Migration vers une architecture orientée service

Charbel Y. Younès

Le Conservatoire National des Arts et Métiers
Centre associé au CNAM Paris

Mémoire Candidat d'Ingénieur

Migration Vers une Architecture Orientée Service

Réalisé par
Charbel Y. YOUNES

Sous la direction de
M. Bachir HARB

Année Universitaire
2010-2011
Beyrouth - Liban

Remerciements

En préambule à ce mémoire, je souhaitais adresser mes remerciements les plus sincères aux personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire ainsi qu'à la réussite de cette formidable année universitaire.

Je tiens à remercier sincèrement Monsieur ***Bachir Harb***, qui, en tant que Directeur de mémoire, s'est toujours montré à l'écoute et très disponible tout au long de la réalisation de ce mémoire, ainsi pour l'inspiration, l'aide et le temps qu'il a bien voulu me consacrer et sans qui ce mémoire n'aurait jamais vu le jour.

Mes remerciements s'adressent à mon université ***ISAE CNAM Liban*** et le directeur de son département informatique également mon professeur, Monsieur ***Pascal Farés*** qui m'a accompagné toutes ces dernières années dans des formations qui m'ont été indispensables à mieux orienter ma vie professionnelle.

Toute ma reconnaissance à l'entreprise ***Omega Financial Solutions*** et à son directeur générale Monsieur ***Elie Geha*** qui a cru en moi et m'a ouvert les portes de sa société pour participer à des chantiers informatiques qui ont eu de grands impacts sur ma carrière dans le domaine de l'informatique financière.

J'exprime ma gratitude à tous les membres de la société Omega Financial Solutions, spécialement sa directrice régionale demoiselle Amal Keyrouf, aisni qu'à son directeur technique demoiselle Carine Maalouf qui a toujours été un soutient pour moi depuis ma première mission informatique datant de l'année 2004.

RESUME

Ce document décrit la démarche de restructuration de l'architecture du Logiciel de gestion de portefeuille de la société ***Omega Financial Solutions*** selon le concept orientée service.

Omega Financial Solutions devait faire face à un système d'information devenu complexe à maintenir et qui devait être simplifié pour pouvoir absorber les évolutions des métiers avec une meilleure réactivité et garder sa renommée sur le marché comme étant un des meilleurs logiciels de gestion de portefeuilles et de fond sur le marché Libanais et International. Une démarche de découplage du code a été lancée en 2008. Cette démarche dans laquelle j'avais un rôle d'analyse et de développement va constituer une excellente occasion pour revisiter l'ensemble du parc applicatif et le réduire en couches logiques qui peuvent être invoquées indépendamment l'une de l'autre. Ce chantier va permettre à la société de normaliser les méthodes de développement, ainsi que l'exposition des fonctionnalités en tant que services indépendants. Elle est aussi une opportunité de partager une vision commune du système d'information entre tous les acteurs concernés.

Mots Clés: restructuration, gestion de portefeuille, concept orientée service, système complexe, réactivité, découplage, couches logiques, normaliser, services indépendants.

SUMMARY

This document describes the process of restructuring the architecture of Omega PM the portfolio management product of ***Omega Financial Solutions*** as service-oriented concept.

Omega Financial Solutions faced an information system that has become complex to maintain and should be simplified to accommodate the changes in occupations with greater responsiveness and keep its reputation in the market as one of the best software to manage portfolios and funds on the Lebanese and international market. An approach of code decoupling was launched in 2008. This approach, in which I had an important role in analysis and development, was an excellent opportunity to revisit the entire fleet and reduce it to logic application layers that can be used independently of each other. This will enable the company to standardize development methods, and exposure features as independent services. It is also an opportunity to share a common vision of the information system among all stakeholders.

Keywords: restructuring, portfolio management product, service-oriented concept, complex, responsiveness, decoupling, application layers, standardize, independent services.

Sommaire

Sommaire 4
Table des figures 6
Table des tableaux 7
Présentation de l'entreprise 8
Introduction 10
Première Partie 13
Environnement et Etat de l'art 13

Chapitre I 14
Environnement de travail 14
I.1 - Structure de l'entreprise 14
I.2 – Fonctionnalités principales d'Omega PM 14
I.3 - Organigramme de l'entreprise et processus d`intégration continue 16
I.4 – Rôle du candidat et nomination 19
I.5 – Définition du problème et Étude des besoins 20
Chapitre II 26
Etat de l'art 26
II.1 Les Application distribuées 26
II.2 Le découplage dans les architectures 35
II.3-Introduction au design Patterns 41
II.4 Introduction à WPF 44
Chapitre III 50
Justification du choix du logiciel 50
III.1– Architecture et Composants du système 56
III .1.2 Les Blocs d'application Enterprise Library 59
III .1.2.1 Introduction aux blocs d`application 59
III .1.2.2 Le noyau des blocs d'applications 61
III .1.2.3 Dépendances entre les blocs d'application 63
III .1.2.6 Le bloc d'accès aux données 64
III .1.2.7 Le bloc de gestion des exceptions 67
III .1.2.8 Le bloc de mise en cache 67
III .1.2.9 Le bloc de piste d'audit(Logging) 67
III .1.2.10 Le bloc de sécurité 69
III .1.2.11 Le bloc de Validation des données 69
III .1.2.12 Outils divers 70
Deuxième Partie 74
Phases de la migration vers l'architecture orientée service 74

Chapitre IV 78
Compilation en .Net 78
IV.1 Phase de recherche 79
IV.1.1.2 Normes de codage et de nomenclatures 79
IV.2 Phase d'analyse du code 81
IV.3 Installation du CodeGear et gestion des composants 82
Chapitre V 89
Conception des composants de l'application 89
V.1 Installation et compilation des Enterprise Library(EntLib) 90
V.2 Conception des tiers de données. 92
V.3 Conception des tiers de gestion 98
V.4 Conception et développement de la politique de sécurité 100
V.5 Conception des composants d'audit 101
V.6 Conception des composants de gestion des exceptions 101
V.7 Conception des composants de validation des données 105
Chapitre VI 107
VI.1 Composants communs et types d'écrans 108
VI.1.1 Composants visuels communs 108
VI.1.2 Type d'écrans 111
Chapitre VII 112
Assistant de conception de l'interface utilisateur 112
VII.1 Phases de Conception de l'assistant 113
VII.2 Etape de développement de l'assistant. 118
VII.3 Le Processus de test des couches 118
VII.3.1 Pré requis 118
VII.3.2 Le plan de tests 119
Conclusion 122
Bibliographie 124
1- Ouvrage de référence 124
2- Articles 124
3- Les sites Internets 124
Glossaire 125
Annexe A 127
1) Aperçu global sur l'interface de développement Delphi 127
2) Bibliothèque de composants VCL 127
Annexe B 128
2 L'EDI de Delphi 129
3)La bibliothèque d'objets de Delphi 130
Annexe C 132
1) Description des différents fichiers d'un projet Delphi 132
Annexe D 134
Classe 134
1)Générateur de code SQL 137
a)Exemple de génération de la requête insert 137
b)Exemple de génération de la requête update 138
Annexe E 140
Annexe F 141
1) Table des matières du plan de Test 141
Annexe G 142
Annexe F 144
a)Exemple de syntaxe du protocole FIX 144

Table des figures

Figure 1 : Omega PM 15
Figure 2 : Organigramme de l'entreprise 16
Figure 3 : Phase de conduite de projet et chaine de responsabilité 18
Figure 4 : Architecture à deux niveaux 21
Figure 5 : Architecture Multicouches 24
Figure 6 :Communication avec les services web 31
Figure 7 : Interactions entre les couches d'une application à trois tiers 35
Figure 8 : Couche BLL 37
Figure 9: Diagramme de classes de la DAL Facture 38
Figure 10: POO 38
Figure 11: SOA 39
Figure 12: Fabrique abstraite 42
Figure 13 : Liaison des données dans WPF 46
Figure 14 : Delphi .Net avec le CLR 53
Figure 15: le modèle de déploiement du système 58
Figure 16: Outil de Configuration des Applications Blocks. 63
Figure 17: La dépendance entre les blocs 64
Figure 18: Organisation du Data Access Application Block 65
Figure 19: Classe publique Logger. 68
Figure 20:Composant QuickReport recompilé 83
Figure 21:Listes des composant .Net installés 84
Figure 22: fenêtre Options du projet 86
Figure 23 :Ecran de gestion de la connexion a la base de donnees Omega 91
Figure 24: Structure des tables de configuration d'accès aux données. 94
Figure 25 : Propagation des exceptions 102
Figure 26:Exemple d'un écran suite à la phase de validation 106
Figure 27:Barre d'outils Omega 110
Figure 28 : exemple d'écran simple, gestion des types de fonds. 111
Figure 29 :IDE de delphi 129
Figure 30 : Hiérarchie des objets Delphi 130
Figure 31 : Table des matières du plan de Test 141
Figure 32:Connaissance et interprétation du concept de SOA 142
Figure 33: Typologie d'entreprises concernées par SOA 142
Figure 34 Pénétration de SOA dans les entreprises 142
Figure 35 : Les freins à l'essor de SOA 142
Figure 36 : Les bénéfices attendus de SOA 142
Figure 37 : Typologie des projets SOA 142
Figure 38 : Les décisionnaires en matière de SOA 143
Figure 39:Les compétences recherchées 143
Figure 40:Appréciation de l'offre SOA actuelle 143
Figure 41:Lacunes des outils SOA 143
Figure 42:Intégration des applications existantes 143
Figure 43: Choix de la plateforme de développement 143

Table des tableaux

Table 1 : avantages et les inconvénients des services Web 28
Table 2: Type et objectifs des Patterns de création..... 42
Table 3: Type de design patterns 43
Table 4 : déclencheur de mise a jour des données(*UpdateSourceTrigger)* 47
Table 5 : caractéristiques de la plateforme Microsoft.NET 52
Table 6 : Exemple de choix du compilateur-Les directives conditionnelles 85
Table 7 :Types de tests appliqués 121

Présentation de l'entreprise

Omega Financial Solutions a été fondée en 1997, elle représente actuellement un acteur majeur sur le marché de l'informatique financière. Elle s'est imposée sur le marché comme l'un des premiers fournisseurs d'un progiciel destiné à la gestion de portefeuilles et des fonds collectifs.

Omega Financial Solutions possède deux agences: Une agence ayant Paris comme siège social (72, Rue du Faubourg, Saint Honore 75008 Paris - France), et l'autre au Liban (201 Sami El Solh Avenue Badaro - Beirut - Lebanon) sous le nom de ***Omega Financial Solutions Offshore***.

Au fil des années, ***Omega Financial Solutions*** fidélisait un nombre important d'établissements financiers en Europe, au Liban, au Maroc et en Arabie Saoudite, qui exercent notamment des exploitations dans les industries suivantes:

- Gestion d'actifs
- Banques d'investissement
- Gestion de fonds commun de placement
- Assurance
- Banques
- Compagnies d'assurance mutuelles
- Retraite et pension
- Fonds de pension
- Courtiers

Omega Financial Solutions offre un large éventail de services qui permet de répondre aux attentes de la clientèle. La liste exhaustive comprend les services suivants:

- Gestion de projet
- Intégration du système
- La personnalisation du système
- Migration de données
- Consultance et formation
- Service Clients

OmegaPM, le produit majeur de la société, est un logiciel de finance adopté par les plus importants investisseurs institutionnels et gestionnaires de patrimoine du monde.

OmegaPM permet l'exécution du progiciel intégré *front to back-office*, la gestion de portefeuilles et de fonds communs de placement, le moteur de compliance, et la gestion des frais et bien d'autres fonctions.

Les services personnalisés sont conçus et développés exclusivement pour répondre aux besoins et exigences spécifiques des clients, tout en respectant un cycle méthodique qui permet de concevoir des programmes à temps et dans le budget. Cette approche comprend:

- Une étude des besoins des clients.
- Organisation du service recommandé.
- Développement du service.
- Test et Intégration sur le réseau.
- Support, suivi, et entretien.

Le cycle précité permet au personnel de clarifier des problèmes, développer des solutions professionnelles basées sur les besoins de clients, et proposer des méthodes alternatives en assurant des disciplines prouvées et des directives pour la gestion des risques.

Pour répondre à la plupart des exigences des départements financiers dans le monde,omega financial solutions ne cesse d'investir dans son logiciel pour satisfaire les clients actuels ci dessous, ainsi que les clients potentiels:

France	**Suisse**	**Maroc**
Rothschild Financial Services Gestion	Diapason	BMCI Gestion
Aviva Gestion d'Actifs	Aforge Switzerland	
Credit Suisse Asset Management		
Banque de France (BDF) Gestion		
Louvre Gestion (HSBC)		
Prado Epargne		
Schelcher Prince Gestion	**Arabie Saoudite**	**Liban**
Fongepar Gf		
Neuflize Gestion	SABB, the Saudi British Bank	BNPI-BNP Paribas Group
Macif	Rana Investement Company	Banque Audi
Eurazeo	HSBC Saudi Arabia Limited	BankMEd
Groupama Asset Management		Med Securities
Aforge		Credit Libanais
Federal Finances		Lebanese Canadian Bank
Federis Ga		Bank of Beirut

Chypre		**Egypte**
Phi Trust	BNP Paribas Cyprus	Bank Audi Egypt
Shanti Gestion		

Introduction

Au cours des quarante dernières années, les systèmes informatiques se sont développés de manière exponentielle, sous forme d'architectures logicielles de plus en plus complexes et difficiles à gérer pour les entreprises.

Les architectures traditionnelles ont atteint leurs limites en termes de capacité, alors que les besoins traditionnels des organisations informatiques demeurent. Les départements informatiques doivent toujours répondre rapidement aux nouvelles exigences, tout en réduisant constamment le coût informatique de l'entreprise en absorbant et en intégrant de manière transparente de nouveaux partenaires et clients commerciaux.

Le secteur d'activité des logiciels a connu de nombreuses architectures conçues pour permettre un traitement entièrement distribué, des langages de programmation destinés à n'importe quelle plate-forme et une myriade de produits de connectivité conçus pour permettre une intégration plus rapide et plus efficace des applications, tout en réduisant considérablement les temps de mise en œuvre. Toutefois, une solution globale ne semble être encore qu'une utopie.

Aujourd'hui plus que jamais, l'avantage concurrentiel des entreprises dépend de leur rapidité à satisfaire les demandes du marché et à saisir de nouvelles opportunités.

La société ***Omega Financial Solutions*** s'est rendu compte de la concurrence sur le marché financier où le système informatique doit être aussi flexible que son processus métier. Toutes les technologies utilisées sont mises au service de son Logiciel de gestion de portefeuilles ***OmegaPM*** qui avec le facteur temps et le changement fréquent des besoins financiers, risque de faire face à des problèmes de **flexibilité** et d'**adaptabilité** sachant qu'il a été conçu pour un seul modèle métier.

La première version d'***OmegaPM*** a été écrite en Pascal, puis en delphi2 et delphi4 pour finir avec delphi6. Toutes ces versions ont eu une valeur ajoutée sur le logiciel mais l'architecture est restée la même, une architecture client serveur.

Avec un seul modèle métier, ***Omega Financial Solutions*** réalise la difficulté d'associer les ressources existantes à de nouveaux objectifs et par conséquence des nouvelles opportunités risquent d'être totalement perdues par manque d'adaptabilité.

L'entreprise a eu besoin d'une solution lui permettant d'isoler les processus et fonctions de base et de les rendre aisément accessibles, réutilisables et combinables afin

de créer de nouvelles fonctionnalités sans reprogrammation coûteuses en temps et en ressources.

Face à ce constat, ***Omega Financial Solutions*** devait mettre en place une infrastructure et une architecture logicielle qui répond aux besoins du marché. D'où la nécessité de la ré-modélisation de l'architecture de l'application ***OmegaPM***. . L'architecture logicielle qui va être adoptée est une **architecture distribuée orientée services**. La mise en place de cette architecture logicielle va assurer l'interopérabilité avec d'autres applications et systèmes.

En effet, ***Omega Financial Solution*** a commencé à chercher la solution adéquate pour pallier aux différents problèmes de l'ancienne architecture. Tout d'abord, les besoins opérationnels et surtout techniques ont été définis dans le cahier de charges rédigé grâce à l'assistance des consultants qui sont en contact régulier avec les clients.

Suite à l'étude des pré-requis du logiciel existant et des compétences du personnel ainsi que l`estimation du temps de la migration, le choix de la plateforme de travail est tombé sur la plateforme **Microsoft.Net** qui assure en plus de la plateforme **win32** des librairies supplémentaires qui permettent un gain dans le temps d'analyse, de développement et de déploiement.

La migration vers l'architecture orientée service a fallu la construction d'une équipe de recherche en développement qui a eu la tâche d'étudier les modules du logiciel et mettre l'analyse adéquate pour une migration souple et rapide.

Ayant une licence en informatique de gestion et possédant une expérience du monde d'analyse et du développement des applications de gestion (Chapitre I, Section I.4), j'ai été nominé pour être membre dans l'équipe de recherches et de développement en tant qu'analyste programmeur.

Le présent document décrit dans ces deux parties les différentes étapes de la migration d'***OmegaPM*** vers une architecture orientée service ainsi que ma contribution au sein de cette migration.

La première partie est décomposée en trois chapitres:

Le premier chapitre, porte sur la structure de l'entreprise et les fonctionnalités principales d***'Omega PM***, les besoins et les objectifs de l'entreprise, son choix de la plateforme de développement et la création de l'équipe de recherche et de développement (R&D) .NET

Dans le deuxième chapitre, intitulé l'état de l'art, j'aborde le besoin de la migration du logiciel vers une nouvelle structure, une nouvelle plateforme et les différents points qui ont mené Omega Financial Solutions à prendre une telle décision. Cette migration comptera en premier lieu sur la programmation orientée objet et les

composants qui en résultent et le découpage du code en couches, ainsi que sur les services web.

Dans le troisième chapitre je justifie le choix de la technologie et des méthodes utilisées et surtout les composants qui assureront la migration d***'Omega PM***.

Dans la deuxième partie, je détaillerai les phases de l'implémentation de l'architecture orientée service dans notre logiciel en question. Cette partie s'étend sur quatre chapitres dans lesquelles sont détaillées les différentes étapes de conception:

Le quatrième chapitre porte sur la phase de la compilation du code du logiciel sous la plateforme .Net et l'influence d'une telle décision sur l'implémentation de la nouvelle architecture.

La conception des composants de l'application fera l'objet du cinquième chapitre au sein duquel je décris le choix des composants, la conception de l'architecture globale et détaillée basée sur SOA, ainsi que les différentes modifications nécessaires pour la mise en œuvre de cette dernière.

Dans le sixième chapitre j'aborde les composants communs et les types d'écrans utilisés dans l'application et leur équivalent après la migration.

Le septième chapitre est un chapitre de présentation du résultat final de la migration, car il apporte le procédé de travail adopté pour la création d'un assistant qui permettra la migration des anciennes interfaces graphiques ou la création d'autres nouvelles comme il le permettra pour les services Web.

Pour conclure, je récapitule les différents points abordés, ainsi que leur impact sur les projets futur concernant ***OmegaPM*** et leur contribution dans leur mise en œuvre tout en profitant du facteur temps et coût.

Première Partie

Environnement et Etat de l'art

Chapitre I

Environnement de travail

I.1 - Structure de l'entreprise

I.2 – Fonctionnalités principales d'Omega PM

Omega PM développé par ***Omega Financial Solutions***, est un logiciel de gestion de portefeuilles sous forme institutionnelle:

- OPCVM de tout type.
- Portefeuilles de placement des banques.
- Des compagnies d'assurance mutuelles.
- Institutions de retraite et de prévoyance.
- Fonds de pension.

Omega PM assure les fonctions suivantes:

- Tenue et valorisation des portefeuilles et des positions (en PRMP ou FIFO assurances).
- Gestion des transactions (carnet d'ordres blocs / Panier ou individuels, simulés…) et leur « routage »
- Aide à la gestion: allocation sous contraintes, simulations, rebalancement, analyses…
- Gestion des contraintes réglementaires, statutaires, conventionnelles, internes.

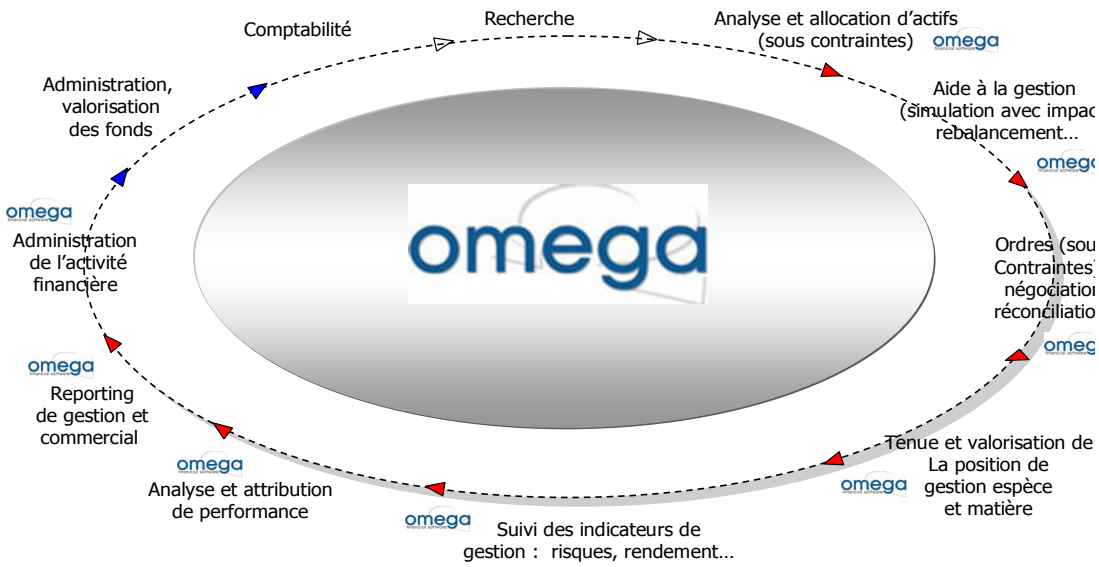

Figure 1 : Omega PM

- Gestion des indicateurs de risques/rendements/performances, VaR et gestion des indicateurs techniques « assurance » (Res. de Capi., Amortissement de S/D, IFRS)
- Gestion de la trésorerie des fonds: prévisions, assistance au réinvestissement…
- Analyse, attribution et contribution à la performance
- Réconciliations: back office comptable et des dépositaires, brokers/contreparties…
- Rapports de gestion.
- Gestion du passif des fonds ou des portefeuilles des clients (Souscription/Rachat, Placements, Apport/Retrait.)
- Gestion administrative et commerciale (gestion des frais des réseaux, rétrocessions.)
- Gestion des frais de gestion, des frais de l'activité et des courtages.
- Fonctions diverses: Piste d'audit, personnalisation de l'application par profil.

La couverture fonctionnelle d'***Omega PM*** permet une gestion complète des opérations, des risques et du suivi administratif. Sa modularité lui permet de s'intégrer aisément dans tous les types d'organisation.

I.3 - Organigramme de l'entreprise et processus d`intégration continue

I.3.1 Organigramme

La figure suivante montre l'organigramme général de l'entreprise ***Omega Financial Solution Offshore***

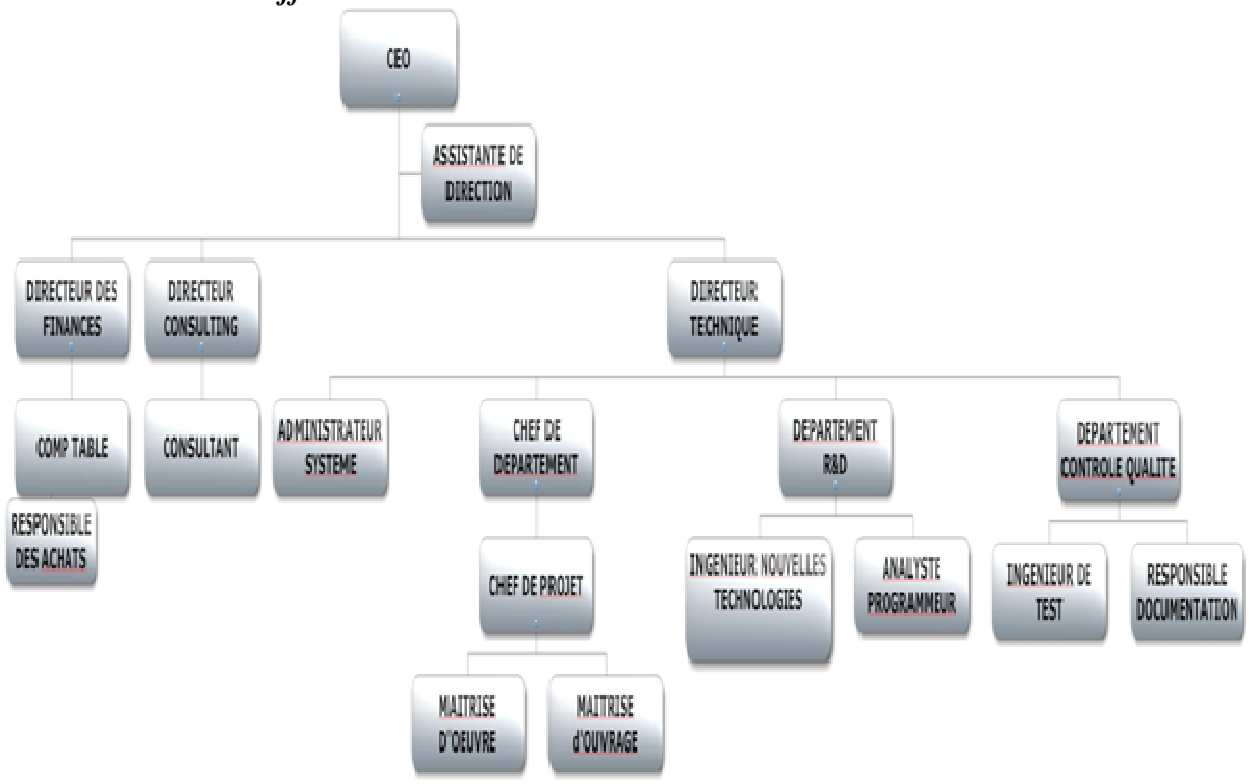

Figure 2 : Organigramme de l'entreprise

Chaque rectangle de cet organigramme représente un rôle qui encapsule plusieurs tâches à accomplir. Une même personne peut se trouver remplir plusieurs rôles.

I.3.2 Processus d`Intégration Continue

Le développement d`applications dans ***Omega Financial Solutions*** suit le **processus d`intégration continue** qui vise à industrialiser les activités liées à la mise en place de nouvelles solutions techniques.

Plus concrètement, notre projet possède une dizaine de développeurs et la taille de l'application implique un temps considérable pour accomplir une compilation sachant que chaque client a sa propre version. De plus, des tests unitaires sont couplés à ce processus, donc la mise en place d'un serveur d'intégration pour assurer la qualité d'exécution du projet.

En particulier les objectifs du **processus d`intégration continue** sont les suivants:

- L`industrialisation du cycle de vie de l`application
- L`automatisation de tâches manuelles récurrentes liées aux déploiements sur les différents environnements de développement, recette et intégration.
- La mutualisation des outils et des méthodes de développement.
- L`amélioration de la qualité de la livraison.
- Une réactivité accrue en phase de recette

I.3.2.1 Principes fondamentaux

Le processus d`intégration démarre à la fin de la phase d`études de solution et se termine à la fin de la recette métier.

Les entrants du processus sont réalisés par les activités suivantes:

- Définition de l`architecture applicative.
- Définition de l`architecture technique, en particulier les technologies et les environnements.

Pendant la phase de spécification, le processus démarre par la mise en place du projet dans l`outil de gestion des versions, en particulier, cela comporte les activités suivantes:

- Réunion de Nommage
- Découpage en bloc d`applications.

Le projet peut être découpé de façon basique de la manière suivante :

- **Phase préparatoire** : Cette phase permet de prendre conscience du projet, puis d'étudier l'objet du projet pour s'assurer que sa mise en œuvre est pertinente et qu'il entre dans la stratégie de l'entreprise. Cette phase, généralement qualifiée d'Avant-projet, doit se conclure par la mise au point de documents formalisant le projet et indiquant les conditions organisationnelles de déroulement du projet.

- **Phase de réalisation** : Il s'agit de la phase opérationnelle de création de l'ouvrage. Elle est menée par la maîtrise d'œuvre, en relation avec la maîtrise d'ouvrage. Cette phase commence par la réception du cahier des charges et se clôture par la livraison de l'ouvrage.

- **Phase de fin de projet** : il s'agit de la mise en production de l'ouvrage, c'est-à-dire s'assurer que l'ouvrage est conforme aux attentes des utilisateurs et faire en sorte que son " installation " et son utilisation se déroule correctement.

Le schéma ci-dessous donne un aperçu sur les différentes phases de conduite du projet au sein de la société.

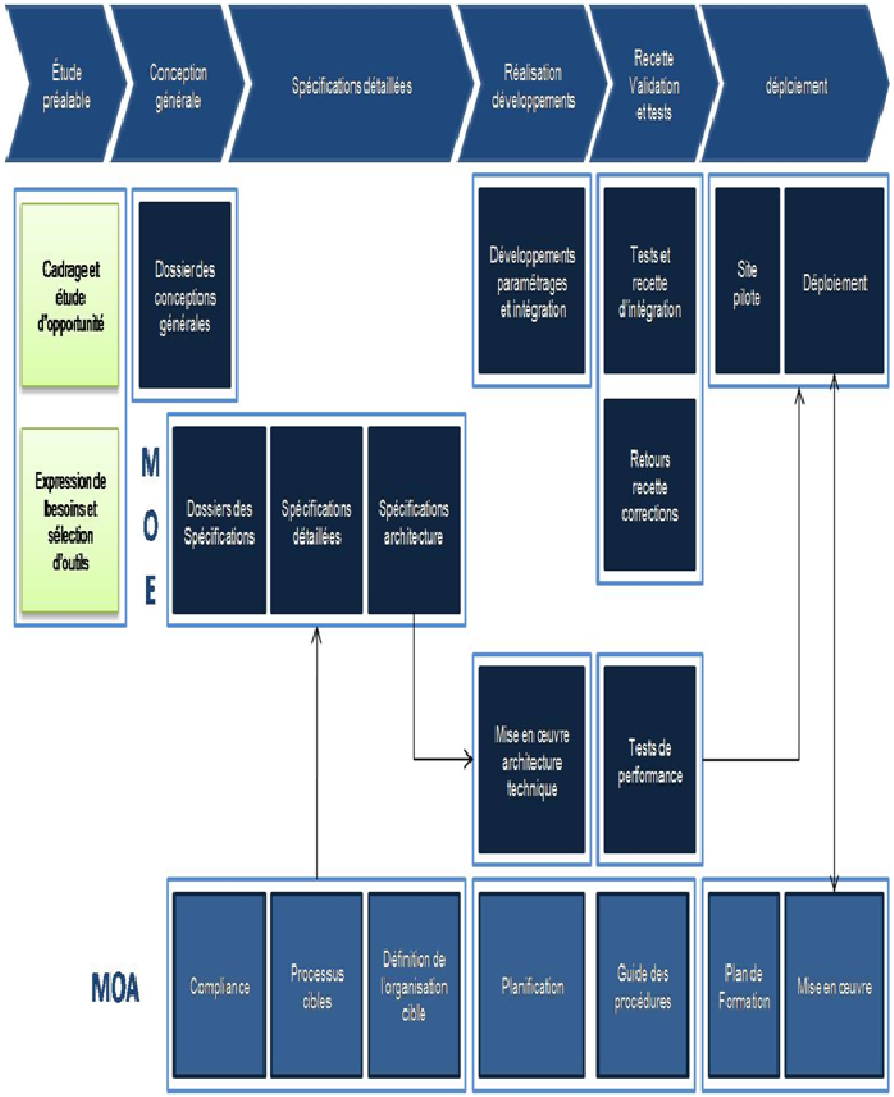

Figure 3 : Phase de conduite de projet et chaine de responsabilité

I.4 – Rôle du candidat et nomination

En juillet 2006, ayant une licence en informatique de gestion et environ 3 ans d'expérience dans le domaine de l'analyse et de développement de logiciel et des services Web, j'ai été recruté en tant qu'analyste programmeur au sein d'Omega ***Financial Solutions***.

J'ai gagné une partie de mon expérience au sein d'*Internet Facilites Group* qui est une entreprise internationale située au Liban et qui développe des solutions pour le secteur privé ainsi que public. J'ai travaillé la première année en tant que développeur au sein d'une équipe sur un projet de gestion des fraudes pour le ministère de l'économie et du commerce. Cette application qui consistait à contrôler les sociétés du marché Libanais, a nécessité un an de travail et a été réalisée sous .Net Framework et Microsoft SQL Server 2000.

Au bout d'un an de développement, je commençais à faire de l'analyse en parallèle au développement d'applications multimédia dédiées aux écoles et aux centres éducatifs en général.

D'autres parts, les différentes tâches que j'ai accomplies au sein de la société ***Omega Financial Solutions*** se résument par ce qui suit :

- Faire de l'analyse et de développement de modules sous Delphi en relation avec la communication TCP/IP.
- Former les développeurs sur le modèle multicouches et l'environnement .NET.
- Voyager et communiquer avec les consultants en France et présenter les nouveaux prototypes qui seront intégrés dans ***OmegaPM***.
- Intervention technique auprès des clients et la documentation des scénarios de déploiement pour chaque composant mis en place.

Tout ce travail m'a permis de gagner la confiance de la société pour que je sois désigné analyste programmeur dans l'équipe de recherche et développement responsable de la migration du Logiciel.

I.5 – Définition du problème et Étude des besoins

Depuis des années, le logiciel ***Omega PM*** s'est réservé une place parmi les meilleurs logiciels financiers au monde et il la garde toujours grâce à un travail laborieux et une expertise dans le domaine. Comme tout autre entreprise, ***Omega Financial Solutions*** fait partie d'un contexte d'édition de logiciel devant atteindre un certain nombre d'objectifs à court et long terme relevant du domaine :

- **commercial**
- **technique**
- **fonctionnel**
- **financier**

Au niveau **commercial**, le logiciel doit répondre à des objectifs de marchés et donc à des contraintes de temps. En effet, pour répondre à un besoin commercial, le logiciel doit pouvoir se lancer publiquement durant une fenêtre de temps bien précise.

Au niveau **technique**, il faut avoir conscience que l'image de l'entreprise est en jeu, donc la fiabilité, la qualité, et la pérennité dans le temps du logiciel est importante.

Au niveau **fonctionnel**, cela correspond tout simplement à l'aspect concurrentiel du monde de l'entreprise qui ne cesse d'augmenter. Il faut donc éviter les situations ou la concurrence propose des fonctionnalités qui ne sont pas encore disponibles dans la solution à réaliser.

Au niveau **financier**, il faut être capable de maîtriser le niveau de qualification exigé des ressources et le temps de développement.

Ces objectifs imposés par le contexte de l'entreprise représentent une équation pas forcément simple à équilibrer. C'est justement à cause de ceci que l'architecture du logiciel se doit prendre en compte dès le départ ces différentes contraintes, car plus la problématique est gérée tôt dans le cycle de réalisation, plus nous avons de chance de nous approcher d'une situation optimale.

A l'issue de la concurrence sur le marché, de la diminution des marges, en vue de fidéliser un nombre plus grand d'institutions financières, et devenir une entreprise *OnDemand,* ***Omega PM*** a été l'objet de plusieurs études et réunions techniques qui ont révélé la difficulté d'atteindre les objectifs planifiés avec l'architecture actuelle du logiciel.

Omega Financial Solutions devait faire face à un système d'information devenu complexe à cause de son architecture qui pose des contraintes sur la réalisation des objectifs tracés à court et long terme dans les différents domaines cités ci-dessus.

L'architecture ***Omega PM*** est une architecture client-serveur à deux niveaux. L'application ***Omega PM*** est une application cliente utilisant la communication ***SDAC*** (SQL Server Data Access Components) directe avec un serveur de base de données ***Microsoft SQL Server.***

Il n'y a pas de couches intermédiaires entre l'application cliente et la base de données.

Figure 4 : Architecture à deux niveaux.

L'architecture client-serveur possède toutefois des inconvénients qui nous ont poussés à utiliser d'autres technologies. Les principaux inconvénients sont :

- **La sécurité** : La difficulté à gérer correctement les questions de sécurité et le coût du déploiement. La sécurité d'un système en architecture client-serveur est gérée au niveau du SGBDR. Celui-ci contrôle l'accès aux données en attribuant des autorisations d'accès aux différents utilisateurs du système. Le problème vient du fait que cette attribution de droit ne peut pas tenir compte des spécificités du logiciel réalisé.

- **Les durées et coûts de déploiement** : Ce problème est souvent considéré comme beaucoup plus important par les entreprises en général, car il est beaucoup plus visible. Il s'agit des durées et coûts de déploiement des logiciels. En effet un logiciel classique, développé en architecture client-serveur, nécessite une installation et une éventuelle configuration sur chaque poste utilisateur. Le déplacement d'un technicien coûte déjà très cher. Mais ce qui reste le plus laborieux est la nécessité de mettre à jour régulièrement le logiciel. Dans une architecture client-serveur, chaque mise à jour du logiciel nécessite un nouveau déploiement accompagné de nombreux coûts.

- Les fonctions de l'application ne peuvent pas être réutilisées facilement.

- Ce type d'architecture rend difficile **l'intégration des applications** existantes sur différents plateformes.

- L'**extensibilité** est impossible puisque c'est difficile de déployer une partie de l'application sur plusieurs machines ou d'ajouter une machine dans le cas où la charge devient lourde.

- Toutes les règles métiers sont contenues dans le code frontal. En conséquence, s'il faut modifier une règle métier, **tous les clients doivent être mis à jour**.

Il faut noter que le logiciel commençait à avoir quelques lenteurs dans le cas de traitement de volume important de données ou dans la génération de certaines chartes graphiques qui sont devenues une nécessité pour les utilisateurs devenus exigeant sur le temps de réponse. Plusieurs développeurs ont assisté au fil de ces dernières années au développement du logiciel sans que les normes de programmations soient respectées de Tous.

L'objectif de ce projet est de trouver un moyen pour pallier à ces problèmes et permettre la présentation du fonctionnel en tant que services indépendants tout en adaptant l'existant avec le minimum de coût et un ROI(retour sur investissement) considérable à court et long terme.

Mettre en œuvre l'implémentation qui répond à ce besoin nécessite la réalisation de trois groupes de fonctions:

1-le stockage des données.

2-la logique applicative.

3-la présentation.

Ces trois parties communiquent entre elles mais sont indépendantes. La conception de chaque partie doit également être indépendante, toutefois la conception de la couche la plus basse est utilisée dans la couche d'au dessus. Ainsi la conception de la logique applicative se base sur le modèle de données, alors que la conception de la présentation dépend de la logique applicative.

Cette division des tâches entre les couches peut être poussée de sorte que la fonctionnalité de l'application soit exposée par un **service** avec lequel la communication se fait via un ensemble de messages. Un **service** ne possède pas une couche de présentation. Par conséquence, il peut être livré à n'importe quelle interface utilisateur sur n'importe quelle plateforme.

Le principe de cette communication ainsi que ses composants principaux seront évoqués dans la partie suivante, mais je me contente dans ce paragraphe de citer Les avantages de l'architecture orientée services:

- **Facilité de déploiement et d'accès** : Les composants peuvent exister sur n'importe quelle machine n' importe où dans le monde et accédés de la même manière. Cette distribution est facilitée par l'utilisation de la programmation orientée objet et plus particulièrement de ce qu'on appelle les composants. Un composant possède entre autre les caractéristiques d'être accessible à travers le réseau. Un composant peut ainsi être instancié puis utilisé au travers du réseau. Il est également possible de trouver un serveur permettant l'utilisation d'un composant, ce qui permet une forte évolutivité ainsi qu'une résistance aux pannes importantes (le service sera toujours disponible sur un serveur même si une machine tombe en panne)

- **Indépendance des plateformes :** Les services et les clients peuvent être de plateformes différentes tant qu'ils utilisent le même standard de communication.

- **Amélioration de la sécurité** : En ajoutant la couche interface de service, il est possible de fournir plus de sécurité. Les autres parties de l'application finale qui nécessite autres mesures de sécurité peuvent être déployés derrière le firewall.

- **La réactivité** : la composition des processus métiers consommant des services permet d'accélérer la mise en œuvre d'une solution pour répondre à un nouveau besoin métier.

- **L'évolutivité** : la possibilité de mettre en place des processus métiers évolutifs qui sont recomposables au gré des modifications des besoins de l'entreprise.

- **La flexibilité** : mise en œuvre de nouveaux processus métiers à partir des services qui peuvent devenir à leur tour de nouveaux services consommables par de futurs processus métiers.

- **L'amélioration de la performance** : L'architecture orientée service améliore la performance puisque la charge d'exécution est distribuée sur plusieurs niveaux et entre différents services.

La nouvelle architecture permettra et grâce à la centralisation des fonctions, à réduire le coût de développement tout en augmentant le niveau de sécurité assuré par les librairies contenue dans la nouvelle plateforme. Le concept des services web permet d'élargir le périmètre du cercle de communication, ce qui permet d'exposer les différents services aux abonnés qui remplissent les conditions de souscription.

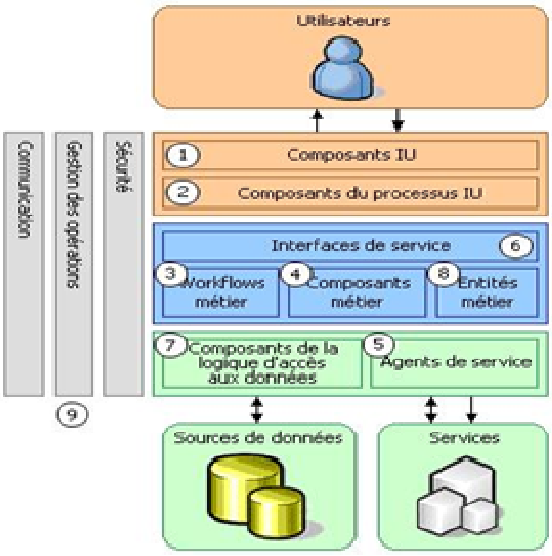

Figure 5 : Architecture Multicouches

Plusieurs facteurs devaient être pris en compte avant de se lancer dans la migration car la nouvelle architecture est faite pour durer, sachant que le retour sur investissement dans ces types de projets n'apparait qu'après un certain temps.

Des mesures ont été prises après l'étude détaillée des contraintes et de l'existant au :

- **Niveau du logiciel :**

  Le fournisseur de l'ancienne plateforme de développement **Delphi** a rendu facile le choix des nouveaux logiciels de développements qui incarneront la nouvelle architecture. Delphi lui même migre vers .Net ; il est devenu possible de créer des libraires et des paquets avec du code Delphi grâce à CodeGear2007 la nouvelle version.

  Microsoft,suite à **Visual Studio 2008** adopté par Omega lance une nouvelle version de l''**Enterprise Library** qui est une collection de blocs applicatifs écrits en **C#** qui représente un complément pour la plateforme .Net (Chapitre II),Après analyse de ces librairies au code source **gratuit**, il s'est avéré qu'elles peuvent être un point de départ sur lequel on peut se baser pour créer un Framework personnalisé basé sur .Net qui répondra aux besoins tracés.

  Windows Presentation Foundation (WPF), de son ancien nom de code **Avalon**, est le nouveau système d'interfaces graphiques adopté dont le but est de proposer une approche unifiée de l'interface utilisateur. [W4]

  Expression Blend sera l' outil d'édition de l'interface graphique selon les normes du **WPF** qui se réduit en un fichier contenant du texte pouvant être édité aussi dans Visual Studio(par le développeur)chose qui diminuera les risques d'erreur entre le responsable graphique et le développeur chose qui

posait un problème auparavant. L'importance d'**Expression Blend** est que le développeur ainsi que la personne responsable du graphisme des écrans peuvent travailler sur le même environnement en même temps.[W6]

- **Niveau du Personnel :**

La formation du personnel sur un nouveau concept de programmation est un besoin dans ce type de projet. Une vue globale du projet doit être communiquée aux différents acteurs qui n'auront pas tous besoin de changer leur langage de programmation sauf en cas de besoin et selon profil.

- **Niveau du Matériel :**

Le contrôle de tous les postes pour valider la **Configuration matérielle minimum requise** par poste qui varie chez les clients selon les servies voulus de Omega**:** Processeur à 2,2 GHz ou plus, 512 Mo de mémoire RAM ou plus, écran 1280x1024, disque dur 7 200 tours/m ou plus.

Chapitre II

Etat de l'art

II.1 Les Application distribuées

II.1.1 Définition

Une application distribuée est une application dont tous les éléments qui la composent (classes, persistance, logique métier) sont distants géographiquement et communiquent entre eux via des réseaux locaux d'entreprise ou par Internet en utilisant le protocole IP. Les composants des applications distribuées sont réutilisables et servent souvent à plus d'une application simultanément. Cependant, l'utilisateur final n'a pas conscience de la nature distribuée de l'applicatif car il utilise plusieurs interfaces dont le but souvent est de fédérer tous les éléments distants pour atteindre l'objectif de l'application tout en masquant tous les mécanismes d'accès.

II.1 .2 Technologies pour applications

Le principe même des applications distribuées n'est pas nouveau. Beaucoup de technologies et de protocoles, souvent incompatibles entre eux ont été mis en œuvre pour permettre aux composants d'une application distribuée, de communiquer entre eux. Parmi ces technologies, on peut citer :

- **COM** (Component Object Model) : Technologie de communication inter application propre à l'environnement Windows. Cette technologie ne permet pas de construire à proprement parler des applications distribuées mais sert d'interface pour faire communiquer des applications sur une même machine. (Les différents logiciels de la suite bureautique Microsoft Office par exemple).

- **DCOM** (Distributed Component Object Model) : version Distribuée de COM.

- **.Net Remoting** : Technologie de communication inter application incluse dans le Framework .Net 2.0, basée sur un modèle client/serveur permettant la communication et la transmission d'objets entre les applications.

- **RMI** : équivalent Java du .Net Remoting

- **Services Web** : technologie permettant de faire communiquer des composants entre eux, indépendamment de la plateforme sur laquelle ils sont hébergés, en utilisant un protocole de communication et un format de fichier unique et standardisé basé sur du XML. Contrairement aux technologies évoquées précédemment, grâce à la nature standardisée des échanges, il devient tout à fait possible de faire communiquer des modules hébergés sous Windows avec d'autres hébergés sous Linux ou Unix. C'est cette interopérabilité associée à une relative facilité de mise en œuvre, qui fait que les services web sont aujourd'hui une des technologies les plus utilisées, pour construire des applications distribuées.

II.1.3 La naissance de l'Architecture Orientée Service [SOA]

Pour comprendre l'avènement et en quoi consiste le SOA, il est nécessaire de bien identifier que le but de la programmation structurée, est d'écrire du code qui soit robuste et réutilisable. Au tout début, il n'existait que les langages purement procéduraux, la seule façon d'écrire du code réutilisable était d'écrire des fonctions et des procédures dans un fichier séparé du corps du programme, et de faire appel à ce fichier chaque fois que c'est nécessaire. Ensuite, est apparue la Programmation Orientée Objet (POO). Elle était innovante dans le sens ou le concept même d' « objet » permet l'encapsulation et donc de masquer la complexité des opérations. Les objets peuvent s'envoyer des messages, grâce aux appels de méthodes exposées par l'objet avec lequel ils souhaitent communiquer sans pour autant savoir comment le dit objet implémente le traitement qu'on lui demande d'exécuter.

Malgré le fait que des technologies comme DCOM, RMI ou .Net Remoting permettent de transporter les objets et donc de dépasser les frontières de la machine grâce au réseau, on s'est souvent heurté à des problèmes de compatibilité entre plateformes, d'où le besoin d'une standardisation et la mise en commun des protocoles (SOAP, XML, …). De là est née la notion d'architecture orientée services (SOA).

.Net Remoting et les services web sont les deux méthodes les plus communes pour écrire des applications **.Net** distribuées, sauf que chacune présente des avantages et des inconvénients et le choix de l'une ou de l'autre repose sur le besoin et le type d'investissement décidé.

Le tableau ci-dessous récapitule les avantages et les inconvénients des services Web face à ceux du .Net Remoting.

	. NET Remoting	**Services Web**
Protocole	TCP, HTTP	HTTP
Format de données	Binaire, SOAP	SOAP, Binaire avec un effort supplémentaire
Type de données	Riche	Limité en support de type de données
Interopérabilité entre les plateformes	Dépendance de la plateforme client. Applications .Net uniquement.	Independence de la Plateforme.
Fiabilité	A besoin de tournure s'il n'est pas hébergé sur IIS.	Grande fiabilité.
Développement	Complexe.	Facile
Facilite d'utilisation	Complexe.	Facile
Audience	Limitée. Réseau abonnés et ports spécifiques.	Pas de limite.
Transfert de données	Plus Rapide	Moins rapide
Déploiement et maintenance	Complexe. Il faut déployer des librairies, des fichiers de configuration, des objets d'abonnement au niveau du client, et ajouter des références pour récupérer les métadonnées de la librairie. Tout changement dans le code nécessite le redéploiement de ces fichiers sur le client.	Facile. Juste ajouter une référence Web ou bien invoquée dynamiquement(Chapitre III section 1.2.12 .1.2) Pas besoin de redéployer l'application
Sécurité	Plus sécurisé. Disponible sur des ports bien spécifiques.	Mois sécurisé si des précautions ne sont pas prises.
Évolutivité	Seulement à travers la méthode SingleCall, mais cela peut être couteux si plusieurs clients appellent l'objet sachant que pour chaque requête un nouvel objet est créé.	Grande marge d'évolutivité.
Dépendance applicative	Dépendant. A besoin d'un déploiement d'une application pour lancer le service.	Indépendant. Peut être déployé seul sur IIS.

Table 1 : avantages et les inconvénients des services Web

II.1.4 Exigences concernant une architecture orientée services

Au vu des problèmes débattus précédemment, il devrait être évident qu'il est important de développer une architecture qui satisfasse toutes les exigences. Ces exigences devraient inclure les éléments suivants :

- Exploitation du parc informatique existant :

Il s'agit de la condition la plus importante. Il est rare que des systèmes existants puissent être éliminés. Ils contiennent certainement des données de grande importance pour l'entreprise. D'un point de vue stratégique, l'objectif est de constituer une nouvelle architecture dont le rendement corresponde aux souhaits de l'entreprise. Mais d'un point de vue tactique, les systèmes existants doivent être intégrés de sorte qu'à terme, ils puissent être décomposés et remplacés par des projets gérables et incrémentiels.

- Prise en charge de tous les types d'intégrations requis :

Cela comprend l'interaction avec l'utilisateur (pour fournir un fonctionnement unique et interactif), la connectivité des applications (pour créer un modèle de communication qui sous-tend toute l'architecture), l'intégration des processus (pour organiser les applications et les services), l'intégration des informations (pour rassembler et déplacer les données de l'entreprise) et la possibilité d'effectuer de futures intégrations (pour créer et déployer de nouveaux services et applications).

- Possibilité de mises en œuvre incrémentielles et de migration du parc :

Si cette exigence est satisfaite, l'un des aspects essentiels du développement de l'architecture pourra être mis en place : la possibilité de produire un retour sur investissement différentiel.

D'innombrables projets d'intégration ont échoué en raison de leur complexité, de leur coût et de temps de mise en œuvre impossibles à réaliser.

- Constitution dans un cadre de composants standard :

Il faut inclure un environnement de développement créé autour d'un cadre de composants standard pour promouvoir une meilleure réutilisation des modules et des systèmes, permettre au parc informatique existant de migrer vers le cadre en question et permettre une mise en œuvre opportune des nouvelles technologies.

- Possibilité de mise en œuvre de nouveaux modèles informatiques :

Des exemples spécifiques incluent les nouveaux modèles client de portails, le **calcul distribué** et l'informatique à la demande.

II.1.5 Web services et standards ouverts

Les services web sont des composants applicatifs capables de communiquer entre eux à travers plusieurs réseaux par le biais de « standards ouverts » non propriétaires basés sur le XML. Les services web permettent à différents types d'applications de communiquer et d'opérer entres eux sans se soucier des plateformes et des langages de programmation utilisés.

Lors du déroulement d'un processus distribué, des ordinateurs s'envoient des messages entre eux pour récupérer des données ou invoquer des procédures. Traditionnellement, ces messages passaient par des solutions middleware propriétaires ou faites maison. Dans un système d'information hétérogène actuel comprenant différents systèmes d'exploitations et des applications programmées avec différents langages, il est souvent nécessaire pour les managers IT d'acheter ou développer une différente interface pour chaque paire de systèmes connectés – une situation coûteuse et complexe.

Un nombre de tentatives ont été faites pour résoudre le problème mais elles ont échoué à cause d'un manque d'adoption par les acteurs de l'industrie et la suprématie des standards propriétaires. Avec l'arrivée d'Internet et des capacités de description universelle du XML, les services web ont apparus.

Le fait que les acteurs de l'industrie se soient mis d'accord sur l'adoption de la technologie émergeante que représentent les services web a eu pour conséquence une révolution dans le monde de l'informatique.

II.1.5 .1 Standardisation et interopérabilité

Pour que les systèmes puissent librement évoluer tout en restant compatible (dans un cadre interopérable), ils doivent s'affranchir d'une marque ou d'un vendeur particulier. Ils doivent obéir à une *norme* clairement établie et comprise.

Les systèmes restent interopérables, tant qu'ils respectent les normes régissant leurs contenus et leurs interfaces. La standardisation a permis de s'entendre sur les :

- Formats de données: X(HT)ML, SVG, SMIL, …
- Protocoles de communication: HTTP, SOAP

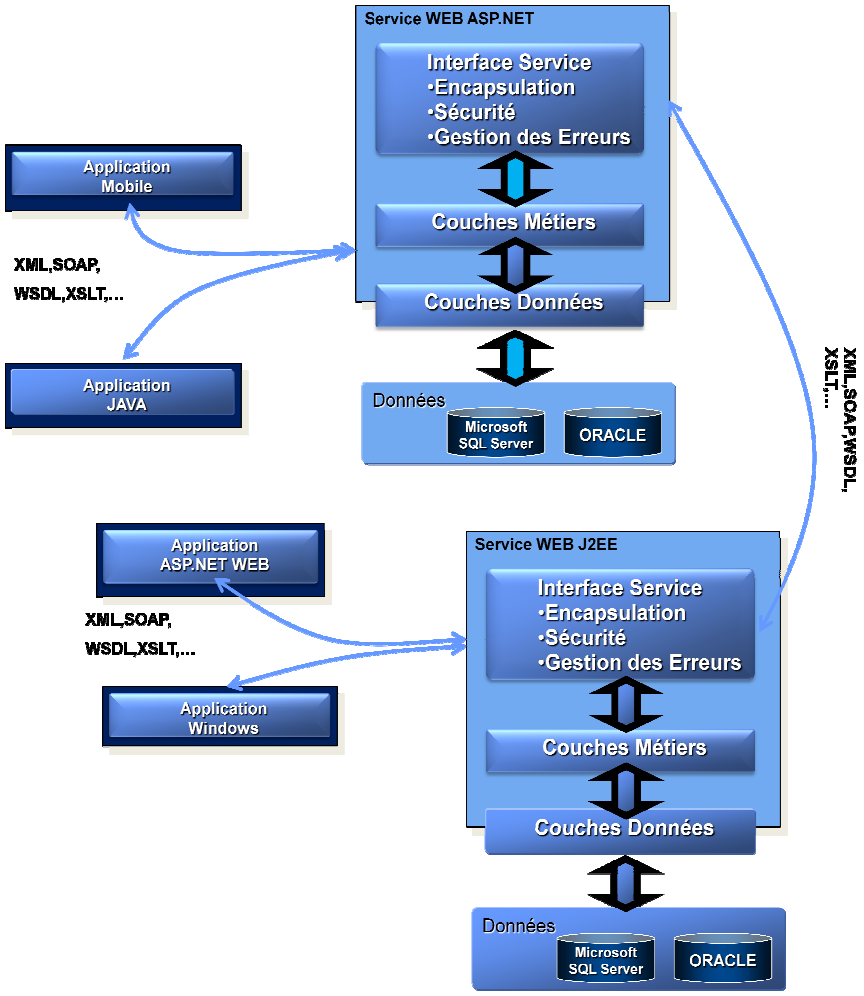

Figure 6 :Communication avec les services web

Il existe des organismes qui émettent des recommandations, à caractère non obligatoire, mais considérées par l'industrie comme standards. Ces standards peuvent être repris par les organismes (ISO, ANSI, AFNOR, SNIMA) nationaux ou internationaux de normalisation.

Parmi ces organismes de normalisations je cite les suivant :

• **IETF**, premier organisme de standardisation Internet, responsable du développement des protocoles et des réseaux basés IP.

• **OASIS,** est un organisme qui encourage le développement et l'adoption de standards dans les services Web et le *e-business*.

• **Consortium W3C**, définit les standards du Web, actuel et futur. L'organisme le plus important, car il touche à l'interaction entre l'utilisateur et Internet, i.e. le World Wide Web.

Ainsi grâce aux services web deux plateformes différentes peuvent communiquer sans aucun problème.

II.1.5.2 Les standards des services Web

L'un des avantages lié à l'utilisation des web services lors de déploiement d'environnements distribués est l'universalité de leurs interfaces.

Un web service dépend de trois standards basés pour bien fonctionner:

- Web Services Description Langage (WSDL) qui est le document qui décrit exactement ce que le web service fait et comment le solliciter.

- Universal Discovery, Description, and Integration (UDDI)

Les services web sont basés sur une architecture logicielle de requêtes et réponses. Un « client » d`un service web sollicite un service web avec une requête SOAP. En retour, le service web effectue l'opération demandée et répond à son tour avec un message SOAP.

Chaque service web a un client et un fournisseur. Grâce à leur nature, les services web peuvent avoir plusieurs clients se connectant à leurs interfaces sans se soucier de la plateforme et des langages de programmation.

Tant que le client envoi un message au format standard SOAP, il n'y a aucune différence au niveau du service web concernant les détails du client.

Pour se décrire au monde extérieur, chaque service web a un document (WSDL) qui fournit au client potentiel du service une explication concernant le fonctionnement et l'accès du service.

Le WSDL décrit comment créer une requête SOAP qui appellera le service web spécifique. Si un développeur veut créer un programme qui fait appel a un web service, il lui faudra juste le WSDL de celui ci.

L'*Universal Discovery Description and Integration* (UDDI) est un annuaire des services web disponibles dans un réseau particulier. Un client potentiel de service web peut consulter l'UDDI pour la disponibilité.

II.1.6 Atouts et faiblesses des SOA

Au-delà de la théorie et du mode de fonctionnement, quels sont donc les avantages que l'on peut identifier autour des SOA? Premièrement, leur mise en place nécessite une réflexion basée sur les notions d'objet. Chaque "service" doit s'apparenter à une fonctionnalité, dont le rôle et le périmètre de couverture doivent être clairement identifiés. La nécessité d'un effort conséquent en termes de modélisation se traduit inévitablement par une implémentation de meilleure qualité. Il en découle des facilités en termes de maintenance, de solidité applicative et de distribution. [W3]

Mais au-delà de ces aspects propres à l'objet et non spécifiques à l'orientation services, ce sont bien entendu les caractéristiques mises en avant par les services web qui ressurgissent. Ainsi, la capacité d'exécuter un traitement, sans avoir connaissance du langage ayant servi à son implémentation, est un atout indéniable. Tout comme la possibilité d'y accéder sans connaître à l'avance sa localisation.

À l'inverse, la nécessité d'effectuer un effort conséquent en termes de conception représente à ce jour l'un des principaux freins à la banalisation des SOA. De même que le besoin de réalisation d'une couche d'interface supplémentaire, pour laquelle les équipes de développement ne sont généralement pas habitués.

Comme on vient de le constater, les SOA seront d'autant plus utiles dans les contextes d'applications d'entreprise - voire de systèmes d'informations dans leur globalité, pour lesquels l'investissement consenti en conception et en interfaçage sera rentabilisé dans la phase d'évolution et de maintenance. Pour des projets départementaux ou locaux, cette approche présente moins d'intérêt, si ce n'est bien sûr pour accéder à des modules existants reposant sur les SOA.

Dans ce qui suit nous allons aborder le découplage entre les architectures car l’architecture orientée service vient pour compléter l’abstraction des couches, et par suite le principe de découpage en couches est une nécessité pour rendre une application flexible.

II.1.6 Résultats de sondage sur la SOA

Un sondage sur la SOA réalisé auprès de 85 entreprises françaises, a été publié par « Groupe Test » dans un article intitulé « SOA perception des entreprises françaises « [a4]

Il est préférable avant de se lancer dans un projet de migration de faire une recherche sur le marché et de comparer les avis des entreprises.

Le détail des statistiques du sondage qui a porté sur les questions communes suivantes peut être consulté dans l'Annexe G à la fin de ce mémoire :

II.1.6.1 Les enjeux de la SOA

- Connaissance et interprétation du concept de SOA(Annexe G- Figure 32)
- Typologie d'entreprises concernées par SOA(Annexe G- Figure 33)
- Pénétration de SOA dans les entreprises (Annexe G- Figure 34)
- Les freins à l'essor de SOA(Annexe G- Figure 35)
- Les bénéfices attendus de SOA(Annexe G- Figure 36)
- Typologie des projets SOA(Annexe G- Figure 37)
- Les décisionnaires en matière de SOA(Annexe G- Figure 38)
- Projets SOA : Les compétences recherchées (Annexe G- Figure 39)

II.1.6.2 L'outillage SOA

- Appréciation de l'offre SOA actuelle (Annexe G- Figure 40)
- Lacunes des outils SOA(Annexe G- Figure41)
- Intégration des applications existantes (Annexe G- Figure 42)
- Choix de la plateforme de développement (Annexe G- Figure 43)

II.2 Le découplage dans les architectures

L'avantage de la SOA est de produire des composants simples, modulaires et faiblement couplés, donc permettant de recomposer rapidement l'agencement applicatif des fonctionnalités qu'ils assurent.

Dans ce qui suit, nous allons aborder les différentes couches et le découpage qui va consolider le niveau d'abstraction de l'architecture orientée service bâtie sur d'autres architectures.

II.2.1 Architecture multicouches

Les architectures multicouches sont maintenant bien acceptées et mises en œuvre dans les applications.

Elles sont au cœur des préoccupations des architectes lorsqu'il s'agit d'aborder SOA et son impact sur les couches logiques et physiques des applications.

Notre premier schéma permettra un rapide rappel sur une architecture classique à trois tiers:

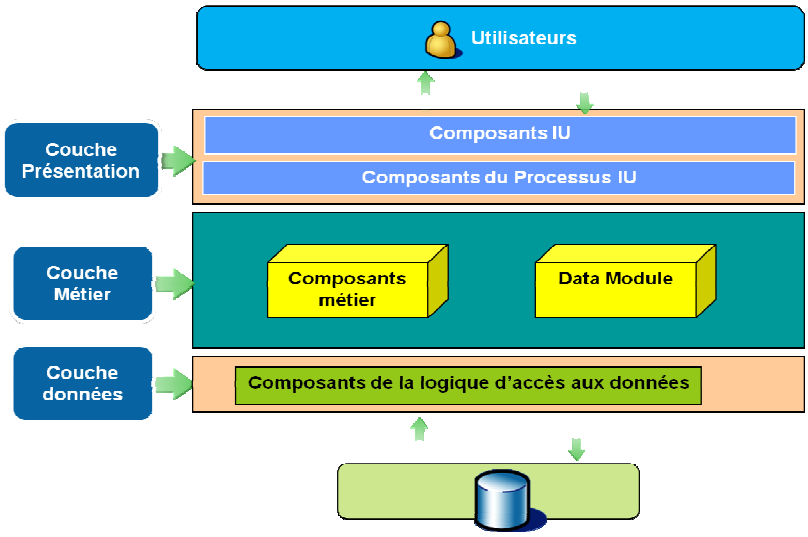

Figure 7 : Interactions entre les couches d'une application à trois tiers

Rappelons-nous des 3 couches principales :

- **Couche Présentation/UIL :**

Elle correspond à la partie de l'application visible et interactive avec les utilisateurs. On parle d'Interface Homme Machine. En informatique, elle peut être réalisée par une application graphique ou textuelle. Elle peut aussi être représentée en **HTML** pour être exploitée par un navigateur web ou XAML.

- **Couche Logique Métier / BLL (second niveau)**

Elle correspond à la partie fonctionnelle de l'application, celle qui implémente la « logique », et qui décrit les opérations que l'application opère sur les données en fonction des requêtes des utilisateurs, effectuées au travers de la couche présentation.

Les différentes règles de gestion et de contrôle du système sont mises en œuvre dans cette couche.

- **Couche Accès aux données /DAL(troisième niveau)**

Elle consiste en la partie gérant l'accès aux gisements de données du système. Ces données peuvent être propres au système, ou gérées par un autre système. La couche métier n'a pas à s'adapter à ces deux cas, ils sont transparents pour elle, et elle accède aux données de manière uniforme (**couplage faible**).

II.2.1.1 Découplage entre les couches et optimisation des échanges

Afin d'assurer de plus grandes facilités d'évolutivité et de réutilisation, il convient de découpler les couches les unes des autres.

La communication entre les couches dépendra de l'architecture physique de l'application. Lorsque les couches se trouvent sur des machines physiquement distinctes, des mécanismes tels que .Net Remoting ou les services web pourront être mis en œuvre. Lorsque les couches d'une application se trouvent toutes sur la même machine, il convient d'optimiser la performance en privilégiant des appels directs entre les couches.

En tout état de cause, les différentes couches devront échanger des données et non des références sur des objets, de manière à grouper les données et réduire les allers-retours entre les couches.

Deux règles essentielles s'appliquent alors :

- Les objets de données doivent être sérialisables sous la forme de flux textuels ou binaires.
- Les objets de données doivent être découplés de leurs sources de données. Un objet de données peut être stocké sous différents formats et dans différents conteneurs, il convient par conséquent de ne pas inclure directement dans les objets de données le code qui assure leur persistance dans ces formats et ces conteneurs.

II.2.1.1 .1 Découplage du format de stockage

Une couche d'accès aux données est chargée de dialoguer avec les bases de données, en exécutant sur celles-ci des requêtes de consultation ou de modification.

Le code contenu dans une couche d'accès aux données est intrinsèquement lié à la base de données particulière pour laquelle elle a été prévue. Par exemple, une couche d'accès aux données prévue pour Oracle ne sera pas compatible avec SQL Server .

Dans le schéma ci-dessous, la couche BLL peut utiliser une des trois couches DAL spécifiques à chacune des bases de données :

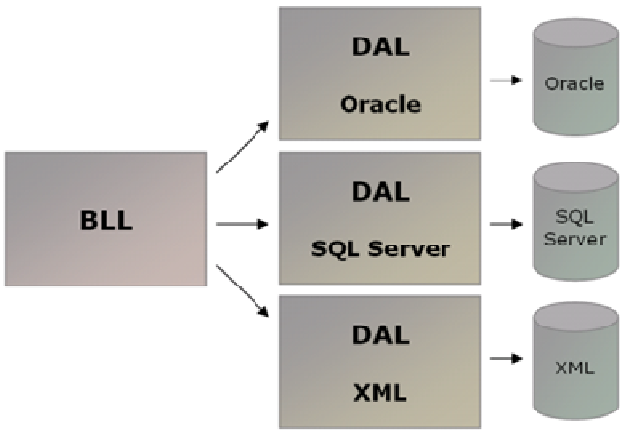

Figure 8 : Couche BLL

Afin de garantir l'indépendance de la couche BLL vis-à-vis d'une implémentation DAL spécifique, il convient de mettre en œuvre un mécanisme d'abstraction. Le mécanisme adéquat est décrit par le modèle dit de la *Fabrication*.

Le design pattern Fabrication (Chapitre II section 3.3.1) fournit une interface pour créer des instances d'objets similaires sans spécifier les classes de leur implémentation concrète.

Le modèle ci-dessous permet par exemple à la couche BLL qui a besoin de manipuler des objets *Facture* d'interagir avec la couche DAL associée, sans que la couche BLL ne

soit conçue spécifiquement pour une implémentation particulière. Voici comment serait modélisée la DAL Facture par exemple :

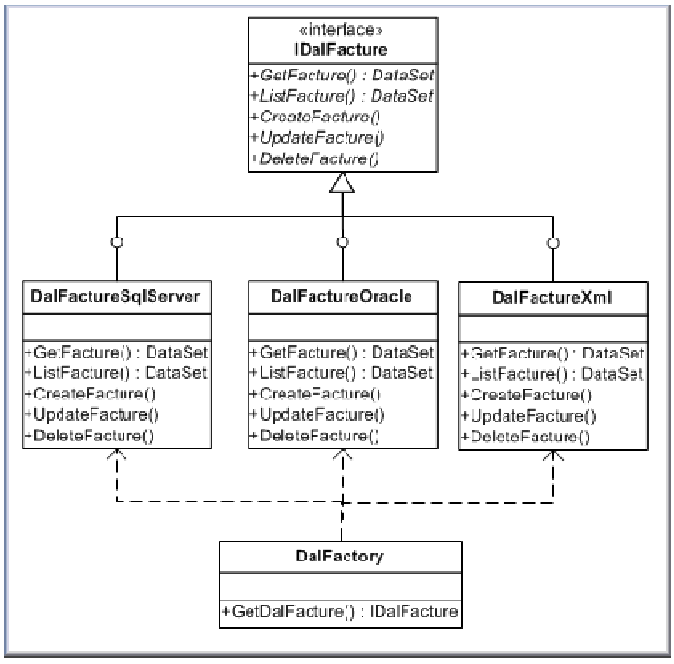

Figure 9: Diagramme de classes de la DAL Facture

La couche BLL aurait alors à utiliser la classe *DalFactory* pour obtenir une DAL spécifique, pour ensuite manipuler cette DAL à travers l'interface *IDalFacture*.

II.2.1.3 Comparaison du modèle orienté objests (POO)et orienté service (SOA)

Il va de soit que c'est l'approche orientée services qui nous intéresse ici, mais il est bon de savoir où se situent les différences. C'est une question récurrente. Voyons donc brièvement ces deux approches :

II.2.1.3 .1 Modèle orienté objets (POO)

Voici une architecture à trois couches classique avec un modèle objet :

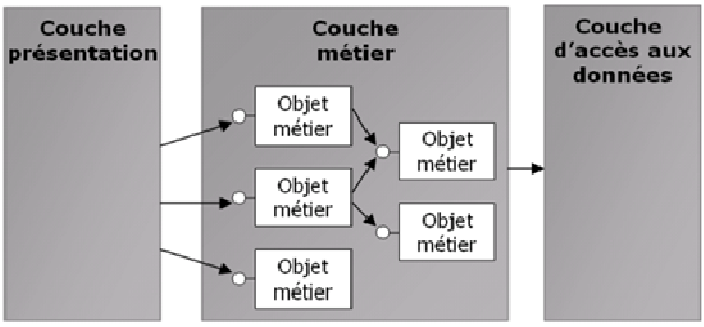

Figure 10: POO

On remarque tout de suite le nombre de liens entre la couche présentation et les objets métiers.

Le code client doit jongler directement avec le modèle objet de la couche métier, ce qui a pour conséquence de lier celle-ci très fortement à un modèle spécifique et requiert un nombre d'appels important entre les deux couches. La multiplication des appels entre les couches pose problème lors de la mise à disposition à distance des objets métiers, et que le nombre d'objets à manipuler réduit l'indépendance entre les couches et complexifie la prise en main de la couche métier.

II.2.1.3 .2 Modèle orienté services (SOA)

Voici une architecture orientée service qui reposerait sur les mêmes objets métiers :

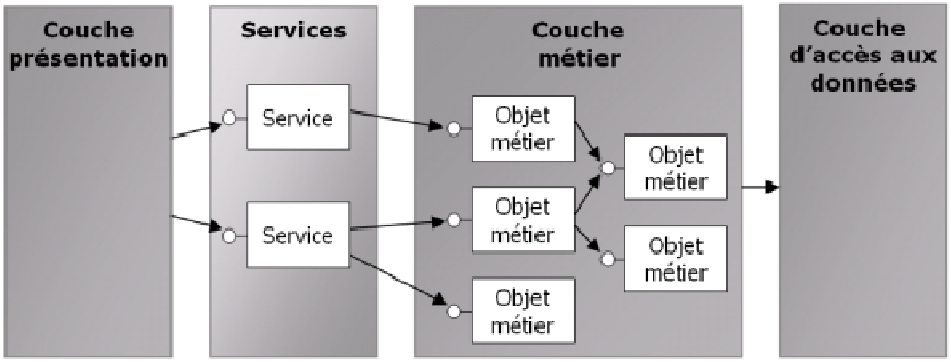

Figure 11: SOA

On remarque dans ce modèle l'introduction d'un niveau d'indirection supplémentaire sous la forme de services. La couche présentation ne manipule plus directement les objets métiers, mais passe par des services.

Les objets métiers se trouvent dans des bibliothèques de classes directement chargées dans le même processus que les services, le coût des appels aux objets métiers est alors très faible.

Les services agissent comme des boîtes noires faisant abstraction de la complexité du modèle objet, présentant un ensemble de fonctionnalités restreintes et permettant de réduire les échanges entre les couches.

Comme on peut le voir ci-dessus, l'architecture "tout objet" a ses limites. Contrairement à ce que l'on pourrait attendre, ces limites incluent jusqu'à la réutilisabilité. Dans le modèle tout objet, les couches sont intimement liées entre elles. En découplant les modules applicatifs en réduisant les liens entre eux par la suppression des appels directs entre objets de couches différentes, il devient plus facile de réutiliser ces modules.

Dans le modèle orienté objets, les couches sont également couplées temporellement, durant toute la durée de vie des objets. On peut considérer qu'un objet retourné par une couche inférieure est vivant tant qu'il est utilisé par la couche supérieure cliente. Cela ne doit pas être le cas dans une architecture orientée services, où les appels entre modules doivent supporter les modes asynchrone et/ou déconnecté.

Pour autant, l'objet ne perd pas sa valeur. Mais c'est à l'intérieur de chaque couche de responsabilité que seront mis à profit la programmation orientée objets et les design patterns.

Opposer POO et SOA, c'est un peu comme dire que la POO sera éclipsé par l'AOP (la programmation par aspects).D'ailleurs, on pourrait même envisager la SOA comme un retour vers la programmation orientée objets pure.

Il est important de se pencher sur des méthodes d'abstraction comme celles qu'on va aborder dans le partie suivante qui met le point sur une nouvelle méthode de concevoir les objets pour diminuer leur couplage ainsi que le coût des évolutions et de la maintenance dans le futur.

II.3-Introduction au design Patterns

Un pattern (modèle) est une façon de faire, une manière d'arriver à l'objectif fixé. Les patterns permettent d'utiliser des principes basés sur l'habileté des précurseurs en la matière. Les modèles sont une façon excellente de capturer et transporter l'habileté.

Les Patterns, sont apparus bien avant l'apparition de l'informatique, étaient utiliser en architecture. Les modèles architecturaux peuvent servir et inspirer les personnes qui occuperont les bâtiments.

Les design patterns répondront au principe de couplage faible et par suite permettront une maintenance plus souple et moins coûteuse.

II.3.1- Avantages des Design Patterns

Un Design Pattern est un modèle de conception, il utilise des classes et leurs méthodes dans un langage orientée objet, .Net dans notre cas. Les Design Patterns permettent de mettre en application des concepts spécifiques sans pour autant avoir à passer beaucoup de temps sur la méthode de développement à employer pour arriver à un objectif. Ils permettent donc de simplifier la réflexion en réutilisant des principes de base de la conception, cela aura donc pour effet de rendre notre programme plus lisible lors de phase de maintenance par exemple.

Les Design Patterns sont tout simplement des architectures de classes permettant d'apporter une solution à des problèmes fréquemment rencontrés lors des phases d'analyse et de conception d'applications. Ces solutions sont facilement adaptables (donc réutilisables), elles sont utilisables sans aucun risques dans la grande majorité des langages de programmation orientée objet.

Les Design Patterns sont fiables. Ils ont une architecture facilement compréhensible et identifiable. Cela améliore donc la communication et la compréhension entre développeurs.

II.3.2. Utilisation des Design Patterns

Les Design Patterns sont des organisations de classes. La difficulté réside dans le fait de savoir identifier les moments où il faut les utiliser et les moments où cela n'est pas utile.

En ce qui concerne les Design Patterns, cette partie a uniquement vocation de présenter les plus utilisés ainsi ceux que j'ai choisi pour faire partie de la migration en question.

II.3.3. Les principales classes de Design Patterns

II.3.3.1 Les Patterns de création(*Creational Patterns*)

Qui ont le rôle de création d'objets sans instanciation directe d'une classe :

Type de design Pattern	Objectif
Abstract Factory ([3] chapitre 6)	obtenir des instances de classes implémentant des interfaces connues, mais en ignorant le type réel de la classe obtenue
Le monteur (*Builder)*	Création d'un objet selon son type et son contenu.
Factory Method	Le pattern "Factory method" explique le moyen de procéder au découplage désiré
Prototype	obtenir une instance d'un objet à partir d'une autre instance
Singleton	s'assurer qu'une seule instance d'un type spécifique existe dans le système et fournir l'accès à cet objet

Table 2: Type et objectifs des Patterns de création

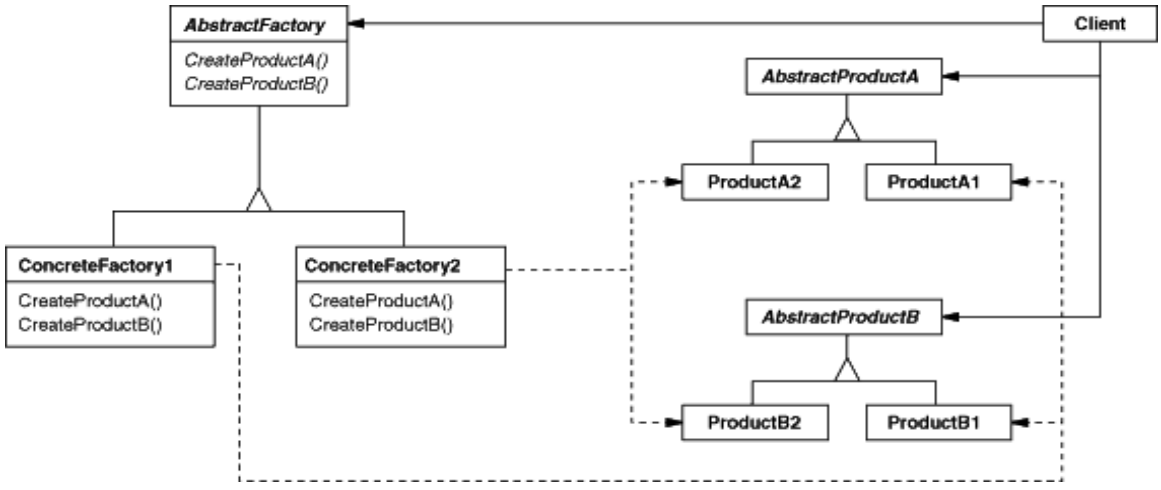

Figure 12: Fabrique abstraite

II.3.3.2 Les Patterns de structure (*Structural Patterns*)

Les Design Patterns de structure sont les patterns qui concernent des problèmes d'assemblage d'objets et de structure de code. Plutôt que de composer des interfaces et des implémentations, ces Design Patterns décrivent les moyens de composer des objets pour réaliser de nouvelles fonctionnalités. ([3] chapitre 3)

Type de design Pattern	**Objectif**
Adapter	convertit l'interface d'une classe en une autre répondant aux attentes du client, soit par composition soit par héritage multiple.
Bridge	découplage entre une abstraction et une implémentation. Exemple : création de fenêtre indépendamment de la plate-forme.
Composite	représentation des structures d'arbres, ou composition récursives.
Decorator	ajoute dynamiquement des caractéristiques (ou des comportements) à un objet (une classe). Exemple : attributs d'un window manager.
Façade	Consiste à donner une interface unifiée et haut niveau à un sous-système composé de plusieurs interfaces et objets aux interactions complexes
Flyweight	objet partagé et susceptible d'être utilisé par plusieurs contextes simultanément. Exemple les caractères d'un texte
Proxy	intercepte les requêtes vers un objet et instancie celui-ci quand il est réellement nécessaire.

Table 3: **Type de design patterns**

Le besoin d'un code de plus en plus dynamique qui répond à la complexité des règles de gestion a poussé les recherches à aboutir à un nouveau type de programmation qui vient augmenter le niveau d'abstraction des applications.

II.4 Introduction à WPF

II.4 .1 Introduction

Pour la plupart des utilisateurs, une application est juste et tout simplement une fenêtre qui s'ouvre et qui leur permet d'interagir avec l'ordinateur. Mais pour le développeur, la gestion de cette fenêtre n'est pas forcément des plus simples. En effet il faut choisir les librairies les plus adaptées pour accomplir l'ergonomie de l'application.

Pour ce qui est de .Net, on a le choix entre les WinForms, et depuis la sortie du Framework .Net 3.0 (2006) avec Windows Vista : WPF ; Même si comme on pourra le voir, ce dernier complète davantage les composants WinForms. On va pouvoir voir dans ce qui suit une introduction à WPF, en quoi consiste WPF, ce qu'il nous apporte et comment l'utiliser.

II.4 .2 Avant WPF

WPF est très récent étant donné qu'il est apparu avec le Framework .Net 3.0. Avant il n'y avait que les WinForms.

Les WinForms, c'est le nom donné à la partie du Framework .Net, responsable de la partie interface utilisateur (GUI). Les WinForms apparaissent similaires aux *Forms* dans le langage *visual basic 6*, tout en ayant apporté leur lot d'avantages. En effet, elles sont très faciles à prendre en main et très orienté objet.

II.4 .3 Pourquoi WPF

A la sortie de Windows Vista, on voit apparaitre des effets 3D et en même temps la sortie du .Net Framework 3.0. Pourquoi ? On s'est aperçu que les WinForms n'étaient pas vraiment adaptés (pour de nombreuses raisons techniques telles qu'elles ne sont pas forcément des plus aisées à personnaliser). Elles posent également un problème au niveau du travail collaboratif entre designers et développeurs, et bien d'autres.

On verra dans les lignes qui vont suivre que WPF apporte son lot de nouveautés qui facilitent le « design » de la GUI. Par exemple, on peut citer :

- les graphismes vectoriels.
- la transparence par pixel.
- les animations.
- l'adaptation à la résolution.
- le support des *template*s de *data binding* (je reviendrai dessus dans le paragraphe 4 .4.1.3 de ce chapitre)

II.4 .4 Les plus et les moins

Comme n'importe quelle technologie, WPF a des avantages comme des inconvénients, même ces derniers sont malgré tout très restreints, comme on va pouvoir le voir.

II.4 .4.1 Les avantages de WPF

II.4 .4.1 .1 Utilisation du GPU

Un des principaux changements avec WPF c'est l'utilisation du GPU. Bref rappel de base, le GPU est en fait le processeur graphique présent sur la carte graphique d'un ordinateur.

Le fait que WPF utilise le GPU change énormément de choses. En effet cela permet de déléguer une partie du travail habituellement délégué au microprocesseur (CPU), au processeur graphique qui va se charger de la manipulation de données graphiques.

II.4 .4.1.2 Séparation code / design

Souvent, lorsqu'un développeur travaille une application en collaboration avec des designers et d'autres développeurs, un problème majeur va se poser. En effet pour customiser votre application, votre designer devra avoir des compétences en développement, il va devoir connaître les objets de votre application et les fonctionnalités des WinForms.

C'est là que WPF permet de justement séparer en couches votre application,il va se charger de séparer le code designer du code *behind* (classe d'arrière plan).C'est à dire que le designer va pouvoir travailler sur le design de l'application,via un langage commun basé sur du XML qui est le XAML qu'on verra plus loin (section 4.5 de ce chapitre). Quant au développeur de son côté via le code *behind* il va pouvoir travailler sur la couche métier. Cela va permettre une meilleure productivité et un support de l'application plus facile par la suite.

II.4 .4.1.3 Plus puissant que les WinForms

Même si au premier abord, WPF peut choquer par la séparation code / design et aux contrôles également sensiblement différents, WPF offre plus de possibilités comme on a pu le voir et plus de « puissance ».

C'est-à-dire que par exemple grâce à WPF et Expression Blend on va pouvoir mettre en place des animations pour notre application, de façon vraiment très simple et très rapide. Cela augmente d'une part la productivité du développeur, celle du designer et pour finir l'ergonomie de l'application pour l'utilisateur final.

De plus on peut parler de la puissance de WPF au niveau du *Data Binding* (liaisons de données). En effet le *Data Binding* est un mécanisme puissant. Nous verrons plus en détail en quoi consiste le *Data Binding* dans le paragraphe suivant.

II.4 .4.1.4 La liaison des données (*Data Binding*)

La liaison de données est la connexion entre l'application et interface utilisateur logique métier. Il ya beaucoup d'amélioration dans le mécanisme de liaison de données en comparant. NET 2.0 à. NET 3.0. Actuellement, le mécanisme de liaison de données peut être décrit en utilisant le langage XAML. Il ya de nouvelles classes d'objets qui sont introduites pour la liaison de données. La source de la liaison de données peut être tout objet contenu dans le Common Language Runtime (CLR) ou bien un fichier XML.

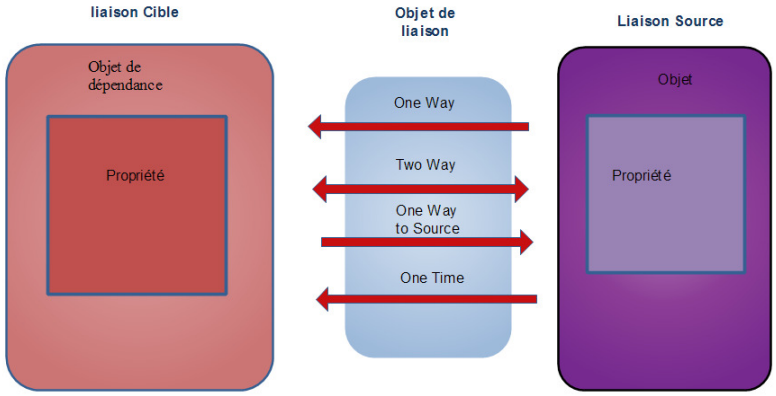

Figure 13 : Liaison des données dans WPF

Liaison cible(*Binding Target*)

Toutes les propriétés de l'objet cible comme la couleur de fond, ou le contenu peut être lié à la source de liaison. Il s'agit de la flexibilité de WPF.

Par exemple, dans les applications d'inventaire la couleur du fond d'écran de la liste des éléments peut être rouge ou vert selon quantité en stock.

Liaison source(*Binding source*)

C'est le fournisseur de données qui peuvent être des objets construits à partir d'une base de données ou des fichiers XML.

Objet de liaison (*Binding Object*)

L'objet de liaison est la classe qui fait le lien source de liaison et sa cible.
Les objets de liaison comme le montre la figure ci-dessus permettent de différentes orientations de flux de données :

- **One Way** - flux de données provenant de la source de données vers la cible.
- **Two Way** - flux de données dans les deux sens, si jamais la valeur cible change, la source est mis à jour également.
- **One Way à la Source** - Le flux de données autorisé n'est que dans le sens inverse.
- **One Time** - le flux de données de la source de liaison à la cible se fait qu'une seule fois (au moment d'initiation de la cible)

La propriété "*UpdateSourceTrigger*" doit avoir une valeur qui détermine quand les données devraient transiter de la source vers la cible. Le tableau ci-dessous montre les valeurs possibles pour cette propriété.

Valeur valide pour *UpdateSourceTrigger*	**Description**	**Exemple**
LostFocus (deault)	La source est mise à jour quand on y est plus positionné.	Utilisé quand un champ de saisi doit être validé quand on passe à un autre.
PropertyChanged	La source est mise à jour quand une propriété change.	La quantité change par exemple.
Explicit	La source est mise a jour sur demande explicite	le click d'un bouton

Table 4 : déclencheur de mise a jour des données(*UpdateSourceTrigger)*

Exemple:

L'exemple ci dessous montre comment lier la couleur du fond d'écran d'un bouton a une autre propriété.

```
<DockPanel>
<DockPanel.Resources>
<c:MyData x:Key="propSource1"/>
</DockPanel.Resources>
<DockPanel.DataContext>
<Binding Source="{StaticResource propSource1}"/>
</DockPanel.DataContext>
<Button Width="150" Height="30" Background="{Binding Path=ColorName}"
>Red<Button>
</DockPanel>
```

II.4 .4.2 Les inconvénients

II.4 .4.2.1 Manque d'interopérabilité

Le principal problème de WPF reste le même que les WinForms c'est-à-dire l'interopérabilité de ce dernier. En effet on ne peut pas, dans l'état actuel des choses, faire du WPF sous linux sous Mac OS ou d'autres systèmes. Le seul portage de .NET fait sur les autres systèmes reste mono et mono ne supporte actuellement pas le WPF.

On peut dire que cela est appelé à changer dans le temps, car quand on regarde du côté de Silverlight, qui est « l'équivalent » de WPF pour les applications web .NET, on peut remarquer qu'il y a un portage réalisé qui s'appelle *Moonlight*. Donc, à suivre.

II.4 .4.2.2 Tout est à refaire

Enfin, on peut dire que la migration des applications WinForms actuelle reste complexe.

C'est pourquoi, le développeur qui souhaite migrer son application WinForms vers WPF, va devoir revoir et recréer toute sa GUI. Pour peu de vouloir utiliser les spécificités de WPF, il faudra également qu'il revoit son architecture, notamment pour l'utilisation du *Data Binding*(section.4 .4.1.4 de ce chapitre).

II.4 .5 L'intérêt du XAML

II.4 .5.1 Pourquoi XAML ?

Le XAML (e**X**tensible **A**pplication **M**arkup **L**anguage) est un langage déclaratif basé sur la syntaxe du XML. Il permet grâce à des balises et des attributs de créer très facilement des objets.

Pour cela, le compilateur XAML se charge de déclarer et définir des objets dynamiquement grâce aux balises (équivalent des classes) et aux attributs (équivalents aux propriétés) XAML.

Malgré sa syntaxe simple, le XAML permet de restituer des graphiques vectoriels, ou des modèles 3D aisément. Les possibilités graphiques sont donc infinies.

Il existe quelques règles élémentaires, issues de la syntaxe du XML, qu'il faut respecter XAML :

- Respecter la casse.
- Les balises ouvertes doivent être refermées sans se chevaucher.
- Chaque attribut doit obligatoirement avoir une valeur inscrite entre guillemets ou apostrophes.

Voici un court exemple de la syntaxe XAML, afin de voir à quoi elle ressemble la. Je vais simplement créer un bouton accompagné d'un texte. Je ne vais pas entrer dans une description détaillée du code, chose qui n'est pas le but de ce mémoire

```
Window x:Class="WpfApplicationCNAM.WindowWPF"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Exemple XAML" Height="300" Width="300">
<Grid Background="AliceBlue">
<Button Height="63" Click="Button_Click" Margin="64,56,64,0"
VerticalAlignment="Top" Background="DarkGray" BorderBrush="DarkBlue"
Foreground="Black" OpacityMask="Azure" Cursor="SizeNWSE">Bouton</Button>
<TextBlock Text="CNAM 2011!" FontSize="26" FontFamily="Vivaldi"
Foreground="Crimson" Margin="67.937,0,64,0" />
</Grid>
</Window>
```

Ce que je dois noter c'est que grâce à ces balises que la génération de l'écran se produit, chaque balise est traitée et les événements sont attachés au code *behind* par un simple attribut comme c'est le cas pour l'événement « Click » attaché au bouton ci-dessus.

L'importance de ces fichiers XAML qui sont du texte en fin de compte, c'est qu'ils peuvent être édités à part et même générés automatiquement tant que la syntaxe est respectée, chose qui fera gagner en flexibilité et en temps de développement des interfaces.

Chapitre III

Justification du choix du logiciel

Les logiciels et composants adoptés pour la mise en œuvre de la migration vers une architecture orientée service ont été choisis pour différentes raisons après avoir étudié de près l'existant ainsi que les objectifs tracés.

Les outils de développement présélectionnés concernant cette partie du projet sont : ***Microsoft .Net*** et ***Expression Blend*** de ***Microsoft*** ainsi que **Delphi.Net** de **Borland.**

La plate-forme Microsoft .NET est une :

- Infrastructure moderne orientée objet pour la construction d'applications d'entreprise multi niveau reparties.
- Stratégie technologique viables pour le développement, le déploiement et la gestion des logiciels d'entreprise.
- stratégie de commerce électronique et de conception de services Web XML.

Microsoft .NET est une plate-forme ***Windows*** pour le développement et le déploiement d'applications de classe entreprise. La plate-forme ***.NET*** en est optimisée pour le langage ***XML*** (***Extensible Markup Language***) et architecturée autour des services Web XML. Elle permet l'utilisation de n'importe quel langage et l'interopérabilité des langages. Du point de vue de ***Microsoft***, les deux langages .NET les plus importants sont le nouveau ***Visual C# (C sharp) .NET*** et le ***VB .NET***, une réécriture complète orientée objet du populaire ***Visual Basic***. ***Microsoft .NET*** intègre un environnement d'exécution analogue à celui de Java, qui permet d'exécuter des programmes compilés sur n'importe quelle plate-forme prenant en charge le ***CLR*** (***Common Language Runtime***). ***Microsoft .NET*** est prise en charge par ***Microsoft*** et nouvellement par ***Borland***. Elle peut tourner sous ***Windows*** *et* ***Unix*** avec le lancement de sa plateforme ***Mono*** , et elle reconnaît plus de 22 langages de programmation différents, incluant Java et **Delphi** récemment .

Le tableau suivant montre les caractéristiques détaillées de la plateforme ***Microsoft .NET***,

	Microsoft .NET
Contrôle d'infrastructure	Microsoft
Site Web	http://www.microsoft.com/net
Axiome	Une plate-forme, n'importe quel langage (optimisation de plate-forme)
Plates-formes	Windows
Éléments compris	.NET Framework .NET Common Language Runtime (CLR) .NET Common Language Specification .NET Languages (VB, C++, C#, JScript, J#) .NET My Services .NET Web Services Framework ADO.NET, ASP.NET Windows Forms, Web Forms Mobile Internet Toolkit
Langages de programmation	APL, C++, C#, COBOL, Component Pascal, Curriculum, Eiffel, Fortran, Haskell, J# (Java Language), Microsoft JScript®, Mercury Mondrian, Oberon, Oz, Pascal, Perl, Python, RPG, Scheme, SmallTalk, Standard ML, Microsoft Visual Basic.
Serveurs d'applications	ASP .NET avec COM+
Environnements de développement interactifs (IDE)	Visual Studio .NET
Environnement d'exploitation	Common Language Runtime (CLR)
File d'attente de messages	MSMQ
Code intermédiaire	Intermediary Language (IL); Common Language Runtime. .NET CLR permet au code de multiples langages d'utiliser un ensemble commun de composants, sur Windows.
Stratégie de composants	Gestion des composants .NET

	Le service des composants .NET est assuré au moyen des services COM+.
Interface utilisateur graphique/Formulaires	.NET Windows Forms,Web Forms et WPF
Normalisation	Microsoft a soumis les spécifications du langage C# et de l'infrastructure CLI (Common Language Infrastructure) à l'ECMA à des fins de normalisation. La majeure partie de .NET est une propriété exclusive.
Pages Web de serveur dynamique	Active Server Pages (ASP .NET) Le code ASP .NET (n'importe quel langage) est compilé en code natif par l'intermédiaire du CLR.
Connectivité de bases de données	SQL Server, ADO .NET,Oarcle…
Connectivité de systèmes existants	Host Integration Server
API de nommage/d'annuaire	Active Directory Services Interface (ADSI)
Prise en charge XML	Inhérente à l'architecture L'espace de noms System.Xml de la bibliothèque .NET Framework Class Library assure la prise en charge du langage XML en fonction des normes établies.
Services Web	Inhérents à l'architecture. L'espace de noms System.Web.Services de la bibliothèque .NET Framework Class Library permet la création de services Web XML à l'aide d'ASP.NET et de clients de services Web XML. Prise en charge des protocoles XML, SOAP, WDSL et UDDI.

Table 5 : caractéristiques de la plateforme Microsoft.NET

Microsoft .NET, actuellement offert sous ***Windows*** seulement, se voulait au départ multi plate-forme de bas en haut. Le langage intermédiaire ***IL*** (***Intermediate Language***) de l'environnement ***CLR*** (***Common Langage Runtime***) est l'équivalent du pseudo code Java. Bien que ***Microsoft*** ne s'y soit pas engagée, l'écriture d'un ***CLR*** pour ***Unix*** ou ***Linux*** appartient au domaine du possible. ***Ximian*** a récemment annoncé le lancement du projet ***Mono***– initiative visant à mettre au point une version à code source

ouvert de l'infrastructure de développement ***.NET***. Il est donc une vérité que ***.NET*** devienne multi plate-forme avec tous les efforts fournis.

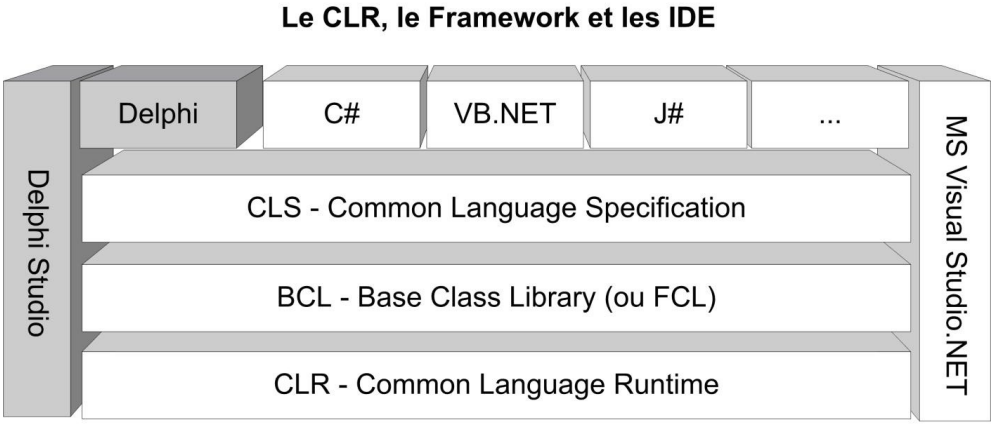

Figure 14 : Delphi .Net avec le CLR

Si l'on met de côté les questions épineuses de portabilité et de préférences en matière de langage de programmation ou de constructeur de logiciels, les principaux enjeux commerciaux lors de la sélection d'une plate-forme d'applications appropriée sont illustrés par ces principaux critères sur lesquels j`avais un avis surtout en ce qui concerne la partie de performance et de développement :

- Sa capacité
- Son coût total de possession
- Sa performance
- Ses Capacités de développement

La majorité des évaluateurs s'entendent pour dire que ***.NET*** est le chef de file en matière de performances et de capacités de développement alors que d'autres avaient une meilleure réputation en matière de fiabilité, sécurité, stabilité et portabilité. Avec le temps les architectes la plateforme ***Microsoft.Net*** se sont appliqués sur les petites failles de la plateforme dont la plus importante réside dans la portabilité de cette dernière mais avec ***Mono*** cette question devient du passé sachant qu`avec le temps la plateforme sera *Open Source* et cela est évident avec les extensions gratuites *Enterprise Library* à cette plateforme que Microsoft n`hésite pas à dévoiler ses secrets chose qui augmente la confiance et ouvre de nouvelles horizons.

Avec l`adoption de **Borland** pour la plateforme .Net, il est devenu possible de profiter de ses caractéristiques tout en gardant le langage de programmation intacte sauf pour les librairies de bases qui concrétiseront l`architecture globale du logiciel. L`invocation d`une fonction dans une libraire .Net à partir d`un code **Delphi.Net** est

devenu une réalité ; le programmeur n`est pas obligé à apprendre un nouveau langage de programmation mais la méthode de référencer les librairies dans son projet ce qui évite le coût de la formation et le respect des délais.

- ***Enterprise Library*** **de Microsoft**

Enterprise Library peut être utile dans différentes situations :

- ***Enterprise Library*** offre suffisamment de fonctionnalités pour prendre en charge de nombreux scénarios courants des applications d'entreprise selon l'architecture SOA.

- Enterprise Library est conçu de sorte que ses blocs d'application puissent fonctionner indépendamment les uns des autres. Il suffit d'ajouter que les blocs d'application que l'application utilisera ; il n'est pas nécessaire d'ajouter la bibliothèque entière ([2], page 41).

- Enterprise Library inclut le code source des **blocs d'application**. Cela signifie qu'il est possible de modifier les blocs d'application à fusionner dans la bibliothèque existante, ou utiliser des parties du code source Enterprise Library dans d'autres blocs d'application ou applications à développer.

- Enterprise Library inclut la documentation, des exemples et le code source. Cela signifie que la bibliothèque peut servir comme outil pour apprendre les méthodes recommandées en termes d'architecture, de conception et de codage.

- Enterprise Library peut servir de base à une bibliothèque personnalisée. On peut tirer parti des points d'extensibilité incorporés dans chaque bloc d'application et étendre le bloc d'application avec de nouveaux fournisseurs. Il est possible également modifier le code source des blocs d'application existants afin d'incorporer de nouvelles fonctionnalités. Enfin, il est possible d'ajouter de nouveaux blocs d'application à Enterprise Library. Il est possible de développer de nouvelles extensions pour les blocs d'application existants et les nouveaux blocs, soit utiliser les extensions et blocs d'application développés par d'autres.

Après avoir contribué dans l`évaluation des spécifications techniques de la plateforme .Net ainsi que les bénéfices de la nouvelle version de *Enterprise Library* sous la direction du directeur technique, il a fallu trancher sur l`adaptabilité de la technologie à utiliser pour le développement des interfaces et des services web et proposer des extensions supplémentaires qui peuvent répondre aux besoins.

Possédant les ressources expérimentées dans l`analyse et le développement des applications sous ***Microsoft .Net*** ainsi qu`un grand nombre de ressources de développement sous le langage Delphi qui migre en vitesse vers la plateforme .Net, les décisions suivantes ont été prises:

- Les interfaces et les services Web XML seront développés avec la technologie **.Net** en utilisant ***Microsoft Visual Studio.Net 2008*** en se basant sur ***Enterprise Library*** comme point de base.

- La nécessite de la migration de la version Delphi6 qui est une plateforme Win32 vers **Delphi.Net** en utilisant ***CodeGear2007*** de ***Borland*** ; chose qui n'affectera pas le langage de programmation et par suite l`expertise dans la programmation en Delphi gardera sa valeur et diminuera les coûts d`apprentissage d`un nouveau langage .La société ***Borland*** permet une migration souple d`une version à une autre sauf dans quelques cas où l'intervention du développeur reste nécessaire pour faire d`une librairie **Win32** une autre équivalente en ***.Net*** avec le minimum de développement.

III.1– Architecture et Composants du système

III.1.1 Architecture Globale du système

III.1.1.1 Les composants du système

La restructuration selon le principe SOA a mené à une nouvelle architecture d'Omega PM qui se résume par les composants suivants :

- La plateforme Omega.Net : La plateforme Omega.Net contient toutes les anciennes fonctionnalités d'***OmegaPM*** sauf qu'elle a été structurée en sous composants:

  - Le noyau de type ***Framework*** : Ce noyau est la structure de base sur laquelle est construite toute la nouvelle architecture, il contient des sous composant responsable de l'accès et de la gestion des données ainsi que leur validation. Les composants de ce noyau seront traités en détail dans la partie de l'architecture détaillée.

  - Le noyau de type ***Core*** : Ce noyau contiendra les composants fonctionnels d'***Omega PM***. Ce noyau contiendra les anciennes fonctions de calculs et de traitement de données et usera du noyau ***Framework*** quand c'est nécessaire pour accomplir des tâches communes comme l'accès à la base de données par exemple.([2] page 6)

- L'interface utilisateur : L'interface utilisateur permet aux utilisateurs de manipuler les différentes transactions et d'accéder aux différents services.

- Les Composants de Services ou ***Utilities :*** Ce sont des composants qui regroupent les fonctions indépendantes du noyau comme l'exportation des données et d'autres fonctionnalités dont le code a été très redondant dans l'ancienne architecture.

- La base de données Omega : Contient les données sur lesquelles porte la gestion. Ce mémoire n'aborde pas la conception ou la mise à jour de cette base.

- L'éditeur du fichier de configuration : L'éditeur du fichier de configuration, qui permet à l'administrateur de mette à jour le paramétrage de l'application ***Omega PM.***

IV.1.1.2 – Les moyens de communications entre les composants du système

L'interface utilisateur a besoin de communiquer avec les différentes couches pour effectuer des calculs, mettre à jour ou insérer des données dans la base de données.

Les différentes interactions entre les composants sont illustrées dans ce qui suit où chaque composant est abordé en détail.

IV.1.1.3 – Modèle de déploiement

Le modèle de déploiement est représenté par un diagramme de déploiement qui montre l'architecture physique du système complet. En effet, il montre la topologie des serveurs ainsi les logiciels qui sont installés et qui fonctionne sur chaque machine.

Le déploiement d'***Omega PM*** nécessite selon le besoin du client 6 types de machines :

1-Le serveur Omega : Ce serveur permet l'hébergement d'***OmegaPM*** contenant la plateforme Omega.Net.

2-Le serveur Web : Ce serveur permet l'hébergement du serveur Web du ***Microsoft***, ***IIS 6.0 Web Server***, d'un ***Firewall*** pour des raisons de sécurité ainsi que toutes les applications Web : interface utilisateur Web en cas ou le client (institution financière ou autres) décide d'exposer les fonctions en ligne, les services Web XML : Service Web ***Calcul***, Service Web d'accès au données Omega et bien d'autres qui peuvent être développés et déployés.

3-Le serveur de base de données.

4-Utilisateur Omega : Chaque utilisateur de ***Omega PM*** possède un ordinateur sur lequel on installe le navigateur Web pour les clients souhaitant avoir une interface Web ou client léger, et l'application cliente pour les utilisateurs souhaitant avoir un client lourd, une application ***Windows***. Il est à noter que la nouvelle architecture offre la possibilité d'avoir des modules ou des services futurs qui fonctionnent sur les portables.

5-Client Administrateur : Ce système offre une interface pour les administrateurs du système qui leur permet de mettre à jour le paramétrage du système. Cette application est pour le moment une interface ***Windows*** avec la possibilité de l'étendre pour devenir une application Web.

La communication entre serveurs, ou entre client/ serveur varie d'un serveur à un autre :

Le Client Omega communique avec le serveur Web via ***http (Hyper Text Transfert Protocol)***. En effet, si l'application cliente est de type Web, le client possède un navigateur Web qui lui permet d'utiliser l'application hébergée dans le serveur Web et

qui s'exécute via le serveur Web ***IIS 6.0***. Si l'application cliente est de type lourd ou Windows, l'exécution est une exécution locale de l'application déployée sur la machine même ou sur le serveur Omega.

Le serveur qui héberge la plateforme Omega.Net communique aussi avec le serveur Web avec les différents services web déployés et ceci en utilisant ***HTTP/SOAP***.

Le serveur Web communique avec le serveur de base de données à travers ***TCP/IP***.

Le client administrateur du système communique avec le serveur de bases de données via ***TCP/IP*** puisque pour le moment c'est un client lourd. Dans le futur, une interface Web serait à la disposition des administrateurs de la base de connaissance. A ce moment là, une application Web serait hébergée sur le serveur Web qui accédera le serveur de la base de connaissance via ***TCP/IP*** alors le client n'aura pas un accès directe avec le serveur de base de données.

La figure suivante montre le modèle de déploiement :

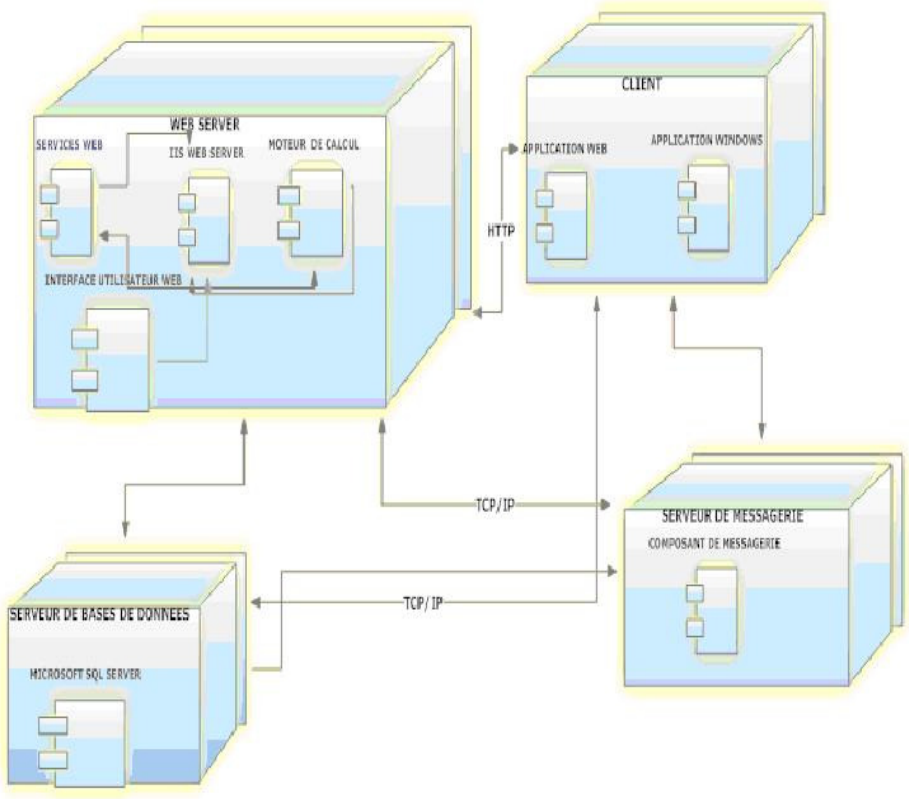

Figure 15: le modèle de déploiement du système

Dans ce qui suit, je vais aborder les composants qui auront le rôle dans la conception et le développement de la nouvelle plateforme Omega ainsi que leur apport en termes de découplage et d'abstraction. Les blocs ou brique d'application de Microsoft nous ont inspirés et ont été la base de nouveaux composants.

III .1.2 Les Blocs d'application Enterprise Library

Quand on a un bagage de programmation de plusieurs applications .Net, on se rend vite compte qu'il existe des parties génériques à toute application qui peuvent être centralisées.

Les blocs d'applications de Microsoft m'ont permis de profiter de leur niveau d'abstraction et de généricité pour bâtir des librairies personnalisées qui répondent à niveau élevé de performance et de maintenance.

III .1.2.1 Introduction aux blocs d`application

Microsoft a développé et testé des extensions à son Framework .Net et a fourni leur code pour les rendre à la disposition des concepteurs d'application .Net. Ces composants ou briques appelées *Application Blocks* évitent d'avoir à développer soi même et à tester (ce qui est long et coûteux) une solution qui répond de façon générique à un problème donné.

En tant qu'analyste programmeur ayant une responsabilité dans la prise de décision pour la migration du logiciel, je ne devais pas opter pour des solutions qui seraient pénalisantes à court et à long terme. Cela veut dire qu'il ne fallait pas réinventer la roue car les délais ne seront respectés dans ce cas et il ne fallait pas non plus céder et prendre le tout comme tel et adapter notre besoin à ces extensions. Je vais présenter dans ce qui suit les différents types de blocs et dans le prochain chapitre je décris mon intervention pour la création de notre propre plateforme et toutes les modifications qui ont été nécessaires de façon à augmenter le niveau d'abstraction en s'inspirant des design patterns(Chapitre II section3).

Enterprise Library est un sous Framework contenant plusieurs *Application Blocks* ("blocs d'application"). Les blocs d'applications encapsulent les méthodes de développement recommandées par les architectures adeptes à la séparation du code et SOA en fait partie.

Enterprise Library contient neuf blocs d'application :

- Le bloc de mise en cache/*Caching*
- Le bloc Commun/*Common*([2] chapitre 4)
- Le bloc de Configuration/*Configuration*([2] chapitre 5)
- Le bloc d'accès aux données/*Data*([2] chapitre 6)
- Le bloc de gestion des exceptions/ *Exception Handling*
- Le bloc d'audit/ *Logging*

- Le bloc de validation/*validation*

- Le bloc de Gestion de la sécurité/ *Security*

Enterprise Library peut être utile dans différentes situations :

- Enterprise Library offre suffisamment de fonctionnalités pour prendre en charge de nombreux scénarios courants des applications d'entreprise.

- Enterprise Library peut servir de base à une bibliothèque personnalisée. On peut tirer parti des points d'extensibilité incorporés dans chaque bloc d'application et étendre le bloc d'application avec de nouveaux fournisseurs. On peut également modifier le code source des blocs d'application existants afin d'incorporer de nouvelles fonctionnalités.

- Enterprise Library inclut le code source des blocs d'application. Cela signifie que ces blocs peuvent être modifiés de manière à fusionner avec les bibliothèques existantes.

- Enterprise Library inclut également un ensemble de fonctions fondamentales, notamment des services de configuration, d'instrumentation et de création d'objets. Ces fonctions sont utilisées par tous les autres blocs d'application.

Il y a plusieurs avantages à utiliser le Framework Enterprise Library :

- **Uniformité.** Tous les blocs d'application utilisent les design patterns.

- **Extensibilité.** Les blocs d'application permettent d'être personnalisés en offrant la possibilité d'ajouter du code dans les classes fournies.

- **Facile d'utilisation.** Enterprise Library contient un outil graphique de configuration, une installation simple, une documentation utilisateur et des exemples d'application.

- **Intégration.** Les blocs d'application d'Enterprise Library sont conçus pour communiquer ensemble.

- **Gratuité.** Enterprise Library est redistribué gratuitement.

III .1.2.2 Le noyau des blocs d'applications

De nombreuses fonctions présentées dans *Enterprise Library* sont communes à plus d'un bloc d'application et sont également utiles dans le code d'application en dehors de l'Enterprise Library. Des exemples existent comme les routines qui permettent la sérialisation les données ou l'accès aux informations de configuration et biens d'autres. Pour favoriser leur appel, ces routines ont été groupées dans une assemblée commune sous le nom de l'***Enterprise Library Core***.

En outre, tous les blocs applicatifs sont conçus pour avoir un nombre limité de dépendances afin qu'ils puissent être utilisés individuellement, ainsi qu'avec d'autres blocs d'application. Tous les blocs d'application dépendent de l'***Enterprise Library Core***, qui est un regroupement logique composé des blocs suivants.

- Le bloc commun.
- Instrumentation.
- Le bloc de configuration.

III .1.2.2 .1 Le bloc commun

Enterprise Library inclut un bloc représenté par une librairie nommée « **Common »** qui contient des éléments qui sont utilisés par plusieurs blocs.

En fournissant un ensemble de fonctions couramment utilisées pour tous les blocs d'application, cette librairie commune réduit la dépendance d'un bloc d'application d' un autre.

III .1.2.2 .2 Instrumentation

La plupart des blocs d'application contiennent de l'instrumentation. Les types d'instruments sont les suivants:

• Compteurs de performance

•Journal des événements

• Les événements WMI (Windows Management Instrumentation)

Par défaut, l'instrumentation est désactivée, mais on peut utiliser les outils de configuration Enterprise Library pour activer chaque type d'instrumentation pour des statistiques appliquées au code.

III .1.2.2 .3 Le bloc de configuration

Le Configuration Application Block a pour objectifs de :

- permettre à l'application de lire et d'écrire des données de configuration complexes à l'exécution.
- pouvoir stocker des données sensibles cryptées (mots de passe par exemple ou les chaines de connexion).
- faciliter et centraliser la configuration d'une application à l'aide d'un outil d'édition de la configuration, simple d'utilisation.
- fournir un moyen de notifier l'application tout changement dans la configuration.

Ce bloc est conçu de façon à être autonome et homogène par rapport aux autres blocs.

L'utilité d'un fichier de configuration dans une application se résume par ce qui suit :

- pour éviter de mettre des valeurs en dur dans le code. Imaginons que nous utilisions une url de service web dans notre application, si l'url change, on aimerait ne pas avoir à recompiler tout le code.
- pour éviter d'utiliser la base de registre et de donner les droits de modification de base de registre.

  Ces fichiers permettent de typer les données à sauvegarder et le Framework .Net dispose des méthodes pour y accéder facilement.

  L'intérêt d'utiliser un fichier de configuration XML plutôt qu'un fichier binaire est que ce fichier est lisible et compréhensible facilement. On peut également le modifier à la main sans un système évolué permettant de faire des modifications.

Quand on dit fichier de configuration il faut penser à XML et voici un petit exemple dans lequel les connexions au serveur sont paramétrées.

```
<configuration>
  <connectionStrings>
         <add name="MaConnection" providerName="System.Data.SqlClient"
                  connectionString="Data Source=localhost; Initial Catalog=MonCatalog;
Integrated Security=true"/>
         <add name="MaConnection2" providerName="System.Data.SqlClient"
                  connectionString="Data Source=localhost; Initial Catalog=MonCatalog;
Integrated Security=true"/>
  </connectionStrings>
/configuration>
```

III .1.2.2 .4 Editeur du fichier de configuration

L'outil de configuration(*Configuration Tool*) est l'outil de l'Enterprise Library permettant de configurer graphiquement les blocs d`applications. Cet outil est un petit programme Windows permettant de générer des fichiers de configuration. Ces fichiers(Web.config ou App.config) viendront remplacer les fichiers de l'application.

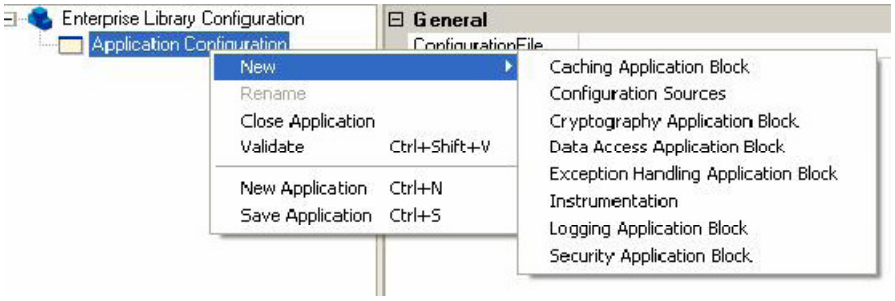

Figure 16: Outil de Configuration des Applications Blocks.

III .1.2.3 Dépendances entre les blocs d'application

Le schéma ci dessous aide à comprendre les exigences pour le déploiement d'applications qui utilisent *Enterprise Library* et aide à assurer le déploiement des assemblages appropriés.

Cette technique utilise le code commun dans le noyau Enterprise Library, qui fait directement appel à l'utilitaire de création d'objets(*ObjectBuilder).* Par conséquent, tous les blocs d'applications ont une dépendance sur la base des bibliothèques de l'entreprise et sur *ObjectBuilder*. La figure ce dessous montre les dépendances obligatoires et optionnelles entre les blocs d'application.

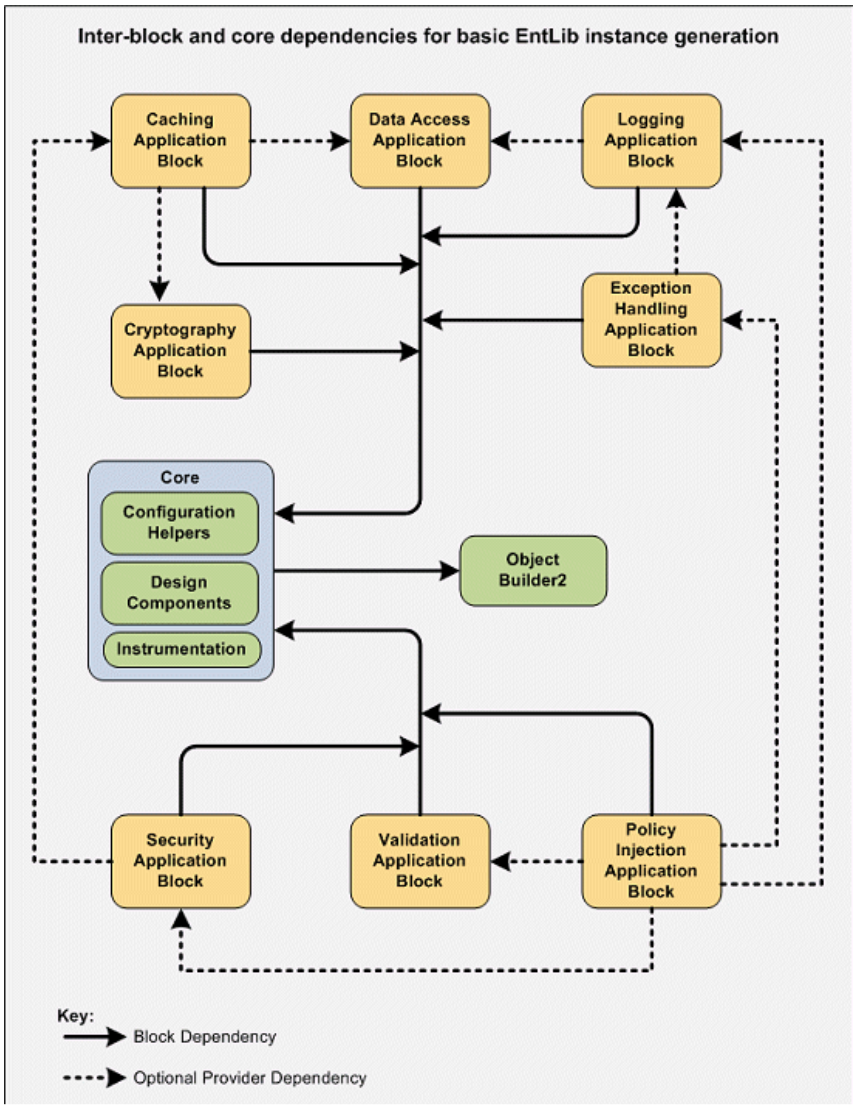

Figure 17: La dépendance entre les blocs

III .1.2.6 Le bloc d'accès aux données

III .1.2.6.1 Champs d'action et caractéristiques

La plupart des logiciels ou applications nécessitent la manipulation d'objets ou de données. Ces données doivent donc être stockées quelque part et il devient donc indispensable d'utiliser les bases de données.

Le serveur d'application doit donc être capable de se connecter aux entrepôts de et d'échanger des flux de données via un langage spécifique au SGDB. Pour faire le lien, il existe plusieurs possibilités :

a)L'Api ODBC (Open DataBase Connectivity). Il s'agit d'un format défini par Microsoft permettant la communication entre des clients bases de données fonctionnant

sous Windows et les SGBD du marché. Cette méthode est universelle et fonctionne toujours mais elle n'est pas très efficace.

b) Le provider fourni par le Framework .NET. Beaucoup plus efficace mais pas optimal.

c)Le provider dédié fourni par la société qui a créé le logiciel de base de données est une solution qui permet d'obtenir de meilleurs résultats en termes de performance.

Le principal problème de ces solutions est la dépendance à la base de données. En effet, si demain, le choix est fait de changer le serveur de base de données, il faut reprendre tout le code pour modifier les requêtes pour qu'elles correspondent au nouveau SGBD (passer des connexions Oracle, aux connexions MySql par exemple).

Dans ce but, Microsoft a développé un bloc applicatif nommé " Data Access Application Block " (DAAB) au sein de " l'Enterprise Library ". Il s'agit d'une couche logicielle qui se trouve au dessus de la couche ADO.NET et qui gère toutes les flux avec les différentes bases de données. Il propose 3 fournisseurs de base de données : SqlDatabase (SqlServer) OracleDatabase (Oracle) et Db2Database (Db2).

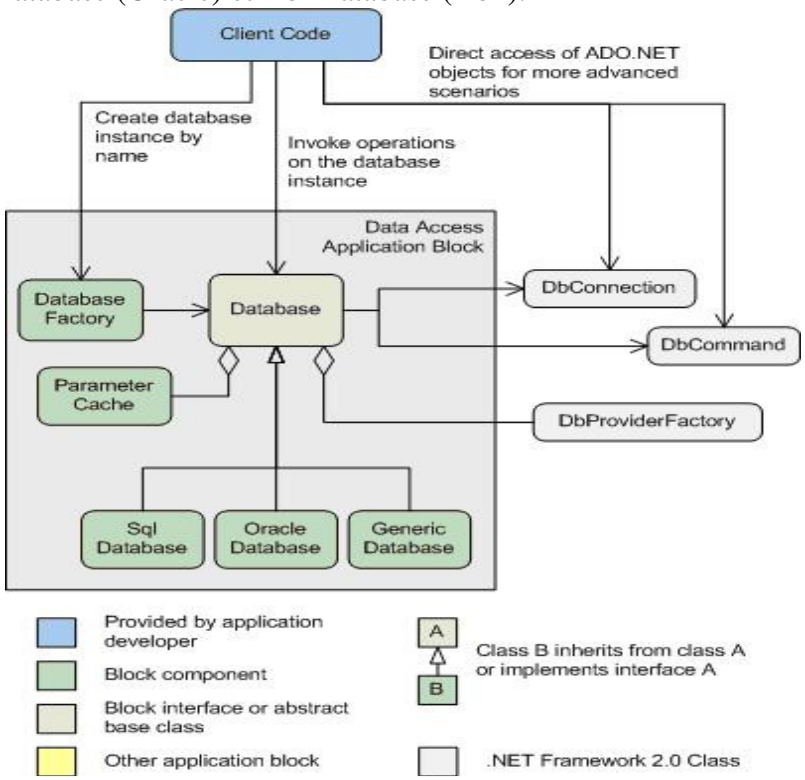

Figure 18: Organisation du Data Access Application Block

La classe de base abstraite *Database* définit l'interface commune et fournit la majeure partie de l'implémentation des méthodes d'accès aux données. Les classes *SqlDatabase*, *OracleDatabase* et Db2Database sont dérivées de la classe *Database*. Elles fournissent à leurs systèmes de serveur de base de données respectifs les méthodes incluant les fonctionnalités courantes (mais implémentées de façon différente selon la

base de données), ainsi que des fonctionnalités propres au système de base de données considéré.

Data Access Application Block a été conçu pour réaliser les objectifs suivants :

- Encapsuler la logique utilisée pour effectuer les tâches d'accès aux données les plus courantes.
- Éviter d'avoir à réécrire le code pour les tâches courantes d'accès aux données.
- Réduire les besoins de code personnalisé.
- Utiliser un nombre réduit d'objets et de classes.
- Garantir que toutes les fonctions du bloc applicatif ont un comportement identique quel que soit le type de base de données.
- Garantir que les applications écrites pour un type de base de données sont, en termes d'accès aux données, identiques aux applications écrites pour un autre type de base de données.
- Utiliser les informations de connexion de base de données stockées dans la configuration.

III .1.2.6.2 Méthodes d'accès aux données

- ExecuteDataSet ;

Cette instruction permet de retourner la liste de résultats d'une requête dans un Dataset.

- ExecuteReader ;

Exécute la requête et renvoie un **SqlDataReader**. L'avantage d'un SqlDataReader est que la lecture ne se fait qu'une fois d'avant en arrière sans la possibilité de revenir en arrière pour un gain de performance.

- ExecuteScalar ;

Exécute la requête et renvoie un seul résultat comme un nombre ou une chaîne de caractères. Le type renvoyé est un objet, il faut donc le transtyper.

- ExecuteNonQuery ;

Exécute la requête et ne renvoie pas de données.

III .1.2.7 Le bloc de gestion des exceptions

Le bloc de gestion des exceptions est conçu pour traiter les tâches les plus courantes auxquelles les développeurs font face quand ils ont à écrire des applications qui utilisent la gestion des exceptions. Ces tâches sont organisées en fonction de scénarios.

Chaque scénario donne un exemple d'une situation du monde réel, traite de l'exception des fonctions de traitement de la situation l'exige, et montre le code qui accomplit la tâche. Le but de l'organisation de ces tâches en fonction de scénarios est de donner le contexte du code.

Au lieu d'afficher un groupe isolé de méthodes, sans aucun sens de l'endroit où ils peuvent être mieux utilisés, les scénarios prévoient un cadre pour le code et de décrire des situations qui sont familières à beaucoup de développeurs dont les applications doivent gérer des exceptions. Ce bloc d'application permet de centraliser la gestion et le traitement des exceptions.

Ce bloc permet également d'élaborer un système de journalisation des exceptions. Cependant, il laisse la possibilité de personnaliser les exceptions.

III .1.2.8 Le bloc de mise en cache

Avec le DotNet framework, il est possible de mettre en cache les données au niveau des couches de présentation.

Le *Caching application block* permet de stocker des données dans la couche métier de l'application. Il évite ainsi au système de persistance d'aller chercher les données lorsqu'on en a besoin.

Les données stockées dans la couche métier sont souvent des données de référence fréquemment utilisées. On peut citer par exemple les différents types de produits proposés, les taux de TVA...

III .1.2.9 Le bloc de piste d'audit(Logging)

La dernière étape avant le déploiement d'une application reste la recherche de bug dans l'application. Pour ce faire il peut être bien utile d'avoir une sortie des événements, des états des variables ou des connexions dans un fichier facilement lisible. Mais ces précieuses informations si utiles au développeur ne sont pas utiles à l'utilisateur et auront même tendance à lui faire peur. De plus, tout le monde sait que l'écriture dans un fichier

ou sur une sortie console est une opération longue, ce qui entraine une perte de performance de l'application. ([2] chapitre 9)

Toutefois il n'est pas question d'écrire du code pour faire le débogage puis de le supprimer avant de recompiler l'application en mode livraison.

C'est la que le Framework,et plus précisément le bloc d'audit intervient en nous fournissant la constante de compilation DEBUG et les classes de manipulation d'informations de débogage.

Ce bloc va permettre d'enregistrer l'activité de l'application que ce soit pour le débogage ou pour un suivi en production.

La figure ci-dessous montre une classe publique utilisée dans ce bloc, cette classe permet d'écrire et de filtrer une information selon son type.

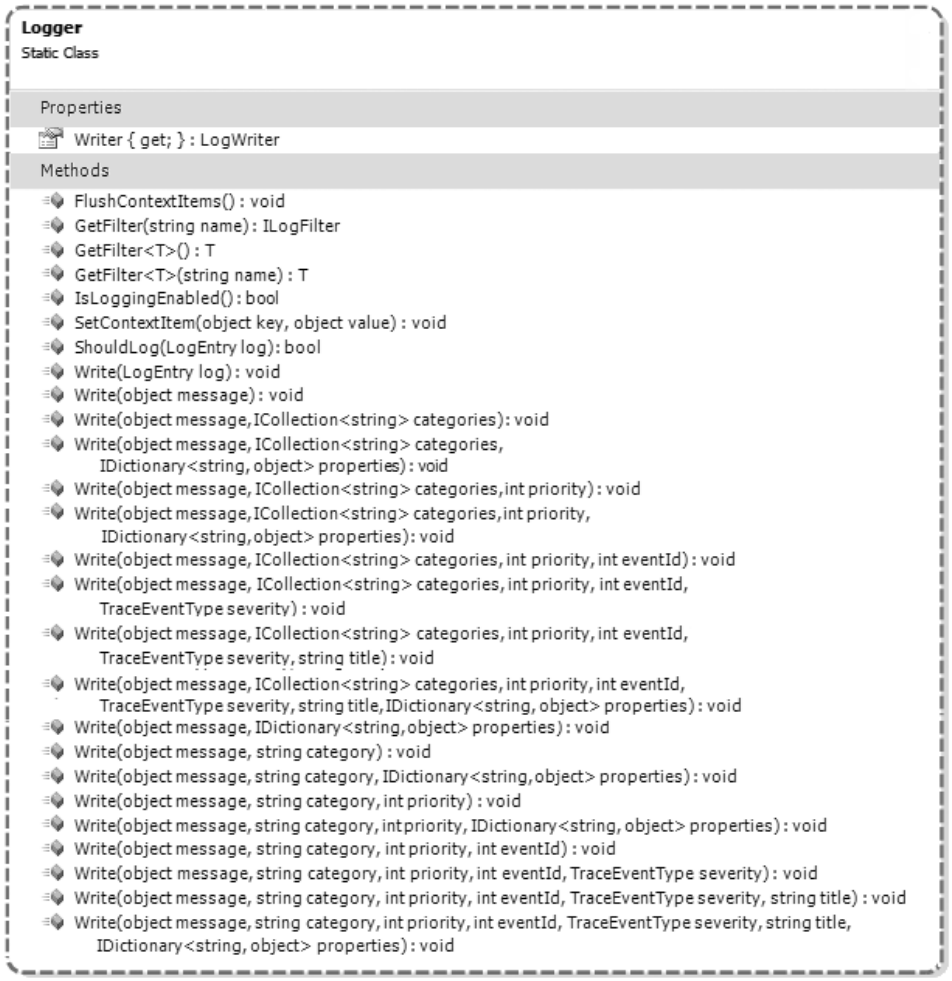

Figure 19: Classe publique Logger.

III .1.2.10 Le bloc de sécurité

Les politiques d'entreprise définissent les règles qui régissent la sécurisation et la gestion d'une application, ainsi que la manière dont ses différents composants communiquent entre eux et avec les services externes. Ces politiques affectent la conception de chaque couche de l'application ou du service.

Ces politiques sont déterminées non seulement au niveau de l'entreprise, mais également au sein des entreprises elles-mêmes. Dans certains cas, il est utile de réfléchir en termes de zones. Toutes les applications, tous les services, voire même tous les niveaux de l'application, sont regroupés dans une même zone si ils/elles partagent un sous-ensemble de politiques.

Par exemple, la connexion vers Internet peut présenter un ensemble de politiques différent du reste de l'infrastructure de l'entreprise, définissant une zone spéciale avec des restrictions plus sévères en matière de sécurité que les autres parties de l'application. Les applications et services de ce centre de données se trouveront par conséquent dans une zone différente de celle des applications et des services de l'intranet. Il est important de bien comprendre les politiques de chaque composant et, de ce fait, de bien définir les zones dans lesquelles chaque composant sera exécuté pour déterminer où déployer ces composants.
Le Security Application Block reprend tous les aspects sécurités liés à l'application.
Il permet de couvrir 5 aspects de la sécurité :

- l'Authentification.
- l'habilitation.
- Les Rôles.
- les profiles.
- Le caching.

III .1.2.11 Le bloc de Validation des données

Le bloc de validation des données offre une bibliothèque de classes nommé validateurs, qui mettent en œuvre des fonctionnalités de validation. NET Framework des types de données.

Il existe également des validateurs nommé *AndCompositeValidator*(pour spécifier la condition et) et *OrCompositeValidator*(pour spécifier la condition ou). Les validateurs de type « And » nécessitent que toutes les conditions soient réunies pour que la validation réussissent et ceux de type « OR » se contentent qu'une seule condition soit Vrai pour déclarer la validation réussie.

Ce bloc permet également de grouper les validateurs dans un ensemble de règles. Un ensemble de règles permet de valider un objet complexe ou un composant graphique en

créant différents validateurs de différents types et de les appliquer à des éléments dans l'objet graphique.

Parmi ces éléments figurent des champs, propriétés et objets imbriqués.

III .1.2.11.1 Avantages des validation applications Block

La Validation Application Block a les avantages suivants:

- Elle permet de maintenir constante la pratique de la validation.
- Il inclut la validation de la plupart des types de données standard de . NET.
- Il permet de créer des règles de validation avec la configuration, les attributs, et le code.
- Il permet d'associer de multiples jeux de règles avec la même classe et avec les membres de cette catégorie.
- Il permet d'appliquer un ou plusieurs jeux de règles lors de la validation d'un objet.

III .1.2.12 Outils divers

III .1.2.12 .1 Générateur dynamique de code

III .1.2.12 .1.1 Introduction

La génération dynamique de code est certainement l'une des fonctionnalités les plus intéressantes du Framework .NET. Cette technique permet de générer dans une application du code à la volée, et ensuite de compiler et exécuter ce code dans le même processus. Il existe une multitude de domaines où cette fonctionnalité peut s'avérer intéressante. Par exemple, dans le cadre d'une application ayant besoin de modifier son comportement dynamiquement afin d'introduire de nouveaux types et ce, sans être arrêtée.

III .1.2.12 .1.2 Le principe de compilation dynamique

La génération dynamique de code ou **Emit Code** donne la capacité pour un environnement donné à intégrer dynamiquement de nouveaux types. Alors, bien entendu cette fonctionnalité n'est pas sans poser de problèmes, notamment au niveau de la sécurité.

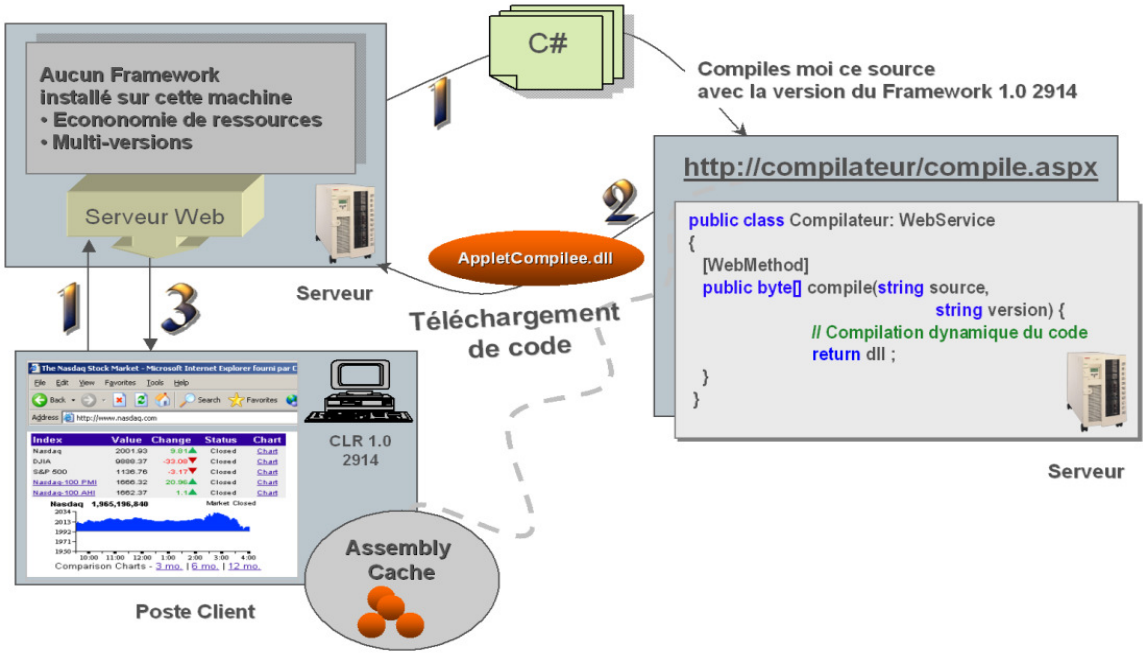

Figure 20: Scenario de génération de code dynamique

System.CodeDom est une autre interface standard, de bien plus haut niveau, dont l'objectif est de représenter un arbre syntaxique abstrait (un AST) sous forme d'objets .NET. Comme son nom l'indique, CodeDom est au code .NET ce que le DOM est à un document XML.

Le gros avantage de ***CodeDom*** est d'être complètement abstrait, indépendant de toute syntaxe des langages de programmation. Une représentation mémoire pourra ensuite être transformée en un fichier C#, VB.NET, ou tout autre langage ***.NET*** à condition de disposer du générateur de code correspondant.

Proxy Dynamique des Services Web

Un autre domaine d'application de la génération dynamique de code pourrait être l'invocation d'un Service Web dont l'interface est spécifiée dans son fichier WSDL(Chapitre II section 1.5.2). Ce même fichier est ensuite utilisé afin de générer un fichier Proxy. Nous pourrions très bien imaginer un programme qui se chargerait de lire le contenu du fichier WSDL et de générer dynamiquement le code du Proxy en fonction de l'interface.

La normalisation des différentes spécifications sur lesquelles s'appuie la technologie service web a permis aux éditeurs de produire des outils automatisant leur utilisation. Ainsi, la plateforme Microsoft.NET s'est dotée, dés sa première version, d'outils (« *Add Web Référence* », « Wsdl.exe ») permettant d'utiliser dans une application.NET des objets et des méthodes fournis par un service Web d'une façon comparable à nos propres objets. Mais la simplicité de ces outils ne peut malheureusement faire oublier leurs limites, révélées lors de leur emploi dans un environnement de production industriel et souvent contraignant.

Contraignant, car il se peut qu'une méthode du service web soit changée sans que le client soit notifié. Par suite sans mise à jour de la référence du service Web, on fait appel à des anciennes méthodes qui ne reflètent la réalité.

Pour ne pas prendre ce risque et pallier à ce problème, il faut penser a l'invocation dynamique des service web.

Génération dynamique d'un proxy

L'objectif est l'implémentation d'un mécanisme offrant une génération dynamique de proxy. Le client n'a besoin que du contrat du service web pour générer un proxy qui reflète une image réelle des méthodes exposées du service Web. Il n'aura plus besoin de savoir a l'avance toutes les méthodes et les paramètres et n'aura plus besoin de faire des modifications en local lorsqu'une modification a lieu dans le service comme l'ajout d'une nouvelle méthode ou le changement des paramètres d'une fonction par exemple.

La génération dynamique du code du permet une flexibilité suffisante à faire jaillir des idées qui dans le passé étaient loin d'être réalisées.

III .1.2.12 .2 Gestionnaire des ressources(Globalisation)

Toutes les applications de production de qualité doivent utiliser des ressources. Une ressource est une donnée non exécutable qui est déployée logiquement avec une application. Une ressource peut être affichée dans une application sous la forme d'un message d'erreur ou comme faisant partie de l'interface utilisateur. Les ressources peuvent contenir des données sous plusieurs formes, telles que des chaînes, des images et des objets rendus persistants. Le stockage des données dans un fichier de ressources permet de changer les données sans avoir à recompiler l'intégralité des applications. Pour écrire

des objets rendus persistants dans un fichier de ressources, les objets doivent être sérialisables.

Le .NET Framework fournit une prise en charge complète pour la création et la localisation des ressources. De plus, le .NET Framework prend en charge un modèle simple pour l'empaquetage et le déploiement des ressources localisées.

Les ressources ont un rôle primordial dans une application multilingue comme la notre. Elles permettent de gérer les messages et la nomeclature des composants graphiques sans passer par la base de données ce qui rend la tâche beaucoup plus simple sachant que ces ressources peuvent être éditées comme tout autre fichier.

Deuxième Partie

Phases de la migration vers l'architecture orientée service

La partie suivante porte sur la mise en œuvre de l'architecture orientée service dans toutes ces étapes avec toutes les préparations de l'environnent pour la faire parvenir à ses bonnes fins.

Devenu familier avec la partie fonctionnelle d'***Omega PM***, je me suis approfondi dans la partie technique du logiciel pour pouvoir faire le diagnostic détaillé des différents problèmes à résoudre dans la nouvelle architecture. Les technologies qui répondent aux besoins fixés et au niveau d'abstraction recommandé m'ont pris un temps de recherche et de test. Je découvre ensuite que la nouvelle version de l'extension du ***Framework .Net*** open source détaillée dans le chapitre précédent, venait d'être lancée et portait avec elle une partie de la solution, et méritait de se pencher dessus.

Mais avant de commencer il faut signaler que plusieurs contraintes nous ont obligés à partager la migration en 3 étapes principales :

1-La Première phase de la migration : Compilation .Net

Cette étape est une simple compilation du Source d'Omega PM pour que l'application s'exécute sous le contexte du Microsoft.Net Framework.

Durant cette étape on a pu avoir une idée concrète préliminaire sur le comportement du logiciel avec la nouvelle plateforme. Cette phase a été importante comme on va le remarquer pour préparer le terrain à la séparation du code et la compilation du code avec les nouvelles librairies créées représentant les différentes couches.

2-La deuxième phase est la restructuration de l'application selon le principe de l'architecture orientée Service.

La remodélisation d'***Omega PM*** comporte plusieurs étapes :

- L'analyse préliminaire qui y a pour but de subdiviser l'application en plusieurs modules légèrement liés tout en étudiant l'inter dépendance de ces modules.
- L'architecture globale du système qui a pour but définir les différents composants du système et leurs interactions ainsi que le modèle de déploiements de ces composants.
- Le développement des modules de gestion des opérations (gestions des exceptions etc…) ainsi que les modules d'héritages.
- La séparation du code de chaque module en plusieurs couches et composants suivants l'architecture multi tiers.
- L'intégration de chaque module migré dans Omega PM sous forme du Delphi.net dll.
- Le test de l'intégration et de la fonctionnalité du module migré.
- Le déploiement des modules migrés dans l'environnement production.

Le déploiement des librairies résultant de cette migration peut se faire sur les différents postes clients ou sur des serveurs différents. Dans le premier cas aucun pré requis autre que Microsoft.Net Framework n'est exigé, mais aucune amélioration au niveau de la performance n'aura lieu.

Dans le second type de déploiement, il existe un Serveur Web Microsoft Internet Information Services Hôte des services Web constituant le moyen de communication entre les différentes couches de l'application. Ces services Web peuvent être accédés par n'importe quelle autre application constituant un moyen d'intégration entre les différentes applications de la société. Selon la configuration, l'application décidera du choix de la communication et des instances qui seront créées.

Plusieurs composants et librairies ont été nécessaires pour assurer le découplage et pour rendre le déploiement moins coûteux et plus rapide.

La figure suivante illustre les différentes étapes à suivre :

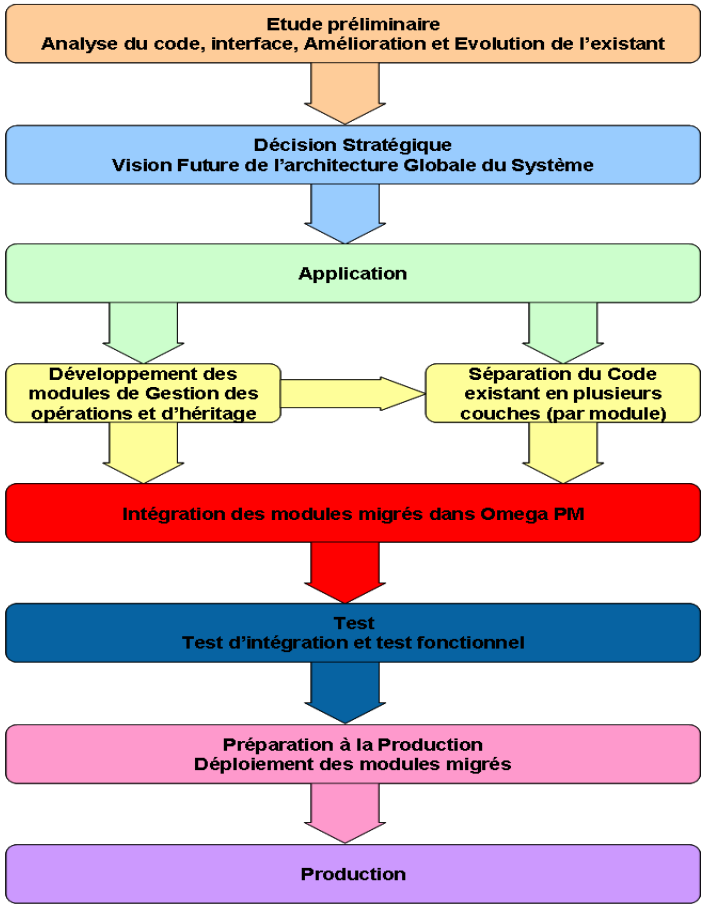

Figure 21: Etapes de la remodélisation d'Omega PM

3-La dernière phase de la migration est la migration des interfaces utilisateurs :

La migration des interfaces utilisateurs ou l'exposition du fonctionnel en tant que services Web exploitables indépendamment de l'application, a nécessité la conception des extensions ou des producteurs qui auront le rôle d'assister le développeur à générer le code nécessaire et de garder le code migré uniforme, chose qui lui facilitera la tâche de maintenance ultérieurement.

Chapitre IV

Compilation en .Net

La première phase de la migration consistait à compiler le code de l'application Omega Win32 développé sous Delphi 6 en Code Delphi.Net permettant ainsi l'exécution d'***OmegaPM*** sous le contexte du Microsoft .Net.

Cette étape représente des avantages au niveau stratégique et technique. D'une part, en offrant au client une partie de code qui s'exécute sous .Net, on va le pousser à préparer l'environnement pour la version Omega.Net et par suite lui permettre de profiter de tous les atouts de la nouvelle version et cela se fait pour les clients qui souhaitent migrer vers la technologie .Net.

D'une autre part, en ce qui concerne les développeurs, ils seront capables de créer tout nouveau module en se basant sur les bases du .Net en respectant les normes et les étapes suivantes mises en place en collaboration avec le directeur technique.

Durant cette phase, je me suis confronté à plusieurs contraintes sachant qu'on était obligé à gérer deux versions en même temps. La maintenance sur l'ancien code écrit sous Win 32 et le développement du nouveau sous .Net devait se faire en parallèle.

En aucun cas les demandes des clients devaient être ignorées ou gelées et en même temps il était hors question d'être pénalisé sur l'avancement du travail sur la nouvelle version.

Au niveau technique, les nouveaux modules demandés par les clients seront créés directement sous .Net diminuant ainsi le nombre de modules à restructurer durant les étapes suivantes de la migration.

La compilation du code est passée par plusieurs phases que je détaillerai avec leurs problématiques et les solutions trouvées dans ce qui suit.

IV.1 Phase de recherche

La première phase de la migration portait sur une recherche concernant les méthodes de migration du code Delphi Win32 vers Delphi.Net. Cette phase était nécessaire pour préparer tous les changements qui devaient être appliqués à l'ancien code.

Il est possible de porter des applications VCL (Annexe A-1) développées sous une version antérieure de Delphi (y compris des applications 16 bits développées sous Delphi 1).

En fait, la société Borland fournit à ces utilisateurs l'ensemble des librairies VCL.Net équivalente aux librairies Win32 offrant ainsi un moyen transparent aux développeurs Delphi d'utiliser les librairies Microsoft .Net Framework. Ces libraires VCL.Net permettent de réduire la taille du travail à faire pour la compilation d'***Omega PM*** en .Net.

En Effet, la migration pointait principalement sur les points suivants :

- Migration des types de pointeurs.
- Migration des types.
- Création et destruction d'objets.
- Appel de l'API Win32.
- Utilisation des ressources.

Les résultats de la recherche ont été documentés et mis à disposition des différents acteurs de la migration pour permettre un meilleur partage de l'information et par suite une meilleure rentabilité.

IV.1.1.2 Normes de codage et de nomenclatures

Avant de commencer les étapes de migration, on devait mettre en œuvre une norme de codage, c'est pourquoi un document a été rédigé pour mettre en place un style standard pour le formatage du code Delphi. Ce document s'est inspiré des documents des règles de nommage élaborés par l'équipe Delphi ([1] Chapitre 25)

Ce document a abordé en particulier les points concernant :

a)Organisation du fichier du code source

Le fichier du code source doit avoir un format standard qui permet de faciliter sa lecture et de donner des informations sur le code en cours.

Dans la déclaration ci-dessous on a un exemple d'entête d'un code Delphi.

```
{*******************************************************}
{       Omega Delphi Visual Component Library           }
{                                                       }
{      Copyright (c) 2006,2007 Omega Financial Solutions}
{                                                       }
{*******************************************************}
unit Buttons;

{$S-,W-,R-}
{$C PRELOAD}

interface

uses
Windows, Messages, Classes,  Controls, Forms, Graphics,
StdCtrls, ExtCtrls, CommCtrl;
```

b) Les conventions de nommage :

- Nommage des unités
- Nommage des classes ou des interfaces
- Nommage des champs
- Nommage des méthodes
- Nommage des variables locales
- Les mots réservés
- La déclaration des types
- Nommages des écrans

c)La gestion de l'espace blanc dans le code :

La gestion de l'espace blanc dans le code est importante pour une meilleure lisibilité du code. Le document décrit dans quels cas il faut espacer le code et quand il faut les blancs dans le code. Il est préférable par exemple de ne pas avoir plus que 80 caractères par ligne de code pour éviter le défilement avec la souris.

d) Gestion des commentaires

La documentation d'une application est l'une des tâches les plus importantes à faire. Sans cette tâche, il serait difficile de suivre les fonctionnalités incorporées dans le système.

Un document a été rédigé dans lequel sont mentionnées toutes les normes pour commenter le code de manière à pouvoir générer la documentation du code automatiquement grâce à un outil gratuit installé(NDOC).

Le processus de la génération de la documentation peut être résumé en trois étapes principales:

1. Document votre code source Delphi en utilisant les balises de documentation XML.

2. Extraction des balises de commentaires dans le code dans un fichier XML.

3. Passer à NDOC le chemin du fichier XML pour générer la documentation dans des fichiers CHM (*Compiled HTML*) qui sont les fichiers d'aide de Windows.

IV.2 Phase d'analyse du code

Une fois la phase de recherche terminée, nous devions procéder à l'analyse du code Omega.

En Effet, pour pouvoir mettre les règles de la migration et du personnel qui va les appliquer, il a fallu faire un certain diagnostic ou rapport du coût de cette compilation en se basant sur plusieurs paramètres comme :

- Le nombre de lignes de code.
- Les types et fonctions utilisés devenus obsolètes en .Net

Cette partie concerne le code de point de vue structure et contenu. Cette opération peut s'avérer délicate si l'on ne prend pas le soin de s'entourer d'un certain nombre de précautions et de respecter certaines règles.

Le résultat de cette étape était un plan de projet établi par le chef du projet dans lequel sont détaillées les différentes phases Concernant l'analyse du code et toutes les manipulations nécessaires pour créer un environnement de développement sain qui va nous permettre de compiler l'application sans oublier des composants à intégrer.

La figure ci-dessous montre le portail d'Omega PM qui est organisé par modules qui apparaissent comme des répertoires à gauche de l'écran. Chacun de ses modules représente physiquement un répertoire sur le disque qui contient les différents fichiers (section 1.1.1 du chapitre en cours) constituant le projet.

Chaque Module va être transformé en un assemblage de fichiers compilés qui vont former la librairie «.dll ».

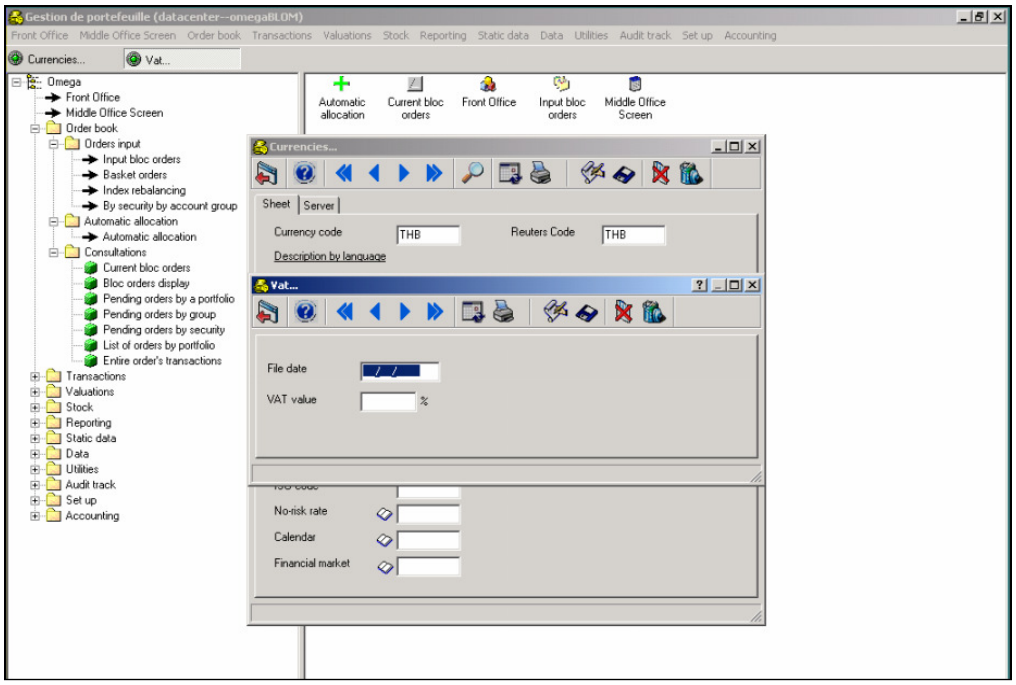

Figure 22: Ecran Omega

IV.3 Installation du CodeGear et gestion des composants

Une fois la planification faite, l'équipe nommée pour la migration procède à la migration en commençant par l'installation de l'IDE Code Gear. Cette installation est documentée pour qu'elle soit utilisée par les développeurs lors de l'installation
La migration d'Omega PM comporte deux parties :

- La migration des composants tiers (*third party*)
- La migration des modules Omega.

IV.3.1 Migration des composants

Une fois l'utilitaire du développement installé, nous procédons par la première partie de la migration celle des composants tiers.

Pour chaque composant, nous devons contacter le vendeur correspondant pour récupérer la version .Net.

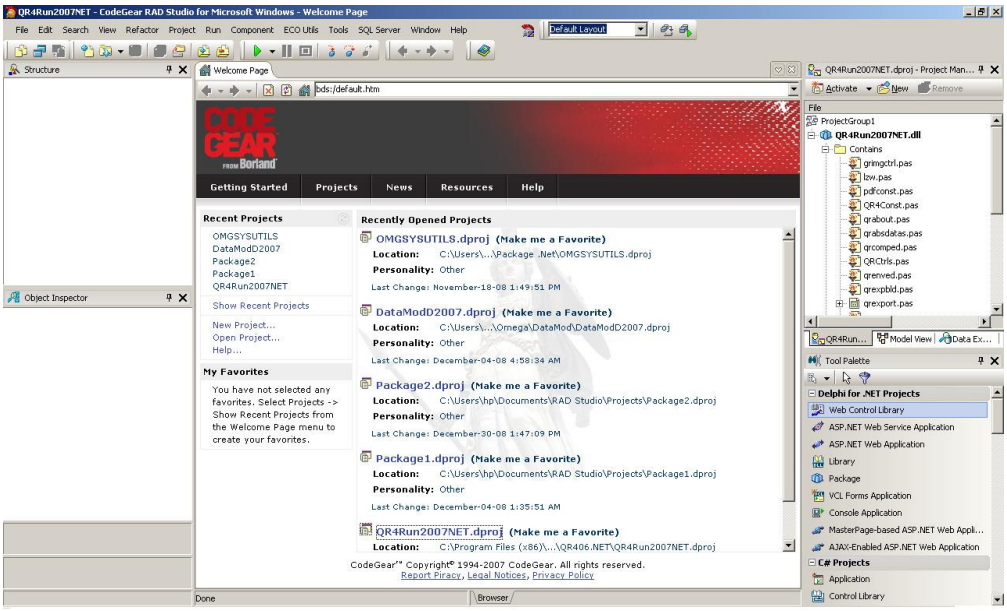

Figure 20:Composant QuickReport recompilé

La plupart des fournisseurs avaient déjà migré leurs composants vers la plateforme .Net et la mise à jour s'est déroulée sans aucun problème de notre côté.

La méthode d'installation de chaque composant est documentée pour faciliter l'installation aux développeurs.

Mais le problème se pose lorsqu'on constate des composants intégrés dans le langage Delphi6 qui n'ont pas été migrés comme par exemple le composant de navigation web ***TWebBrowser*** pour lequel aucun plan de migration n'a été prévu et du coup la compilation bloquait là où le composant est référencé dans le code. Plusieurs relances ont été faites auprès du personnel de support Delphi pour avoir une solution à ce problème qui était loin d'être résolu si je n'avais pas contourné le problème pour utiliser le composant grâce à un ***Wrapper*** qui enveloppera toutes les fonctionnalités du composant mais sans avoir la possibilité de le paramétrer visuellement chose qui ne nous a pas pénalisé.

La Migration des composants consistait en sept étapes :

- La préparation du composant sous Delphi
- Le chargement sous Delphi.net :

Il faut ouvrir le fichier package du composant(.dpk),les éléments du composants sont visibles dans le gestionnaire de projet.

- Modification de certains fichiers :

- Modification du package :

Les seules modifications essentielles du package concernait la liste des « package(.dpk)»(Annexe C,dpk) requis par ce composant et ces derniers ne sont pas toujours les même dans delphi.net et il m'a fallu chercher leur équivalent en VCL.Net.

Une fois le changement fait, je lançais la compilation du package et si aucune erreur n'est signalée, le composant est donc prêt à être installé.

- Installation du composant :Comme tout langage Delphi.net possède ses outils pour pouvoir installer de nouveaux composants afin de les visualiser dans la palette d'outils sinon pouvoir les utiliser par simple référence.

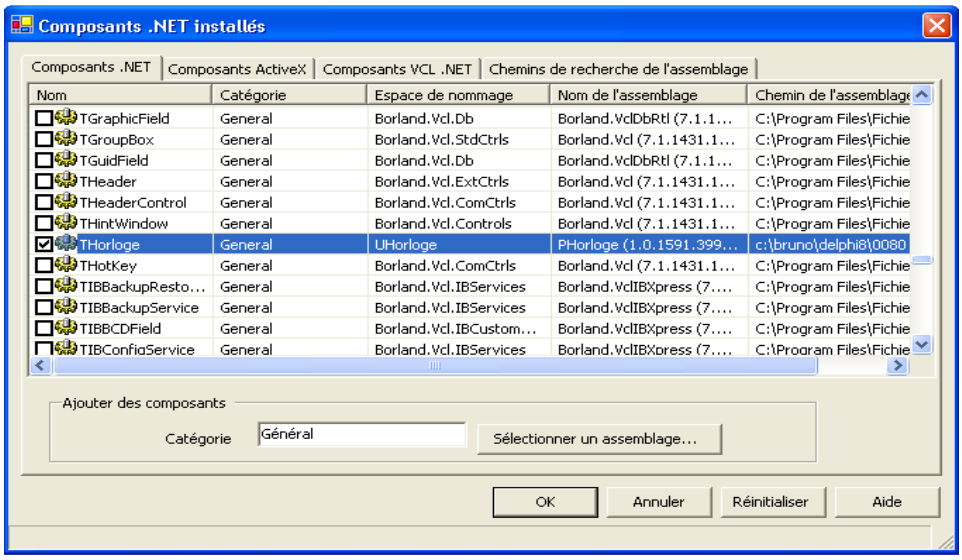

Figure 21:Listes des composant .Net installés

- Vérification de l'installation **:**

Après la fin de l'étape précédente le nouveau composant est installé dans l'EDI de Delphi .Net, mais pour s'assurer que tout est bien passé il me suffisait de fermer tous les fichiers ouverts et de démarrer un nouveau projet VCL vierge.

- Chargement de l'application de test du composant :

Le test de ma part se faisait sur les fonctions basiques du composant, mais un autre test d'intégralité se faisait après chaque livraison d'un composant migré.

Chaque package ou paquet créé devait être déposé sur le serveur des sources où il était accessible à toute l'équipe de la migration sachant que la dépendance entre les modules à migrer oblige une communication entre les différents développeurs travaillant aussi sur la migration.

Parfois on avait du retard et l'ordre de terminaison des modules à compiler n'a pas été respecté, mais le travail ne s'arrêtait grâce à une simulation d'appel des fonctions lorsqu'un package avait besoin d'un autre non encore livré.

Ce retard est parfois dû au contact avec les fournisseurs des composants quand c'était nécessaire, soit pour acquérir de nouvelles versions, soit pour commenter à propos de quelques fonctionnalités omises par erreur de la part du fournisseur et il m'a fallu parfois aller dans le code source de ses composants et modifier puis compiler comme c'était le cas dans SDAC dont on avait le code source heureusement.

IV.3.2 Migration des modules Omega

Une des contraintes principales est de pouvoir compiler le même code sous Win32 et .Net.

En fait, on doit veiller surtout à garder le fonctionnel intacte et penser comment faire l'équivalence technique en cas ou un changement est nécessaire si l'ancien code ne compile pas directement.

Pour cela, il fallait trouver une solution technique pour pouvoir le faire. Toute recherche faite, nous avons établi une solution basée sur les directives de compilation.

- Au début par : {$IFDEF CLR}, pour indiquer si le code sera compilée avec La CLR(Chapitre III) ou {$IFNDEF CLR }pour les autres compilateurs.

- A la fin par :{$ENDIF} qui ferme toute directive ouverte.

Nom	Définition
VER180	Version du compilateur Delphi 2006
CLR	Indique que le code sera compilé pour la plate-forme .NET.
WIN32	WIN32 Indique que le système d'exploitation est Win32 API. Utilisez WIN32 pour distinguer les diverses plates-formes Windows, comme Windows 32 et 64 bits.

Table 6 : Exemple de choix du compilateur-Les directives conditionnelles

Lors de l'importation, Delphi .NET ajoute une référence au modèle de composant dans la clause ***Uses*** de la partie Interface de l'unité. Pour pouvoir permettre au code d'être compilé par les deux environnements, il a fallu modifier cet ajout automatique en utilisant un test de commutateur ou ce qu'on appelle les directives conditionnelles qui permettent de guider le compilateur au bloc de code qui doit être pris en compte selon la version voulue.

Exemple ;

```
uses
Windows, Classes, Graphics, Forms, Controls,
{$IFDEF CLR} System.ComponentModel, {$ENDIF} StdCtrls;
```

Le code ce dessus veut dire si la compilation est à faire en .Net alors référencer la librairie *System.componentModel* comme référence.

IV.3.4 Création des paquets

Lors de l'implémentation de cette étape on s'est confronté à plusieurs problèmes :

1)- Le langage Delphi prend en compte la migration d'une version à une autre pour cela il met à disposition une fenêtre de paramétrage des options du projet pour minimiser l'intervention des développeurs et par suite le temps global de la migration. Cette méthode a été essayée sur une portion du code mais des changements supplémentaires ont été nécessaires, chose qui a poussé à créer de nouveaux paquets avec de nouveaux espace de nommage.

2)-Le problème principal à ce stade a été détecté à un stade avancé et était imprévu.

La compilation du code sous .Net avec l'ajout de certaines références sur d'autres librairies retournait parfois des échecs de compilation. Une solution qui nécessitait d'écrire un petit bout de code mais qui nous a libéré du blocage consistait à invoquer les méthodes des librairies dynamiquement en passant le nom de la librairie ainsi que de la méthode à invoquer et ses paramètres. Ainsi, la dépendance entre les librairies n'existe pas à la compilation mais à l'exécution.

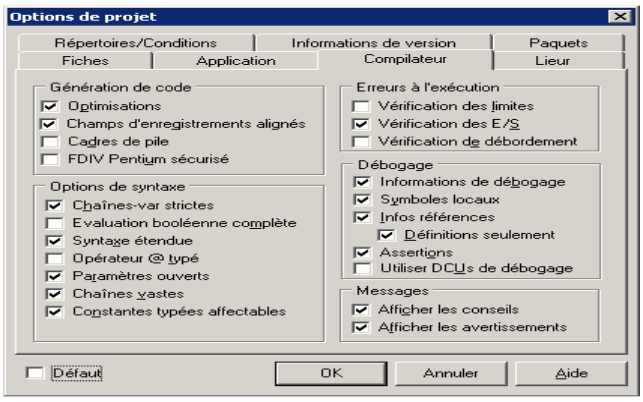

Figure 22: fenêtre Options du projet

Les écrans et les classes doivent être ajoutés au paquet dans CodeGear pour ensuite essayer de le compiler et faire les modifications nécessaires pour récupérer le fichier « .dll » ou le code final compilé (Annexe C,dll).

3)- les fichiers ayant une extension « .DFM » ne s'ouvrent pas dans le designer VCL.Net. Pour pouvoir visualiser correctement les écrans dans le nouvel outil de développement, il a été utile de créer les fichiers ayant l'extension « .NFM » pour la VCL.NET à partir de ceux de WIN32 ayant l'extension « .DFM ».La conversion a été possible après le développement d'une petite application qui prend en entrée le chemin d'accès au fichier « .DFM » et crée en sortie un nouveau de type « .NFM » dans le même

répertoire. L'annexe C donne un aperçu sur les différents types de fichiers présents dans un projet Delphi et leur utilité.

Les modifications au sein du code varient d'une classe à une autre selon les types et les fonctions utilisées, car il existe des types qui n'existent plus en VCL.NET ainsi que des fonctions qui sont devenues obsolètes.

IV.3.5 Optimisation du Code

Les nouvelles librairies équivalentes aux anciennes de point de vue fonctionnel ont permis une optimisation du code inattendue. En effet, la fonction composée de dizaines de lignes dans l'ancienne version de Delphi n'est en VCL.Net que quelques lignes et parfois intégrée dans la Plateforme .Net directement .La vitesse d'exécution du code a augmentée de même grâce aux nouvelles méthodes d'accès dans VCL.Net ainsi que la destruction automatique des objets devenus inutiles en mémoire grâce au *Garbage Collector.*

Le changement apporté au code n'a pas toujours été prévisible. Une liste de types et de fonctions a été générée montrant toutes les équivalences en .Net pour des bouts de code à migrer.

IV.4 Documentation

Toute la Compilation.net a été documentée étape par étape depuis l'installation de ***CodeGear*** jusqu'à la création de l`exécutable final « Omega.exe » .La documentation a été importante pour accélérer le processus de modification du code et de la manipulation des composants et de tracer surtout les différents problèmes et la manière de les résoudre.

Préciser la méthodologie de travail dans un projet est essentielle pour permettre une meilleure gestion, chose qui le rend uniforme facile à lire et à maintenir.

IV.5 Phase de Test

Le fait de garder une version compilée en win32 permet de comparer la nouvelle version à l'ancienne et de valider les résultats. Plusieurs tests basés sur le plan de test initial ont été effectués.

On essaye cependant, heuristiquement, de faire en sorte que si un bogue est présent, le test le mette en évidence, notamment en exigeant une bonne couverture des tests :

- couverture en points de programme : chaque point de chaque module compilé a été testé au moins une fois.

- couverture en chemins de programme : chaque séquence de points de programme possible dans une exécution a été testée au moins une fois de notre part avant de la passer à l'équipe de test.

- couverture fonctionnelle : chaque fonctionnalité métier de l'application a été vérifiée par au moins un cas de test.

On n'a pas attendu la compilation de tous les modules pour débuter les tests. En effet, les jeux de tests commençaient dès qu'une fonctionnalité se terminait.

La dépendance entre les modules nous obligeait à compiler un code avant un autre, comme c'était le cas pour les fonctions communes à plusieurs modules.

J'ai du écrire des bouts de code mis à la disposition des testeurs avant le déploiement de la version finale. Ce code a servi à simuler unitairement l'appel des méthodes des librairies grâce au principe de réflexion (appelé reflection en Anglais) ou l'invocation dynamique du code (Annexe D) supporté par Visual studio. Le nom de la fonction à tester est passé en paramètre ainsi que ses propres paramètres d'entrée, La réflexion permet de charger un assemblage ou une librairie, de découvrir des types et d'invoquer leurs membres à l'exécution.

Les résultats sont affichables ou exportables dans un fichier Excel pour permettre de les circuler entre les testeurs et les développeurs en cas des erreurs ou de résultats incorrects.

Les tests unitaires passés, une version de l'application compilée en .Net a été livrée sur l'environnement de recette où est installée l'ancienne version win32.Le testeur peut ainsi se connecter sur les deux versions en comparant les résultats des mêmes scénarios de tests sur les mêmes écrans.

Des erreurs (ergonomiques et fonctionnelles) ont été remontées, mais en général les tests étaient concluants. Il faut savoir que le code compilé ainsi que le code SQL ont été aussi optimisés, chose qui a augmenté les performances de l'application et cela a été noté en comparant les temps de réponses.

Chapitre V

Conception des composants de l'application

Une fois l'application Omega compilée en .Net. On a abordé la première phase de la migration en architecture SOA qui consistait à créer un Framework Omega.Net qui a le rôle d'uniformiser et consolider le code source commun d'**Omega PM** tel que l'accès à la base de données et la gestion des erreurs et l'audit sous formes de librairies .Net. Après plusieurs jours de recherche sur les méthodes de mise en place de ce Framework, le choix du point de départ est tombé sur l'extension gratuite *Enterprise Library de Microsoft* qui est un ensemble de librairies dont le code source est accessible et modifiable.

Le code source de ce Framework apportait une partie de la solution mais pas la solution entière, c'est pourquoi j'ai travaillé sur la personnalisation de ce dernier pour qu'il s'adapte de la façon la plus étroite à nos besoins d'exposer le fonctionnel en des services indépendants pouvant communiquer avec d'autres langages et tout en étant facile à maintenir et à évoluer. Ces objectifs ne peuvent être atteints sans une restructuration des composants de façon à séparer toutes les règles métiers du code frontal (Chapitre I Section 5).

Le Framework contiendra des composants appartenant à un des groupes suivants :

1-le stockage des données.

2-la logique applicative.

3-la présentation

La conception de notre propre Framework Omega selon l'architecture orientée service s'est déroulé en sept étapes:

- Installation et compilation des différents Blocs d'application
- Conception des tiers de données. (Chapitre II section 2.1)
- Conception des tiers de gestion
- Conception des composants de gestion des exceptions
- Conception des composants d'audit.
- Conception des composants de sécurité
- Conception des interfaces utilisateurs

V.1 Installation et compilation des Enterprise Library(EntLib)

Le Framework **Enterprise Library** est une extension du Framework **.NET**. Cette extension peut être téléchargée gratuitement sur le site Microsoft .On peut être demandé de s'enregistrer pour pouvoir compléter le téléchargement.

L'installation fournit les librairies et le code correspondant qui a été travaillé à son tour pour construire nos propres librairies. Dans ce qui suit, je détaillerai les différentes étapes de construction du Framework Omega.Net, ainsi que ma contribution personnelle dans chacune d'elle.

V.1.1 Configuration requise

Pour pouvoir compiler et utiliser le nouveau Framework, il est important de disposer des éléments suivants:

1. Système d'exploitation Microsoft Windows 2000, Windows XP Professionnel,Windows Server 2003 ou Windows vista
2. Microsoft .NET Framework 2.00 ou version ultérieure.
3. Microsoft Visual Studio® .NET 2008.

Après avoir installé les blocs d`application, nous allons procéder à leur mise en œuvre.

V.1.2 Fichiers de configuration

Les librairies développées se basent sur un fichier de configuration de type XML où elles trouvent le paramétrage nécessaire pour fonctionner.

Le fichier de configuration est essentiel pour le paramétrage de l`application en ce qui concerne les variables globales comme la chaîne de caractères qui contient la connexion à la base de données et d`autres données qui pointent vers les différentes librairies et leur emplacement physique ainsi que tous les différents espaces de nommage indispensables surtout lors de la création dynamique des objets.

Dans l'ancienne version d'Omega PM, la configuration de la chaine de connexion était gérée dans un autre fichier ***.OCF*** accessible via un utilitaire **conadmin.exe** développé pour ce but**.**

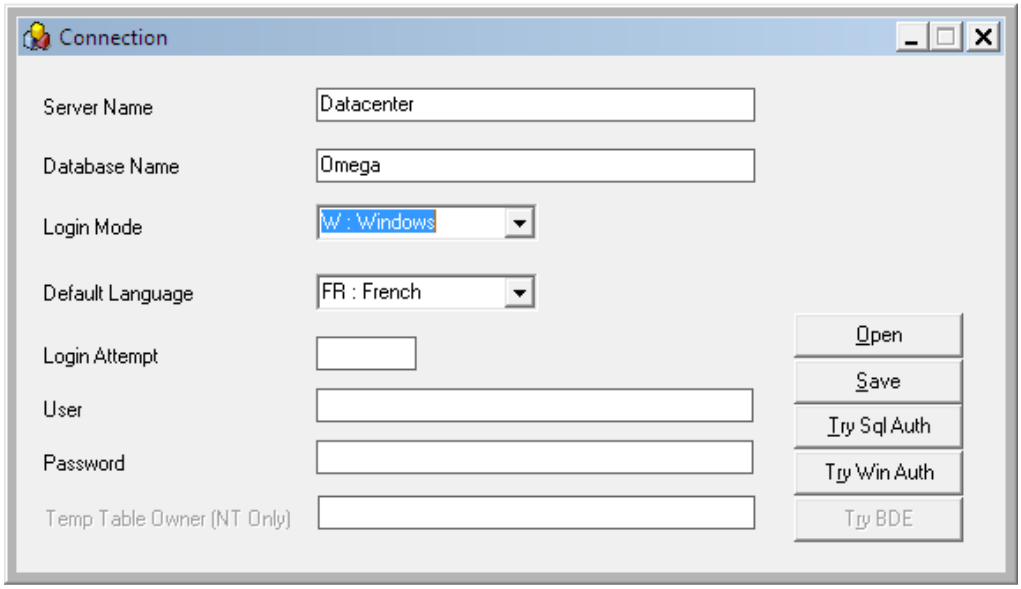

Figure 23 :Ecran de gestion de la connexion a la base de donnees Omega

Le problème qui s'est posé, est que la nouvelle librairie d'accès aux données a besoin de lire la chaîne de connexion dans le fichier de configuration comme on verra dans ce qui suit.

Donc, il a fallu retravailler le code **conadmin** de manière à faire passer la chaine de connexion au fichier de configuration pour que les nouvelles librairies fonctionnement normalement.

Ce fichier ***XML***(*App.config*) contient aussi les paramètres de configuration propre aux blocs d`application.

Un autre problème commun auquel on a été confronté est souvent relié au déploiement de ce fichier sans lequel l'application ne démarre pas et génère une exception. La maîtrise de toutes les sections du fichier est nécessaire pour pouvoir faire le paramétrage initial.

V.2 Conception des tiers de données.

Le but de créer la couche d'accès aux données ne s'arrête pas sur le fait de séparer le code frontal de celui qui permet la connexion aux entrepôts de données.

La gestion d'un projet de la taille de cette migration doit anticiper les demandes futures des clients, comme par exemple la migration de la base de données d'un fournisseur à un autre.

Il faut rendre paramétrable l'accès aux données et éviter tout codage en dur, c'est pourquoi on a pensé à regrouper tout le code SQL dans un fichier .SQL ou dans des procédures stockées.

Les nouveaux composants ont pris en compte les catégories d'IHM qui ont été détectées pour pouvoir créer une modèle de chacune à partir duquel toutes peuvent hériter.

Parmi ces interfaces, on trouve celles qui sont simples comme par exemple celles qui gèrent les tables de référentiel à l'exemple de la table des devises par exemple. Une autre catégorie d'interface est commune dans l'application [Chapire VI,section1.2].

Les **blocs d`application** ont été conçus à la base pour répondre à la gestion des entités statiques et non dynamiques. Nos besoins consistaient à gérer les données dynamiquement sans se limiter à des entités figées. Nos entités deviennent des tables stockées dans un conteneur ayant des caractéristiques assez importantes. Ce conteneur est appelé le Dataset.

Les tables de la base de données peuvent être modifiées à tout moment par les développeurs selon le besoin du client ou par un utilisateur ayant des privilèges avancés, et cette modification ne porte pas sur les données uniquement, mais sur la structure de la table elle-même.

Le code pour la persistance se révèle vite long, fastidieux et difficile à écrire et à mettre au point dès que le modèle objet se complexifie et qu'on recherche des bonnes performances dans un environnement chargé.

La solution consiste à stocker les données dans un objet dynamique qui peut contenir les données et les organiser d'une manière identique à leur entrepôt initial. Cet objet n'est que le Dataset fourni le par le ***.Net Framework*** et plus précisément le ***ADO.NET***.

Les principales avantages de l`approche Dataset sont les suivantes :

- C`est un moyen performant d`échanger des données complexes avec d`autres composants ou applications
- Le stockage des données dans un Dataset est un bon moyen d`échanger des données entre les différentes couches applicatives.

- Le travail en mode déconnecté permet de ne pas monopoliser les connexion a la base de données. C`est particulièrement utilise pour réaliser des traitements longs sur chaque enregistrement.
- Intègre des fonctions natives de manipulation XML de sérialisation
- Permet de parcourir plusieurs fois les mêmes données, les trier, les filtrer
- Facilite le codage :
  - Les données issues du Dataset peuvent être liées directement a des contrôles graphiques.
  - Le code généré dans le cas des Dataset typés permet de naviguer plus facilement dans le modèle objet du Dataset.
  -

La conception de cette couche a nécessité un travail dans la configuration du bloc pour profiter de la généricité d'accès aux données apportée grâce à *design pattern Factory* (Chapitre II section 3.3.1) .Dans ce qui suit je vais détailler les modifications apportées pour sortir avec une couche de données qui répond aux besoins actuels et futurs avec la possibilité d'extension.

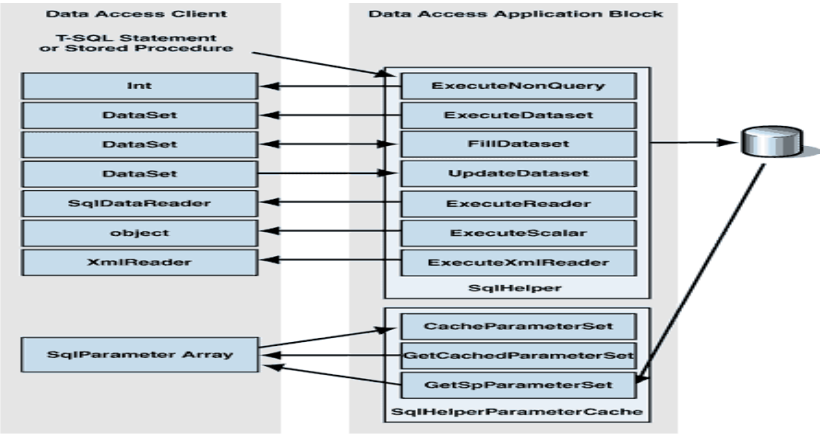

Figure 27: Type de données et les fonctions d'appel dans le Data Access Block

Des modifications ont été apportées à ce composant ainsi que des additions de code comme je vais le montrer dans ce qui suit :

V.2 .1 Création du fichier de configuration des commandes SQL

En m'inspirant du fichier de configuration globale qui permet le paramétrage principal de la solution, je me suis penché vers la configuration de l'accès à la base de données et la manipulation des données. Je m'explique :

L'idée consiste à grouper tout le code SQL(Ajouter,Modifier,Supprimer) dans un Dataset Typé ou un Fichier XML, chose qui centralisera le code et le rendra plus facile à maintenir. Cette méthodologie de travail a pris en compte :

- Le concept des tables maîtres-détails.

- Le mode transactionnel d'exécution.
- Les commandes et leurs paramètres.
- liaison des colonnes de la table aux paramètres.

Le schéma suivant montre un Dataset qui est l'entité ou le modèle sur lequel se base le fichier de configuration en question avec ses différentes tables et les relations qui les lient.

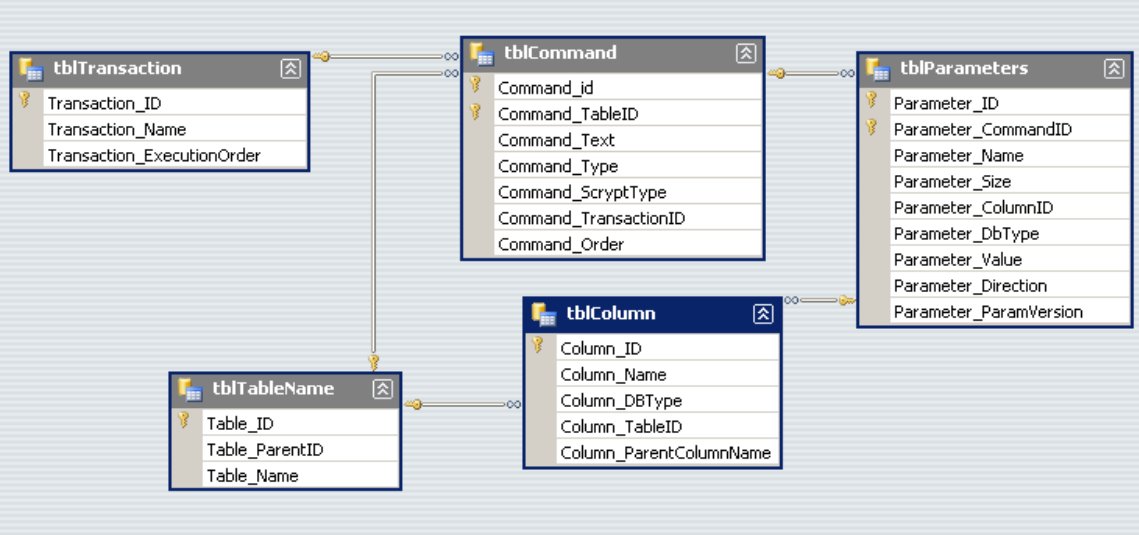

Figure 24: Structure des tables de configuration d'accès aux données.

Nom de la Table	Description
tblTransaction	Table spécifiant les transactions
tblTableName	Table Contenant les noms des tables en question avec leur parent s'il existe.
tblCommand	L'ensemble des commandes par table
tblColumns	Ensemble des colonnes par table(utile pour le mapping entre la colonne et son paramètre)
tblParameters	Ensemble des paramètres des commandes

Tableau 3 : Description des tables

La construction du Dataset selon la logique fixée pour chaque traitement. Toutes les tables qui seront utilisées doivent être déclarées dans la table *tblTableName*,ensuite il faut préciser leurs différentes Commandes SQL dans tblCommand avec leurs paramètres SQL s'ils existent dans *tblParameters* et enfin faire le *Mapping* entre les colonnes et les paramètres en cas ou la valeur du paramètre doit être tiré d'un certain champs durant l'exécution.

Le fichier de configuration créé, il faut qu'il passe un test de validation nécessaire pour s'assurer de la validité de la syntaxe SQL, ainsi que toutes les relations entre les entités de configuration.

V.2 .2 Validation du fichier de configuration des commandes SQL

Le fichier de configuration des commandes SQL est construit par le développeur selon le module à développer. Pour minimiser le risque d'erreur et par suite accélérer le temps de développement, j'ai conçu une fonction qui prend en entrée le fichier construit et retourne un autre fichier qui sera le résultat de la validation effectuée sur le format du fichier de configuration ainsi que les données qu'ils contient comme la syntaxe SQL qui serait préférable de tester avant l'appel sur le serveur réel de base de données.

V.2 .3 Automatisation de la gestion du Magasin des données

L'automatisation de la gestion des données est une tâche qui va permettre de normaliser les méthodes d'appels à la base de données ainsi qu'elle va permettre un gain dans le temps de développement. L'automatisation porte sur la lecture, l'écriture et la suppression des données.

V.2.3.1 Lecture des données automatiques

Avant l'arrivée de l'extension du Framework,le développeur devait écrire plusieurs lignes de codes pour pourvoir se connecter à la base de données et manipuler ses tables :

- Créer la connexion
- Préparer la commade SQL et ses parametres
- Instanciation des objets qui vont recevoir les donnees
- Invocation du code SQL
- Récuperation du resultat de la requete avant l'affichage.

Parfois la différence entre un appel et un autre ne diffère que par le nom de la procédure stockée et quelques paramètres ;mais la préparation de l'appel reste un travail répétitif qui consomme des ressources.

Avec les enterprise library et les modifications effectuées sur les différentes librairies,le développeur n'a plus besoin de se mêler de la création de la connection et l'instanciation de plusieurs objets pour la lecture des données .

Ce que je veux dire par lecture automatique, c'est le fait d'importer les données voulues dans une structure qui est le Dataset. Le développeur a une seule tâche qui est remplir le fichier de configuration (Tableau 3 : Description des tables) et le faire passer en paramètre à une fonction dans la couche d'accès aux données.

La lecture du fichier de configuration dans un Dataset permet son passage en paramètre d'entrée à une fonction avec une liste des codes des tables à retourner dans le Dataset de retour.

Il en est de même pour la modification des données qu'on détaillera le principe dans ce qui suit.

V.2 .3.2 Sauvegarde des données automatiques

La manipulation des données impose une lecture et une écriture des tables de la base. Une Fonction joue le rôle de la lecture du fichier de configuration pour préparer les commandes SQL nécessaires à sauvegarder les changements qui ont lieu dans le Dataset.

Autrement dit, les données récupérées via la requête SQL sont stockées dans un Dataset. Ces données sont ensuite modifiées via l'interface pour ensuite être renvoyées au serveur pour la mise à jour.

L'importance de la mise à jour automatique est que le développeur n'a pas à créer les commandes de sauvegarde pour chaque écran. Il suffit de bien paramétrer le fichier SQL et passer le Dataset modifié en tant que paramètre à la méthode responsable à la mise à jour de la base.

ADO.net permet de détecter la nature de chaque modification dans le Dataset,ce qui permet de préparer les commandes SQL qui seront exécutées coté serveur.

Une insertion prendra en compte l'auto incrémentation de la clé primaire en cas où elle existe et toutes les contraintes.

L'ordre d'exécution est lu dans le fichier de configuration des commandes SQL. Cette fonction prend en compte le principe des tables Maitres-Détails et gère la lecture des valeurs des paramètres à la source.

La logique de cette fonction prend en compte les transactions SQL ainsi que la gestion des exceptions lors des erreurs avec un système d'audit.

V.2 .4 La sérialisation des données et des paramètres

Le but de la migration pose une contrainte de type pour ce qui concerne la communication via les services WEB et par prévention, la conception des couches et surtout les fonctions publiques doivent prendre en compte le type de retour qui est en général de type texte (format XML) pouvant être désérialisé pour reconstruire l'objet.

Face à cette contrainte j'ai développé une classe contenant les fonctions de sérialisation et de désérialisation des données de type différents portant sur des données devant être sérialisé en un Dataset ou des paramètres SQL.

V.2 .5 Les types des SQL scripts et les méthodes d'invocations

La configuration pour l'accès à la base de données invoquée précédemment montre que le code SQL peut être de types différents :

- Texte
- Procédure stockée
- Fichier

Le type du Code SQL est passé en tant que paramètre d'entrée dans les fonctions accédant à la base de données.

Exemple d'appel :

```
public string ExecuteNonQueryCommand(string command, DbParameter[]
ParamArray, string ScryptType)
```

La fonction précédente donne un échantillon sur l'appel des méthodes,leur types de paramètre en entrée et le type de retour qui est un *string* résultant de la serialisation évoquée precédemment.

Lors de l'appel le developpeur qui a besoin d'executer une commande SQL n'a quà envoyer la script SQL,un tableau de parametere et la nature du script(Texte,Procedure Stockée ou fichier SQL)

V.2 6 Génération automatique du code SQL

Le .Net Framework possède des outils de génération du Code SQL comme *SqlCommandBuilder*.Ce dernier permet à partir de la commande « Select », de générer celle de l'insertion, de modification et de suppression des données.

Cette génération possède des limitations, car les commandes INSERT, UPDATE ou DELETE créées sont pour des tables autonomes et les relations avec les autres tables au niveau de la source de données n'ont pas été prises en compte.

En conséquence, il est possible que l'appel à Update par exemple, pour soumettre les modifications pour une colonne qui participe à une contrainte de clé étrangère dans la

base de données, échoue. Pour éviter cette exception, j'ai dû spécifier explicitement les instructions utilisées pour effectuer l'opération.

La logique de génération automatique de commandes échoue si les noms de colonne ou de table contiennent des caractères spéciaux (notamment espaces, points, points d'interrogation ou autres caractères non alphanumériques). Notre générateur du code SQL a pris en compte les caractères spéciaux et évite l'occurrence des erreurs de syntaxe par l'utilisation des crochets. (Annexe D, Générateur de code SQL)

V.3 Conception des tiers de gestion

Dans cette couche une classe générale a été créée qui apporte les fonctionnalités de base pour les couches supérieures. La gestion de la sécurité, des erreurs, la validation ainsi que l'audit dans la couche métier est soumise comme dans les autres couches au principe d'abstraction qui fait que la configuration des règles de sécurité soit paramétrée manuellement ou par un code qui peut être invoqué selon le besoin.

C'est à la charge du développeur d'hériter les fonctions et de fonder dessus les couches métiers avancées spécifiques à chaque module *OmegaPM.*

La tâche principale dans cette phase est de s'assurer que l'invocation des fonctions de la couche d'accès se fait sans problèmes même si quelques fonctions d'accès étaient encore en cours de modification, cela n'a pas retardé le travail, car ce qui m'importait c'est la communication et le retour du type du résultat correctement indépendamment des données réelles.

La configuration n'était pas une tâche facile, il fallait changer les sections et tester toujours le déroulement de l'interception du code ou plus précisément son tissage et la gestion des événements.

Ce tiers contient trois composants :

- Les composants de gestion
- L'agent de service
- Les entités de gestion

V.3.1 Les composants de gestion

Notre application exige un composant qui implémente les règles de gestion et performe des tâches de gestion. C'est le rôle de composants de gestion qui sont incarnés dans la couche métier.

V.3.2-L'agent de service

Lorsque le composant de gestion a besoin d'utiliser des fonctionnalités assurées par un service externe, on a besoin d'assurer quelques codes pour gérer les sémantiques de communications avec ce service particulier. Dans notre application, l'accès aux fonctionnalités d'Omega PM peut être assuré par un service web XML. Pour pouvoir utiliser ce service, on devait écrire une classe contenant les signatures des méthodes implémentées par ce service WEB. Cette classe est généré par l'utilitaire WSDL.exe fournit par .Net Framework.

Ce composant est responsable de l'établissement de la correspondance entre le format de données exposées par le service et le format de l'application requis.

V.3.3-Les entités de gestion

La plupart des applications exige des données à être transférées entre les composants. Par exemple, dans notre application, une liste de problèmes doit être passée du composant d'accès logique de données au composant de l'interface utilisateur pour que la liste des problèmes soit affichée aux utilisateurs. Les données sont utilisées pour représenter des objets du monde réel comme problèmes ou applications. Les entités de gestions qui sont utilisées d'une façon interne dans l'application sont des Dataset mais peuvent être implémentés en utilisant des classes orientées objet qui représentent des entités du monde réel que l'application manipule.

V.4 Conception et développement de la politique de sécurité

Plusieurs principes généraux en matière de sécurité ont été pris en compte lors de l'élaboration de la politique de sécurité d'***OmegaPM*** dans la nouvelle architecture.

OmegaPM possède des méthodes de sécurité pour éviter toute vulnérabilité au logiciel mais sachant que cette sécurité était appliquée dans l'ancienne architecture, il a été primordial de la restructurer en un modèle plus abstrait qui peut répondre aux contraintes de la séparation des couches et de l'architecture orientée service en général.

Les Tests d'authentification et d'autorisation dans l`ancienne version ont été faits par des conditions réécrites à chaque appel par le développeur, ce qui augmente le volume du code et du risque d'erreur et par suite un temps de test supplémentaire pour chaque module.

La politique est de faire de la sécurité un module à part dont les mêmes fonctions seront réutilisées pour minimiser le nombre de lignes de code à écrire.

Pour mettre en œuvre cette politique j'ai du encore faire appel aux blocs d'application avec leur propre configuration pour la sécurité.

Le bloc de sécurité est un utilitaire très utile dans la sécurisation des applications. Il a les moyens d'authentifier et d'autoriser les utilisateurs. Ce bloc est semblable aux autres en ce qui concerne l'utilisation de la fabrique d'objet pour la création d'une instance du fournisseur de l'autorisation. Il a une seule méthode,(Authorize ou autoriser) qui renvoie une valeur booléenne indiquant si l'utilisateur est autorisé à la règle paramétrée dans le fichier de configuration déjà abordé précédemment.

V.5 Conception des composants d'audit

Pour une meilleure gestion de la migration, il fallait implémenter une fonctionnalité d'audit afin d'effectuer le suivi des activités de l'utilisateur et de l'entreprise dans l'application à des fins de sécurité. Généralement l'audit en local se fait en copiant l'erreur ou un message déterminé dans un fichier texte pas toujours structure. Je savais très bien que les fichiers d'audit sont autant importants pour le client que pour notre société, car c'est une réflexion de l'activité en coulisse d' ***Omega PM***.

V.5 .1 Log.xml v/s Log.txt

Le fichier texte ***log.txt*** à été remplace par ***log.xml*** dans le but de le rendre plus structuré de telle sorte a appliquer toutes les fonctions relative a une table de données en ADO.Net où des méthodes de d'insertion de mise à jour et suppression sont possible ainsi que la fonction de filtrage.

V.6 Conception des composants de gestion des exceptions

Il est important pour une application d'avoir une gestion centralisée des fonctions de gestion des exceptions(ou plutôt erreurs) pour permettre un meilleur audit des erreurs, une sélection dans le traitement, l'accélération du temps de développement et la minimisation du taux d'erreur.

L'intérêt de la gestion des exceptions et de l'architecture en général est l'apport d'un nouveau concept de travail.il faut prendre en compte a ce stage la bonne conception des exceptions et la façon de les présenter de façon qu'elles soient utilisées d'une façon optimisée et c'est ce que j'ai essayé de faire suite a la configuration et la conception des classes de base pour la gestion des exceptions.

La gestion des exceptions couvre l'interception et le rejet des exceptions, leur conception, la transmission des informations les concernant et leur publication auprès des utilisateurs.

V.6 .1 Conception des classes d'exception

Les classes d'exception ont été dérivées de la classe *ApplicationException*. J'ai décidé de construire notre propre exception pour ajouter des données à l'exception pour obtenir des fonctions supplémentaires.

On part généralement de deux branches d'exception principales : les exceptions métiers et les exceptions techniques. Cette conception a permis d'intercepter et de publier le type d'exceptions approprié dans les différentes parties de notre application.

V.6 .2 Transmission des informations sur les exceptions

Les exceptions fournissent un flux d'informations en amont. Il était important de penser à leur sérialisation pour qu'elles soient transmises en amont à travers les niveaux. Cela est important, en particulier lors de l'accès à une interface de service ou à une interface utilisateur à travers laquelle on ne souhaite pas transmettre l'exception textuellement mais plutôt sous la forme d'une traduction que l'appelant puisse activer, le tout sans exposer d'informations techniques ou métier sensibles sur notre application et le service (telles que la chaîne de connexion à la base de données en cas de connexion défectueuse) qui risqueraient d'être utilisées contre notre logiciel dont les fonctionnalités seront exposées en tant que services, ou la nécessité de la prévention.

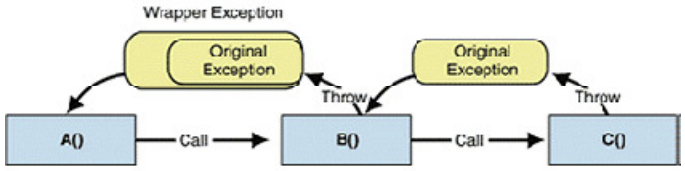

Figure 25 : Propagation des exceptions

V.6 .3 Publication des informations sur les exceptions

En cas d'exception, l`application doit en informer les personnes concernées. Les équipes de support technique doivent être informées des exceptions techniques et il est parfois nécessaire d'informer les responsables et les usagers du service d'assistance sur les exceptions métier. Chaque type d'audience souhaitera des informations d'environnement supplémentaires sur l'exception afin de remplir son rôle.

V.6 .4 Gestion des exceptions dans les composants d'interface utilisateur

Les composants du processus utilisateur devront gérer les exceptions provenant des composants du processus métier et des composants d'accès aux données et décider s'ils doivent effectuer les tâches suivantes :

- relancer l'opération.
- exposer le problème auprès de l'utilisateur.
- interrompre, relancer ou poursuivre le flux de l'interface utilisateur de l'application.

Les composants du processus utilisateur peuvent avoir besoin de masquer des exceptions à l'utilisateur, selon l'opération. Si l'exception doit être affichée, le processus utilisateur liera probablement l'exécution du contrôle à une représentation visuelle de l'erreur et ne la transmettra pas à l'appelant.

V.6 .5 Gestion des exceptions dans les composants du processus métier

La gestion des exceptions dans les composants métier requiert souvent l'interception des exceptions et des erreurs renvoyées par les objets métier et leur extraction sous la forme d'une exception qui puisse être comprise par l'appelant. Les composants métier doivent gérer les exceptions des composants d'accès aux données. Il s'agit notamment :

- des exceptions techniques (par exemple, l'échec d'une connexion à une base de données) ;
- des exceptions métier (par exemple, la violation d'une contrainte de clé étrangère).

Les composants métier doivent créer de nouvelles exceptions dans les cas suivants :

- L'appelant essaie d'effectuer une opération, mais dispose de données insuffisantes ou incorrectes.

- Une violation de contrainte se produit au cours d'une opération.

Les composants métier doivent propager toutes les exceptions des composants d'accès aux données, par exemple dans les cas suivants :

- Des problèmes techniques surviennent lors de l'accès aux données ou aux erreurs entraînées par les composants d'accès aux données d'arrière-plan. La plupart de ces exceptions peuvent être propagées sans nouveau recours au bouclage.

- Il faut utiliser un schéma de verrouillage optimiste :Cela se produit généralement lorsque les entités métier sont utilisées à partir des couches d'interface utilisateur et une mise à jour risque d'écraser des données qui ont été mises à jour depuis leur dernière lecture.

En général, les composants métiers ne doivent masquer aucune exception produite par les couches qu'ils appellent. Masquer des exceptions risque d'induire les processus métiers en erreur en ce qui concerne l'état transactionnel et faire croire à l'utilisateur que certaines opérations ont réussi.

Les exceptions ont été publiées dans les couches métiers, car c'est là que le résultat des transactions se trouve et que les accords au niveau du service interne sont définis.

V.6 .6 Gestion des exceptions dans les composants d'accès aux données

Les composants d'accès aux données devront généralement gérer deux classes d'exceptions principales :

- les exceptions dérivant des erreurs techniques lors de la connexion au magasin de données et lors de son invocation ;
- les exceptions métiers dérivant de procédures stockées qui implémentent une logique métier comportant un grand nombre de données.

La gestion des exceptions dans les composants de données requiert souvent l'interception des exceptions et des erreurs renvoyées par la source de données sous-jacente (ou l'API d'accès aux données) et leur mappage vers le schéma d'exception utilisé dans le reste de l'application. Les composants d'accès aux données doivent propager les exceptions, les bouclant automatiquement sous forme de types d'exception compréhensibles par leurs clients. Le bouclage automatique des exceptions en deux types d'exceptions principaux (exceptions métier et exceptions techniques) améliore la structure de gestion des exceptions et la logique de publication des exceptions pour les différents appelants éventuels.

- La fonctionnalité permettant de mapper des exceptions de sources de données (par exemple, **SqlExceptions**, qui représentent les erreurs de SQL Server soulevées par **RAISERROR** dans les procédures stockées) vers notre schéma d'exception de l'application basé sur la technologie .NET a été implémentée dans les composants d'accès aux données.

V.6 .7 Gestion des exceptions dans les composants d'entités métiers

Les entités métiers peuvent être appelées à partir des composants de processus métier ou d'interface utilisateur, c'est pourquoi il est important de générer et de propager des exceptions pouvant être consommées par les deux.

Il a été important de compter sur une bonne stratégie de gestion des exceptions et de filtrer les exceptions et de les classer selon des priorités d’importance, chose qui permet un retour sur erreur plus précis et rapide.

V.7 Conception des composants de validation des données

Le principe de la validation apporté par .Net et son extension *Entlib* promettaient une efficacité et une accélération dans le temps développement.

La validation des blocs d'application compte sur les règles et les attributs.les validateurs étaient propres aux propriétés des entités statiques conçues à l'avance en d'autres termes j'ai remarqué que pour pouvoir utiliser ces validateurs il faut avoir une entité possédant des propriétés accessibles.

V.7.1 Règles et attributs de la validation

L'application d'attributs s'effectue par l'ajout de blocs d'attributs à des éléments de programme, tels que propriétés, méthodes, événements, classes et assemblées. Un bloc d'attributs se compose d'une liste de déclarations attribut encadrée par des crochets.

Par exemple, le code suivant définit une propriété (nommée DescriptionProduit) avec une composition de validateurs pour la validation d'une propriété de type texte qui doit être vide ou entre 5 et 50 caractères :

Dans ce cas l'attribut a pour but la validation de cette propriété comme le montre ce bout de code :

```
[IgnoreNulls]
[ValidatorComposition(CompositionType.Or, Ruleset = "RuleSetA", MessageTemplate = "La valeur doit
être vide, ou  entre 5 et 50 characteres")]
      [StringLengthValidator(0, Ruleset = "RuleSetA")]
      [StringLengthValidator(5, 50, Ruleset = "RuleSetA")]
      public string DescriptionProduit
      {
         get { return descriptionproduit; }
         set {descriptionproduit = value; }
       }
```

Déjà avec ces options,la validation des données est devenue presque automatique mais ne répondait pas encore au dynamisme de validation requis sanchat que notre code ne porte pas sur des entités fixes comme on l'a evoqué précédemment.Parsuite je me trouve devant deux choix pour realiser la validation :

- Réécrire nos propres classes de validation et s'en passer des classes de validation apportées par le framework.
- Trover un moyen pour bénéficier de cette nouvelle technologies de validation de données.

Avant de trancher, je me suis donné un temps de recherche et d'essais de méthodes pour passer des variables aux attributs.

Les résultats de la recherche et les essais n'ont pas donné une solution adéquate, c'est pourquoi j'ai opté pour la deuxième solution qui consiste à :

- Créer un fichier de configuration dans lequel le développeur peut configurer tous les champs à valider avec le type de validation à y appliquer.
- Lire le fichier et le remplir dans une structure.
- Passer la structure en paramètre à une fonction d'une classe.
- Utiliser la compilation dynamique pour créer une classe qui jouera le rôle d'une entité temporaire avec tous les validateurs des propriétés voulues.
- Exécuter la fonction de validation dans la classe créée dynamiquement
- Retourner le résultat de la validation de l'entité créée.

Normalement l'appel se fait ainsi :

```
Produit produit = CreerProduit();
 if (produit != null)
 {
Validator<Produit> validator = ValidationFactory.CreateValidator<Produit>();
ValidationResults results = validator.Validate(produit);

 }
```

La classe *produit* passé en paramètre est la classe qui doit être créée dynamiquement pour être validée avec ses propriétés.

Par cette méthode un objet est retourné contenant les résultats de la validation qui seront les données en entrée des composants visuels de validations.

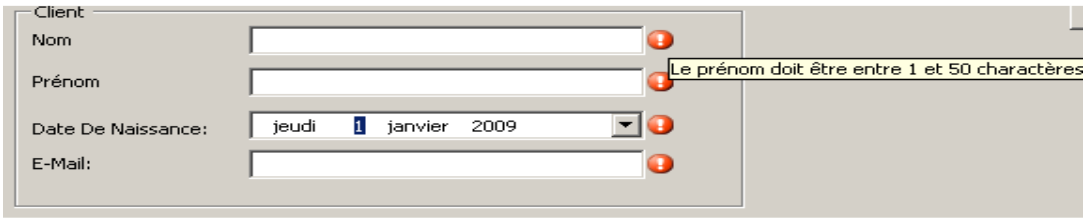

Figure 26:Exemple d'un écran suite à la phase de validation

Une fois toutes les configurations et le développement des couches achevés et testés je pouvais participer à d'autres défis comme l'architecture et le développement des Assistants de la couche présentation et des services Web que j'évoquerai dans ce qui suit.

Chapitre VI

Avant de se lancer dans la migration de chaque écran d'Omega PM, un plan de travail de travail a été dressé dans lequel j'ai classifié les composants visuels et non visuels communs utilisés dans les écrans d'***OmegaPM*** pour m`assurer de la possibilité de leur migration en WPF, car il a été décidé de profiter des atouts de WPF détaillés précédemment (Introduction à WPF Chapitre II.4) dans ce mémoire pour migrer les ancien écrans et créer les nouveaux selon les normes du WPF.

En fait, on peut considérer que WPF complète, plutôt que remplace, Windows Forms.Voici une liste non exhaustive des fonctionnalités proposées par WPF :

- graphismes vectoriels.
- support des animations.
- support de la transparence par pixel.
- support des effets bitmap (flou, ombre portée…).
- support de la modélisation et du rendu en trois dimensions (3D), avec possibilité d'intégration d'une interface 2D dans une scène 3D .
- rendu adapté à la résolution de l'écran.
- accélération matérielle pour le rendu, à travers DirectX.
- supports des styles, des *templates* et du *data binding*.

A cela s'ajoute, WPF permet d'écrire des applications d'une plus grande qualité et maintenabilité grâce au principe de séparation de l'interface utilisateur (UI) et de la logique.

Les nouvelles fonctionnalités telles que le *data binding* (Chapitre II section4 .4.1.4) permettent de réduire considérablement la quantité de code requis pour la liaison entre l'interface graphique et la couche métier.

En résumé, WPF apporte un gain en qualité et en productivité, grâce à de nouvelles techniques qui sont compatibles avec le principe de séparation de code selon l'architecture adopte, en y rappelant que notre logiciel est un logiciel de finance dans lequel la représentation graphique est importante lors de l'élaboration des états de sorties et des statistiques.

Dans ce chapitre, je vais décrire l'existant en termes d'écrans et expliquer comment j'ai pu classer les écrans par type ainsi que les composants visuels à migrer.

VI.1 Composants communs et types d'écrans

La classification d'un composant ou d'un écran selon son type a été basée sur le but et la fréquence d'utilisation de ces derniers dans l'application.

Le problème des composants développés dans l'ancienne version ne diffère pas du problème rencontré au niveau de tous les écrans où il y' a peu de séparation entre le code frontal et le code responsable de l'accès aux données.

C'est pourquoi ces composants ont été refaits en WPF de manière à respecter les règles de découplage de la nouvelle architecture. L'importance de ses composants qui seront migrés est qu'ils feront partie du processus de génération des écrans WPF,

Dans ce qui suit je vais faire un aperçu sur les composants communs ainsi que le des différents types d'écrans.

VI.1.1 Composants visuels communs

Avant la conception de l'assistant il fallait s'assurer que tous les composants de base qu'on doit utiliser seront disponibles, cela a fallu une recherche et une communication avec les fournisseurs de logiciel de développement.

La phase de la compilation .Net a permis avec l'analyse détaillée de réduire le temps d'analyse du code à cette phase. Il fallait décider quels composants utiliser et s'il y'a possibilité de la migration des anciens comme ça a été le cas en VCL.Net (Chapitre V,section V).Des composants nouveaux ont été développés et d'autres fournis par leur fournisseurs.

Après avoir défini les étapes futures pour la création de l'assistant, je devais établir une correspondance entre les composants d'***Omega PM*** et les nouveaux composants WPF.

Cette correspondance a été importante pour le développement de cet assistant car la personne qui va participer au développement devait être au courant de l'équivalence des anciens composants dans la nouvelle plateforme pour pouvoir traduire correctement ce ***Mapping*** au compilateur qui doit générer le code WPF.

Omega PM se base sur des composants visuels dont je cite les suivants :

Dans une migration on ne peut pas s'en passer de l'apport de nouveaux composants en cas ou leurs ancêtres ne peuvent pas être migrés où si des nouveaux besoins imposent l'existence de composants dont le fonctionnel ou le graphisme est meilleur.

Parmi les nouveaux composants qu'on a eus besoin de modifier ou d'insérer dans notre champ de travail je cite les suivants:

- **La Grille de données WPF** : Ce composant est passé par plusieurs phases avant d'être stable et selon la version qui répond le mieux au besoin du logiciel de point de vue affichage et filtrage de données, ainsi que leur format. La principale phase de ce composant c'est son inexistence !!! Oui, en premier lieu la grille en WPF n'existait pas, bien que le planning comptait sur une version dite alpha promise par Microsoft.

  Sans la grille, composants principale dans tout logiciel, le développement de l'assistant prévu ne pouvait démarrer, c'est là où je contacte le directeur technique pour proposer le développement de notre propre grille qui a été développée basée sur un autre composant proche visuellement mais ne possède pas les fonctionnalités voulues. Grâce à des recherches d'exemples de code, dans une semaine la grille était prête à être testée. [W5]

  Pas trop de regret sur le fait que la grille promise venait d'être disponible avec l'accomplissement de l'assistant. Grâce au découplage du code, un petit paramétrage a suffit de remplacer l'ancienne grille par la nouvelle qui présentait des fonctionnalités encore plus avancée vu la taille de l'équipe qui l'a travaillée.

- **le Validateur des données**: C'est le composant visuel qui apparaitra près du champ à valider. Il diffère des autres validateurs déjà existants dans le Framework par son paramétrage et le résultat de la validation qui dépendent de la validation mise en place dans une phase précédente.(Chapitre V section7.1)

- **Le TextBox** : Le TextBox requis doit avoir la possibilité de compléter le mot en écrivant (Auto Complété), ainsi que d'être paramétré pour n'accepter que les entiers et devenir configurable pour les champs de dates, les numéros de téléphone et autres.

- **LOV**(List Of Values) :Ce composant permet d'afficher une liste des valeur qui ne sont que les données relatifs a des champs de tables passés en paramètres. Il permet le filtrage et le retour de la valeur choisie dans la liste.

- **Barre d'outils Omega** : Cette barre consiste en des boutons qui encapsulent toute la fonctionnalité de l'écran :
    - Navigation
    - Recherche
    - Impression
    - Création de nouvel enregistrement
    - Sauvegarde des données.
    - Annulation d'une modification
    - Suppression des données
    - Bouton de rechargement

Figure 27:Barre d'outils Omega

Cette barre a été conçue pour être paramétrée de façon à communiquer avec les libraires des couches basses .elle a besoin de lui fournir l'entité Dataset qui contient les données de l'écran .Son rôle est de détecter le lien(**Binding**) qui existe entre le composant de la forme et le champs de la base de données. A savoir que toute modification de données au niveau de l'écran modifie le statut de la ligne (ajoutée, modifiée, supprimée).En se basant sur ce statut et avec le paramétrage concernant les tables en questions, la barre est capable de gérer tous les événements se produisant sur l'écran dans lequel elle est présente.

Pour une meilleure gestion, les composants qui sont en relation avec l'interface graphique ont été groupés dans une librairie ; chose qui facilitent leur accès ainsi que la gestion de leur espace de nommage

VI.1.2 Type d'écrans

Dans différents modules de l'application plusieurs écrans se répètent, chose qui nous a poussée à les grouper par type pour bénéficier de plus du concept d'héritage apporté par la programmation orientée objet.

On remarque les écrans du type suivant :

- **Ecran Simple** :

Ce type d'écrans est dédié au simple affichage d'un résultat d'une requête dans une grille de donnée. Ce type d'écran peut s'ouvrir par exemple en cliquant sur la grille de la barre d'outils Omega

- **Ecran Liste des valeurs :**
  Cette écran contient la liste des valeurs d'une table référentiel dans la plupart des cas, on a besoin de l'ouvrir pour sélectionner une valeur qui sera affectée a un autre champ lie a la table référentiel par une contrainte de clé étrangère.

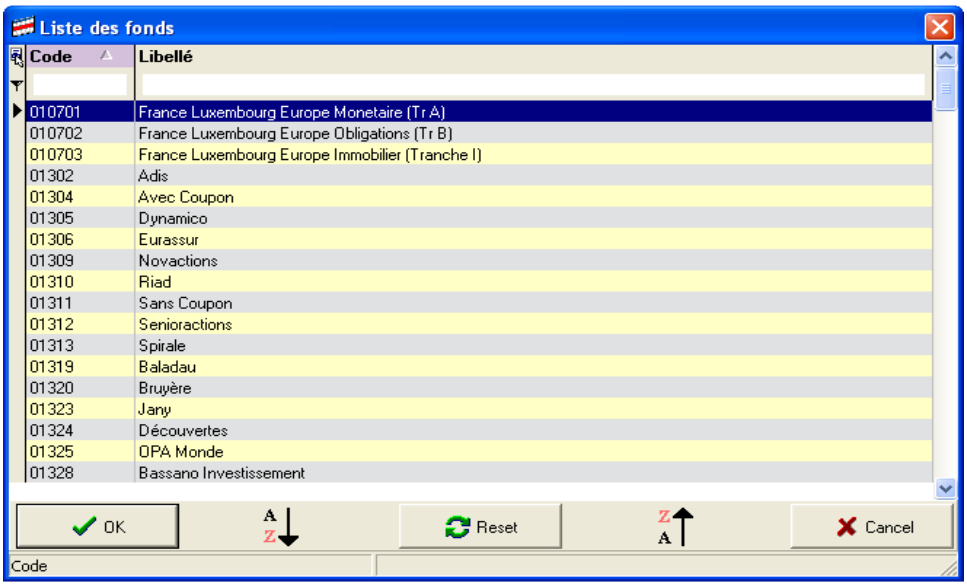

Figure 28 : exemple d'écran simple, gestion des types de fonds.

- **Ecran maître-détail :**

  Dans ce type d'écran sont gérées les données reliées par une relation maitre-détails. Dans le chapitre V, j'ai expliqué le travail qui a été fait à ce niveau de point de vue paramétrage et automatisation d'accès et de modifications de données.

Chapitre VII

Assistant de conception de l'interface utilisateur

Arrivés à cette phase, nous sommes munis de tous les outils permettant de créer des composants visuels WPF (Chapitre I,section V) qui communiquent avec des couches métiers robustes comptant à leur tour sur les couches métiers déjà conçues. En fait, toute la fonctionnalité apportée par les couches sera mise en œuvre dans l'environnement réel grâce à ces composants.

Visual Studio 2008 et ***Expression Blend*** et ***CodeGear2007*** sont les logiciels de développement utilisés pour la conception de cette couche dont j'expliquerai la procédure de travail qui a permis sa mise en œuvre.

WPF (Chapitre I,Section V) offre de nouveaux composants visuels dont on a profité pour améliorer la présentation des données dans ***OmegaPM*** tout en prenant en compte la possibilité de la migration des anciens comme on a remarqué dans la phase de la première compilation sous .Net(Chapitre IV).

Pour permettre la réalisation de la couche de présentation et des couches métiers d'un module Omega selon le principe de la normalisation adopté pour cette migration, il a été décidé de construire un **assistant** ou extension qu'on a nommé ***OmegaCodeGen*** qui permet au développeur l'élaboration de nouveaux écrans et de migrer les anciens vers WPF pour permettre le découplage de code tant attendu. ***OmegaCodeGen*** porte donc sur la génération de la couche présentation et tous les métiers nécessaires avec leur accès souple à la base de données via les composants d'accès déjà créés ainsi que la construction des services Web.

En d'autres termes cet assistant sera intégré dans l'outil de développement de façon à permettre la création plus ou moins automatique de l'interface graphique avec la création des couches basses avec lesquelles elle doit communiquer.

Ayant travaillé sur la compilation .Net (Chapitre IV), la conception des composants de la couche métier et celle de l'accès aux données, ainsi que différents gestionnaires cités précédemment, j'ai été nommé responsable de la conception de cet assistant au sein de l'équipe de recherche et développement.

VII.1 Phases de Conception de l'assistant

Visual studio permet l'ajout des extensions (Addins en anglais) de programmation qui peuvent être exploitées comme toute autre fonctionnalité de l'outil. On a profité de cet atout pour intégrer une fonctionnalité d'assistance dans l'outil de développement de sorte que le module à migrer ou le nouveau module à développer soit bâti selon les normes de la nouvelle architecture tracée.

Pour pouvoir bénéficier de cet assistant, le développeur doit bâtir son paramétrage qui porte sur la nomenclature du module, la connexion, les objets métiers et les services externes en cas ou il y'a lieu.

L'assistant permet de bâtir un fichier de configuration au fur et à mesure comme si on remplissait une application d'inscription en ligne constituée de plusieurs étapes dont chacune d'elle doit être sauvegardée avant de passer à la suivante. La différence consiste dans l'utilité de ce fichier construit, qui sera lu et interprété pour produire du code qui sera intégré dans le projet à compiler. Produire le code dynamiquement sera la tâche de chacun des sous assistants développés que j'ai nommés dans ce qui suit "les **producteurs** ".Chacun de ces producteurs a une tâche bien déterminée qui sera détaillée dans ce qui va suivre.

La Conception de l'assistant de la couche présentation porte sur trois Etapes principales :

- Etape de conception de La configuration : Qui est l'étape de conception du fichier de configuration et du format des données qu'il va contenir.
- Etape de conception et de développement des Générateurs du code ou des producteurs.
- Etape de développement de l'interface de l'assistant.

VII.1 .1 Etape de conception de La Configuration

Cette étape porte sur la conception des différentes configurations de l'assistant qui seront passées à des producteurs pour permettre la génération de l'interface utilisateur ou des services Web selon le standard de la SOA expliqué précédemment.

VII.1 .1.1 Le fichier de configuration, point d'entrée du modèle

En règle générale, ce fichier contient le paramétrage du projet ***OmegaCodeGen*** ainsi qu'une référence vers chaque fichier XML faisant partie du modèle. Cette fonctionnalité d'import permet un découpage bien adapté à un mode de travail en équipe.

Chaque projet généré par ***OmegaCodeGen*** contient un nœud ou un attribut **<Omg:project>** permettant de déclarer un paramétrage global à tout le projet. Il existe des nœuds obligatoires tels que l'espace de nommage XML de ***OmegaCodeGen*** permettant de déclarer l'espace de nommage[Chapitre V section 1.2] racine de notre

projet et d'autres permettant de spécifier au producteur SQL le nom des colonnes qui vont être générées .

VII.1 .2 Etape de Conception et développement des producteurs

Dans ***OmegaCodeGen***, on a dû créer quatre producteurs spécialisés:

- Le producteur SQL, en charge de la génération du code SQL pour l'entité en question qui est une ou plusieurs tables appartenant à un conteneur de données. Ce producteur permettra l'accès à la base pour extraire des données et à partir de la structure de la table, il doit générer le code SQL permettant l'insertion, la modification et suppression.

- le producteur **CodeDom** en charge de fabriquer la librairie des classes permettant de manipuler les entités métiers, il doit permettre l'héritage à partir d'autres librairies qui permettent la gestion des entités.

- le producteur WPF responsable de la génération de la couche de présentation constituée par défaut des écrans WPF ou fichier .XAML.

- le producteur des services Web, responsable de l'exposition du fonctionnel en services Web.

Tous les producteurs cités ci-dessus seront accessibles à partir des exécutables ou dans les outils de développement en tant qu'extensions dans le menus de ces derniers ; le développeur peut les paramétrer et exécuter les méthodes voulues visuellement chose qui diminuera la quantité du code et le taux d'erreur.

VII.1.2 .1 Le producteur SQL

Ce producteur est responsable de la génération du code SQL nécessaire pour la gestion des tables paramétrées. Le développeur a le choix de choisir le type de la génération (fichiers .SQ L, procédure stockée, ou texte).

En se basant sur les librairies développées précédemment pour l'accès aux données, il est devenu simple de se connecter aux bases et de lire les objets existants sur le serveur de données comme les tables et le code SQL déjà présent ainsi que toutes les propriétés nécessaires à la génération de nouveau code qui SQL .

A part la création du code SQL, ce producteur a la tâche de créer les fonctions de base dans un objet d'accès aux données propre aux tables sélectionnées. Cet objet et grâce au principe d'héritage possède déjà les fonctionnalités de base de lecture, d'écriture et de suppression de données. Selon le besoin fonctionnel, le développeur interviendra pour ajouter des fonctions spécifiques ou des contrôles recommandés.

VII.1.2 .2 Le producteur CodeDom - Business Object Model (BOM)

Ce producteur est en charge de la génération de la couche objet .NET des entités métiers. Ce producteur est un producteur important d'***OmegaCodeGen***. Il n'est pas lié au producteur SQL Server vu ci-dessus car le code qu'il génère peut fonctionner avec toute base de données prenant en charge les procédures stockées.

Quelques attributs importants dans le fichier de configuration de ce producteur :

- L'attribut **compile** spécifié si une compilation automatique doit être effectuée par ***OmegaCodeGen*** à la fin de la génération. Nous allons compiler le code généré avec Microsoft Visual Studio et avons choisi donc de désactiver la compilation pilotée par ***OmegaCodeGen***.

- L'attribut **outputName** permet de spécifier le nom du fichier « assembly » qui sera généré.

VII.1.2 .3 Le producteur des écrans WPF

Le producteur WPF est en charge de générer les fichiers XAML prêt à l'emploi permettant de manipuler les entités de notre modèle. Pour l'utiliser, nous allons le déclarer et le configurer à l'intérieur du nœud racine **<omg:project>** en rajoutant le contenu XML suivant :

Lors de la génération des fichiers XAML, la fabrique utilisera le modèle disponible dans le répertoire spécifié en tant que répertoire de sortie des fichiers**.**

Ce producteur a en charge de générer des composants WPF spécialisés (de type Grid, Form, ListBox, CheckBoxList, etc.) permettant de faciliter l'affichage et la saisie des objets de type complexe de l'application (entités, énumérations).

Le producteur WPF a été conçu de façon à pouvoir l'ajouter en tant qu'une extension à l'interface de développement de façon qu'elle puisse être accessible par les développeurs pour faire le paramétrage portant sur la chaîne de connexion et les données concernées et leur visualisation à l'écran.

VII.1.2 .4 Le producteur de Services Web

Un des objectifs de la migration vers la nouvelle architecture est de rendre les fonctionnalités d'***OmegaPM*** accessibles via les services web. Les interfaces de programmation fondées sur la messagerie et les services Web exposés permettent une intégration rapide des systèmes.

Ce travail a nécessité une petite recherche pour voir s'il existe un outil sur le marché qui puisse accomplir le travail de génération dynamique des services Web. J'ai remarqué la présence d'outils de génération semi automatique qui ne répondent pas au niveau de généricité attendu.

Afin de pouvoir exposer nos services avec ***.ASMX*** et ***WCF*** dans un temps futur, nous allons déclarer l'utilisation du sous-producteur ServiceModel dans la configuration du producteur CodeDom:

Ce sous-producteur enrichit le code généré par le producteur BOM pour le rendre conforme avec ***WCF*** (*Windows Communication Foundation*) en créant donc une couche de service. Il nous génère également une bibliothèque proxy qui est une représentation quasi à l'identique du BOM et qui s'occupe de toute la tuyauterie entre la couche client et la couche serveur. C'est un proxy amélioré, mais qui n'est pas obligatoire pour utiliser la couche de service ajoutée au BOM.

Les méthodes d'une classe implémentant un service Web n'ont pas automatiquement la possibilité de recevoir des demandes de service et de renvoyer les réponses, mais avec les services Web créés à l'aide d'ASP.NET, il est très facile d'ajouter cette fonctionnalité. Il faut appliquer un attribut *WebMethod* aux méthodes publiques. Les méthodes d'un service Web avec lesquelles il est possible de communiquer sur le Web sont appelées des méthodes de service Web.

Ci-dessous un exemple d'une classe héritant de la classe webservice du .Net Framework exposant la méthode « CalculerTaux » pour pouvoir l'invoquer via le service web .

```
public class ServicePosition : System.Web.Services.WebService
{
    [System.Web.Services.WebMethod]
    public double CalculerTaux(string typeproduit)
    {
       return this.calculertaux(typeproduit);
    }
}
```

Comme peut le remarquer que le code du service web ressemble au code normal de la librairie contenant les méthodes fonctionnels sauf qu'il faut intervenir pour spécifier l'héritage dans les classes et injecter les attributs au dessus des méthodes à exposer.

Le producteur des services web a été conçu de façon à transformer un code passé en format texte ou une librairie compilée (Annexe C .dll). Le développeur en général doit pouvoir choisir les méthodes publiques qui feront partie du service web à générer grâce au principe de Génération de code et l'invocation dynamique [Chapitre III partie 1.2.12 .1.2].

VII.2 Etape de développement de l'assistant.

La mise en œuvre de cette partie du projet a nécessité la collaboration de deux développeurs avec lesquels j'ai discuté l'analyse et délivré les outils nécessaires dont ils ont besoin pour réaliser cet assistant.

Le travail a été divisé de sorte à avoir deux étapes à programmer par chacune des collaborateurs qui assistent à la mise en œuvre.

En cours de développement, des informations supplémentaires ont été ajoutées, des problèmes de connexion ont été réglés ainsi que des problèmes de communication avec les nouvelles librairies mises en place comme la couche d'accès aux données qui ne fonctionnera si le fichier de configuration est mal paramétré. Mon intervention était nécessaire pour voir l'avancement du travail et pour avoir une idée sur les résultats de l'assistant ; ce qui m'a permis de modifier quelques méthodes d'appel ainsi que des paramètres.

La collaboration et la communication des informations et des résultats a permis de porter l'assistant à ses bonnes fins. Nos premiers tests ont montré la bonne génération des écrans ainsi que les services qu'ils soient fonctionnellement nouveaux ou basés sur des écrans déjà présents dans ***OmegaPM***.

VII.3 Le Processus de test des couches

Lors du test des couches applicatives de l'architecture, on a suivi les mêmes étapes de base indépendamment des méthodes qu'on a utilisées pour développer et tester chaque couche. Le but de ces mesures est de s'assurer que chaque couche est conforme à toutes les exigences et spécifications fonctionnelles.

Ce qui suit permet de décrire le processus adopté pour tester les couches. Quand il y'a eu une nécessité, j'ai assisté à la personnalisation des processus de test en fonction de la stratégie de développement auprès du département de contrôle qualité.

VII.3.1 Pré requis

Les documents sont les suivants :

- Les spécifications fonctionnelles des couches et des composants utilisés.
- Les objectifs de performance
- Le scénario de déploiement.

VII.3.2 Le plan de tests

Avec l'assistance du directeur technique et des responsables contrôle et qualité, on a pu élaborer un nouveau plan de test qui sera appliqué dans un environnement de recette où tous les scénarios prévus et non prévus seront exécutés et dont le résultat sera tracé avec la date d'exécution.

Le but de du document de plan de test est de mettre en œuvre une stratégie de tests qui présente l'approche recommandée d'évaluation des cibles des tests.

Chaque type de test a été fourni une description, ainsi que la méthode de son implémentation et de son exécution.

Lorsque les tests ne sont pas implémentés et exécutés, il faut décrire la raison et en cas où le test n'est pas concluant, il faut noter toutes les informations correspondant à la fonctionnalité dont les résultats diffèrent de ceux qui sont attendus.

Ce type de document nous a permis une meilleure gestion des régressions dues à la migration où aux nouveaux composants développés.

Les différents types de tests qui appliqués ainsi que leur objectif sont les suivants :

Type de test	Objectif
Tests fonctionnels	Vérifier que la fonctionnalité rencontre la bonne cible de test, incluant la navigation, la saisie de données, le traitement et la récupération.
Tests d'interface utilisateur	Évaluer: • La navigation des cibles de test représentent adéquatement les fonctions d'affaire et les exigences, incluant d'écran à écran, de champ à champ, et l'utilisation des modes de déplacement tells les clés de tabulation, le mouvement de la souris et les clés d'accès rapide. • Les composants de l'écran et les caractéristiques tels les menus, la taille, la position, la cible de saisie et l'état sont conformes aux normes.
Tests de données et d'intégrité de base de données	Vérifier les méthodes d'accès à la base de données ainsi que les processus fonctionnent correctement et sans corruption de données.
Profilage de performance	Vérifier la performance d'exécution de transactions désignées ou de fonctions d'affaires.

Tests de charge	Vérifier les temps de performance pour des transactions données ou pour des cas d'affaire selon différentes conditions de charge de travail.
Tests de stress	Vérifier que la cible de test fonctionne correctement et sans erreur dans les conditions de stress suivantes : • Mémoire vive du serveur (RAM) ou unité de stockage à accès direct (DASD) disponible absente ou minime. • Nombre maximum de clients, actuels ou potentiels, connectés simultanément. • Plusieurs utilisateurs exécutent les mêmes transactions sur les mêmes données. Le pire des cas en volume et en nature de transaction.
Tests de volumétrie	Vérifier que la cible de test fonctionne correctement et sans erreur avec les scénarios de volume suivants : • Nombre maximum de clients connectés qui exécutent la même fonction qui, en termes de performance, est le pire des cas pour une fonction d'affaire sur une longue période de temps. Taille maximum de base de données et exécution simultanée de plusieurs transactions de requêtes ou rapports
Tests de sécurité et de contrôle d'accès	La sécurité au niveau de l'application: Vérifier qu'un acteur ne peut accéder aux seules données et fonctions pour lesquelles son type d'utilisateur lui donne les autorisations. La sécurité au niveau du système : Vérifier que seuls les acteurs ayant l'autorisation, ont un accès au système et aux applications.
Tests de basculement et de récupération	Vérifier que le processus de récupération, manuel ou automatique, restaure correctement la base de données, les applications et le système dans un état connu désiré. Les types de conditions suivantes doivent être prises en compte : • Mise hors tension du poste client. • Mise hors tension du serveur.

	• Interruption de communication du réseau • Mise hors tension ou de communication du DASD ou du contrôleur DASD. • Cycles incomplets: processus de filtrage de données interrompu, processus de synchronisation de données interrompu. • Pointeur ou clé de base de données invalide. Élément de données de base de données corrompu ou invalide.
Tests de configuration	Vérifier que la cible de test fonctionne correctement avec les configurations logicielle et matérielle définies
Tests d'installation	Vérifier que la cible de test est installée correctement pour chaque configuration matérielle requise, avec les conditions suivantes : • Nouvelle installation, sur une nouvelle machine où l'application n'a jamais été installée auparavant. • Mise à niveau d'une machine avec une application de la même version. • Mise à jour d'une machine avec une application d'une plus vieille version.

Table 7 :Types de tests appliqués

Conclusion

L'architecture orientée services constitue la prochaine vague de développement d'applications. Les services Web et les architectures orientées services sont sur le point de concevoir et de créer des systèmes à l'aide de composants logiciels hétérogènes et adressables par le réseau.

L'architecture orientée services comprend des propriétés spécifiques, des composants et des interconnexions qui mettent l'accent sur l'interfonctionnement et la transparence d'emplacement. Elle peut souvent évoluer à partir de systèmes existants et ne requiert donc pas de réécriture système complète. Elle tire avantage des systèmes existants de l'organisation en profit tant des ressources actuelles, y compris des développeurs, des langages logiciels, des plates-formes matérielles, des bases de données et des applications, et permet de réduire les coûts et les risques tout en stimulant la productivité.

Cette architecture adaptable et flexible est la base d'une réduction du temps de mise sur le marché, ainsi que des coûts et des risques de développement et de maintenance. Les services Web constituent un ensemble de technologies dynamisantes pour les architectures orientées services qui deviennent l'architecture de choix pour le développement de nouvelles applications réactives et adaptatives.

L'objet de ce mémoire était de décrire à la fois le besoin de la migration du logiciel ***Omega PM*** vers une architecture orientée service qui le rend flexible aux changements. Il a été utile dans la deuxième partie de ce mémoire de détailler plus ou moins les solutions techniques qui ont été adoptées pour la mise en œuvre de la solution finale.

Le travail dans l'équipe de recherche et développement était un vrai bonheur malgré les difficultés rencontrées lors la mise en œuvre des solutions qui en théorie paraissaient facilement applicables mais en réalité un simple détail pouvait remettre en question une décision toute entière mais la plupart du temps on savait comment s'en sortir avec la meilleure rentabilité à un coût pas très cher.

La société Omega Financial Solutions est bien consciente du rythme des changements dans l'informatique financière dont les besoins en ressources ont augmenté surtout avec l'exposition des traitements sur le WEB.

Omega Financial Solutions est consciente de même que l'investissement fait dans cette migration est à long terme et par suite les revenus seront réalisés au fur et à mesure des modifications recommandées par les clients ou la communication avec d'autres plateformes sera une contrainte si l'architecture n'aurait pas changée.

Chaque fonction du Logiciel peut devenir un service à part dont peut profiter chaque entité qui s'y enregistre.

Conquérir de nouveaux marchés est le but de toute société mais ce n'est toujours facile d'y accéder si on n'a pas les outils nécessaires nous permettant d'être parmi les pionniers à promouvoir de nouveaux services se basant sur les technologies les plus récentes.

Trouver de nouveaux marchés et augmenter le chiffre d'affaire est un des buts principaux des sociétés financières, mais peu parmi ces dernières parviennent à conquérir des nouveaux marchés et les fidéliser.

Les ressources humaines et techniques sont les indicateurs principaux de la compétitivité d'une société, c'est d'ailleurs le but des formations continues du personnel dont j'ai fait partie au sein d'une amélioration technique primordiale dont les revenus seront prometteurs.

En effet, de point de vue technique beaucoup de chantiers se préparent au sein de Omega Financial Solutions. Parmi ces projets:

- La préparation d'une version Oracle :

Il est important de faire le paramétrage nécessaire pour adapter l'application à une base de données Oracle, sachant que plusieurs demandes ont été constatées au niveau des clients potentiels ou les anciens clients qui pensent à migrer vers une base de données Oracle.

- plateforme personnalisée FIX

Le travail en cours porte la création d'une plateforme personnalisée dans laquelle sera intégré le protocole FIX qui est un protocole standard de messagerie développé spécialement pour l'échange électronique en temps réel des transactions sur titres.

- Interface pour mobile :

La prolifération des téléphones mobiles dans le monde avec la déréglementation du marché des services financiers a fourni de nouvelles opportunités pour les marques de confiance. Soudain, les entreprises avec des millions de clients et canaux de distribution large, qu'ils soient opérateurs de téléphonie mobile, des détaillants ou des marques en ligne, ont l'occasion de participer à des marges élevées des services financiers, dont ils bénéficiaient précédemment par les banques et sociétés de services financiers associés.

Grâce aux messages XML, les données peuvent être exploitées sur n'importe quelle plateforme y inclus les téléphones mobiles. Omega Financial Solutions se lancera dans le domaine des produits financiers sur mobile et proposera à ces clients la création des applications financières mobiles sur demande.

Avec une telle architecture Omega Financial solution a une marge d'adaptabilité plus grandes qu'auparavant qui va lui permettre de faire face aux changements fonctionnels et techniques dans le monde de la finance.

Bibliographie

1- Ouvrage de référence

[1] Olivier Dahan(Eyrolles,2004)
Delphi 8 pour .NET
" Lu et approuvé par Borland France "

[2] Keenan Newton(2007)
"The Definitive Guide to the Microsoft Enterprise Library"

[3] Laurent DEBRAUWER
"Design Patterns pour C#"

[4] Laurence Moroney (Apress,2006)
" WPF An Introduction to Windows Presentation Foundation"

2- Articles

[a1] David Soto (IBM 2008)
"Augmenter la flexibilité de l'entreprise grâce à l'architecture orientée service"

[a2] François Tonic(Programmez-Le magazine du développement - Juillet / Août 2006)
"FOCUS SOA"

[a3] Laurent Henriet(Décision informatique N° 650 / 17 OCTOBRE 2005)
"Le point sur… les services web"

[a4] Groupe Tests (Février 2007)
"SOA perception des entreprises françaises"

[a5] Alain Zanchetta, MCS, Microsoft France(Magazine Programmez Octobre 2006)
« Introduction à WPF »

3- Les sites Internets

[W1] http://www.jcolibri.com/articles/articles.html
[W2]http://www.microsoft.com/biztalk/solutions/soa/overview.mspx
[W3]http://www.zdnet.fr/actualites/soa-comprendre-l-approche-orientee-service-39206712.htm
[W4]http://msdn.microsoft.com/fr-fr/netframework/bb870269
[W5] http://www.codeproject.com/KB/WPF/GuidedTourWPF_3.aspx
[W6] https://www.microsoft.com/france/expression/products/blend_overview.aspx

Glossaire

Framework: En programmation orientée objet, un Framework est typiquement composé de classes mères qui seront dérivées et étendues par héritage en fonction des besoins spécifiques à chaque logiciel.

CLR :

ASMX: Le fichier « .asmx « représente le point d'entrée adressable des services Web XML créés en code managé. La façon dont d'y accédez via le protocole HTTP détermine le type de réponse obtenue.

WCF: (**Windows Communication Foundation**) est un sous-système de communication.

WPF : Windows Presentation Foundation est la spécification graphique de Microsoft .NET 3.0. Il intègre le langage descriptif XAML qui permet de l'utiliser d'une manière proche d'une page HTML pour les développeurs.

Mapping : C'est généralement utilisé pour désigner la mise en cohérence entre deux types d'informations distincts.

OPCVM : Un **OPCVM**, ou *Organisme de Placement Collectif en Valeurs Mobilières*, est une entité qui gère un portefeuille dont les fonds investis sont placés en valeurs mobilières.

TWebBrowser :Composant Delphi

XAML: eXtensible Application Markup Language est un langage d'interface utilisateur graphique universel pour les applications Web riches (RIA) et les logiciels de bureau. Il utilise un format XML facile à éditer et à réutiliser. Il fonctionne sous .NET ou systèmes compatibles comme Mono, ou avec un plug-in pour navigateurs.

VCL : Il s'agit de la bibliothèque des composants visuels (Visual Component Library, VCL). Elle est composée de classes hiérarchisées toutes descendantes de la classe TComponent. Tous ces composants sont accessibles sur la palette de composants de Delphi. Il est possible de créer de nouveaux composants en dérivant les classes existantes.

Resx: Les fichiers de ressources sont généralement des fichiers de données non exécutables utilisés dans le contexte de la localisation d'applications, les fichiers de ressources font référence aux fichiers .resx, que vous pouvez déployer dans des assemblys satellites localisés.

IHM :Interface Homme Machine

Front-Office : Parfois appelé également *Front line*, désigne la partie frontale de l'entreprise, visible par la clientèle et en contact direct avec elle, comme les équipes de marketing, de support utilisateur ou de service après-vente.

Back Office : A l'inverse désigne l'ensemble des parties du système d'information auxquelles l'utilisateur final n'a pas accès. Il s'agit donc de tous les processus internes à l'entreprise (production, logistique, stocks, comptabilité, gestion des ressources humaines, etc.).

Annexe A

1) Aperçu global sur l'interface de développement Delphi

L'environnement de développement de Delphi s'appuie sur un éditeur d'interface graphique associé à un éditeur de code source.

Delphi (le programme) auto-génère du code, il maintient une correspondance automatique entre la vue de conception (la fenêtre que le programmeur bâtit en déposant des composants graphiques) et l'éditeur de code (la vue affichant le code source qui créera ces composants à l'exécution). Les données spécifiques aux composants sont stockées dans des fichiers d'extension « .DFM » tandis que le code source *Pascal Objet* est sauvegardé dans des fichiers d'extension « . PAS ».

L'interface de développement permet l'ajout de composants tiers (graphiques ou non) via un dispositif de composants. La modularité est obtenue à la conception mais peut aussi être exploitée à l'exécution via un dispositif de chargement dynamique de paquets d'exécution, Borland ayant étendu le concept de bibliothèques partagées et le format Windows DLL en introduisant un modèle propriétaire permettant d'enregistrer dynamiquement et d'exporter des classes entre modules. Le même dispositif sera repris par Microsoft sous Visual Basic avec le format VBX, puis ensuite à l'échelle du dispositif avec les composants COM et ActiveX.

2) Bibliothèque de composants VCL

Avant d'aller plus loin il fallait étudier les composants utilises sous win32 pour pouvoir trouver l'équivalent optimal sous .Net.

Delphi est livré avec des composants faisant partie d'une hiérarchie de classes appelée la VCL (Visual Component Library). La VCL comprend des objets visibles à l'exécution, comme les contrôles de saisie, les boutons et les autres éléments d' interface utilisateur, ainsi que des contrôles non visuels. Certaines des principales classes constituant la VCL sont détaillées dans l'Annexe

Les objets dérivés de *TComponent* possèdent les propriétés et les méthodes autorisant leur installation dans la palette de composants et leur ajout aux fiches Delphi. Les composants de la VCL étant étroitement liés à l'EDI, vous pouvez utiliser des outils comme le concepteur de fiches pour développer des applications très rapidement.

Les composants apportent un haut degré d'encapsulation. Par exemple, les boutons sont pré-programmés pour répondre aux clics de souris en déclenchant des événements *OnClick*. Si vous utilisez un contrôle bouton de la VCL, vous n'avez pas à écrire de code gérant les messages Windows lorsque le bouton est cliqué ; vous n'êtes chargé que de la logique de l'application qui doit s'exécuter en réponse à cet événement.

Annexe B

1) Les unités

Une unité est un fichier séparé pouvant contenir des constantes, des types, des variables et des sous-programmes disponibles pour la construction d'autres applications. L'utilisation des unités permet de partager des données et des sous-programmes entre plusieurs applications.

a) Squelette d'une unité

```
Unit Nom_unite;
Interface
{Déclarations publiques}
Implementation
{Déclarations privées}
{Corps des procédures et des fonctions}
Initialization
{
```

b) Visibilité d'une unité

En-tête
C'est le nom de l'unité (8 caractères maxi). Il figure dans la partie `uses` d'un programme ou d'une unité qui utilise l'unité.
NB : Éviter les références croisées entre unités (ex. l'unité A inclut l'unité B et vice-versa).

Interface
Tout ce qui est placé ici est visible pour toute entité utilisant l'unité.

Implémentation
Cette partie définit les procédures et les fonctions déclarées dans la partie interface. On peut également y définir des types, constantes, variables et sous programmes locaux.

Initialisation
Cette partie permet d'initialiser les variables de l'unité si besoin est et d'effectuer des traitements avant de redonner la main au programme principal.

2 L'EDI de Delphi

a) L'interface de Delphi

La figure ci-dessous représente l'interface typique de Delphi. Elle est composée de :

- la barre de menus (en haut),
- la barre d'icônes (à gauche sous la barre de menus),
- la palette de composants (à droite sous la barre de menus),
- le concepteur de fiche (au centre),
- l'éditeur de code (au centre sous le concepteur de fiche),
- l'inspecteur d'objets (à gauche).

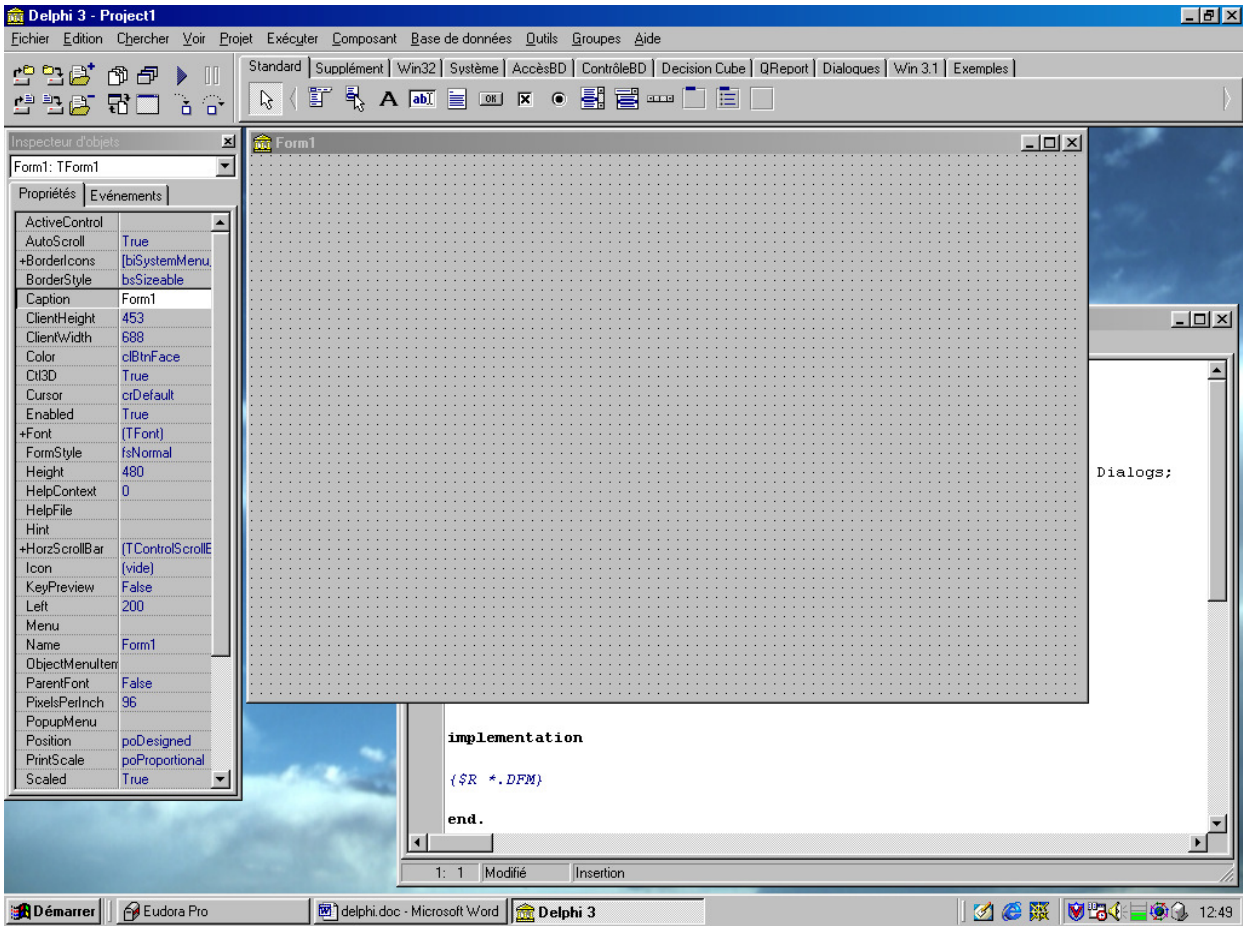

Figure 29 :IDE de delphi

3)La bibliothèque d'objets de Delphi

3.1)Hiérarchie des objets Delphi

La bibliothèque d'objets de Delphi est constituée d'une hiérarchie de classes Pascal dont le sommet est *Object*. Tous les composants et contrôles sont des objets dérivés de cette classe.

Une (petite) partie de cette hiérarchie est présentée ci-dessous.

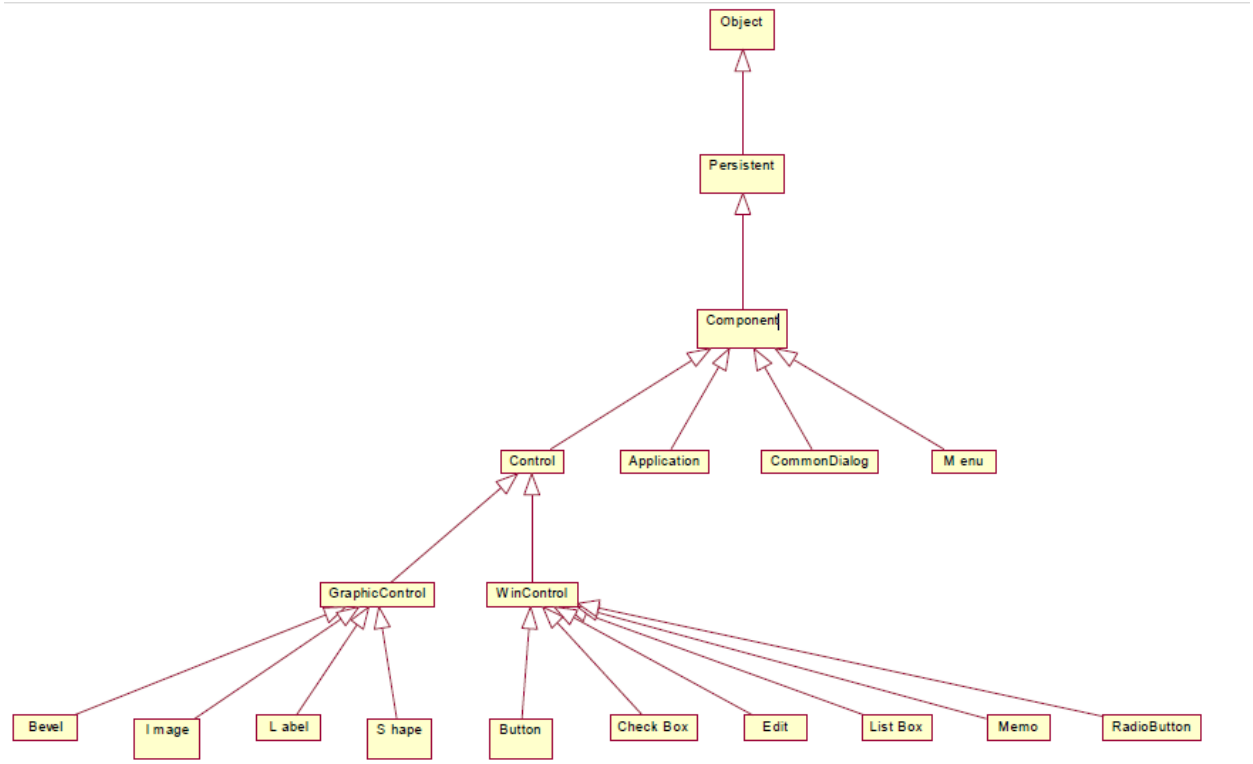

Figure 30 : Hiérarchie des objets Delphi

a) Objets Delphi

Le terme d'objet prend, dans le contexte de la bibliothèque de Delphi, un sens particulier : ce sont les objets de base du système, par opposition aux composants visuels.

Ces objets, qui constituent la partie supérieure de la hiérarchie de la bibliothèque, ne sont accessibles que par programme.

b) Composants et contrôles

Composants

Ce sont des objets descendant de la classe *Component*, éléments fondamentaux à partir desquels sont construites les applications.
Ex. Application, boîtes de dialogue, barres de menu, groupe de boutons radio…

Contrôles

Les contrôles représentent l'ensemble des composants visuels manipulables grâce à la palette des composants. La cohésion des contrôles est basée sur la dérivation de la classe *Control*.

Les contrôles se subdivisent en deux groupes :

- les *contrôles fenêtrés*, héritiers de la classe *WinControl*, qui sont capable de recevoir le focus de l'application, disposent d'un descripteur de fenêtre ou *handle* Windows et peuvent contenir d'autres contrôles (ex. boutons de commande, cases à cocher, boîtes de liste, boîtes d'édition…) ;

- les *contrôles graphiques*, héritiers de la classe *GraphicControl*, qui ne possèdent pas de descripteur de fenêtre, ne peuvent pas recevoir le focus de l'application et ne peuvent pas contenir d'autre contrôle (ex. cadres biseautés, images, étiquettes, formes géométriques…).

Annexe C

1) Description des différents fichiers d'un projet Delphi

Lors de la création et de la compilation d'une application sous Delphi, des fichiers portant différentes extensions sont générés automatiquement. Voici leur signification :

BPL :
Borland Package Library (Librairie de package Borland) : Le package compilé et chargeable. Ce fichier est une librairie partagée (DLL) contenant des spécificités propres à Borland. Le nom de base du package est le même que le fichier de source dpk.

CAB :
C'est le format de fichier que Delphi propose aux utilisateurs déployant sur le WEB. Ce format compressé est une manière efficace d'empaqueter de multiples fichiers.

DBI :
Ce fichier texte contient les informations d'initialisation pour l'explorateur de base de données. Ce fichier ne doit pas être édité excepté par l'explorateur de base de données.

DCI
Ce fichier texte contient le code des templates clavier prédéfinis et ceux définis par l'utilisateur de code pour l'usage dans l'EDI.

DCP :
Fichier de symbole d'un package : Une image binaire contenant le header du package ainsi que la concaténation de l'ensemble des fichiers dcu (Windows) ou dcu(Linux) du package. Un seul fichier dcp est créé/compilé pour chaque package. Le nom de base du fichier dcp est le même que celui du fichier source dpk.

DCU :
Un fichier par fiche (TForm) ou par unité. Il contient le code compilé correspondant à la fiche ou l'unité de même nom.
Delphi utilise ces fichiers lors de la compilation. Ils ne sont recompilés que si le fichier .PAS ou DFM a été modifié.
Dans le cas d'une distribution de composants sans les sources il sont obligatoires.

DFM :
Un fichier par fiche (TForm). Ce fichier contient les détails des composants contenus dans une fiche. Il peut être visualisé comme un fichier texte par un clic-droit sur la la forme et en choisissant le menu contextuel "voir comme texte" ou converti à l'aide de l'outil *Delphi\Bin\convert.exe* (à utiliser avec précaution).

DLL :
Un fichier par Projet. Il s'agit de la version compilée de votre projet DLL.

DPK
Fichier source d'un package : Détermine quelles unités sont contenues dans le package et quels autres packages sont requis.

Analogue au fichier .DPR d'un projet Delphi standard.

EXE
Un fichier par Projet. Il s'agit de la version compilé correspondant à votre application.

MAP
Un fichier par Projet. Contient la description de l'implémentation mémoire des objets et unités de votre projet.

OCX
Un fichier OCX est une DLL spécialisée qui contient tout ou parties des fonctions associées à un contrôle ActiveX.

PAS :
Un fichier par fiche (TForm) ou par unité. Il s'agit du fichier source, dans le cas d'une fiche il porte le même nom.
Note : L'édition d'un fichier associé à une fiche en dehors de l'EDI peut avoir comme conséquence la perte d'informations entre le fichier .DFM et le .PAS.

RES :
Un fichier par ressource externe (possibilité de les regrouper). Ce sont les ressources externes utilisées par l'application (Bitmap, icônes, dialogues importés…)
Il est créé/recréé lors d'une compilation.

RSM :
Ce type de fichier est généré par le compilateur (Dcc32- VR) et est utilisé pour le débogage à distance. Le fichier est placé sur la machine distante et ce dans le répertoire de l'application.

TLB :
Le fichier binaire d'une librairie de type. Permet d'identifier les types d'objets et d'interfaces disponibles dans un serveur ActiveX.

Annexe D

Classe

```
using System;
using System.Collections;
using System.Reflection;

namespace DynamicInvoker
{
   public class DynaClassInfo
   {
      private Type _type;
      private Object _ClassObject;

      public DynaClassInfo()
      {
      }

      public DynaClassInfo(Type t, Object c)
      {
         _type = t;
         _ClassObject = c;
      }
      public Type type { get { return this._type; } set { this._type = value; } }
      public Object ClassObject { get { return this._ClassObject; } set { this._ClassObject = value; } }}

   public class DynaInvoke
   {
      public static DynaClassInfo CreateClassObj(string AssemblyName,string ClassName, Object[]
paramarray)
      {
         return CreateClassObjAbsolute(AppDomain.CurrentDomain.BaseDirectory + "\\" +
AssemblyName,ClassName, paramarray);
      }
      public static DynaClassInfo CreateClassObjAbsolute(string AssemblyName, string ClassName,
Object[] paramarray)
      {
         // load the assemly
         Assembly assembly = Assembly.LoadFrom(AssemblyName);
         // Walk through each type in the assembly looking for our class
         foreach (Type type in assembly.GetTypes())
         {
            if (type.IsClass == true)
            {
               if (type.FullName.EndsWith("." + ClassName))
               {
                  try
                  {
                     // create an instance of the object
                     object ClassObj = Activator.CreateInstance(type, paramarray);
                     return new DynaClassInfo(type, ClassObj); ;
                  }
                  catch (Exception exc)
                  {
```

```
                    //string asd = exc.Message;
                    throw (exc);
                }

            }
        }
    }
    throw (new System.Exception("could not invoke method"));
}
public static Object InvokeMethodByName(DynaClassInfo ci, string MethodName, Object[]
paramarray)
{
    Object[] args = null;
    args = paramarray;
    try
    {
        // Dynamically Invoke the method
        object Result = ci.type.InvokeMember(MethodName,
        BindingFlags.Default | BindingFlags.InvokeMethod,
            null,
            ci.ClassObject,
            args);
        return Result;
    }
    catch (Exception exc)
    {
        //string str = exc.Message;
        throw (exc);
    }

    throw (new System.Exception("could not invoke method"));
}
// this way of invoking a function
// is slower when making multiple calls
// because the assembly is being instantiated each time.
// But this code is clearer as to what is going on
private static Object InvokeMethodByName(string AssemblyName,string ClassName, string
MethodName, Object[] paramarray)
{
    Object[] args = null;
    args = paramarray;
    // load the assemly
    Assembly assembly = Assembly.LoadFrom(AssemblyName);
    // Walk through each type in the assembly looking for our class
    foreach (Type type in assembly.GetTypes())
    {
        if (type.IsClass == true)
        {
            if (type.FullName.EndsWith("." + ClassName))
            {
                // create an instance of the object
                object ClassObj = Activator.CreateInstance(type);
                // Dynamically Invoke the method
                object Result = type.InvokeMember(MethodName,
                BindingFlags.Default | BindingFlags.InvokeMethod,
                    null,
```

```
                    ClassObj, args);
                return (Result);}}}
        throw (new System.Exception("could not invoke method"));
        }

        // ----------------------------------------------
        // now do it the efficient way
        // by holding references to the assembly
        // and class
        // this is an inner class which holds the class instance info
        private static Hashtable AssemblyReferences = new Hashtable();
        private static Hashtable ClassReferences = new Hashtable();
        private static DynaClassInfo
        GetClassReference(string AssemblyName, string ClassName)
        {
           if (ClassReferences.ContainsKey(AssemblyName) == false)
           {
              Assembly assembly;
              if (AssemblyReferences.ContainsKey(AssemblyName) == false)
              {
              AssemblyReferences.Add(AssemblyName,  assembly = assembly.LoadFrom(AssemblyName));
              }
              else
              assembly = (Assembly)AssemblyReferences[AssemblyName];
              // Walk through each type in the assembly
              foreach (Type type in assembly.GetTypes())
              {
                 if (type.IsClass == true)
                 {
                    // doing it this way means that you don't have
                    // to specify the full namespace and class (just the class)
                    if (type.FullName.EndsWith("." + ClassName))
                    {
                       DynaClassInfo ci = new DynaClassInfo(type,
                                Activator.CreateInstance(type));
                       ClassReferences.Add(AssemblyName, ci);
                       return (ci);}}}
              throw (new System.Exception("could not instantiate class"));
           }
           return ((DynaClassInfo)ClassReferences[AssemblyName]);
        }

        private static Object InvokeMethod(DynaClassInfo ci,  string MethodName, Object[] args)
        {
           // Dynamically Invoke the method
              Object Result = ci.type.InvokeMember(MethodName, BindingFlags.Default |
             BindingFlags.InvokeMethod,  null, ci.ClassObject, args);
              return (Result);
        }
        // --- this is the method that you invoke ------------
        public static Object InvokeMethod(string AssemblyName,string ClassName, string MethodName,
Object[] args)
        {
           DynaClassInfo ci = GetClassReference(AssemblyName, ClassName);
           return (InvokeMethod(ci, MethodName, args));
        }}
```

1)Générateur de code SQL

a)Exemple de génération de la requête insert

```
public string GenerateInsertStatement(DataRow row, string tableName)
        {
           string Result = "/* Generated by Omega Sql Generator */\n\n";
            string inString = "";
            string valueString = "";
            string TableName = tableName;
            DataTable dt =row.Table;
            List<string> LstAuto=new List<string>();
            inString = "INSERT INTO " + tableName + "(";
            foreach (DataColumn col in dt.Columns)
            {
                if (!col.AutoIncrement)
                {
                    inString += "[" + col.ColumnName + "], ";
                }
                else
                    LstAuto.Add(col.ColumnName);

            }
            inString = inString.Substring(0, inString.Length - 2) + ")
\nVALUES (";


            for (int i = 0; i < row.ItemArray.Length; i++)
            {
                if (!LstAuto.Contains(row.Table.Columns[i].ColumnName))
                {
                    if (row.ItemArray[i].GetType() ==
System.Type.GetType("System.DBNull"))
                        valueString += "null, ";
                    else if (dt.Columns[i].DataType ==
System.Type.GetType("System.String"))
                    {
                        if (row[i] == DBNull.Value)
                            valueString += "null, ";
                        else
                        valueString += "'" +
row.ItemArray[i].ToString().Replace("'", "''") + "', ";
                    }
                    else if (dt.Columns[i].DataType ==
System.Type.GetType("System.DateTime"))
                    {
                        DateTime dtime = (DateTime)row.ItemArray[i];
                        valueString += "'" +
dtime.ToString().Replace("'", "''") + "', ";
                    }
                    else if (dt.Columns[i].DataType ==
System.Type.GetType("System.Guid"))
                        valueString += "'" +
row.ItemArray[i].ToString().Replace("'", "''") + "', ";
                    else if (dt.Columns[i].DataType ==
System.Type.GetType("System.Byte[]"))
                        valueString += "null, ";
```

```
                    else if (dt.Columns[i].DataType ==
System.Type.GetType("System.Boolean"))
                        valueString += row.ItemArray[i].ToString() ==
"True" ? "1, " : "0, ";
                    else
                        valueString += row.ItemArray[i].ToString() + ",
";

                }
            }
                Result += inString + valueString.Substring(0,
valueString.Length - 2) + ")\n\n";
                valueString = "";


            return Result;
        }
```

b)Exemple de génération de la requête update

```
 public string GenerateUpdateStatement(DataRow schema, string
tableName,string pkClause)
        {
            string Result = "/* Generated by Omega Sql Generator
*/\n\n";
            string inString = "";
            string valueString = "";
            string WhereString = pkClause;
            if (tableName != null)
            {
                inString = "UPDATE " + tableName + "\nSET ";
                for (int i = 0; i < schema.ItemArray.Length; i++)
                {
                if (!schema.Table.Columns[i].AutoIncrement)
                    {
                        if (schema.Table.Columns[i].DataType ==
System.Type.GetType("System.String") ||
                        schema.Table.Columns[i].DataType ==
System.Type.GetType("System.DateTime"))
                        {
                            if (schema[i] == DBNull.Value)
                            valueString +=
schema.Table.Columns[i].ColumnName + " = "+"null, ";
                            else
                            valueString +=
schema.Table.Columns[i].ColumnName + " = " + "'" +
schema.ItemArray[i].ToString().Replace("'", "''") + "', ";
                        }
                        else if (schema.Table.Columns[i].DataType ==
System.Type.GetType("System.Guid"))
                        {
                         string v = "'" +
schema.ItemArray[i].ToString().Replace("'", "''") + "', ";
                         if (v == "'', ")
                         v = "null, ";
```

```
                            valueString +=
schema.Table.Columns[i].ColumnName + " = " + v;
                        }
                        else if (schema.Table.Columns[i].DataType ==
System.Type.GetType("System.Byte[]"))
                        {
                            // valueString = valueString
                        }
                        //valueString += dt.Columns[i].ColumnName + " =
" +"null, ";
                        else if (schema.Table.Columns[i].DataType ==
System.Type.GetType("System.Boolean"))
                        {
                            if (schema.ItemArray[i].ToString() ==
"True")
                                valueString +=
schema.Table.Columns[i].ColumnName + " = " + "1, ";
                            else
                                valueString +=
schema.Table.Columns[i].ColumnName + " = " + "0, ";
                        }
                        else
                        {
                            string v = "'" +
schema.ItemArray[i].ToString().Replace("'", "''") + "', ";
                            if (v == "'', ")
                                v = "null, ";
                            valueString +=
schema.Table.Columns[i].ColumnName + " = " + v;
                        }
                    }
                }
                    Result += inString + valueString.Substring(0,
valueString.Length - 2) + "\n" + WhereString + "\n\n";
                    valueString = "";
                }
                return Result;
        }
        /// <summary>
```

Annexe E

Configuration du Producteur SQL

```
<!-- configuration du producteur SQL -->
<cf:producer
      typeName="OmegaCodeGen.Producers.SqlServer.SqlServerProducer,
OmegaCodeGen.Producers.SqlServer">
      <cf:configuration
            targetDirectory="..\..\{0}.Persistence"
            updateDatabase="true"
            connectionString="database=Omega;server=(local)\sqlexpress;T
rusted_Connection=true"
            />
</cf:producer>
```

BOM

```
<!-- configuration du producter CodeDom - Business Object Model (BOM)-->
<omg:producer
typeName="OmegaCodeGen.Producers.CodeDom.CodeDomProducer,
OmegaCodeGen.Producers.CodeDom">
<cf:configuration
compile="false"
outputName="{0}.dll"
resourceFileFormat="resx"
runtimeResourceBaseName="{0}.{0}"
targetDirectory="..\..\{0}">
 </cf:configuration>
</omg:producer>
```

Annexe F

1) Table des matières du plan de Test

Omega.net	Version : <1.0>
Plan de tests	Date :

Table des matières

1. Introduction 4
   - 1.1 Objectif du document 4
   - 1.2 Portée du document 4
   - 1.3 Historique 4
   - 1.4 Définitions, acronymes et abréviations 4
   - 1.5 Identification des artéfacts 4
   - 1.6 Vue d'ensemble 5

2. Exigences à tester 6

3. Stratégie de tests 6
   - 3.1 Types de tests 6
     - 3.1.1 Tests fonctionnels 6
     - 3.1.2 Tests d'interface utilisateur 6
     - 3.1.3 Tests de données et d'intégrité de base de données 7
     - 3.1.4 Profilage de performance 7
     - 3.1.5 Tests de charge 8
     - 3.1.6 Tests de stress 9
     - 3.1.7 Tests de volumétrie 10
     - 3.1.8 Tests de sécurité et de contrôle d'accès 11
     - 3.1.9 Tests de basculement et de récupération 12
     - 3.1.10 Tests de configuration 14
     - 3.1.11 Tests d'installation 14
   - 3.2 Outils 15

4. Ressources 15
   - 4.1 Travailleurs 15
   - 4.2 Système 17

5. Jalons du projet 17

6. Biens livrables 17
   - 6.1 Modèle de test 17
   - 6.2 Journaux de test 18
   - 6.3 Rapports d'anomalies 18

7. Annexe A: Tâches du projet 19

Figure 31 : Table des matières du plan de Test

Annexe G

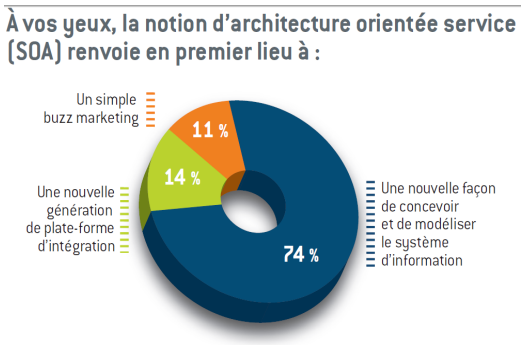

Figure 32:Connaissance et interprétation du concept de SOA

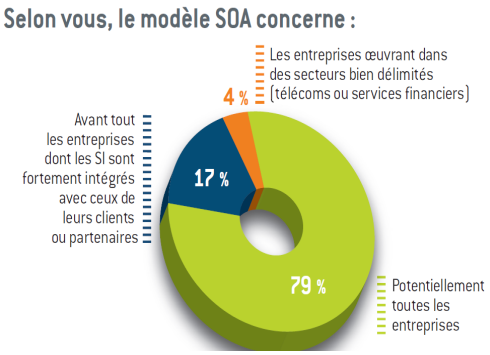

Figure 33: Typologie d'entreprises concernées par SOA

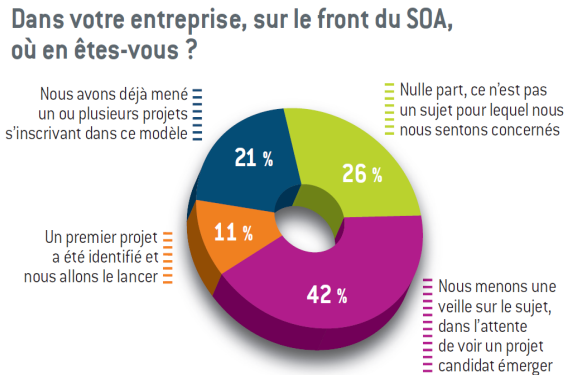

Figure 34 Pénétration de SOA dans les entreprises

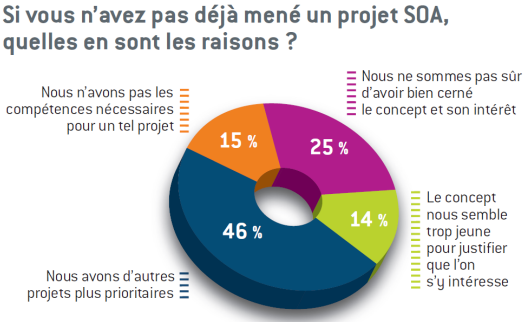

Figure 35 : Les freins à l'essor de SOA

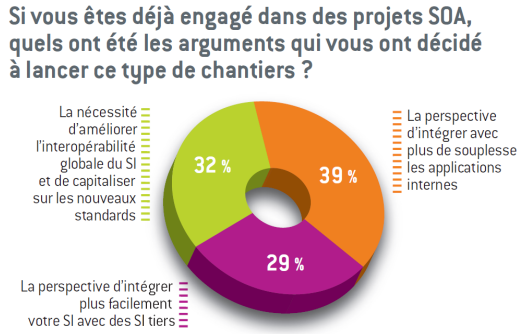

Figure 36 : Les bénéfices attendus de SOA

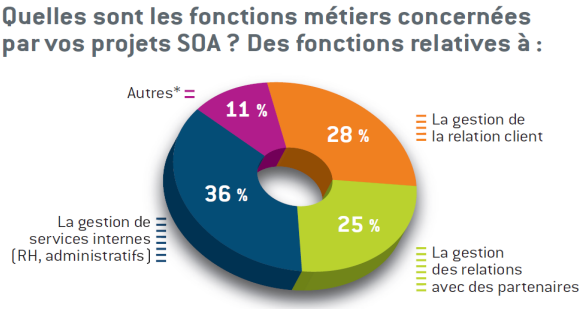

Figure 37 : Typologie des projets SOA

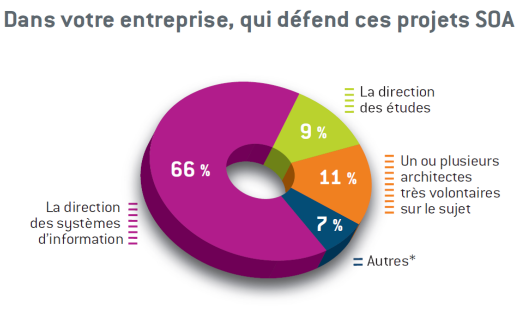

Figure 38 : Les décisionnaires en matière de SOA

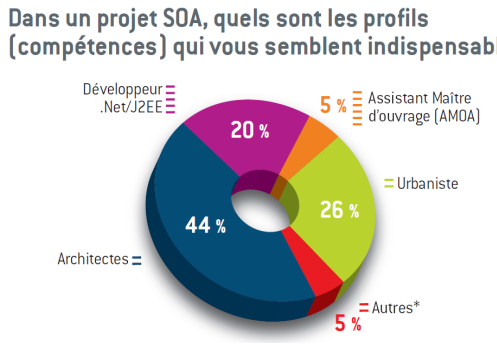

Figure 39:Les compétences recherchées

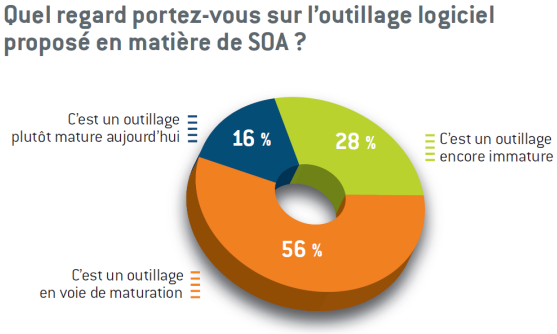

Figure 40:Appréciation de l'offre SOA actuelle

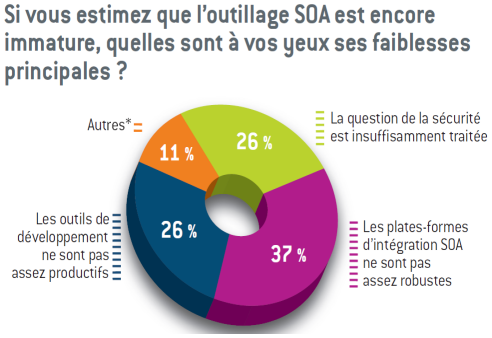

Figure 41:Lacunes des outils SOA

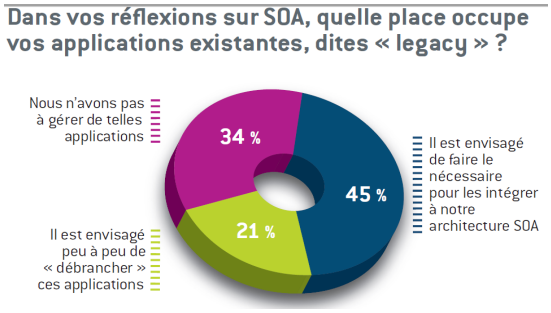

Figure 42:Intégration des applications existantes

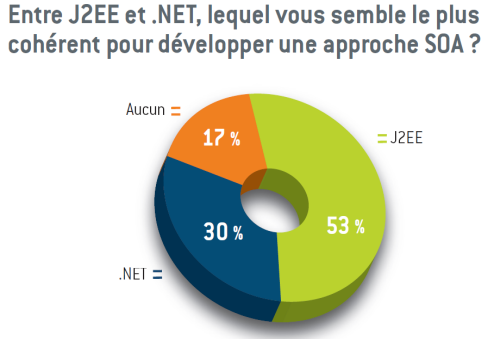

Figure 43: Choix de la plateforme de développement

Annexe F

a)Exemple de syntaxe du protocole FIX

8=*FIX.4.3*^9=*199*^35=*D*^34=*10*^49=*VENDOR*^115=*CUSTOMER*^144=*BOSTONEQ*^56=*BROKER*^57=*DOT*^143=*NY*^52=*20010907-09:25:58*^
11=**ORD_1**^21=**2**^110=**1000**^55=**EK**^22=**1**^48=**277461109**^54=**1**^60=**20010907-09:25:56**^38=**5000**^40=**2**^44=**62.5**^15=**USD**^528=**A**^
10=*165*^

8=*FIX.4.3*^9=*1043*^35=*D*^34=*10*^49=*VENDOR*^115=*CUSTOMER*^144=*BOSTONEQ*^56=*BROKER*^57=*DOT*^143=*NY*^52=*20010907-09:25:58*^
212=**937**^213=**<FIXML><FIXMLMessage>...omitted ...</FIXMLMessage></FIXML>**^
10=*038*^

```
<FIXML><FIXMLMessage>
<Header>
  ... omitted ...
</Header>
<ApplicationMessage>
 <Order>
  <ClOrdID>ORD_1</ClOrdID>
  <HandInst Value="2" />
  <MinQty>1000</MinQty>
  <Instrument>
   <Symbol>EK</Symbol>
   <SecurityIDSource Value="1"/>
   <SecurityID>277461109</SecurityID>
  </Instrument>
  <Side Value="1" />
  <TransactTime>20010907-09:25:56</TransactTime>
  <OrderQtyData>
   <OrderQty>5000</OrderQty>
  </OrderQtyData>
  <OrdType Value="2"/>
  <Price>62.5</Price>
  <Currency Value="USD" />
  <OrderCapacity Value="A" />
 <Order>
</ApplicationMessage>
</FIXMLMessage></FIXML>
```

b) Exemple d'interface de Trading