relier les bases de données relationnelles aux langages de programmation orientés objet. Le but de l'ORM est de "modéliser" une base de données sous forme d'objets. Aujourd'hui, encore beaucoup de développeurs choisissent de réaliser leur propre mapping entre objets et base de données relationnelle.

Cependant, il existe des outils gratuits ou payants qui sont capables de réaliser ces opérations de mapping. Ces outils permettent de générer les fichiers de code contenant les classes de mapping de la base de données.

Notre projet a pour but de réaliser un outil d'ORM. En effet, nous avons remarqué que l'environnement de développement Visual Studio .NET 2003 ne dispose pas d'outil d'ORM intégré. Visual Studio .NET 2003 peut ainsi être enrichi par un Add-In afin de disposer de ces fonctionnalités d'ORM. Il existe des Add-In payants et gratuits mais aucun ne fait figure de "standard" car Microsoft n'en propose aucun. C'est pourquoi il nous est venu l'idée de réaliser un prototype d'Add-in réalisant le mapping ORM pour Visual Studio .NET 2003. Nous avons choisi comme nom pour cet add-in "Hornet".

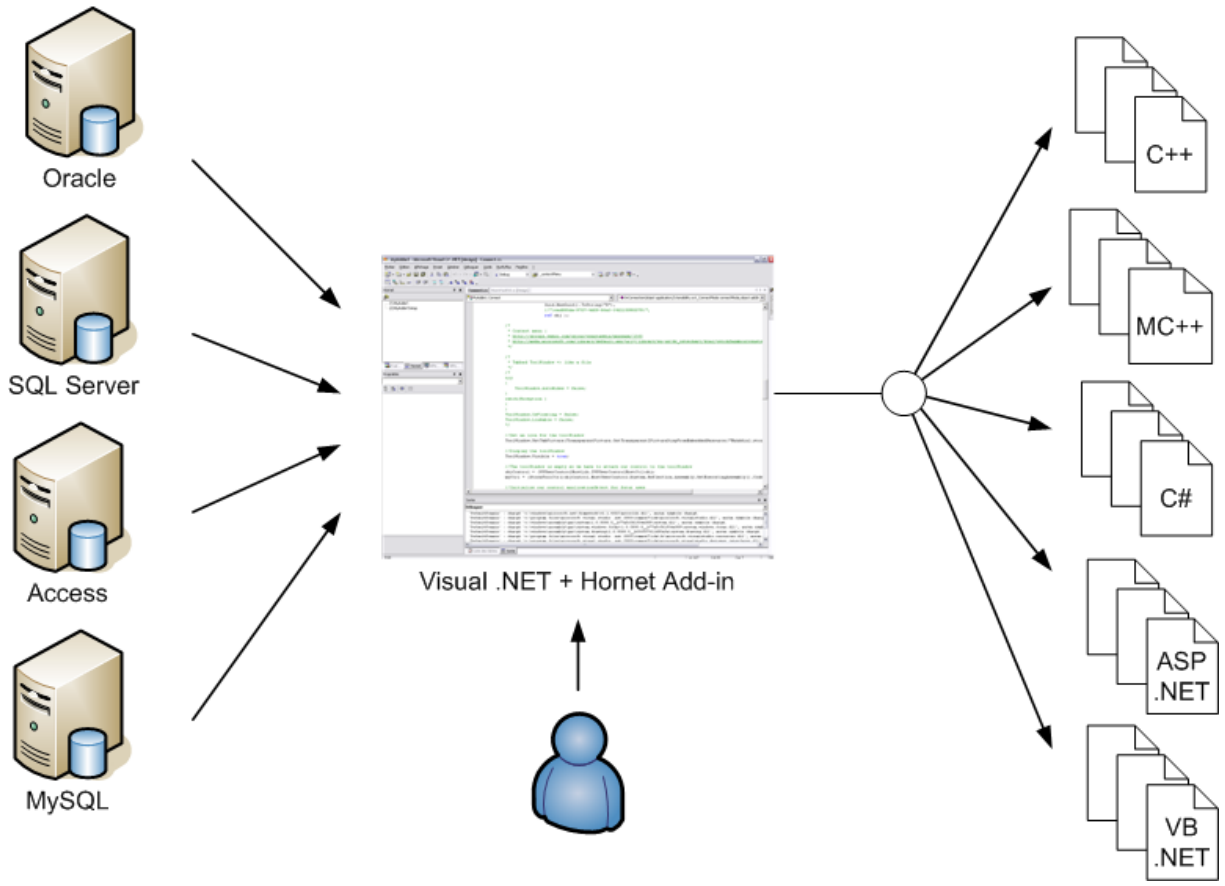


FIG. 3.1 – Principe de fonctionnement de l'Add-In Hornet pour Visual Studio .NET

Ce schéma résume la philosophie de notre outil. A partir d'une base de données, l'Add-in Hornet génère des fichiers sources dans le langage souhaité. Ces fichiers contiendront le code de mapping de la base de données. Il sera offert des fonctions de "configuration"' à l'utilisateur. Cela enrichira ainsi les possibilités de l'utilisateur

3.2 Bases de données et langages de programmation

3.2.1 Systèmes de gestion de base de données :

Plusieurs systèmes de gestion de base de données existent sur le marché. L'idéal serait que l'Add-in soit compatible avec les cinq systèmes de gestion de base de données suivants :

- Oracle
- SQL Server
- Access
- MysQL
- PostgreSQL (A confirmer)

3.2.2 Langages de programmation :

Hornet, étant un Add-in destiné à Visual Studio .NET, le but final serait qu'il propose de générer du code dans les différents langages reconnus par l'environnement de développement .NET.

- C++
- MC++ (Managed C++)
- C#
- ASP .NET
- VB .NET

3.3 Un outil d'aide au développement

3.3.1 L'utilisateur : le développeur

L'outil Hornet s'adresse aux développeurs de logiciel utilisant un système de gestion de base de données. Afin de réaliser le mapping d'une base de données dans une application, un développeur doit avoir des compétences en SQL et SGBD en plus de celles en programmation. L'utilisation de notre outil lui permettra de se passer de compétences poussées en bases de données. L'outil permettrait à un développeur de réaliser un mapping évolué sans pour autant avoir à gérer les problématiques de bas niveau de gestion de la BDD.

3.3.2 Intégration au processus de développement

Par sa forme d'Add-In pour Visual Studio .NET 2003, Hornet s'intègre parfaitement au processus de développement. En effet, le développeur aura à sa disposition l'outil d'ORM de manière permanente et directe. En outre, l'environnement de développement Visual Studio .NET est déjà très répandu.

L'Add-In Hornet généra une bibliothèques de classe de façon à intégrer facilement le code d'accès à la base de données dans des programmes et modules différents. L'utilisation de librairies et non

du code directement intégré dans un module permettra une meilleure lisibilité du code source de l'application réalisée par le développeur utilisateur.

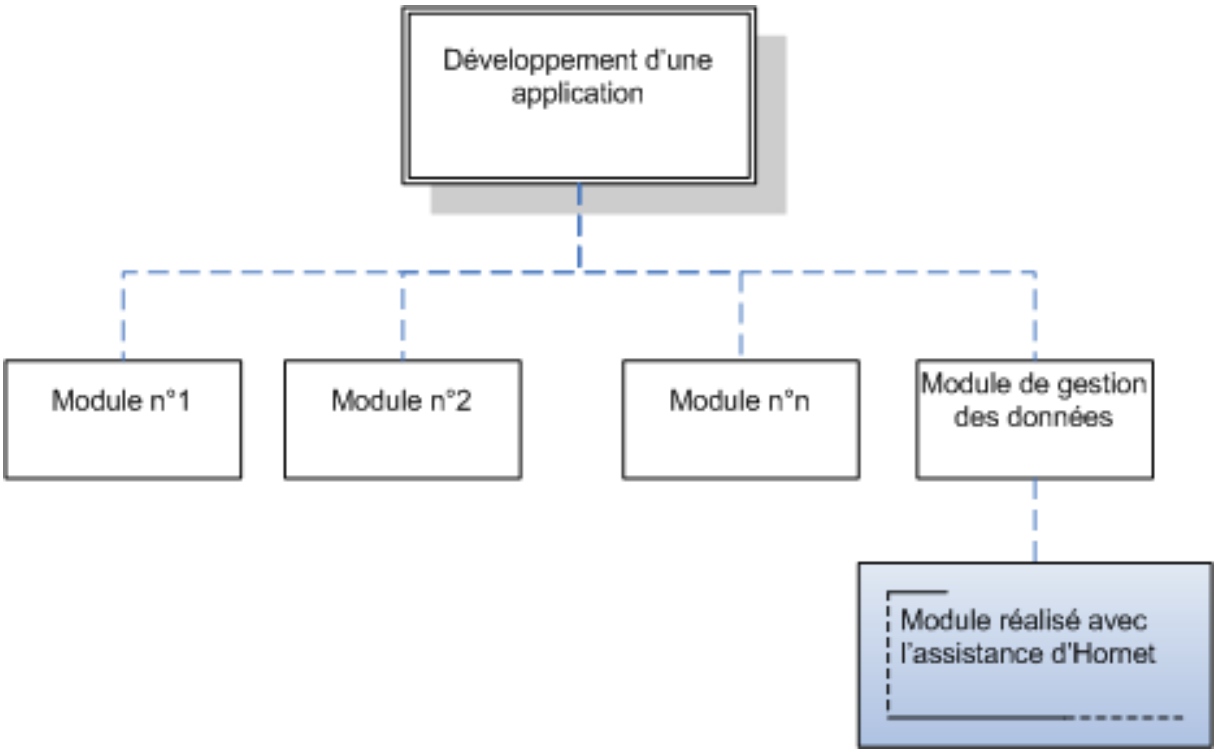


FIG. 3.2 – Un module entier d'application développé par l'Add-In Hornet

Notre outil soulage ainsi les développeurs dans leur travail dans la mesure où un module d'importance majeure sera conçu rapidement et simplement. Le résultat de la génération est facilement exploitable du fait de sa forme (bibliothèque de classes). Cette bibliothèque peut ainsi être exploitée par n'importe quelle autre module de l'application.

3.3.3 Récapitulatif des avantages d'Hornet

- Support multi-formats de bases de données ;
- Support multi-formats de langages de programmation ;
- Accessibilité de l'outil du fait de sa forme d'Add-In pour Visual Studio .NET ;
- Intégration dans le processus de développement (développement d'un module séparé de l'application - bibliothèque de classes en résultat) ;
- Le développeur garde le contrôle de la conception de la base de données ;
- Gain de temps de développement ;
- Accessibilité (connaissances en SQL et SGBD nécessaires réduites) ;
- Paramétrage : l'utilisateur pourra apporter une personnalisation à la bibliothèque de classes générées (renommage des entités et attributs, ajout de relations d'héritage pour les objets générés, sélection des entités à mapper, ...).

Chapitre 4

Spécifications

4.1 Fonctionnalités de l'application

4.1.1 Entrées et sorties de l'outil

L'application Hornet reçoit des données en entrée à plusieurs moments. On distingue ainsi les entrées suivantes :

- La base de données ;
- Les informations de connexion à la base de données ;
- Les paramètres de génération de la bibliothèque de classes.

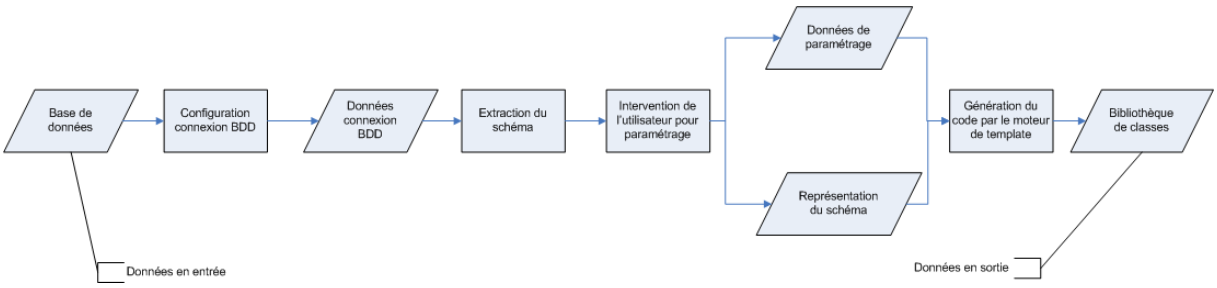


FIG. 4.1 – Séquence de fonctionnement attendu de l'outil

La base de données est l'entrée fondamentale de notre application. Afin d'accéder à la base, l'outil demande à l'utilisateur de fournir les paramètres de configuration pour se connecter à la base de données. Une fois l'extraction du schéma de la base de données effectuée, l'outil permet à l'utilisateur de configurer la génération du code de la bibliothèque de classes.

En sortie, l'utilisateur obtient une bibliothèque de classes qui s'intègre dans la solution *.sln du projet de développement sous la forme d'un projet supplémentaire. L'utilisateur pourra ainsi ajouter dans d'autres projets des références simples vers le projet contenant la bibliothèque.

4.1.2 Compatibilité du prototype avec les SGBD et langages

Dans notre Projet de Fin d'Etude, l'Add-In Hornet est limité en fonctionnalités. En effet, le temps imparti pour ce projet étant limité, notre objectif est de fournir un prototype de l'application.

En terme de compatibilité avec les SGBD, l'Add-In doit fonctionner avec :
– SQL Server ;
– Oracle ;
– MySQL.
L'outil devra s'appuyer sur un standard pour communiquer avec les SGBD. Cela permettra ainsi d'implanter aisément la gestion d'autres SGBD.

Le langage de référence pour le prototype est le C#. Notre objectif est de générer une bibliothèque de classes fonctionnelle dans le langage C#. Cependant, la solution doit proposer une méthode de génération générique qui doit permettre l'implantation de nouveaux langages de manière souple et fiable. Au final, notre but est de pouvoir implémenter le support d'autres langages de programmation sans avoir à retoucher à l'Add-In.

4.1.3 Environnement de travail

Hornet est un Add-In pour Visual Studio .Net 2003. Par conséquent, la configuration requise pour faire fonctionner cet environnement de développement est la même pour Hornet.

La station faisant fonctionner l'Add-In devra être configuré avec :
– Un système d'exploitation Windows 2000 ou supérieur ;
– Le Framework Microsoft .NET Version 1.1 ou supérieure ;
– Le fournisseur de données .NET ODBC ;
– Le pilote ODBC du Système de Gestion de Base de Données utilisé par la base de données utilisateur.

4.2 Description de l'application côté utilisateur

4.2.1 Installation

L'installation de l'Add-In se fait de la manière la plus simple. Un fichier install à exécuter est proposé afin d'intégrer l'Add-In dans Visual Studio .NET.

4.2.2 Configuration

L'utilisateur pourra choisir s'il souhaite rendre le démarrage de l'Add-In automatique, à chaque lancement de Visual Studio .NET.

4.2.3 Utilisation

Par le biais du menu de Visual Studio .NET, l'utilisateur peut lancer l'Add-In à tout moment.

A partir de ce moment, l'utilisateur doit pouvoir lancer une génération de bibliothèque de classes. Pour cela, l'Add-In doit lui proposer une fenêtre pour paramétrer une connexion vers la base de données.

Ces paramètres sont :
- Le nom du serveur ;
- Le login ;
- Le mot de passe ;
- La sélection du type de SGBD utilisé ;
- La sélection du pilote ODBC à utiliser ;
- Le mode de connexion (Remote ou Local).

Une fois l'extraction du schéma de la base de données effectuée, l'Add-In permettra à l'utilisateur de paramétrer la génération de la bibliothèque de classes. Ce paramétrage se fera par le biais d'une fenêtre d'affichage intégrée dans Visual Studio. L'application devra être en mesure d'afficher des classes dans la notation UML (diagramme des classes).

L'utilisateur peut :
- Sélectionner les entités qu'il désire inclure dans le mapping ;
- Renommer les entités et attributs dans les classes ;
- Modifier les paramètres de base des attributs comme la valeur par défaut ;
- Ajouter des relations d'héritage entre classes générées ;
- Ajouter des relations de composition entre classes générées ;
- Paramétrer les associations entre classes telles qu'elles étaient prévues dans le modèle conceptuel de données initial ;
- Paramétrer les cardinalités.

4.3 Représentation UML du fonctionnement de l'Add-In

4.3.1 Cas d'utilisation

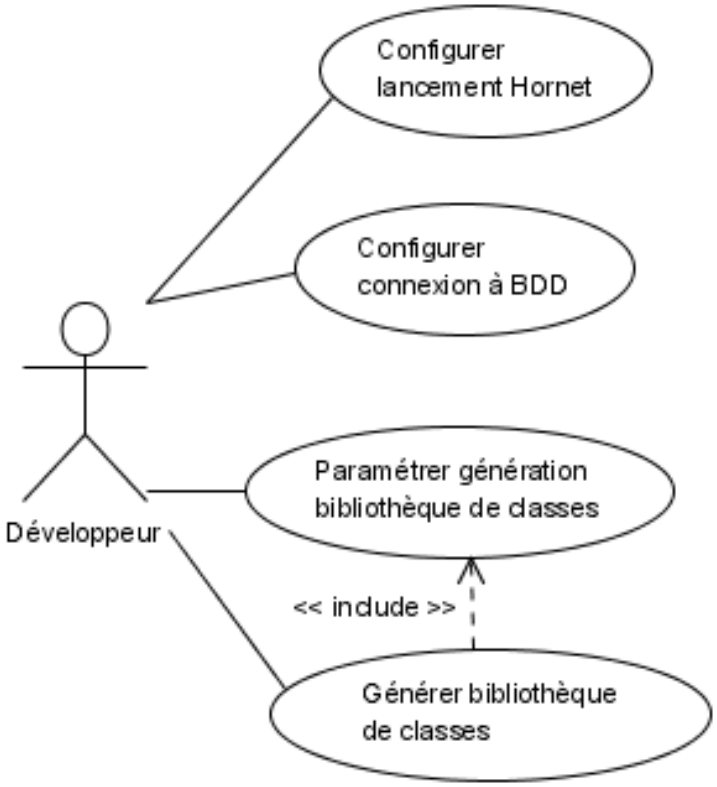


FIG. 4.2 – Diagramme des cas d'utilisation de l'application

4.3.2 Description des cas d'utilisation

4.3.2.1 Configurer le lancement de l'Add-In Hornet

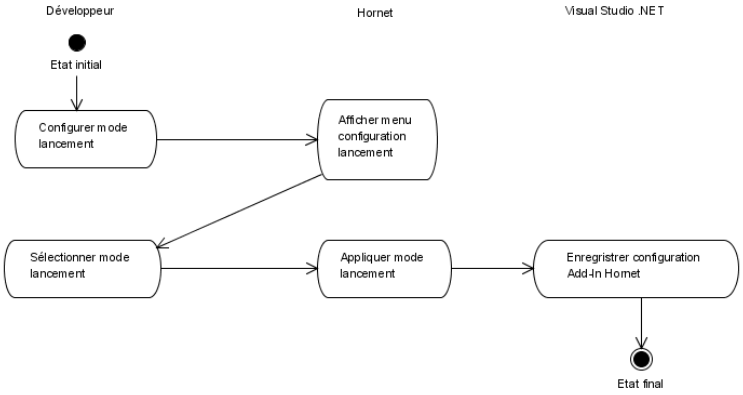


FIG. 4.3 – Diagramme d'activité du lancement de l'Add-In

4.3.2.2 Configurer la connexion à la base de données

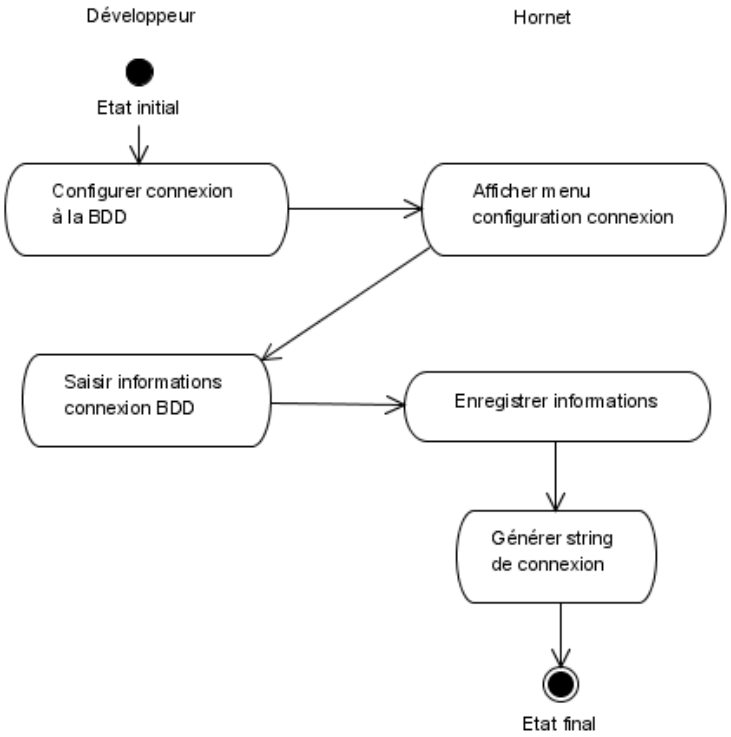


FIG. 4.4 – Diagramme d'activité de la configuration à la base de données

4.3.2.3 Paramétrer la génération de la bibliothèques de classes

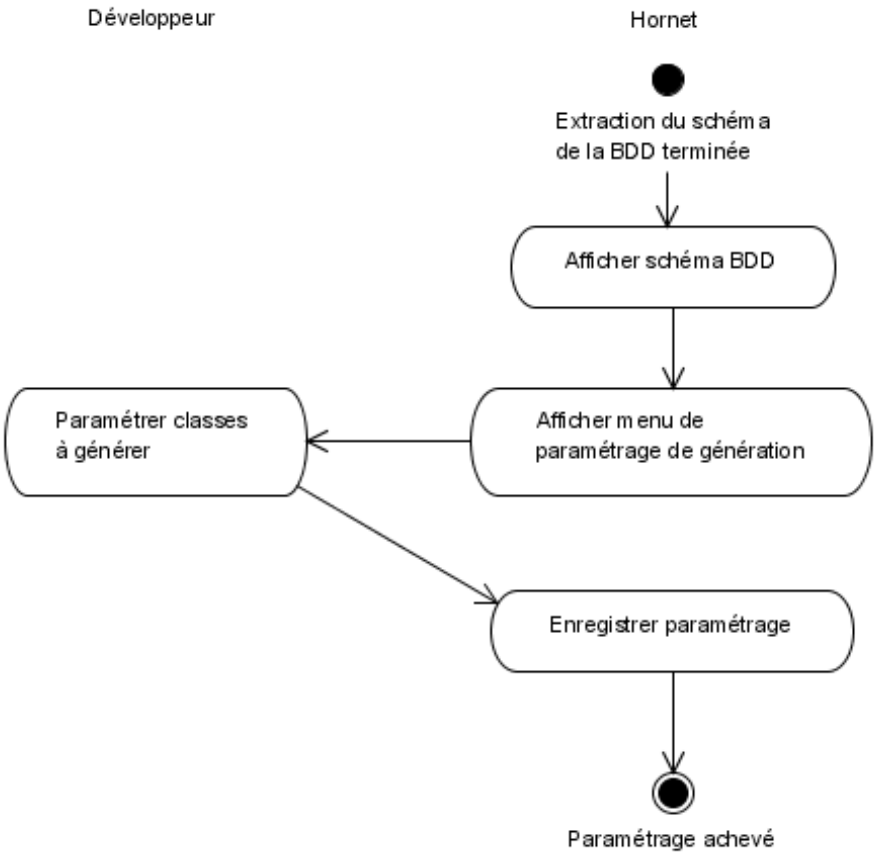


FIG. 4.5 – Diagramme d'activité du paramétrage de la génération de la bibliothèque de classes

4.3.2.4 Générer la bibliothèque de classes

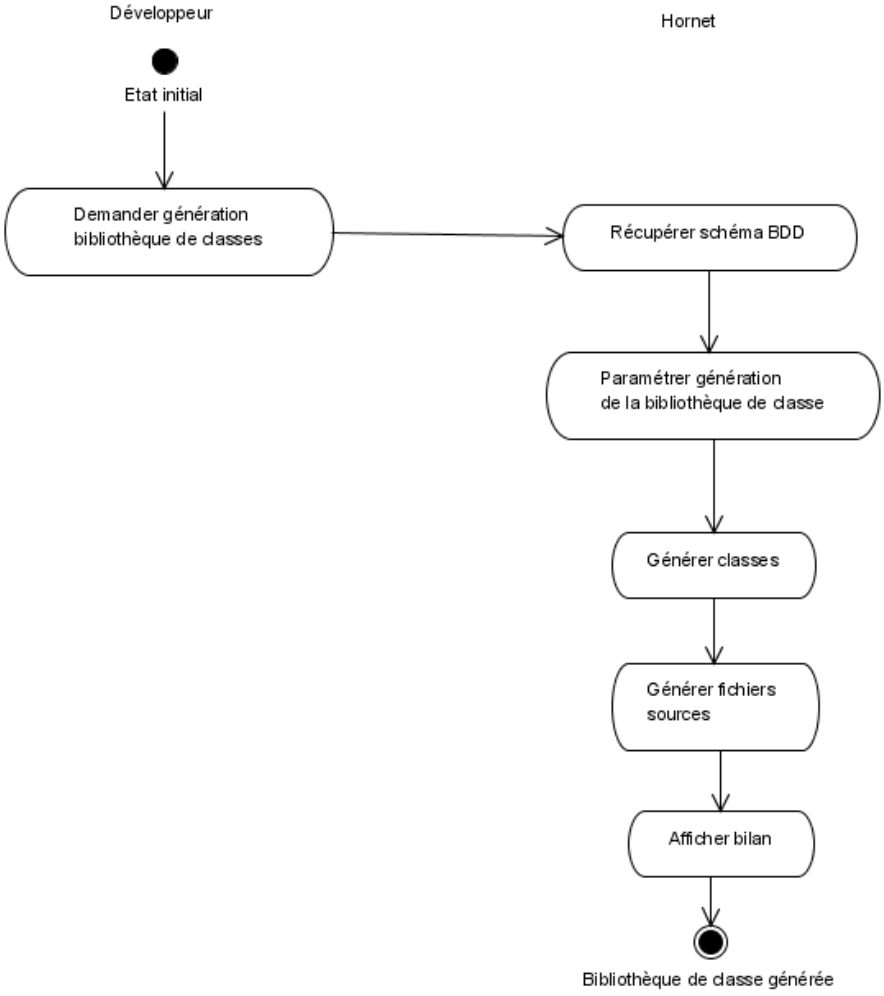


FIG. 4.6 – Diagramme d'activité de la génération de la bibliothèque de classes

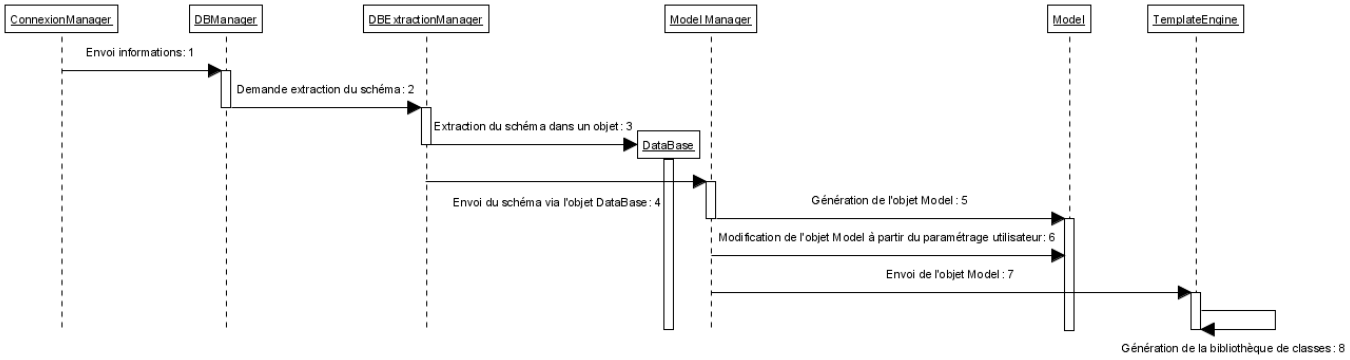


FIG. 4.7 – Diagramme de séquences de la génération de la bibliothèque de classes

Nous désignons par "DataBase" la classe contenant le schéma de la base de données. "Model" correspond à la classe contenant le schéma de la base de données modifié par le paramétrage de

l'utilisateur.

4.4 Spécifications de la bibliothèque de classes générée

4.4.1 Cas d'utilisation

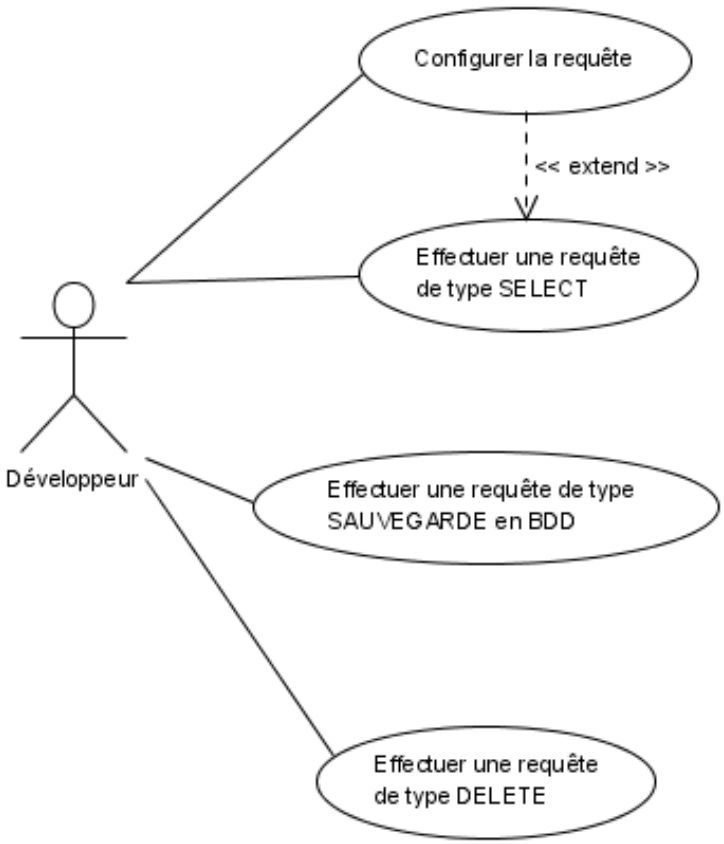


FIG. 4.8 – Diagramme des cas d'utilisations de la librairie générée par l'application

4.4.2 Problèmatiques

La bibliothèque de classes générée devra ainsi répondre aux problèmatiques suivantes :
- Gestion de l'héritage entre objets ;
- Gestion des relations de composition ;
- Gestion multi-table ;
- Gestion des cardinalités (1-1, 1-n, n-n, ...) ;
- Gestion des clés étrangères en général ;
- Gestion des suppressions en cascade dans le cas des clés primaires utilisées ailleurs en tant que clés étrangères.

Chapitre 5

Gestion de projet et PDL

Cette partie présente le plan de développement logiciel adopté pour le projet. Il détaille entre autre la méthodologie de développement, la planification des différentes taches du projet et les outils et procédures mis en oeuvres pour assurer la qualité du logiciel final en terme de réalisation et de documentation.

5.1 L'équipe du projet Hornet

L'équipe projet est composé de trois personnes dont un chef de projet et deux développeurs. La taille de l'équipe étant relativement réduite, le chef de projet assure, en plus de l'animation de l'équipe, une partie du développement à part égale par rapport aux autres membres de l'équipe.

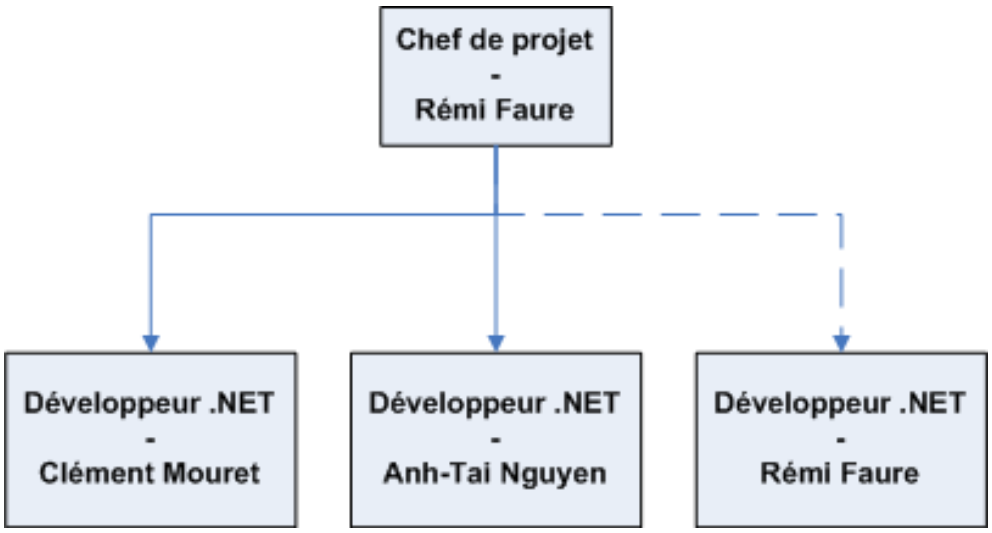


FIG. 5.1 – Equipe projet Hornet

Le chef de projet assure, dans le cadre sa fonction managériale du projet :
- La conduite du projet dans son ensemble (vision globale,...)
- La cohérence théorique entre les différents modules
- Les relations avec l'école et le personnel encadrant
- L'animation des réunions
- La rédaction des comptes-rendus de réunion
- Le dynamisme de groupe

5.2 Logique de déroulement et planification

5.2.1 Méthodologie de développement

Le projet suit un déroulement itératif par prototypes successifs. Au regard de la difficulté de l'étude en terme de conception et réalisation et de la courte durée de développement (4 mois de Octobre 2005 à Janvier 2006), la finalité de notre projet de fin d'étude n'est pas d'obtenir à la fin un logiciel abouti et complet, mais un premier prototype fonctionnel. Le prototype livré à la fin du mois de Janvier permettra d'exécuter la chaîne applicative (du modèle relationnel des données vers le modèle objet) du début jusqu'a la fin. Il s'agit d'une étape indispensable pour une application finale de qualité.

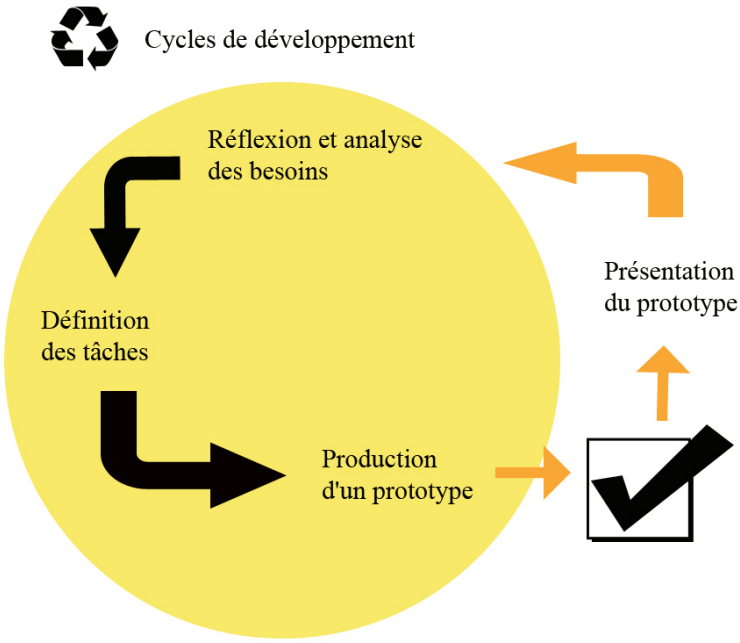


FIG. 5.2 – Cycle de développement

Le développement du logiciel se décompose en cinq étapes :
- Organisation de l'équipe et mise en place des outils de développement
- Analyse des besoins et rédaction des spécifications
- Architecture logicielle et choix technologiques
- Développement des différents modules
- Intégration, présentation du prototype

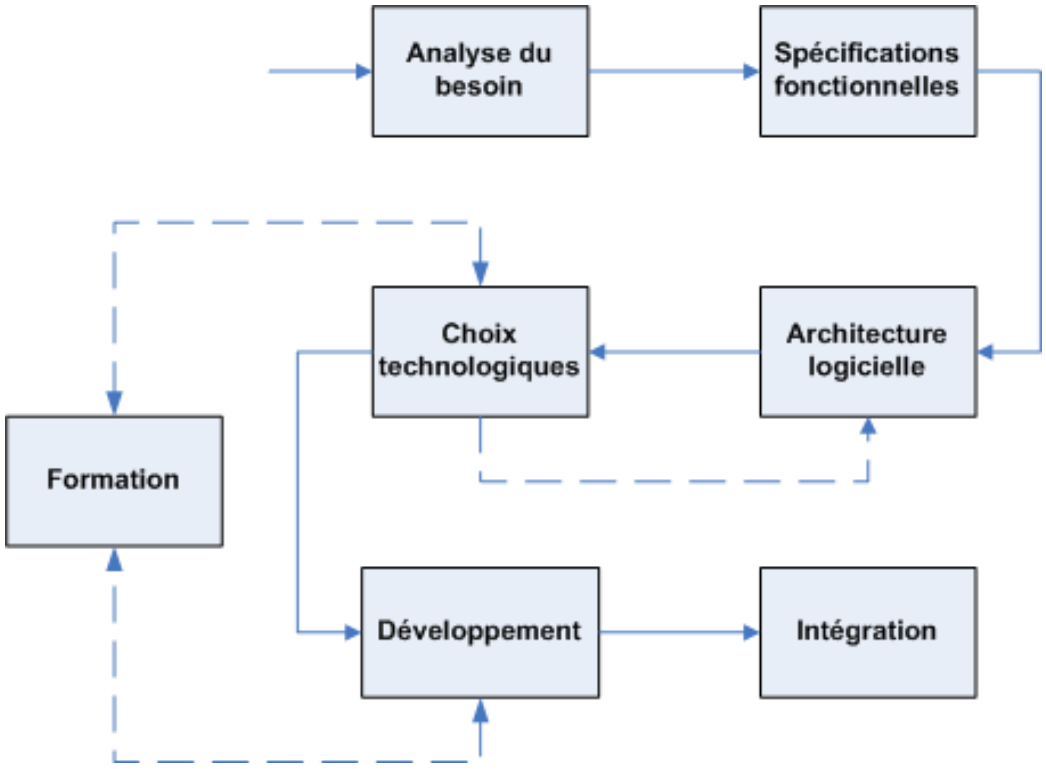


FIG. 5.3 – Détail du cycle de développement

5.2.2 Objectifs des étapes

La phase de préparation comprend les activités suivantes :
– Organisation de l'équipe ;
– Mise en place des outils internes : liste de diffusion, outil de reporting, serveur CVS.

L'itération 1 comprend les activités suivantes :
– Analyse des besoins ;
– Définition des spécifications fonctionnelles ;
– Rédaction d'un document d'avant projet (analyse du besoin & spécifications) ;
– Planification générale.

L'itération 2 comprend les activités suivantes :
– Architecture logicielle ;
– Choix technologiques ;
– Formations éventuelles ;
– Planification du développement.

L'itération 3 comprend les activités suivantes :
– Développement des différents modules ;
– Validation de la cohérence des modules.

L'itération 4 comprend les activités suivantes :

- Intégration ;
- Tests ;
- Développement application de test ;
- Rédaction d'un document de projet de fin d'étude ;
- Rédaction de la documentation technique ;
- Rédaction de la procédure de recette ;
- Rédaction du manuel utilisateur ;
- Présentation du prototype.

5.2.3 Fournitures

Etape	Fourniture	Délai
Itération 1	Définition de l'assurance qualité	2 semaines
	Analyse du besoin & Spécifications fonctionnelles	1 mois
	Calendrier (V1)	1 mois
Itération 2	Architecture logicielle (schémas)	2 mois
	Calendrier (V2)	2 mois
Itération 3	Compte Rendu d'Avancement	3 mois
Itération 4	Document de projet de fin d'étude (VF)	4 mois
	Prototype final	4 mois
	Documentation technique	4 mois
	Procédure de recette	4 mois
	Manuel utilisateur	4 mois

5.2.4 Détails du plan d'itération

Un responsable et une liste de travaux ont été attribués à chaque tâche du projet. Notons que chaque personne de l'équipe a participé à l'ensemble des tâches planifiées en dehors des modules de développement réalisés de façon plus individuelle au niveau de l'implémentation.

5.2.4.1 Tâches à l'itération 1

Tâche	Responsable	Travaux	Documents et produits
Déploiement des outils projets	Anh-Tai Nguyen	Mise en place d'outil de travail collaboratif (ML,Bug Report,CVS)	Tutoriaux et méthodes d'utilisation via email
Analyse des besoins et spécifications fonctionnelles	Clément Mouret	- Etude du besoin et de la problématique - Spécifications fonctionnelles	Spécifications fonctionnelles
Planification	Rémi Faure	- Vue d'ensemble du projet - Découpage en étapes et modules	Planning V1
Assurance qualité	Anh-Tai Nguyen	Définition de l'assurance qualité (outils, normes, ...)	Récapitulatif des normes & processus qualité

5.2.4.2 Tâches à l'itération 2

Tâche	Responsable	Travaux	Documents et produits
Archictecture logicielle - Add-In	Rémi Faure	- Analyse et conception de l'Add-in .NET - Méthode d'intégration à Visual .NET - Interface graphique utilisateur	Schémas UML
Archictecture logicielle - Moteur de template	Rémi Faure	Analyse et conception du moteur de template	Schémas UML
Archictecture logicielle - *Persistent Data Object Library*	Clément Mouret	- Analyse des méthodes de mapping - Analyse et conception de la bibliothèque de classes Hornet	Schémas UML
Archictecture logicielle - Interface SGBD	Anh-Tai Nguyen	Méthode d'extraction du schéma relationnel de la base de données	Schémas UML
Planification	Rémi Faure	- Calendrier de développement - Dépendance entre les modules	Planning V2

5.2.4.3 Tâches à l'itération 3

Tâche	Responsable	Travaux	Documents et produits
Développement Add-In .NET	Rémi Faure	- Intégration application à Visual .NET (fenêtre de configuration, toolbox, ajout/modification projet,...) - Interface graphique utilisateur	Add-in .NET
Développement moteur de template	Rémi Faure	Développement générateur de fichier de classe	Application moteur de template
Développement *Persistent Data Object Library*	Clément Mouret	- Implémentation du mapping des données - Développement log Hornet - Module de requête	Hornet Library
Développement interface SGBD	Anh-Tai Nguyen	Extraction du schéma relationnel de la base de données	Classes
Fichiers template	Clément Mouret	Ecriture des fichiers templates	Fichiers templates
Application test V1	Anh-Tai Nguyen	Développement de l'application test (Interface)	Application test V1

5.2.4.4 Tâches à l'itération 4

Tâche	Responsable	Travaux	Documents et produits
Intégration	Rémi Faure	Intégration de tous les modules développés	Prototype 1
Documentation	Clément Mouret	Rédaction/Génération documentation technique	Documentation technique
Tests	Anh-Tai Nguyen	Test de la solution (prototype)	Rapport de test via email
Application test V2	Anh-Tai Nguyen	Finalisation du développement de l'application test	Application test V2
Recette	Rémi Faure	Rédaction de la procédure de recette	Procédure de recette
Manuel utilisateur	Anh-Tai Nguyen	Rédaction du manuel utilisateur	Manuel utilisateur
Rapport final	Clément Mouret	Rédaction du rapport de projet de fin d'étude	Rapport de projet de fin d'étude

5.2.5 Calendrier

Le calendrier ci-dessous a été construit en deux étapes :
– Planification générale du projet : les grandes étapes et jalons ;

– Révision de la planification : Intégration du détail du développement des modules.

Task Name	Duration	Start	Finish	Resource Names
Itération 1	**14 days**	**Mon 17/10/05**	**Sun 30/10/05**	
Déploiement des outils projets	7 days	Mon 17/10/05	Sun 23/10/05	Rémi Faure
Analyse des besoins et spécifications	14 days	Mon 17/10/05	Sun 30/10/05	Clément Mouret;Rémi Faure;Anh-Tai Nguyen
Assurance qualité	7 days	Mon 17/10/05	Sun 23/10/05	Anh-Tai Nguyen
Jalon 1 : Fin lancement du projet et spécification	0 days	Sun 30/10/05	Sun 30/10/05	
Itération 2	**21 days**	**Mon 31/10/05**	**Sun 20/11/05**	
Architecture logicielle générale	7 days	Mon 31/10/05	Sun 06/11/05	Rémi Faure;Clément Mouret;Anh-Tai Nguyen
Archictecture logicielle - Interface SGBD	14 days	Mon 07/11/05	Sun 20/11/05	Anh-Tai Nguyen
Archictecture logicielle - Add-In	14 days	Mon 07/11/05	Sun 20/11/05	Rémi Faure[50%]
Archictecture logicielle - Moteur de template	14 days	Mon 07/11/05	Sun 20/11/05	Rémi Faure[50%]
Archictecture logicielle - Persistent Data Object Library	14 days	Mon 07/11/05	Sun 20/11/05	Clément Mouret
Jalon 2 : Fin de l'analyse & conception	0 days	Sun 20/11/05	Sun 20/11/05	
Itération 3	**45 days**	**Mon 21/11/05**	**Sat 07/01/06**	
Développement Add-In .NET	45 days	Mon 21/11/05	Sat 07/01/06	Rémi Faure[50%]
Développement moteur de template	45 days	Mon 21/11/05	Sat 07/01/06	Rémi Faure[50%]
Développement interface SGBD	30 days	Mon 21/11/05	Tue 20/12/05	Anh-Tai Nguyen
Développement Persistent Data Object Library	30 days	Mon 21/11/05	Tue 20/12/05	Clément Mouret
Application test V1	10 days	Wed 21/12/05	Mon 02/01/06	Anh-Tai Nguyen;Clément Mouret
Jalon 3 : Fin du developpement partie 1	0 days	Sat 07/01/06	Sat 07/01/06	
Itération 4	**15 days**	**Sun 08/01/06**	**Sun 22/01/06**	
Intégration	10 days	Sun 08/01/06	Tue 17/01/06	Rémi Faure
Tests	5 days	Wed 18/01/06	Sun 22/01/06	Anh-Tai Nguyen
Application test V2	5 days	Fri 13/01/06	Tue 17/01/06	Anh-Tai Nguyen;Clément Mouret
Documentation	5 days	Sun 08/01/06	Thu 12/01/06	Clément Mouret
Procédure de recette	5 days	Wed 18/01/06	Sun 22/01/06	Rémi Faure
Rapport projet de fin d'étude	15 days	Sun 08/01/06	Sun 22/01/06	Clément Mouret;Rémi Faure;Anh-Tai Nguyen
Jalon 4 : Fin du projet	0 days	Sun 22/01/06	Sun 22/01/06	

FIG. 5.4 – Planning projet

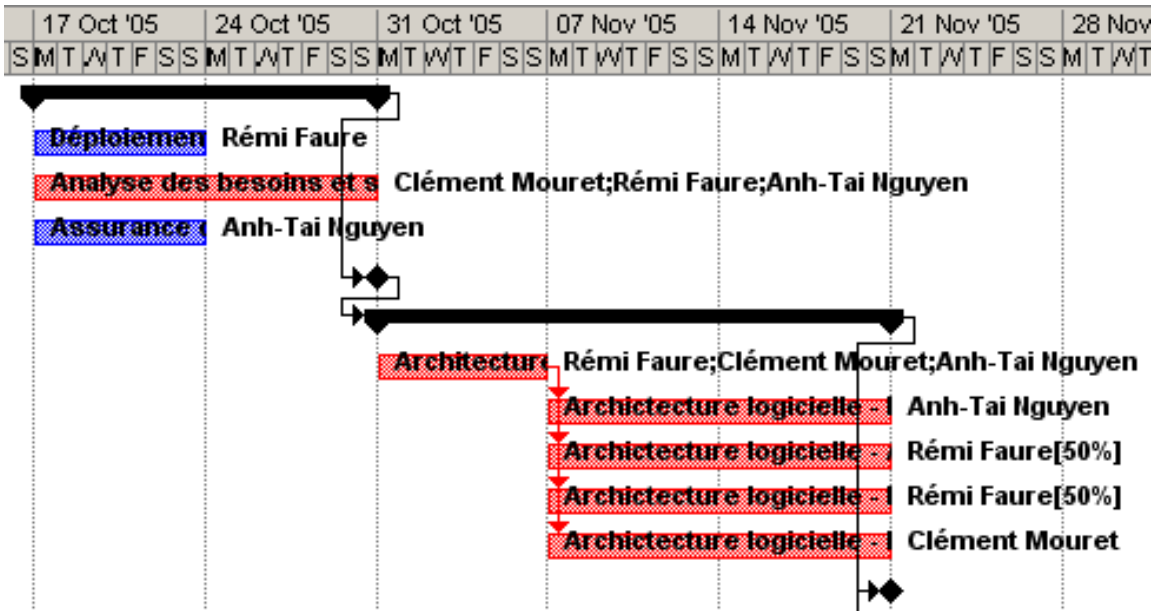


FIG. 5.5 – Diagramme de Gantt itération 1 et 2

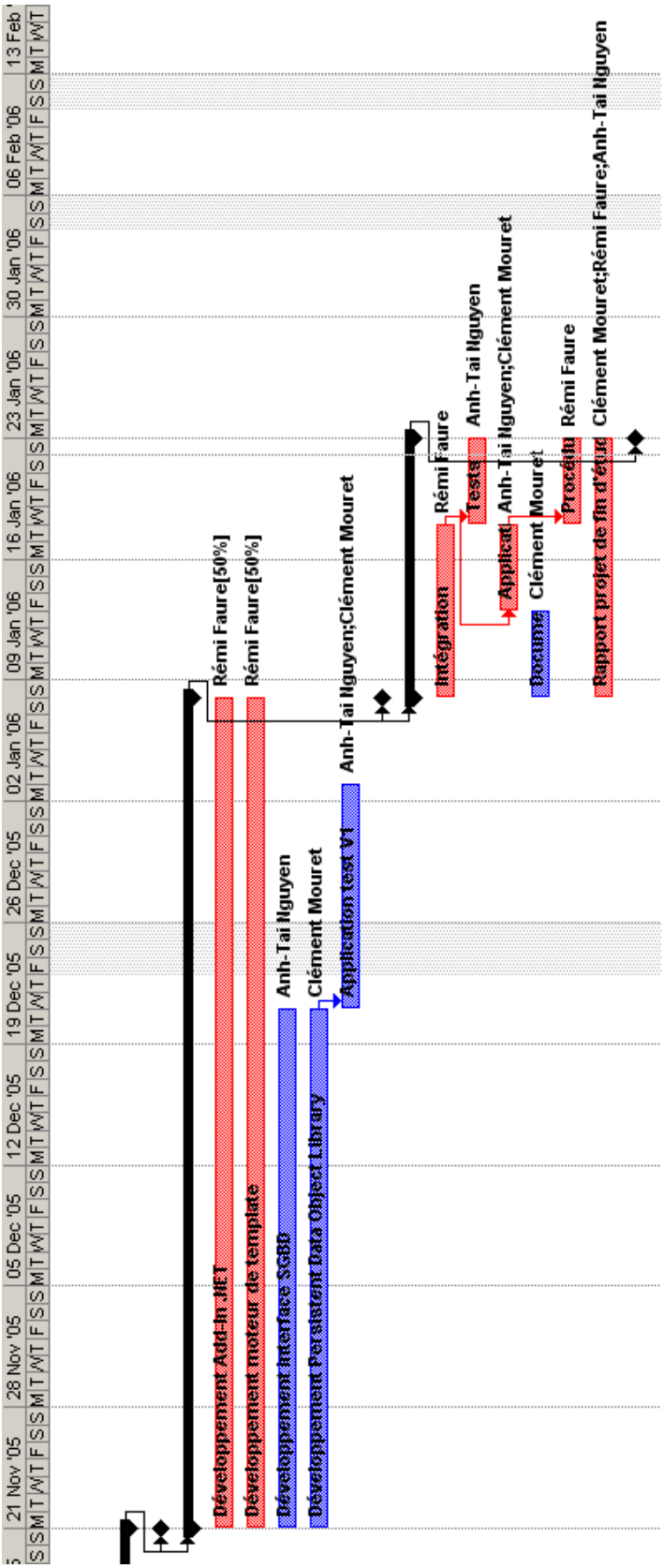


FIG. 5.6 – Diagramme de Gantt itération 3 et 4

5.2.6 Tableaux de bord

Le tableau de bord du projet, mis à jour au fur et à mesure du développement, est fourni en annexe de ce document.

5.3 Méthodes, techniques et outils

5.3.1 Conception & Développement

La méthode de conception est une méthode objet basée sur UML.

Les principaux outils utilisés sont :

- MS Project pour la planification ;
- Poseidon For UML 3.0 pour la conception ;
- MySQL - SQL Server - Oracle pour les bases de données ;
- Le langage C# ;
- Visual .NET pour l'environnement de développement ;
- Windows XP Professionnal pour le système d'exploitation ;
- Des ordinateurs Pentium 4 & Athlon, plus un portable pour les éventuelles démonstrations.

5.3.2 Organisation de la communication

Afin de communiquer de la façon la plus efficace possible entre les membres de l'équipe projet, les outils utilisés sont :

- Liste de diffusion (hornetproject@yahoogroupes.fr)
- Conférence audio sur IP, Skype (http ://www.skype.com)
- Outil de reporting de bug (http ://fluminis.free.fr)

Ces outils completent les réunions (au rythme de une réunion toute les deux semaines et demi) dans l'enceinte de l'école. Chaque réunion possède un ordre du jour, une liste de tâches à effectuer avant et un compte-rendu final.

Historiques des réunions :

Date	Ordre du jour	REF compte-rendu
11-10-2005	- Définition projet Hornet	HPREF_CR_111005
18-10-2005	- Analyse des besoins - Mode de diffusion - Nom du projet	HPREF_CR_181005
24-10-2005	- Liste des fonctionnalités du logiciel - Mode de paramétrage de l'outil - Liste des fonctionnalités de la *Persistent Data Object Library*	HPREF_CR_241005
17-11-2005	- Etat de l'art du concept d'ORM - Reflexion sur l'architecture	HPREF_CR_171105
02-12-2005	- Architecture logicielle	HPREF_CR_021205
04-01-2006	- Point d'avancement développement 1	HPREF_CR_040106
11-01-2006	- Point d'avancement développement 2 - Définition des fonctionnalités du prototype présenté fin Janvier	HPREF_CR_110106

5.4 Maîtrise des risques

Voici la présentation des différents risques identifiés et des actions prévues pour réduire leur danger.

Risque	Actions
Mauvaise intégration au cycle de développement	Fourniture de prototypes successifs Test utilisateurs Développement d'application test
Complexité de l'interface de configuration	Réalisation d'une maquette IHM
Mauvaise interprétation des attentes des utilisateurs	-
Performance de l'outil final	Benchmark
Pas de gestion des spécificités des SGBD	Analyse de leurs taux d'utilisation

5.5 Assurance qualité

Cette partie synthétise les processus mis en oeuvre pour assurer la qualité du logiciel final. Elle décrit entre autre le recueil et l'analyse des problèmes, la gestion de la documentation technique, la gestion de configuration et la définition et le respect des normes de développement.

5.5.1 Recueil et analyse des problèmes

Deux canaux de diffusion ont été installés pour recueillir les problèmes éventuels :

– Une liste de diffusion (ML) ;
– Un outil de reporting de bug.

La liste de diffusion est utilisée pour les problèmes généraux, c'est à dire les problèmes d'analyse et de conception ou les problèmes liés à la partie organisationnelle du projet. Chaque problème est ensuite résolu directement en utilisant le même canal (ML) ou abordé lors de réunions ou conférences audio.

L'outil de reporting de bug, utilisé pour des problèmes liés à l'exécution de logiciel (bug), permet de :
– Classifier les bugs ;
– Suivre l'état des bugs ;
– Connaitre le responsable de la correction ;
– Décrire la méthode de résolution ;
– Tracer l'évolution du logiciel.

Classification du bug :
– Nouveau : Nouveau bug, soumis par un membre du projet ;
– Pris en charge : Le bug a été pris en charge par un développeur ;
– Résolu : Le bug a été résolu.

Le projet étant au stade du prototype 1, l'aspect reporting de bug as été exploité uniquement sur la dernière itération (phase d'intégration). Il devrait se montrer très utile dans la suite du projet.

5.5.2 Gestion de la documentation technique

La documentation technique complète la description de l'architecture logicielle. Elle sera générée automatiquement à partir des fichiers sources commentés par l'ensemble des développeurs. La documentation automatique inclue une description, brève ou détaillée, des modules, des namespaces, des classes, des fonctions... Elle est complétée par des schémas (diagrammes des classes, diagrammes de collaboration,...) aux normes UML générés par *reverse-engineering*.

La génération de documentation sera réalisée à partir du logiciel Doxygen (www.doxygen.org). La syntaxe à respecter, au format XML, est décrite en annexe de ce rapport (Norme de développement).

5.5.3 Gestion de configuration

Documents et codes sources seront gérés en configuration à l'aide d'un outil CVS permettant de partager les sources entre les différents participants du projet. Dans le cas de la validation d'une modification, l'outil CVS réalise la fusion des codes de manière automatique et transparente. En cas de problème, il est possible de consulter les versions précédentes des fichiers.

CVS : https ://freepository.com

5.5.4 Respect de normes de développement

La qualité du code réalisé doit respecter les normes syntaxiques de codage décrites en annexe de ce document.

Chapitre 6

Architecture logicielle

6.1 Architecture générale

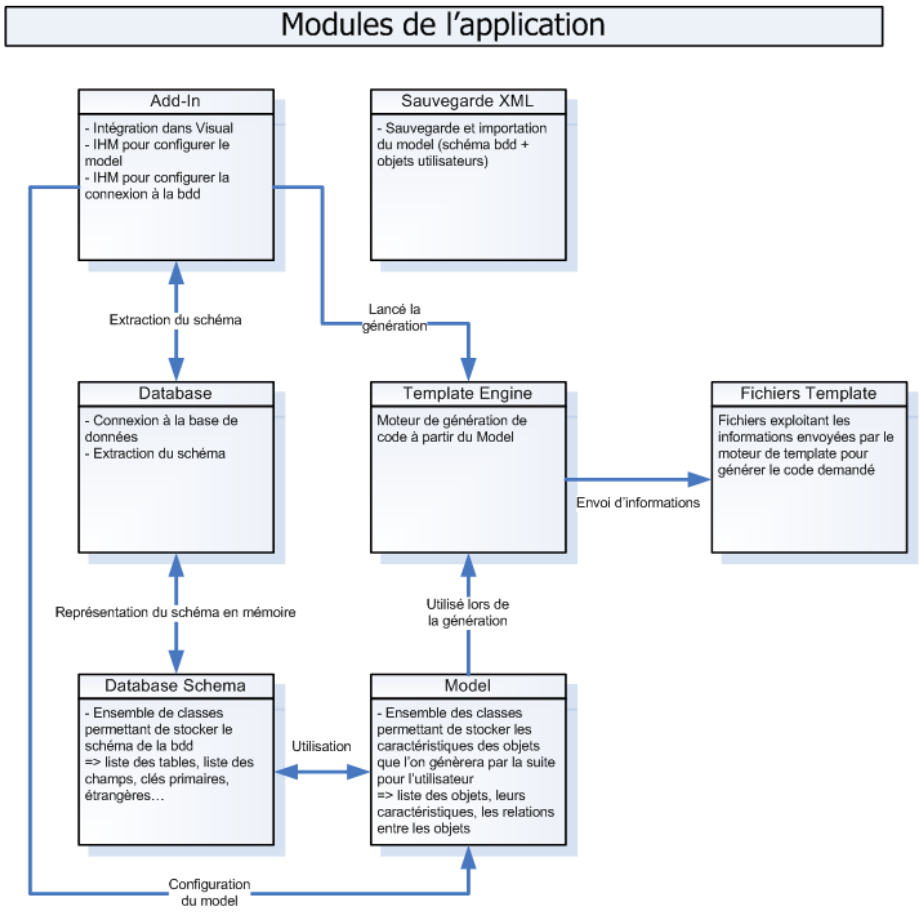


FIG. 6.1 – Modules de l'application

Afin de répondre au mieux à la problématique posée par l'outil, un découpage de l'application en plusieurs modules a été fait.

- Le module Add-In correspond à la couche d'intégration dans Visual Studio et aux objets de présentation des IHM afin de gérer les interactions avec l'utilisateur ;
- Le module Database est chargé de la connexion au serveur de base de données et de l'extraction du schéma de la base ;

- Le module Database Schema doit permettre de conserver et représenter en mémoire le schéma extrait de la base ;
- Le module Model est le module contenant le résultat de l'extraction du schéma et du paramétrage de l'utilisateur ;
- Le module Template Engine exploite le module Model pour la génération de code et envoie les informations nécessaires au module Fichiers Template ;
- Le module Fichiers Template génére le code à partir des informations fournies par le module Template Engine ;
- Le module sauvegarde XML est là pour sauvegarder des objets sous forme de fichiers XML quand cela est requis.

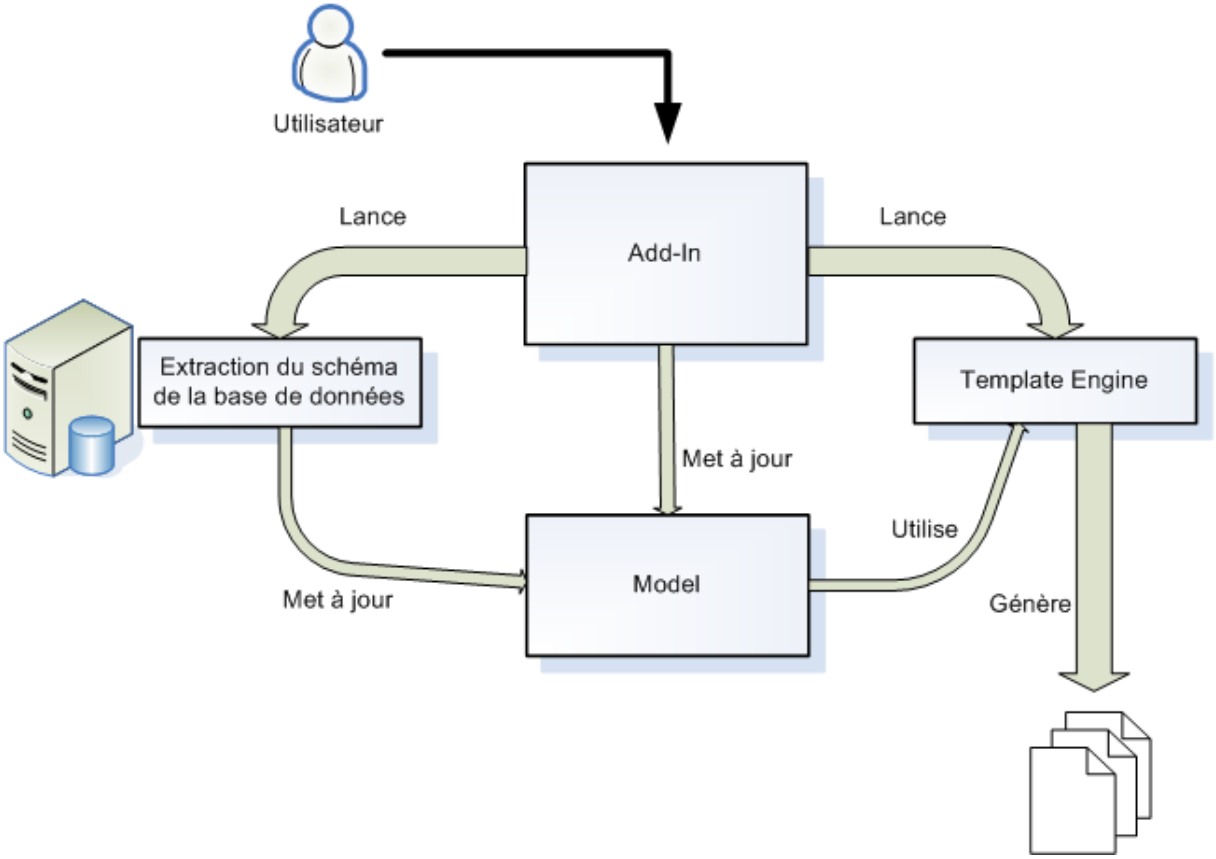


FIG. 6.2 – Interactions entre différents modules

Le schéma ci-dessus récapitule le fonctionnement global de l'application. L'utilisateur passe par le module Add-In pour interagir avec l'outil. Ce schéma montre également comment les différents modules communiquent entre eux.

6.2 Interface SGBD : Extraction du schéma de la base de données

6.2.1 Concept

Le rôle de ce module est de se connecter à un serveur de base de données afin d'extraire le schéma de la base demandée. Ce module doit donc répondre aux exigences suivantes :

- Capacité à se connecter avec plusieurs SGBD ;
- Capacité à employer les requêtes SQL spécifiques selon le type de SGBD ;
- Récupérer les informations du schéma ;
- Stocker ces informations dans un objet standard quelque soit le type de SGBD employé.

Parmi les informations du schéma, nous retrouvons :

- La liste des tables ;
- La liste des colonnes de chaque ;
- Les clés primaires ;
- Les clés étrangères et à quelles clés primaires elles se rapportent ;
- Les informations de base caractérisant une colonne (Type, Auto-incrémentation, longueur, valeur par défaut, Nullité).

La connexion à la base de données ainsi que l'envoi de requêtes SQL repose sur le format d'interfaçage ODBC.

6.2.2 Fonctionnement

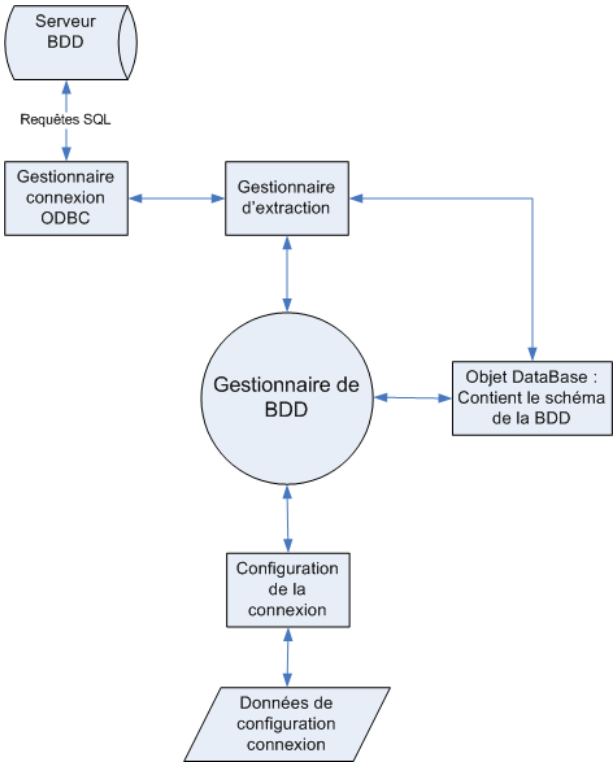


FIG. 6.3 – Fonctionnement de l'extraction

Ce module reçoit en entrée les paramètres de connexion au serveur de base de données sous la

forme d'un objet de configuration de connexion qui est capable de générer une string de connexion ODBC. Cet objet contient :

- Le type de base ;
- Le nom du serveur ;
- Le nom d'utilisateur ;
- Le mot de passe ;
- Le pilote ODBC à employer ;
- Le mode de connexion.

Le gestionnaire de base de données emploie des modules de connexion au serveur de base de données et d'extraction du schéma. Le module d'extraction est une interface dont les instances varient selon le type de base de données employé. C'est ce module qui effectue les requêtes SQL afin d'extraire le schéma. Le schéma est ensuite contenu dans un objet DataBase qui se retrouve en sortie.

6.3 Add-In Visual .NET 2003

6.3.1 Visual Studio

Les developpeurs de Microsoft ont pensé leur logiciel Visual Studio afin que les développeurs puissent rajouter des fonctionnalités dans le programme qu'ils n'auraient pas eux même imaginées.

Ces *Add-Ins* (litéralement "ajouté dedans") s'intègrent au coeur même de l'application Visual Studio. Il devient possible de controler l'application directement au seins même de l'AddIn. On peut ainsi :

- ajouter et/ou modifer des fichiers de la solution
- ajouter des projets à la solution
- ajouter des commandes dans les menus
- ajouter des boutons dans les barres d'outils
- ajouter des fenêtres
- ...

Pour être reconnu par Visual Studio, notre Add-In s'appuie sur un type de projet un peu particulier nommé *Complément de Visual .Net*. Ce projet est configuré pour développer un Add-In.

La classe centrale de l'application se nomme *Connect* et c'est par elle que passe toutes les informations que nous envoie Visual Studio. La fonction *OnConnection* par exemple est excecutée lors ce que l'Add-In est chargé dans Visual. C'est là que nous devons créer nos fenêtres et s'enregistrer auprès des gestionnaires d'évènements.

6.3.2 Architecture

Visual Studio charge notre Add-In. Il exécute la méthode *OnConnection* de notre classe *Connect*. Là nous créons nos fenêtres et ajoutons nos commandes dans les menus de Visual.

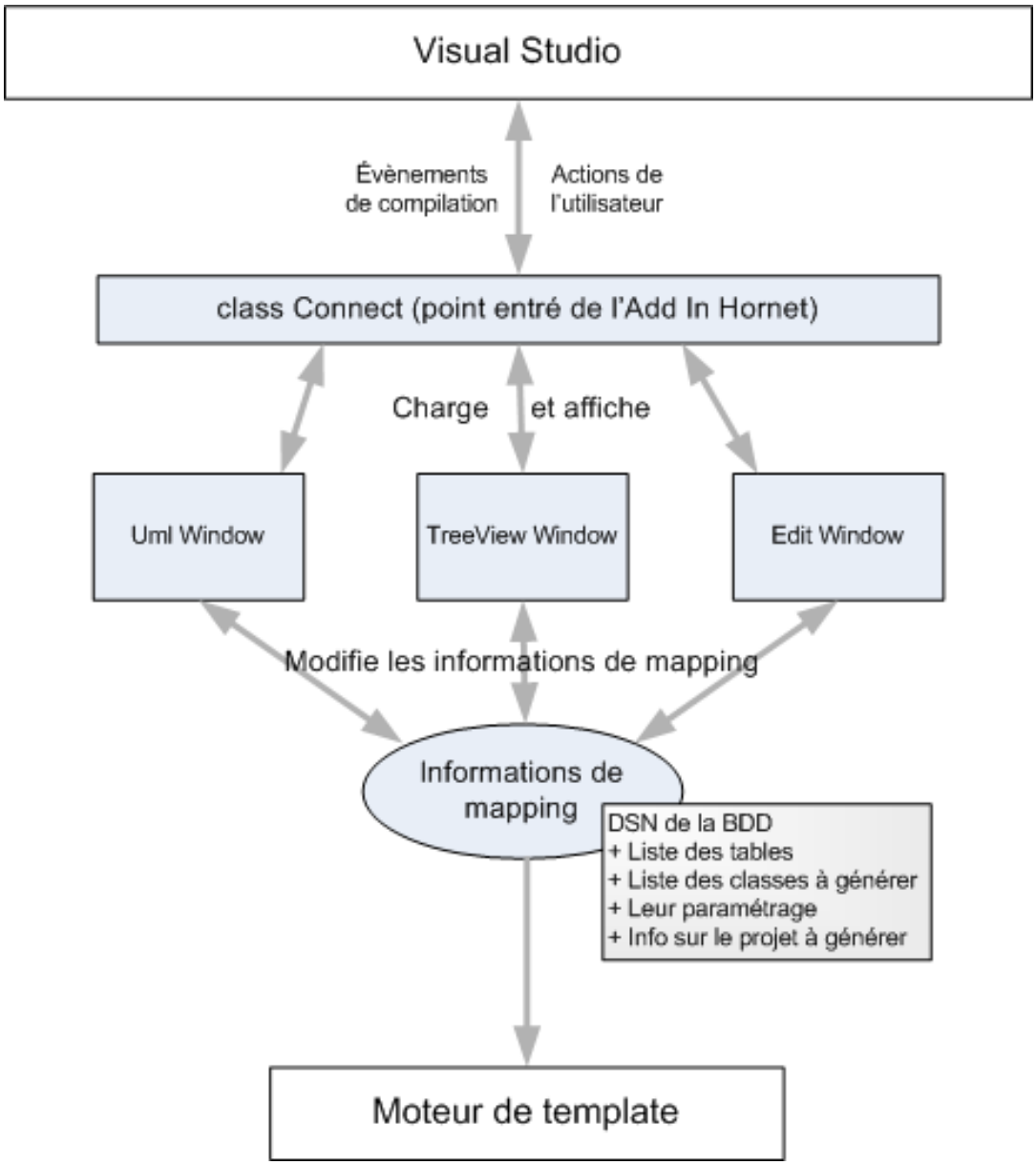
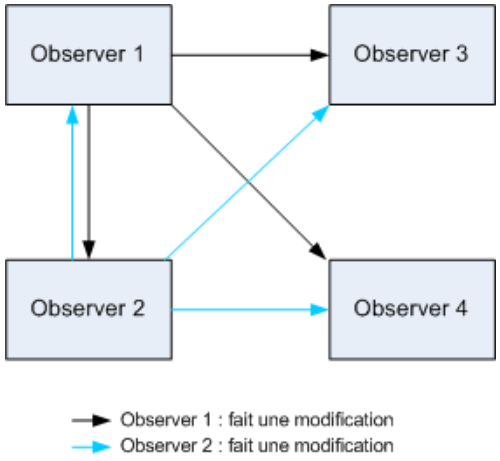


FIG. 6.4 – Architechture de l'Add-n

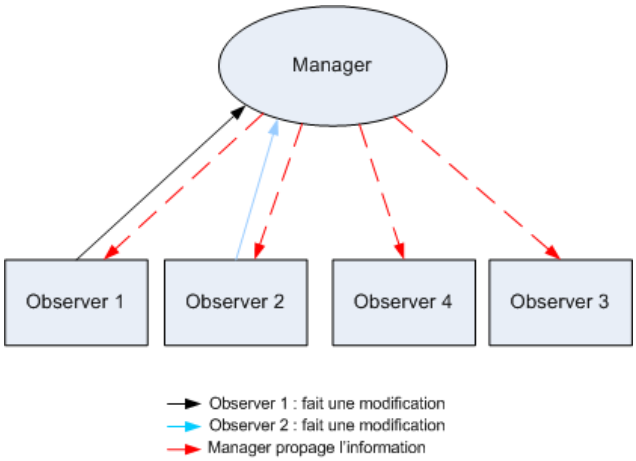
6.3.2.1 Model manager

Nous avons dû surmonter un problème majeur : plusieurs fenêtres ont besoin de connaitre les informations de mapping. Mais chaque fenêtre ne connaît pas toutes les autres fenêtres qui sont ouvertes et qui ont besoin d'être alertées lors d'un changement sur le model.

En effet, quand l'utilisateur ajoute une classe dans la fenêtre UML, il faut que la fenêtre avec le TreeView reflète immédiatement les changements. Quand l'utilisateur sélectionne une relation dans le TreeView, il faut que la relation s'affiche comme sélectionnée sur le dessin des classes, et il faut également que la fenêtre d'édition d'une relation s'affiche au bas de la fenêtre de dessin des classes.



Pour contourner ce problème nous avons utilisé un design pattern appelé *Observer*. Chaque objet devant être averti lorsque survient un changement sur les données, s'inscrit auprès d'un manager. Ensuite, c'est le manager qui se charge d'avertir tous les observateurs lors d'un changement sur le model. Ainsi, chaque élément n'a plus qu'à connaître uniquement le manager. De cette façon, il devient possible d'ajouter ou de retirer des observateurs sans que cela influe sur les autres.



Les observateurs ont uniquement à implémenter une interface contenant le prototype des fonctions qui seront appelées lorsque le model sera rechargé, ou qu'une classe est ajoutée... Charge ensuite aux observateurs de traiter ou non tous les messages qu'ils reçoivent.

6.4 Moteur de template : Génération des fichiers sources Hornet

6.4.1 Concept

Notre idée était de réaliser une application capable de générer des objets capables de manipuler une base de donnée dans plusieurs langages objets. C'est à dire que nous proposons à l'utilisateur une seule interface graphique pour récupérer les informations sur la base de données (les identifiants, l'ip du serveur, ...) et les configurations sur le mapping souhaité par l'utilisateur (je veux tel objet avec tel et tel attributs qui correspondront à tel et tel champs de la base de données).

Ensuite, à partir de ces informations, nous allons générer du code dans un langage objet. Oui mais les langages ont tous des spécifications différentes et notre idée est de ne pas re-compiler l'Add-In à chaque ajout de nouveaux langages supportés.

La solution est d'utiliser des fichiers de template.

6.4.1.1 Moteur de template mêne la danse

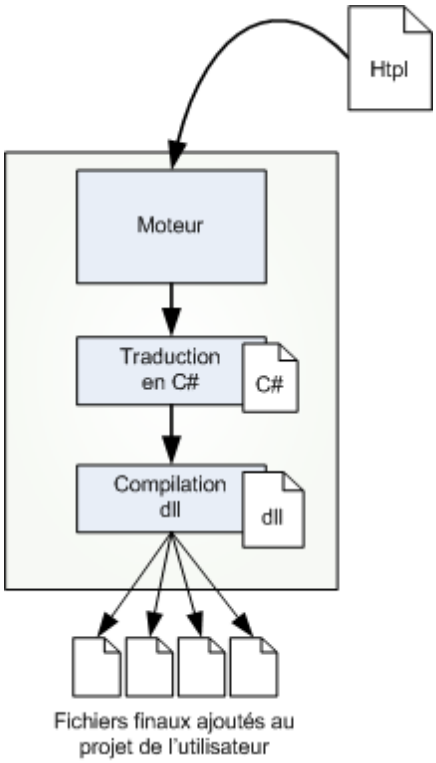
Le moteur de template lit le fichier en sachant à l'avance ce qu'il va trouver dans le fichier. Par exemple il sait qu'il peut remplacer %MaVar1% par "hornet" et %MaVar2% par "project". Il sait également quelles boucles existent. Mais il est impossible d'ajouter des boucles suplémentaires.

6.4.1.2 Le fichier template mêne la danse

Le template ne sait pas ce que fait le fichier de template. Il sait uniquement quelles informations il doit donner au fichier. Le fichier fait ensuite ce qu'il veut.

C'est cette deuxième méthode que nous avons utilisé.

6.4.2 Fonctionnement du template



Le template charge un fichier ".htpl", le tranforme en C#.
Par exemple, <%=*MaVar*%> est converti en *Writer.Write(MaVar) ;*
Le template compile ensuite ce fichier temporaire pour en faire une dll. Cette dll est chargée et exécutée. C'est alors elle qui mène la danse et qui crée les fichiers qui vont être ajoutés dans le projet de l'utilisateur.

6.5 Bibliothèque de classes Hornet : *Data Persistent Objects Library*

6.5.1 Concept et structure de la bibliothèque de classes

La bibliothèque de classe est composée de deux couches distinctes :
- Une couche de base (non générée) permettant l'implémentation d'un mapping générique
- Une couche de paramétrage (générée par la solution Hornet)

6.5.2 La couche de base Hornet

La couche Hornet est la base de la couche de paramétrage. Elle implémente la structure du mapping et fourni un ensemble d'outils et de classes de base permettant de communiquer avec la base de données : transfert de données, stockage de données, mapping des données ...

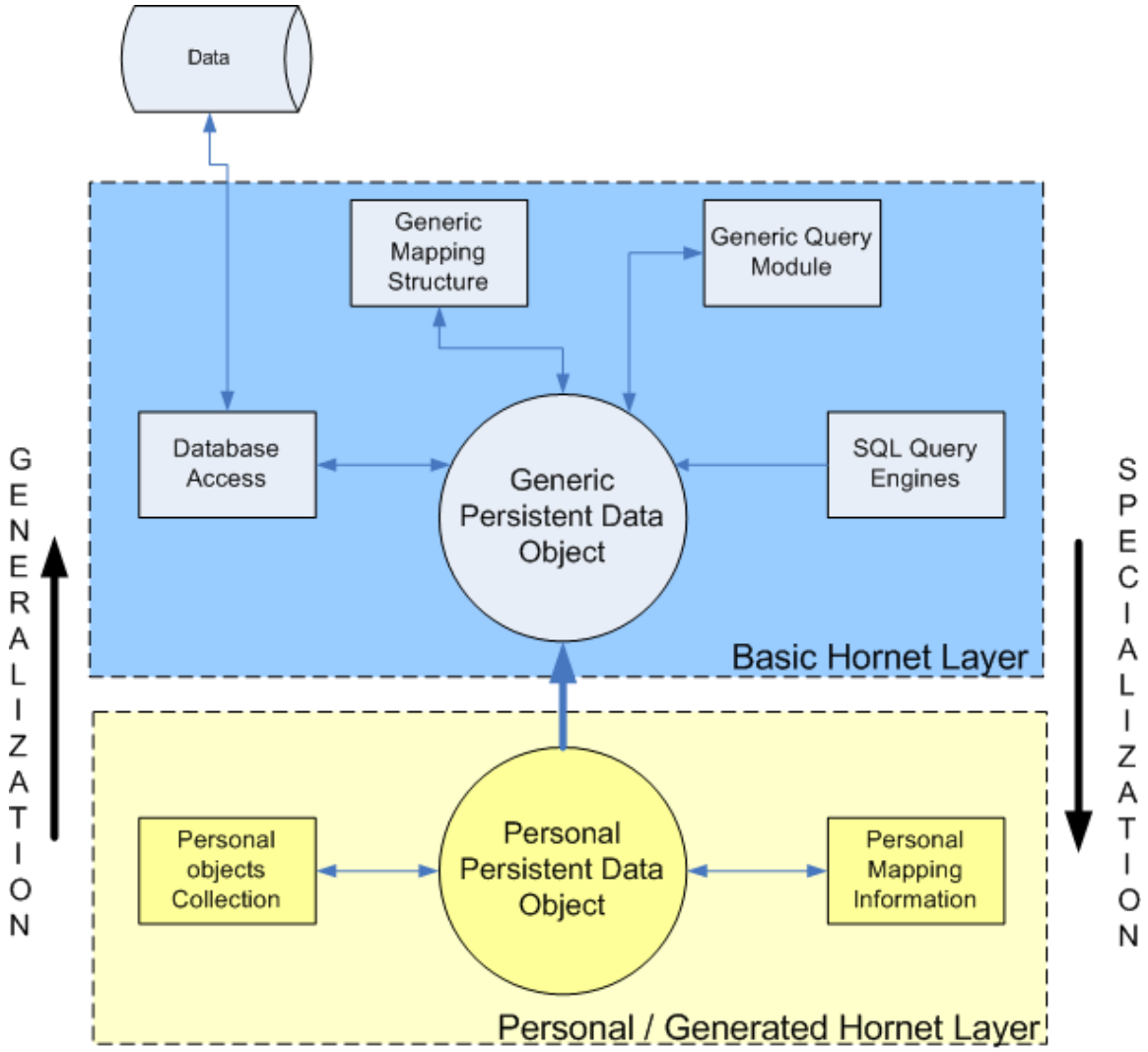


FIG. 6.5 – Concept architecturale de la bibliothèque de classe

La couche de base contient notamment les modules suivants :
- Objets de base pour le stockage et l'accès aux données (*Generic Persistent Data Object*)
- Ensemble de structures pour le mapping des données (*Generic Mapping Structures*)
- Accès à la base de données (*Database Access*). Indépendant du SGBD
- Module de requête (*Generic Query Module*)
- Moteur de requêtes SQL (*Generic SQL Query Engine*). Prise en compte des spécificités des SGBD

Les diagrammes des classes associés sont fournis en annexe de ce rapport.

6.5.3 La couche de paramétrage

La couche de paramétrage est générée par l'outil hornet, via le moteur de template et la configuration de l'utilisateur. Elle spécialise la couche générique Hornet implémentant le mapping des données.

La couche de paramétrage contient notamment les modules suivants :
- Les informations de mapping pour chaque classe générée : table, champ, type, valeur par défaut, propriété *NullAble*, clé étrangère, clé primaire, ... (*Personal Mapping Information*)
- Les classes de données définies par l'utilisateur (développeur) spécialisant l'objet générique de donnée (encapsulation de l'interface générique) (*Personal Persistant Data Object*)
- Les classes de collection d'objets permettant de gérer des listes d'objets de données (*Personal Object Collection*)

Les diagrammes des classes associés sont fournis en annexe de ce rapport.

6.5.4 Encapsulation des données

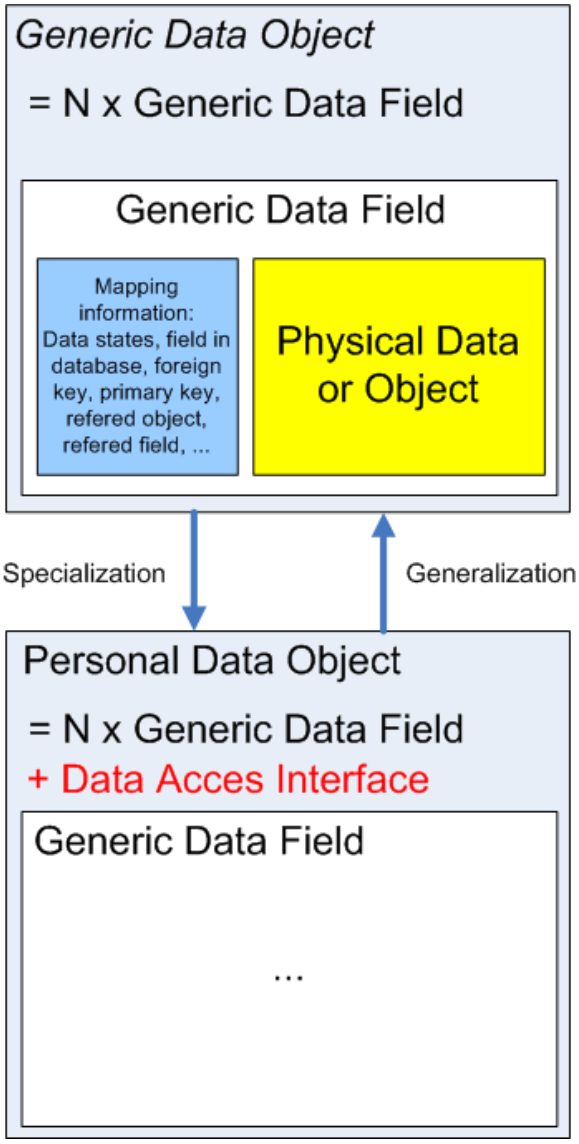


FIG. 6.6 – Concept d'encapsulation des données dans les objets génériques

Chapitre 7

Bilan

7.1 Résultat obtenu

Au final, nous avons bien une application s'installant en tant qu'Add-In dans Visual Studio .NET 2003.

Cet outil parvient à être conforme aux spécifications :
- L'Add-In s'installe et s'intègre dans Visual Studio .NET 2003 ;
- L'utilisateur peut démarrer la génération à partir d'une base de données existante ou d'un projet de génération précédent ;
- L'utilisateur peut configurer la connexion à la base de données ;
- L'application extrait le schéma de la base de données ;
- L'utilisateur peut paramétrer la génération de la bibliothèque de classes telle que c'est décrit dans les spécifications ;
- L'application génère la bibliothèque de classes et l'intègre à la solution de l'utilisateur sous la forme d'un projet intégré à la solution du développeur ;
- La librairie générée permet d'effectuer les opérations spécifiées : SELECT, DELETE et SAVE. Les différentes problématiques décrites dans les spécifications ont été résolues.

7.2 Difficultés rencontrées

7.2.1 Recherche d'informations

Lors de la phase de conception générale de l'outil, nous avons effectué des recherches préliminaires. La première étape fut de comprendre le principe de fonctionnement l'ORM. Les informations précises furent rares.

Parmi les outils gratuits d'ORM que nous avons pu récupérés, aucun n'était réellement exploitable. Certains ne pouvaient se lancer ou s'installer tandis que d'autres nécessitaient une phase de configuration complexe.

Recenser les critères selon lesquels un outil d'ORM est bon fut également délicat. En effet, certaines informations récoltées auprès de différentes sources se révélèrent contradictoires par moment.

Il fut également très difficile de récolter des informations sur la manière de faire des Add-In pour

Visual Studio .NET. Les tutorials bien expliqués sont très rares et les apprendre à les appliquer a pris beaucoup de temps.

7.2.2 Application de la méthodologie de travail

Bien qu'en dernière année d'école d'ingénieur, nous manquons encore d'expérience en méthodologie de projet. Le projet à huit en I2 fut notre première expérience en la matière et notre stage technique ne nous a pas permis de voir de projets dans leur globalité.

Ainsi, en raison de notre manque d'expérience, nous ne pouvions pas encore proposer de solution définitive à l'issue notre phase d'analyse. En outre, du fait que nous ne maitrisions pas encore la technologie utilisée, il nous a fallu passer à la phase de développement afin de valider nos choix conceptuels. Nous avons néanmoins réussi à séparer les phase de conception et de développement.

7.2.3 Gestion du temps

La gestion du temps fut un point crucial de notre projet. En plus des différents cours et projets, il a fallu effectuer notre recherche de stage. Au démarrage du projet, nous avions défini un planning de réalisation du projet.

Au moment d'établir ce planning qui comptait des délais très serrés pour chaque étape, nous n'avons pas la connaissance de toutes les autres contraintes (planning des cours, projets et devoirs). Appliquer la méthodologie de projet fut alors plus difficile.

Enfin, le mois de Janvier, traditionnellement réservé à la finalisation des PFE, fut plus chargé que ce qui nous avait été annoncé. Une nouvelle matière (Enterprise Ressource Planning) débuta et d'autres cours ne s'achevèrent que durant ce mois.

7.3 Evolution

La première évolution à apporter à l'application serait d'offrir la compabilité avec davantage de Systèmes de Gestion de Bases de Données. L'Add-In devrait ainsi être capable d'extraire le schéma d'autres SGBD et de générer une librairie ces SGBD. Nous pensons ainsi aux SGBD Access et PostGre.

Une autre évolution serait d'apporter la possibilité de générer une librairie dans d'autres langages, à savoir :
- C++
- MC++ (Managed C++)
- ASP .NET
- VB .NET

Les dernières évolutions concerneraient la librairie générée. Beaucoup de travail a déja été effectué mais certaines améliorations mériteraient d'être apportées. Elles concerneraient :

- Dans les opérations de SELECT, inclure la gestion des opérateurs (SUM, AVG, MIN, MAX) et de l'instruction GROUP BY ;
- Les types spécifiques à une bdd (ex : ENUM et SET pour MySQL) ;
- Le support des trigger et des procédures stockées ;
- Les requêtes imbriquées ;
- Les jointures récursives.

7.4 Apports du projet pour ses membres

7.4.1 Apport sur le plan technique

Par le biais de ce projet, nous avons pu découvrir une nouvelle technique, l'ORM.

Ce projet nous a également permis de nous former à un langage évolué qui sera de plus en plus utilisé dans le monde professionnel, le C#. Travailler avec C# fut l'un des points les plus attrayants du projet. Outre le fait de découvrir un nouveau langage, nous avons été séduits par les possibilités offertes par ce langage.

Un autre apport de ce projet fut le fait pouvoir travailler avec des Systèmes de Gestion de Base de Données relationnels différents. Cela nous a apporté une expérience supplémentaire sur les systèmes Oracle et MySql tandis que nous avons également pu découvrir SQL Server.

7.4.2 Méthodologie de travail

Le travail de conception et modélisation fut également très porteur. Bien qu'Add-In, Hornet a l'architecture d'une application complète. Cette expérience se révéla bénéfique car nous avons eu l'opportunité de réaliser un travail de conception complexe. En outre, afin de découvrir un nouvel outil de modélisation, nous avons utilisé Poseidon For UML de GentleWare.

L'effectif réduit de notre équipe nous a permis de communiquer de manière rapide et efficace. Nous avons ainsi pu organiser des réunions de travail à un rythme soutenu et selon les besoins du moment. En outre, afin de reporter les différents bugs de l'application, nous avons utilisé un outil de reporting de bug.

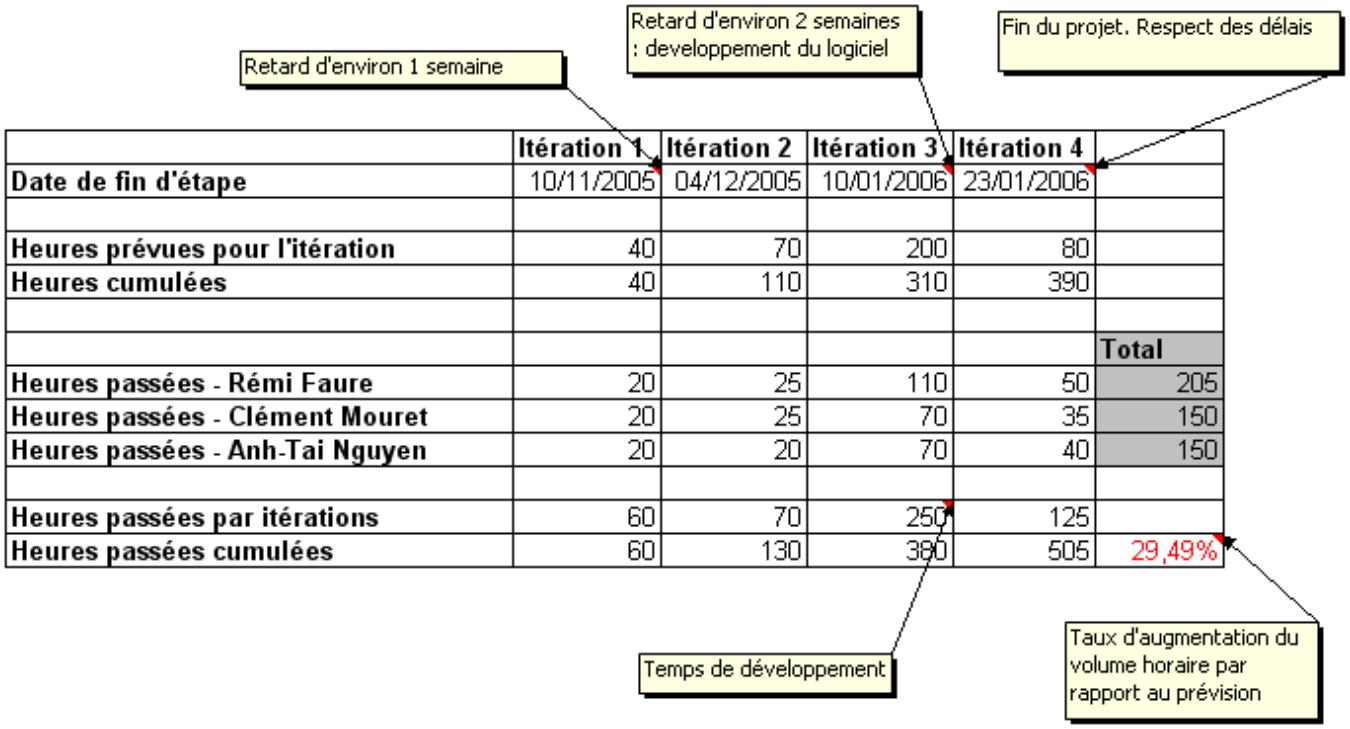
Ce projet nous a donné l'opportunité d'aller plus que dans le projet à 8 l'an passé au niveau de la méthodologie. Nous avons réussi à respecter les étapes que nous avions fixés au démarrage du projet. Nous avons eu des phases d'analyse, de conception et de développement. Ces phases ont été faites sérieusement. En effet, le produit obtenu à la fin de la phase de développement s'est révélé fidèle et conforme à ce qui avait été modélisé lors de la phase de conception.

7.4.3 Un travail valorisant vu de l'extérieur

Ce projet de fin d'étude s'est avéré difficile à mener. Le défi technique à relever était d'un niveau élevé et l'intérêt pédagogique du projet était indéniable. C'est pourquoi lors de nos différents entretiens pour la recherche de stage, ce projet nous a servi d'atouts. En effet, la difficulté technique du projet et son côté ambitieux ont séduit.

Annexe A

Tableau de bord



	Itération 1	Itération 2	Itération 3	Itération 4	
Date de fin d'étape	10/11/2005	04/12/2005	10/01/2006	23/01/2006	
Heures prévues pour l'itération	40	70	200	80	
Heures cumulées	40	110	310	390	
					Total
Heures passées - Rémi Faure	20	25	110	50	205
Heures passées - Clément Mouret	20	25	70	35	150
Heures passées - Anh-Tai Nguyen	20	20	70	40	150
Heures passées par itérations	60	70	250	125	
Heures passées cumulées	60	130	380	505	29,49%

FIG. A.1 – Tableau de bord du projet

Annexe B

Normes syntaxiques

B.1 Mise en place de la norme

Les recommandations qui suivent s'appuient principalement sur des standards fréquemment rencontrés dans la communauté des développeur C++/C#, nos expériences individuelles, des besoins propres à notre projet et une norme syntaxique rédigée pour C++ particulièrement bien faite :

http ://www.geosoft.no/development/cppstyle.html

Nous nous appuierons d'ailleurs sur cette norme pour présenter et organiser la notre.

Chaque recommandation est numérotée et sera toujours présentée de la manière suivante :

0. Description / Nom de la recommandation
`Exemple (facultatif)`
Provenance, justification, informations complémentaires

B.2 Conventions de nommage générales

1. Les noms des types (classes, structures, énumération...) doivent avoir une majuscule au début de chaque mot et le reste en minuscule
`Line, SavingAccounts`
Pratique courante en C++/C#.

2. Les noms de variables et objets doivent avoir une majuscule au début de chaque mot sauf le premier et le reste en minuscule
`line, savingsAccount`
Pratique courante en C++/C#. Permet de s'y retrouver dans une déclaration comme Line line ;

3. Les noms de constantes et les valeurs des énumérations doivent être en majuscules
`MAX_ITERATIONS, COLOR_RED, PI`
Pratique courante en C++/C#.

4. Les noms des méthodes doivent commencer par un verbe et doivent avoir une majuscule au début de chaque mot sauf le premier et le reste en minuscules
`getName(), computeTotalWidth()`
Pratique courante en C++/C#. Identique aux noms de variables mais se différencie par les parenthèses.

5. Les noms d'espace de noms (*namespaces*) doivent être en minuscules
`analyzer, iomanager, mainwindow`
Pratique courante en C++/C#.

6. Les variables privées d'une classe doivent toujours être suivies d'un underscore.

`class SomeClass`
`{`
`  private int length_ ;`
`}`

En plus d'indiquer la portée de la variable, l'underscore permet d'écrire des lignes de la forme suivante : `length_ = length ;` Ce qui est très utile lorsque l'on utilise les getteurs (`get` et les setteurs `set`)

7. Les variables génériques devrait avoir le même nom que leur type

`void setTopic (Topic *topic)` `// NOT : void setTopic (Topic *value)`
`// NOT : void setTopic (Topic *aTopic)`
`// NOT : void setTopic (Topic *x)`
`void connect (Database *database)` `// NOT : void connect (Database *db)`
`// NOT : void connect (Database *oracleDB)`

Simplifie le code en réduisant le nombre de termes et de noms utilisés. Pas besoin de chercher de noms.

8. Tous les noms doivent être écrit en anglais.
L'anglais est le langage de référence pour le développement international.

9. Les variables ayant une large portée doivent avoir des noms longs, les variables ayant une portée réduite doivent avoir des noms courts.
Ceci permet au programmeur de savoir immédiatement si une variable est importante pour la compréhension du programme ou pas. Les variables comme `i`, `j`, `k`, `m` et `n` sont utilisées pour stocker temporairement des entiers, `c` et `d` pour les caractères.

10. Les nom de l'objet est souvent implicite et ne doit pas apparaître dans le nom des méthodes
`line.getLength() ; // NOT : line.getLineLength() ;`
Cette erreur se produit généralement parce que le nom de l'objet nous vient naturellement lors de la conception de la classe mais il est superflu lors de son utilisation.

B.3 Conventions de nommage spécifiques

11. Le terme `compute` **peut être utilisé pour préfixer les fonctions dans lesquelles quelque chose est calculé.**
`valueSet.computeAverage() ; matrix.computeInverse() ;`
Informe le lecteur qu'une utilisation trop fréquente de cette fonction peut s'avérer être gourmande en ressources processeur. Il pourra alors, dans certain cas, stocker le résultat une fois pour toute.

12. Le terme `find` **peut-être utilisé pour préfixer les fonctions dans lesquelles on cherche quelques chose.**
`vertex.findNearestVertex() ; matrix.findMinElement() ;`
Informe le lecteur que cette fonction nécessite un minimum de calcul.

13. Les variables représentant des élément d'IHM devraient être préfixées par le type d'objet.
`frmEdit (form), txtName (champ texte), picPhotograph (image), lblName (label), lwStuff (listview)`
Indique immédiatement au lecteur le type d'élément auquel il a à faire.

14. Le suffixe `List` **devrait être utilisé pour les noms représentants des listes d'objets. Le nom de l'objet devrait être au singulier.**
`point (un point), pointList (une liste de point) // NOT : pointsList`
Indique au lecteur qu'il a à faire à une liste d'élement. C'est notation est plus claire que d'utiliser les pluriels (`point` représente un point, `points` représente un liste de points) car dans cette notation, il n'y a qu'un caractère qui change ce qui augment le risque d'erreur et diminue la lisibilité.

15. Le préfixe `n` **devrait être utilisé pour les variables qui représentent un nombre d'objets**
`nPoints, nLines`
Pratique courante en mathématiques pour indiquer un nombre d'objet.

16. Le suffixe `No` **devrait être utilisé pour représenté le numéro (l'identifiant) d'un objet.**
`tableNo, employeeNo`
Pratique courante en mathématiques pour représenter le numéro d'un objet.

17. Les itérateurs devraient être appelés `i, j, k,...`
`for (int i = 0 ; i < nTables) ; i++) ;`
Pratique courante en mathématiques pour indiquer un itérateur.

18. Le préfixe `is` **devrait être utilisé pour les variables et méthodes booléennes**
`isSet, isVisible, isFinished, isFound, isOpen`

Pratique courante en C++/C#.
Parfois il y a des préfixes qui s'adaptent mieux :
`bool hasLicense() ;`
`bool canEvaluate() ;`
`bool shouldSort() ;`

19. Des noms complémentaires doivent être utilisés pour des opérations complémentaires
`add/remove, create/destroy, start/stop, insert/delete, increment/decrement, old/new, begin/end, first/last, up/down, min/max, next/previous, old/new, open/close, show/hide, suspend/resume, etc.`
Reduit la compléxité par symétrie.

20. Les abréviations dans les noms devraient être évitées
`computeAverage(); // NOT : compAvg();`

Il y a 2 types de mots à considérer. Ceux que l'on trouve dans le dictionnaire, ceux là ne doivent jamais être abrégés. Ne pas écrire :
`cmd` instead of `command`
`cp` au lieu de `copy`
`pt` au lieu de `point`
`comp` au lieu de `compute`
`init` au lieu de `initialize`
`etc.`
Et il y a les groupes de mots ou phrases que l'on a l'habitude d'utiliser de manière abrégée. Ne pas écrire :
`HypertextMarkupLanguage` au lieu de `html`
`CentralProcessingUnit` au lieu de `cpu`
`PriceEarningRatio` au lieu de `pe`
`etc.`

21. Les variables booléennes négatives devraient être évitées

`bool isError; // NOT : isNoError`
`bool isFound; // NOT : isNotFound`

Cela complique la solution lorsqu'une telle variable est utilisée avec une double négation. La signification de `!isNotError` n'est pas tout de suite évidente.

B.4 Autres recommandations

22. Les `using` au début du fichier sont regroupés par namespaces, dans l'ordre alphabétique et on commence toujours par les namespaces standard puis les namespaces de l'utilisateur.

`using System;`
`using System.Collections;`
`using System.Collections.Specialized;`
`using System.IO;`
`using mainwindow;`

Ceci permet de savoir immédiatement si un namespace est disponible ou pas dans un fichier.

23. Les variables de classes doivent toujours être déclarées en privé (`private`)
C'est le concept de la programmation orienté objet et de l'encapsulation. Si l'utilisateur a accès aux données de la classe, il s'expose à des problèmes de stabilité et de sécurité.

24. Les commentaires `/* ... */` ne doivent pas être utilisés

`// Voici un commentaire`
`// Et voici une seconde ligne de commentaire`
`/// <summary>`
`/// Ceci est un commentaire façon VS.NET`
`/// </summary>`

Depuis que les commentaires à niveaux multiples (`/* /* ... */ */`) ne sont plus supportés, l'utilisation des `//` permet d'avoir la possibilité de commenter des sections de code entière avec les `/* ... */` lors du debuggage par exemple.

B.5 Documentation du code

25. Commentaires XML
`commentaire ///` `balises XML`
Forme standardisée compréhensible par de nombreux générateurs de documentation automatique

26. Balises principales

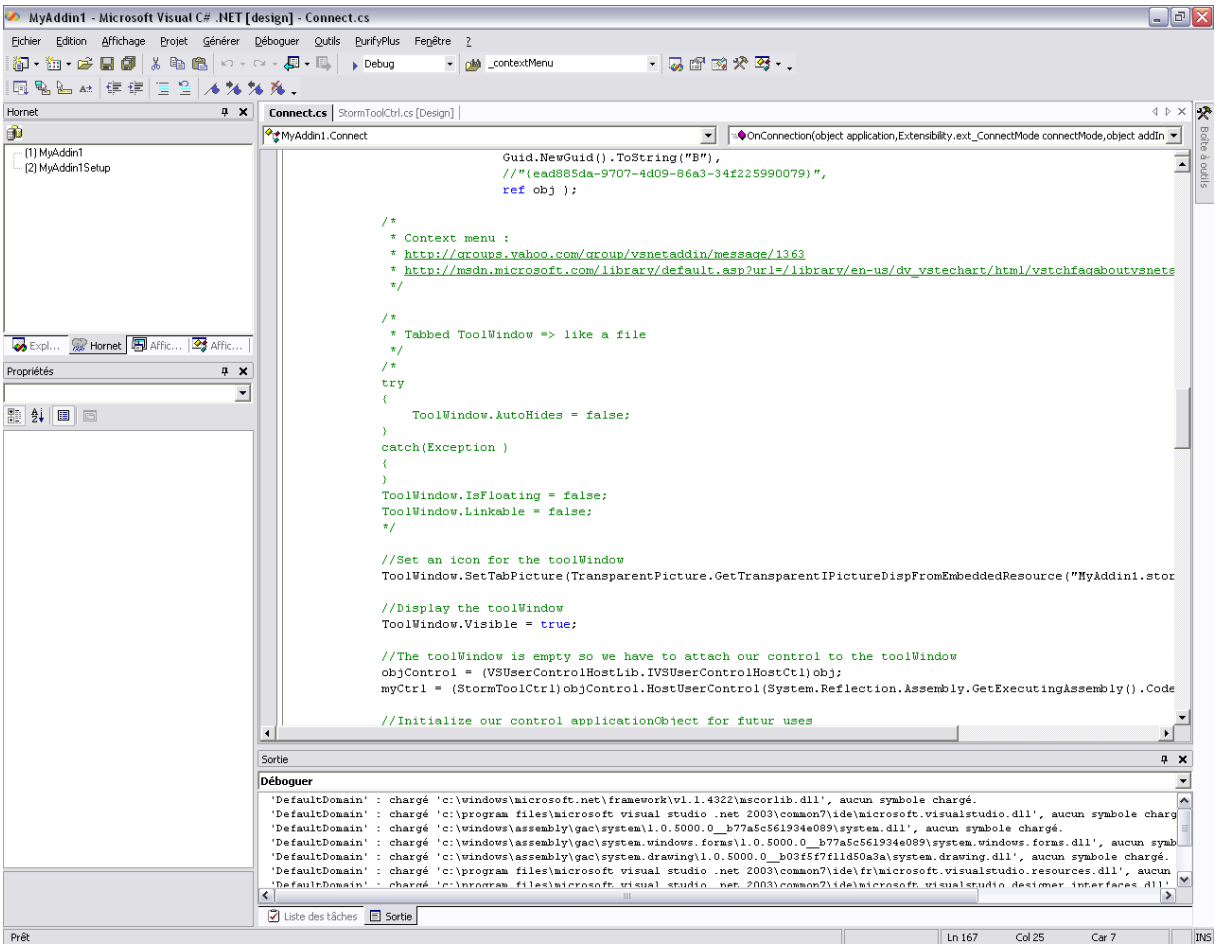
`<summary> description générale de l'élément (classe, méthode, Ě)`
`<param name="X"> description d'un paramètre`
`<value> description d'une propriété (ou attribut)`
`<returns> description du paramètre de retour`
`<seealso cref="X"> élément en relation`
`<exception> description d'une exception levée`
`<permission> accessibilité de la méthode`

27. Balises autorisées en fonction des éléments

`class <summary> <remarks> <seealso>`
`struct <summary> <remarks> <seealso>`
`interface <summary> <remarks> <seealso>`
`delegate <summary> <remarks> <seealso> <param> <returns>`
`enum <summary> <remarks> <seealso>`
`constructor <summary> <remarks> <seealso> <param> <permission> <exception>`
`property <summary> <remarks> <seealso> <value> <permission> <exception>`
`method <summary> <remarks> <seealso> <param> <returns> <permission> <exception>`
`event <summary> <remarks> <seealso>`

Annexe C

Environnement Visual Studio



- En haut : des menus
- En dessous : des barres d'outils
- A gauche : les toolWindows (Solution Explorer, fenêtre de propriété, ...)
- Au centre : les fichiers ouverts
- En bas : la fenêtre de Sortie (résultat de la compilation d'un projet)

Annexe D

Diagrammes annexes

D.1 Bibliothèque de classes Hornet : *Data Persistent Objects Library*

D.1.1 *Data Field*

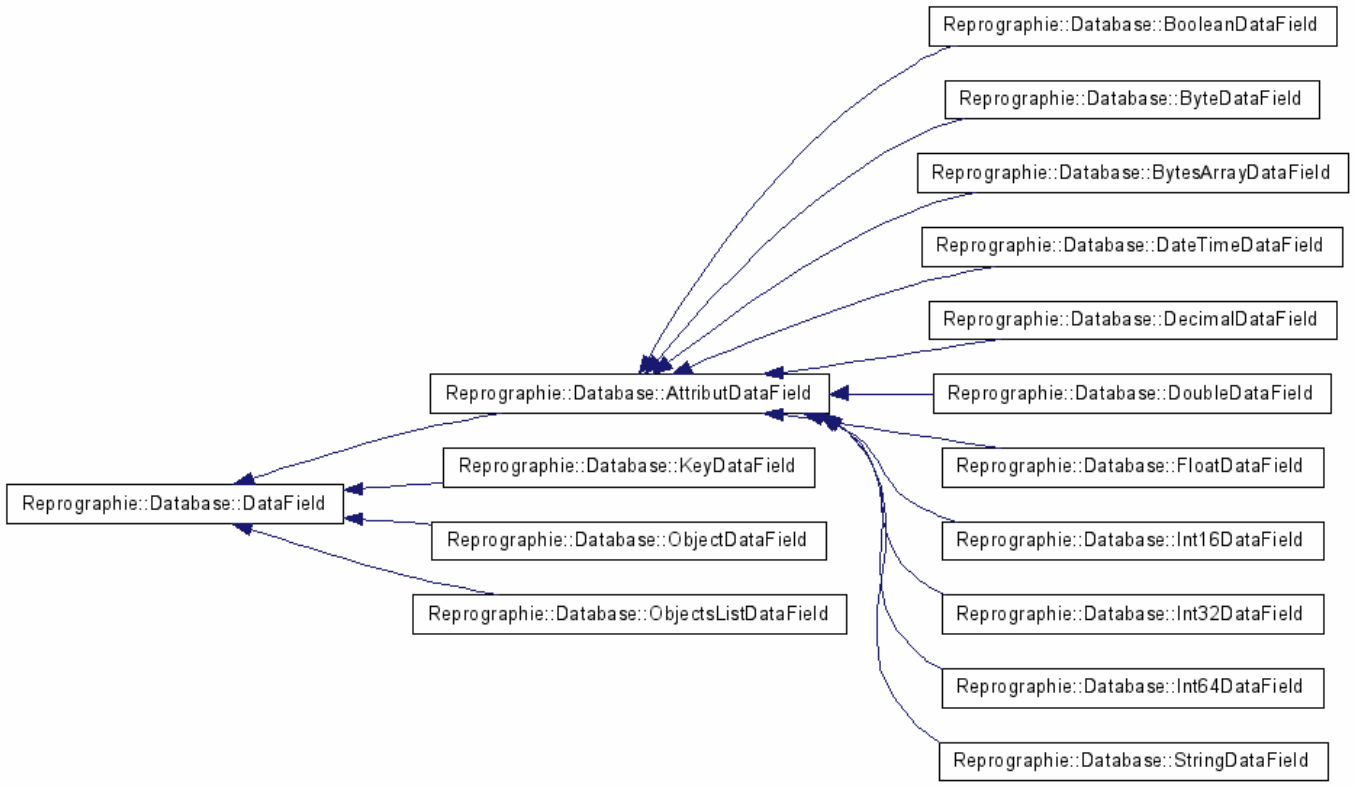


FIG. D.1 – Diagramme des classes : All *Data Field*

D.1.2 *Data Object*

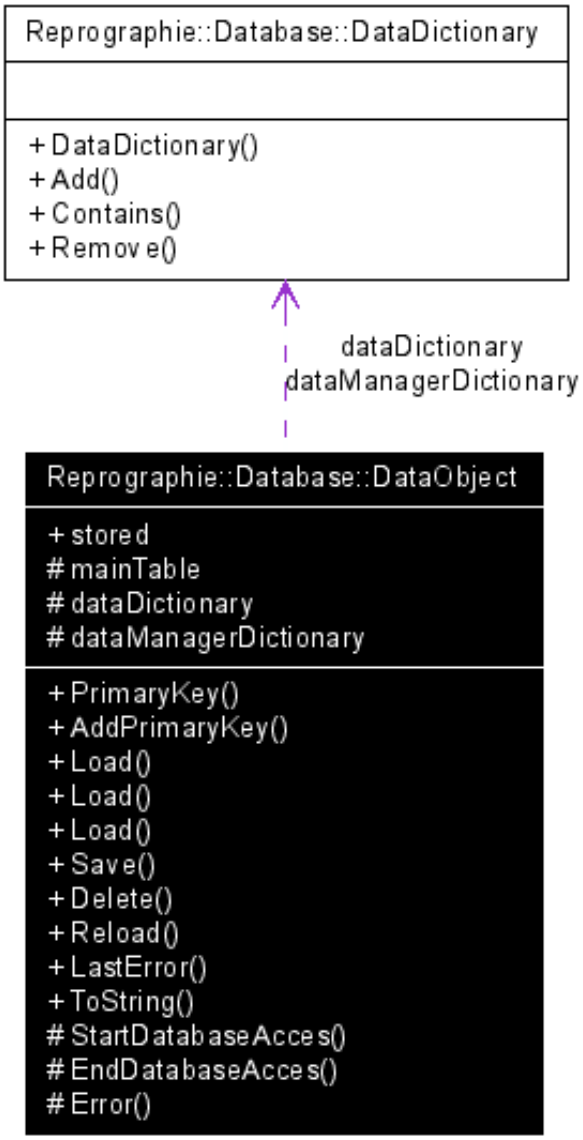


FIG. D.2 – Diagramme des classes : *Data Object*

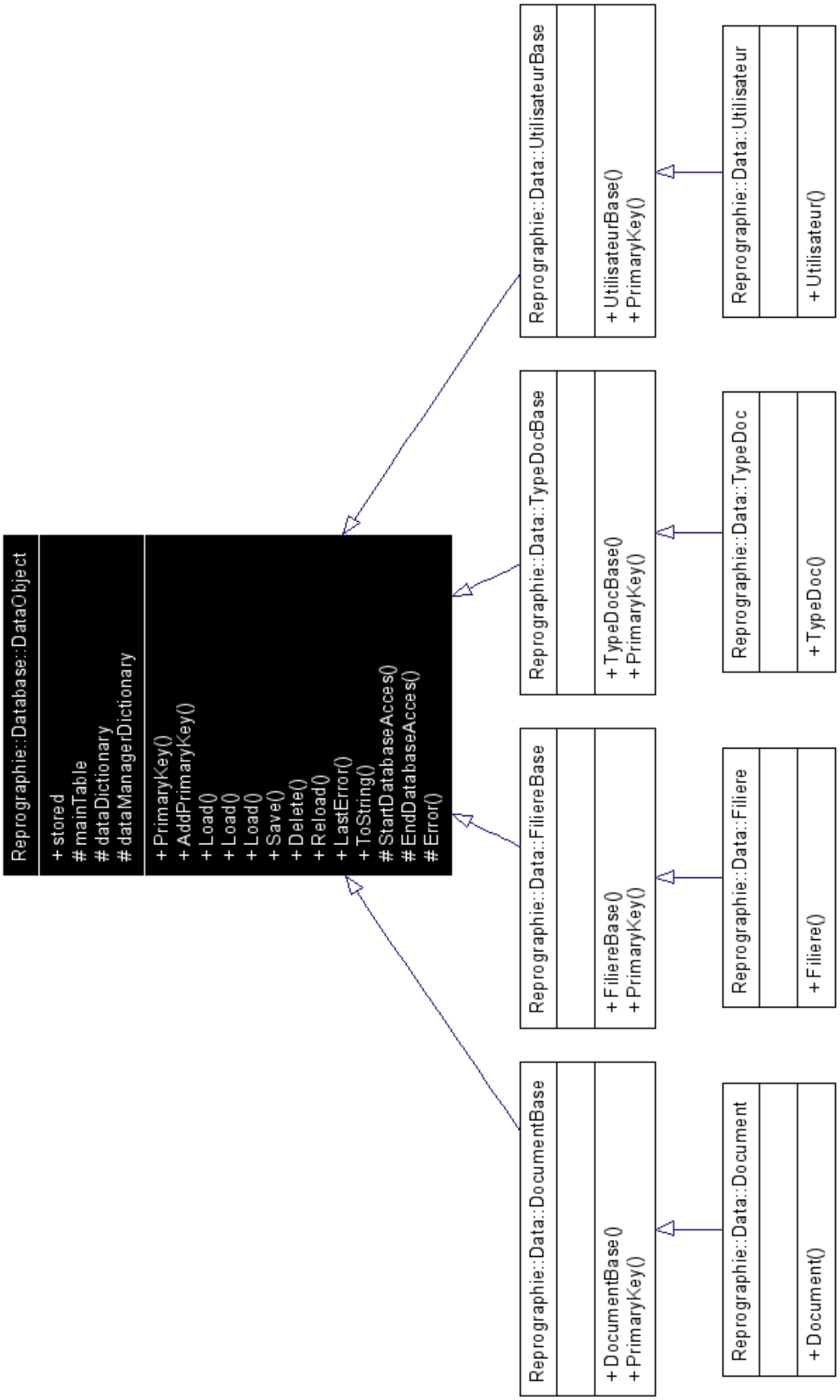


FIG. D.3 – Diagramme des classes : Classes générées

D.1.3 *SQL Query Engine*

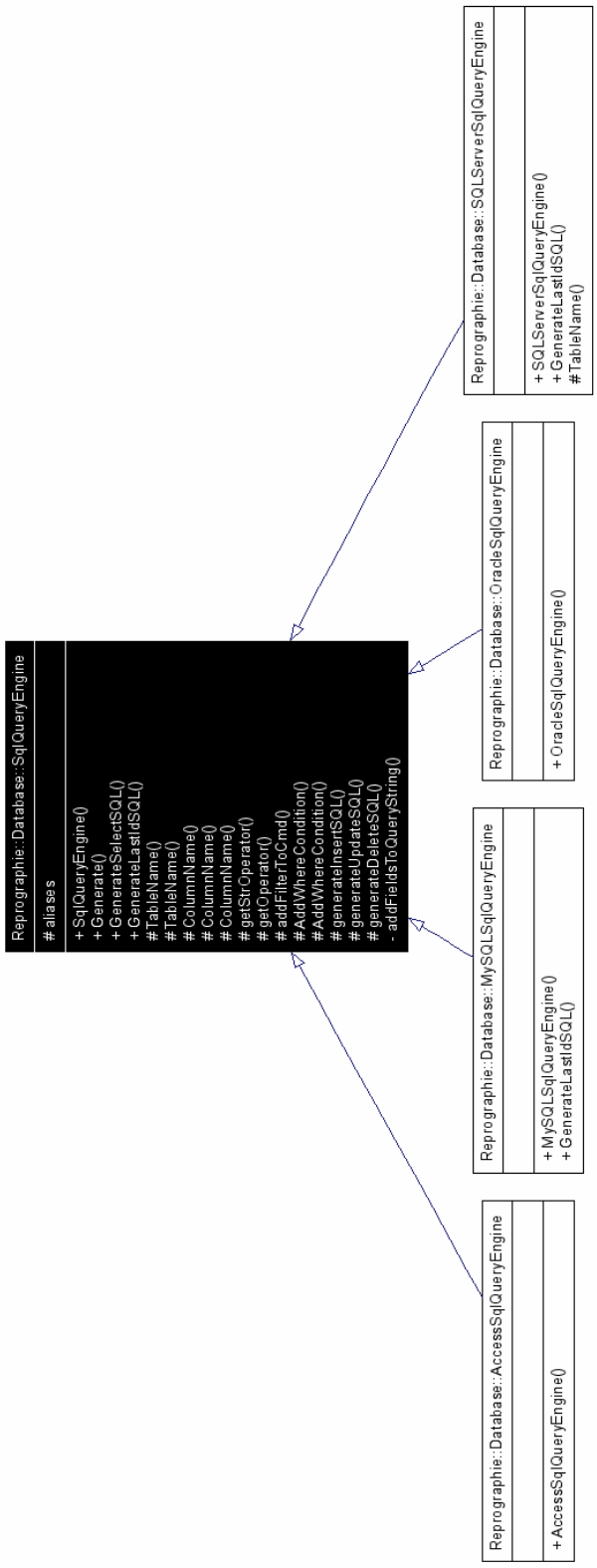


FIG. D.4 – Diagramme des classes : *SQL Query Engine*

Annexe E

Critères d'évaluation d'un ORM

http ://madgeek.com/Articles/ORMapping/EN/mapping.htm

E.1 Basic features

- Be able to use inheritance, create hierarchies between entities, and use polymorphism (we are using objects !). The tools can support a variety of combinations for tables and classes to allow these mechanisms.
- Handle any type of relations (1-1, 1-n, n-n)
- Support for transactions
- Aggregates (equivalent to SQL's SUM, AVG, MIN, MAX, COUNT)
- Support for grouping (SQL's GROUP BY)

E.2 Useful extended features

- Supported databases. A big advantage of mapping tools is that they provide an abstraction of the underlying database engine. Most of them allow switching easily between RDBMSs (Relational Database Management Systems).
- Query language (OQL - Object Query Language, OPath). We very frequently have to execute dynamic queries. It's the case at least with searches based on filters. It is important to be able to use a powerful query language.
- Support for DataBinding (to be able to bind data objects to visual components). Note some specificity exists with Windows Forms.

E.3 Flexibility

- Customization of queries. We often need to go beyond what is possible with the provided query language. In these cases, we need to be able to provide custom SQL queries. HQL, which is a strong point of Hibernate/NHibernate, allows for this. We could also wish a dynamic mapping to be possible from developer provided SQL queries.
- Support any type of SQL joins (inner join, outer join)
- Concurrency management (support for optimistic and pessimistic approaches)
- Support for the data types specific to the database management system (identity columns, sequences, GUIDs, autoincrements)

- Be able to map a single object to data coming from multiple tables (joins, views). Most of the tools handle a direct mapping of a class to one table. We often need more.
- Be able to dispatch the data from a single table to multiple objects.

E.4 Assistance, ease of use

- GUI to set up the mapping. Such a graphical tool presents the relational data model and lets you specify the objects to be created or at least the links between the objects and the tables.
- Generation of the classes. This can speed up the development, even if in a lot of cases we prefer to map the database to hand-coded classes or to classes generated from UML for example. Check which scenarios are supported by the tools.
- Generation of the database schema. Some tools work only with a database they generated. This can be a big constraint, especially if you have to work with a legacy database of course ! Otherwise, it all depends on whether you are an expert in database modeling, or if you prefer not to have to deal with the database schema. If you have a DBA who takes care of your databases, or if you prefer to design them by yourself, be sure to select a mapping tool that doesn't require its own data model.

E.5 Optimizations, performance, design

- Global performance (good implementation of the object-relational mapping concept)
- Lazy loading (the loading of some data is deferred until it's needed) o for the data through relations o for some columns. When we want to display just a list of names, we don't need all the columns of a table to be loaded. We may need the blob fields only at certain point, under certain conditions, and so it's better to load them only at that time.
- Cache dynamically generated queries, so that they don't get rebuilt at each call.
- Cache some data to avoid too many calls to the data source.
- Optimized queries (update only the modified columns ; detect situations where the number of executed queries can be reduced ; ...)
- Handle circular references without duplication of objects ("account == account.Client.Account")
- Handle cascade updates. Deleting a master record should delete the linked details if wished so.
- Bulk updates or deletions. When we want to update or delete thousands of records at a time, it's not possible to load all the objects in memory, while this can be easily and quickly done with a SQL query (DELETE FROM Customer WHERE Balance < 0). Support from the tool is welcome to handle such massive operations without having to deal with SQL. Hibernate is not very good on this point for example.

E.6 Evolution, compatibility

- Maintainability (what happens if the database schema changes ? If I need to add a new collection ?)
- Possibility to move to a new mapping tool (what would it imply ? At what cost ?)
- Serialization. Serialization can be used to persist data outside of the database. Serialization can be done into a binary format, or more important, in XML (see the section on SOA below)

- Distributed objects (remoting ; web services ; requires support for serialization) Welcome additional features
- Freedom in the design of the classes (no base class for the entities ; no mandatory interface ; no specific class for collections). Think POJO (Plain Old Java Object).
- Less constraints as possible on the database schema (eg. support for composite data keys)
- State information on data. It can be useful to know by looking at an object if the entity has been added, modified, deleted.
- External mapping file or not ? Attributes (annotations) in code or not ? o Advantages of external files : mapping entirely externalized ; no intrusion in the classes ; can be generated o Disadvantages of external files : one or multiple additional files to deal with ; a syntax to learn if no GUI is provided ; understanding the links between the code and the database requires some effort ; can become out of sync with the code o Advantages of attributes/annotations : everything at hand at code-level ; the mapping is obvious since directly present on classes and class members ; can be used to generate external mapping files using reflection if needed ; if the mapping tool isn't used anymore, they are ignored o Disadvantages of attributes/annotations : the code is "polluted" ; the code depends on a specific mapping framework
- Advanced compatibility with the development platform. For example, compatibility with .NET's DataSets can be useful. It can be important to be able to convert object graphs into DataSets to interact with components that require them (reporting tools for example).
- Support for disconnected mode (fill objects from a database, close the connection, the session, create/update/delete some objects, and apply this modifications back to the database later)
- Interceptors and delegation mechanisms to be able to react when the handling of the persistence happens (eg. to be able to log what happens)
- Support for stored procedures. The advantages of stored procedures compared to dynamic SQL queries make for a hot debate, but it is better to have the choice.
- One advantage of some tools for .NET could be their popularity in the Java world. This is the case for iBatis.NET or NHibernate. NHibernate is a port of Hibernate, a tool widely used in Java, and so can benefit from a large community of developers and existing documentation.
- Ability to specify constraints between objects and on properties (OCL - Object Constraint Language). This can avoid having to wait for the data to reach the database before being validated.
- Filtering objects in memory (without having new queries executed on the database)
- Be able to defer the updates on the database, and apply them at a given time using a specific method call, instead of having them systematically applied immediately.

Annexe F

Licences

Rappel des différents types de licences existants dans le cadre d'un logiciel.

F.1 Freeware ou Shareware

Parmi les oeuvres dont le code de la propriété intellectuelle prévoit la protection, le logiciel se distingue par une caractéristique inhabituelle : l'utilisateur peut la reproduire à l'infini et à l'identique, presque sans coût ni effort. Ce constat conduit le législateur à restreindre le champ l'application de la copie privée (L. 122-5 2 du code de la propriété intellectuelle) à la copie de sauvegarde (L. 122-6-1 1 du C. prop. intellectuelle) Faut-il en déduite qu'en dehors le cette dernière hypothèse, toute duplication d'un progiciel constitue une contrefaçon ? -certainement pas - parallèlement au marché classique, existe un système mondial de diffusion le progiciels en libre copie dont l'avenir repose sur le développement des réseaux en ligne.

Lorsque la conception d'un progiciel est le fait d'un auteur individuel, sa commercialisation correspond à celle d'une oeuvre littéraire. L'auteur confie son programme à un éditeur, pour que ce dernier le commercialise et le mette à disposition d'une clientèle d'utilisateurs finaux. Différents contrats s'enchaîneront. Tout d'abord, auteur et éditeur signeront un contrat d'édition dont le régime juridique résulte des articles L. 132-1 et suivants du Code de la propriété intellectuelle. Puis l'éditeur conclura des licences de commercialisation avec des distributeurs pour le cas où la distribution passe par l'intermédiaire de points de vente. Enfin, des licences de concession d'usage du logiciel permettront de définir les prérogatives que l'éditeur entend transmettre à l'utilisateur final. Il convient néanmoins d'observer qu'une partie substantielle de la distribution des progiciels échappe au schéma évoqué ci-dessus.

Les auteurs individuels, dans leur majorité, excluent un recours au contrat d'édition. Leurs logiciels ne sont donc pas distribués en boutique sous forme de produits édités mais sont mis à disposition sur des serveurs qui les hébergent afin de permettre aux utilisateurs de les télécharger presque sans frais.

L'originalité de ce mode de distribution, propre à l'informatique, repose donc sur la divulgation de l'oeuvre (droit moral, L 121-2 al. 1 C. prop. Intellectuelle) directement sur le réseau et sur une gestion particulière du droit de reproduction (droit patrimonial, L 122-3 al. 1 C. prop. Intell). car l'auteur sollicite les utilisateurs pour contribuer à sa diffusion, en encourageant la duplication et la transmission des copies à titre gratuit.

Puisqu'en cette matière, l'existence a précédé l'essence, il convient désormais d'en rendre compte

au plan juridique. Le concept de libre copie constitue le dénominateur commun à différentes licences d'utilisation qui se distinguent sous l'angle des droits transmis à l'utilisateur (et accessoirement par leur caractère onéreux). Ce critère permet d'établir un classement des différentes licences rencontrées, de la plus libérale à la plus restrictive, nous évoquerons donc successivement les logiciels mis dans le domaine public F.1, les "freewares" F.1 puis les " sharewares" F.1.

Les logiciels mis dans le domaine public

Les logiciels appartenant au domaine public sont ceux spontanément offerts par l'auteur à la communauté des utilisateurs. En effet, le logiciel constitue un genre d'oeuvre protégeable apparu trop récemment pour avoir épuisé les délais légaux de protection, en particulier depuis que la durée de protection a été alignée sur celle de droit commun. La divulgation du logiciel portera sur le code-objet, la documentation et s'étendra au code-source. L'auteur renonce donc à l'ensemble de ses droits patrimoniaux, ainsi chacun peut s'approprier le logiciel sans avoir à acquitter de redevance (en particulier au titre de son utilisation ou de son adaptation).

S'agissant des droits extra- patrimoniaux, rappelons au préalable que la protection réservée aux auteurs de logiciels est moins étendue qu'en droit commun de la propriété littéraire et artistique. Cependant, pour le droit français, le placement d'une oeuvre dans le domaine public n'a de conséquences ni sui le droit de paternité ni sur le droit au respect de l'oeuvre puisque l'article L 121-1 al. 3 C. prop. intell. les répute perpétuels, inaliénables et imprescriptibles (ĚĚ.). L'objectif des auteurs qui placent leur oeuvre dans le domaine public est de faciliter la libre appropriation de celle-ci afin de favoriser le progrès de leur art. C'est la raison pour laquelle ils encouragent la libre adaptation du programme (en divulguant le code-source) et renoncent de facto à l'exercice de leurs droits moraux. Envisageons successivement un exemple, puis un contre-exemple.

Du " copyrigt " au " copyleft "

A titre d'exemple, nous évoquerons le cas de Philip Zimmermann qui est l'auteur d'un logiciel de cryptologie dénommé "pretty good privacy" (PGP). Il a placé son programme dons le domaine public afin d'offrir à tout utilisateur le moyen de garantir la confidentialité de son courrier électronique contre les velléités inquisitoires des services secrets. Ainsi, grâce à Internet (où le code-source était disponible), des dizaines d'universités s'en sont emparées pour l'abriter, le faire connaître et étudier sa structure interne afin de concevoir de nouvelles versions. Placer un logiciel dans le domaine public est donc le moyen le plus simple d'offrir un logiciel à la communauté des utilisateurs.

Le contre-exemple nous est fournit par R.Stallman. Ce dernier précise à juste titre que l'oeuvre originaire, placée dans le domaine public, peut être détournée par un intermédiaire moyennant quelques modifications, elle sera alors une oeuvre dérivée. C' est la raison pour laquelle Richard Stallman a recours au copyright, et propose une licence d'utilisation originale nommée "copyleft" ; "ĚĚ.to copyleft a program, first copyright it ; then we add distribution terms, whith a légal instrument that gives everyone rights to use, modify, and redistribute the program's code or any program derived from it, only if the distribution terms are unchanged. Proprietary software developers use copyright to take away the users's freedom ; we have a copyright to guarantee their freedom. that's vue reverse the name, changing "copyright" in "copyleft"Ě..".

En conclusion, le critère permettant de déterminer si un logiciel a été placé dans le domaine public ou s'il fait l'objet d'un " copyleft " est la vulgarisation du code-source, a contrario, celui-ci n'est jamais divulgué c'est que l'auteur entend se réserver la maîtrise de l'évolution des futures versions.

Les licences d'utilisation que nous évoquerons au cours des développements qui suivent s'adressent qu'à un public d'utilisateurs (à l'exception d'informaticiens susceptibles d'adapter les progiciels). Par ailleurs, chacune d'elles implique strict respect des droits moraux de l'auteur (les auteurs le rappellent fréquemment en affirmant leur droit à la paternité et en étendant le droit respect à l'ensemble des fichiers associés au logiciel. Ces licences d'utilisation pourront être transférées soit à titre gratuit (licence en freeware soit à titre onéreux (licence en sharewre ou en "cripplewore"). Envisageons désormais mais les particularités de la licence freeware

La licence en " freeware "

Les conditions d'une licence en freeware doivent être distinguées d'un placement de l'oeuvre dans le domaine public. Comme dans le cas précédent, l'auteur renonce à percevoir une redevance au titre de l'utilisation de son logiciel mais il en interdit toute adaptation. Il conserve ainsi la maîtrise de l'évolution des futures versions dont l'exclusivité de la commercialisation lui reviendra. Ainsi, contrairement à un placement de l'oeuvre dans le domaine public, l'auteur revendique le respect de ses droits moraux. Il en résulte pour l'utilisateur l'interdiction de modifier le programme ou les fichiers qui lui sont associés.

En diffusant son progiciel sous licence "freeware" l'auteur renonce donc à exploiter commercialement une version déterminée. En effet, dès lors qu'il autorise la diffusion de son oeuvre par copie et qu'il précise que l'utilisation du progiciel est gratuite, il n'a plus prise sur les exemples que tout utilisateur peut distribuer et utiliser sa guise. Certes l'auteur pourrait contractuellement limiter les conséquences d'une telle diffusion par la suspension de l'autorisation de distribuer à compter d'une date préalablement déterminée, ou la limitation de l'autorisation de diffusion à une zone géographique strictement délimitée mais dans ce cas il ne peut que s'en remettre à la bienveillance des utilisateurs à défaut de pouvoir les identifier et veiller au respect de ces stipulations.

On doit, la plupart des logiciels diffusés sous licence " freeware" soit à des centres de recherche publics, soit à des auteurs indépendants qui souhaitent acquérir une certaine notoriété tout en se réservant pour l'avenir une faculté de commercialisation en cas de succès.. Parfois, la diffusion sous licence "freeware" permet de promouvoir un progiciel, une technologie qui font déjà l'objet d'une commercialisation. Dans cette hypothèse, la mise à disposition d'un produit gratuit ne semble pas entraver le libre jeu de la concurrence, mais une appréciation au cas par cas s'impose. L'actualité récente offre un exemple où la diffusion d'un progiciel au moyen d'une licence en "freeware" a pour effet d'entraver le fonctionnement normal du marché. Voici les faits :

" freeware " et respect du marché

Après avoir négligé le phénomène Internet, la société Microsoft décide de rattraper rapidement son retard et projette d'acquérir son principal concurrent (la société Netscape qui détient alors 80 % lu marché des navigateurs W3). Malheureusement pour la société de Bill Gates cette stratégie de croissance horizontale échoue. Pour atteindre son objectif, celui-ci décide de développer un produit

concurrent diffusé sous licence " feeware' (gratuité de la licence d'utilisation) alors que les sociétés déjà présentes sur ce marché commercialisent des produits comparables par le biais de licences en " cripplewaree" (cession à litre onéreux de la licence d'utilisation).

La concurrence prend la menace au sérieux. Elle saisi le département américain de la Justice afin qu'il ouvre une enquête et sanctionne ces pratiques qualifiées d'anticoncurrentielles.

Si un juge français avait à connaître de telles pratiques, l'issue du procès ne ferait guère de doute comme le souligne une décision du conseil de la concurrence rendue en 1986 : " Le groupe a mis en oeuvre une stratégie délibérée visant à éliminer les publications concurrentes des siennes... afin d'acquérir une situation de monopole sur ces marchés lui permettant ultérieurement de dégager des marges importantes. Celle politique, rendue possible par la puissance économique du groupe X. s'est traduite par des baisses artificielles des prix auxquelles les publications concurrentes n'avaient pas moyen de résister. Elle a été poursuivie jusqu'à leurs disparitions Ces agissements ont eu pour objet et pour effet d'entraver le fonctionnement normal des marchés considérés ".

Aux Etats-Unis, la réglementation antitrust ne permet pas de prévenir de telles pratiques puisqu'elle date de l'ère industrielle. Celte situation ne semble pas devoir évoluer à court terme, en l'absence d'une volonté politique du ministère de la justice américaine. Elle consacre l'influence d'économistes libéraux selon lesquels il n'est pas nécessairement mauvais de s'appuyer sur un pouvoir de monopole pour intervenir dans de nouveaux secteurs si cela se traduit par une meilleure efficacité commerciale el si cela permet aux consommateurs de faire une bonne affaire.

Les licences d'utilisation que nous évoquerons dans les développements qui suivent sont à titre onéreux. Elles se distinguent en considérant l'étendue de la divulgation (droit moral, L. 121-2 al. 1 C. prop. intell.) Si l'auteur délivre l'essentiel des fonctionnalités du logiciel, il s'agira d'une licence en "shareware " dans le cas contraire, ce sera une licence en "crippleware" Envisageons les particularités de chacune d'elles.

La licence en " shareware"

La licence en " shareware" quant à elle s'entend d'un contrat par lequel l'auteur d'un progiciel en autorise la reproduction à titre gratuit afin de le diffuser et de permettre à tout utilisateur de l'essayer avant d'acheter. En pratique, l'auteur divulgue (droit moral, L. 121-2 al. 1 C. prop. intell.) son progiciel sur un serveur Internet, qui l'héberge, afin de permettre une mise à disposition du public. Ce dernier précise que son logiciel peut être librement dupliqué (droit patrimonial, L. 122-3 al. 1) dans le respect de l'oeuvre, L 121-1 C prop. intell. par tout utilisateur, afin de le communiquer à litre gratuit et d'en permettre l'achat en cas d'utilisation effective.

Celle licence ne se distingue de la licence en "freeware" que sous l'angle de sa finalité marchande. Cette finalité échappe parfois aux utilisateurs dans la mesure où la version transmise recèle l'essentiel des fonctionnalités de la version enregistrée. Néanmoins, la mise à disposition des fonctionnalités se justifie par la faculté accordée à l'utilisateur d'effectuer un libre essai avant d'acheter (try before you buy) et non par une quelconque intention libérale de l'auteur (Voir supra " logiciel placé dans le domaine public, licence "copyleft", licence en "freeware ").

Au terme de la période dite de "libre essai", l'utilisateur a une alternative. S'il décide de conserver le logiciel, il devra verser une redevance à l'auteur, Ce dernier lui adressera en retour soit la dernière

version, soit une clef électronique permettant d'établir que le montant de la licence d'utilisation a été acquitté. Pour le cas où l'utilisateur décide de ne pas rémunérer l'auteur, la licence en "shareware" stipule que l'utilisateur doit cesser d'utiliser le progiciel et l'effacer.

Outre une diminution substantielle du prix, la licence en "shareware" permet aux utilisateurs de vérifier préalablement la compatibilité du logiciel et d'en apprécier, par l'usage, les qualités intrinsèques (ergonomie, fonctionnalités ...). Ce mode de commercialisation permet donc de prévenir une erreur sur les qualités substantielles alors que la commission des clauses abusives précise dans sa recommandation 95-02 qu'éditeurs et distributeurs manquent parfois à leur devoir de conseil.

Cependant, malgré un succès certain, la diffusion de logiciels commerciaux sous licence "shareware" n'a pas encore atteint ses objectifs. Si elle permet d'établir la notoriété d'un auteur, elle ne lui assure qu'une rémunération aléatoire. Dans cette perspective, les juristes ont un rôle à jouer, d'une part, en qualifiant ce contrat afin d'en préciser le régime juridique, d'autre part, en contribuant à lutter contre le pillage des oeuvres diffusées par le biais d'une telle licence.

S'agissant du manque à gagner des progiciels diffusés au moyen d'une telle licence, nous distinguerons suivant que la contrefaçon est le fait, soit d'utilisateurs (une solution préventive pourra être recherchée), soit de professionnels (une solution répressive sera privilégiée) Envisageons au préalable la possibilité pour l'acquéreur leur d'agir en contrefaçon.

S'agissant du mode classique de commercialisation des progiciels, nous avons vu que la cession du droit de reproduction par l'auteur- l'éditeur, permet la commercialisation des exemplaires par des revendeurs aux utilisateurs finaux. Un tel mode de distribution repose sur l'exclusion du droit de reproduction concédé par l'auteur- l'éditeur, de telle sorte que ce dernier se sera le seul habilité à produire des exemplaires du progiciel Ainsi, dans le système classique de commercialisation, la valorisation du progiciel repose sur la concession du droit de reproduction de l'éditeur.

Si l 'utilisateur ne lève pas l'option, l'autorisation qui lui a permis de reproduire le logiciel devient caduque, dès lors, il devra effacer le logiciel sous peine d'être possible du délit de contrefaçon. Certes l'auteur est démuni face aux utilisateurs peu scrupuleux qui négligent de le rémunérer puisqu'il lui est matériellement impossible de les identifier.

L'auteur devra donc privilégier une attitude préventive face au problème de la contrefaçon. Elle consistera à mettre l'accent sur les risques qu'implique l'utilisation du logiciel au-delà de la période d'essai. Il pourra par exemple conditionner le premier lancement du logiciel à l'accomplissement d'une formalité empruntée aux ventes en forme authentique. L'installation serait alors subordonnée à la saisie par l'utilisateur d'une formule résumant ses principaux engagements. Cependant, précisons que si ce formalisme constitue un progrès dans la mesure où il permet d'établir l'acceptation de l'utilisateur, il ne permet pas de prouver son identité. Sur ce point, l'identification des utilisateurs au moyen de leurs empreintes biométriques peut constituer une première ébauche de solution.

La contrefaçon de logiciels diffusés sous licence "shareware" peut également être imputée à des professionnels. Il s'agit plus particulièrement d'éditeurs dont l'activité consiste à collecter, éditer ; commercialiser des compilations. Le grief de contrefaçon se justifie à plusieurs titres.

D'une part, la licence en " shareware " stipule impérativement que la reproduction du logiciel ne peut être monnayée. D'autre part, l'article L. 132-7 du C.prop. intell. dispose que le consentement donné par écrit de l'auteur est obligatoire, pour toute édition d'une oeuvre. Enfin et surtout, l'article L. 132-5 du C. prop. intell. pose le principe d'une rémunération de l'auteur proportionnelle aux

ventes en cas de conclusion d'un contrat d'édition. Or, un même logiciel ne peut à fois faire l'objet d'un contrat d'édition, lequel implique que le prix comprend la rémunération de l'éditeur, et continuer à être désigné "sous licence shareware". ce qui implique que la rémunération de l'auteur reste due par l'acquéreur du produit. Cette dernière contradiction apparaît de manière criante lorsqu'on examine les clauses destinées à informer l'utilisateur sur les droits qui lui sont transférés, ou plus exactement, sur les droits qui ne lui sont pas transférés. De telles clauses méritent d'être citées, au moins pour l'ironie dont elles témoignent puisqu'elles soulignent que l'achat ne porte que sur le contenant (conditionnement et support numérique) à l'exclusion du contenu (le produit lui-même, c'est-à-dire le ou les logiciels) :

" Attention, le shareware n'est pas gratuit, pour continuer à bénéficier de logiciels de qualité, n'oublier par de verser votre modeste contribution" à l'auteur "

" Tirez profit du principe du shareware : testez les logiciels et achetez-les (enregistrez-vous) si ces derniers vous plaisent. Vous avez donc tout loisir d'essayer chaque programme avant de décider d'en acquérir les droits d'utilisation".

Le constat est donc amer pour les auteurs de logiciels diffusés sous licence 'shareware". Pourtant, plusieurs arguments d'opportunité plaident en faveur d'un strict respect des droits que la loi leur reconnaît en qualité d'auteur. Premièrement, nous avons vu que la faculté de " libre essai " accordée aux utilisateurs permet de prévenir dans une large mesure une erreur sur les qualités substantielles. Deuxièmement, la distribution en libre copie mérite d'être saluée puisqu'elle repose sur le plébiscite des utilisateurs satisfaits. Cette démarche des auteurs indépendants revient à accepter une sélection naturelle des oeuvres où seul compte le mérite. Tel n'est pas le cas des éditeurs qui s'appuient essentiellement sur la publicité pour promouvoir leurs progiciels. Or il est permis de penser que celle-ci a parfois un caractère descriptif si l'on considère certaines clauses au terme desquelles différentes formes de publicité préalables leur sont inopposables. Troisièmement, l'existence d'un secteur alternatif de distribution de progiciels commerciaux permet d'entretenir un certain degré d'innovation dans un marché où les principaux intervenants témoignent d'une certaine complicité objective.

F.2 Licences Open-Source

" Open Source " implique plus que la simple diffusion du code source. La licence d'un programme " open-source " doit correspondre aux critères suivants :

Libre redistribution

La licence ne doit pas empêcher de vendre ou de donner le logiciel en tant que composant d'une distribution d'un ensemble contenant des programmes de diverses origines. La licence ne doit pas exiger que cette vente soit soumise à l'acquittement de droits d'auteur ou de royalties. (voir justification)

Code source

Le programme doit inclure le code source, et la distribution sous forme de code source comme sous forme compilée doit être autorisée. Quand une forme d'un produit n'est pas distribuée avec le code source correspondant, il doit exister un moyen clairement indiqué de télécharger le code source, depuis l'Internet, sans frais supplémentaires. Le code source est la forme la plus adéquate pour qu'un programmeur modifie le programme. Il n'est pas autorisé de proposer un code source rendu difficile à comprendre. Il n'est pas autorisé de proposer des formes intermédiaires, comme ce qu'engendre un préprocesseur ou un traducteur automatique. (voir justification)

Travaux dérivés

La licence doit autoriser les modifications et les travaux dérivés, et leur distribution sous les mêmes conditions que celles qu'autorise la licence du programme original. (voir justification)

Intégrité du code source de l'auteur

La licence ne peut restreindre la redistribution du code source sous forme modifiée que si elle autorise la distribution de fichiers ń patch ż aux côtés du code source dans le but de modifier le programme au moment de la construction. La licence doit explicitement permettre la distribution de logiciel construit à partir du code source modifié. La licence peut exiger que les travaux dérivés portent un nom différent ou un numéro de version distinct de ceux du logiciel original. (voir justification)

Pas de discrimination entre les personnes ou les groupes

La licence ne doit opérer aucune discrimination à l'encontre de personnes ou de groupes de personnes. (voir justification)

Pas de discrimination entre les domaines d'application

La licence ne doit pas limiter la champ d'application du programme. Par exemple, elle ne doit pas interdire l'utilisation du programme pour faire des affaires ou dans le cadre de la recherche génétique. (voir justification)

Distribution de la licence

Les droits attachés au programme doivent s'appliquer à tous ceux à qui le programme est redistribué sans que ces parties ne doivent remplir les conditions d'une licence supplémentaire. (voir justification)

La licence ne doit pas être spécifique à un produit

Les droits attachés au programme ne doivent pas dépendre du fait que le programme fait partie d'une distribution logicielle spécifique. Si le programme est extrait de cette distribution et utilisé ou distribué selon les conditions de la licence du programme, toutes les parties auxquelles le programme est redistribué doivent bénéficier des droits accordés lorsque le programme est au sein de la distribution originale de logiciels. (voir justification)

La licence ne doit pas contaminer d'autres logiciels

La licence ne doit pas apposer de restrictions sur d'autres logiciels distribués avec le programme qu'elle couvre. Par exemple, la licence ne doit pas exiger que tous les programmes distribués grâce au même médium soient des logiciels ń open-source ż.