

Edité à PAU le lundi 22 décembre 2003.

SOMMAIRE

REMERCIEMENTS	4
AVANT-PROPOS	5
I - INTRODUCTION	6
II - ENVIRONNEMENT DE TRAVAIL	7
1) Présentation de la société	7
2) L'équipe de travail	7
3) Méthode de travail	8
4) Environnement technique	9
III - FORMATION	10
1) Généralités	10
2) COM+	10
3) Outils de développement	10
4) Accès aux données	12
IV - ÉTUDE DU PROGICIEL DE BILLETTERIE	14
1) Généralités	14
2) Données	15
3) Traitements	15
4) Architecture technique	17
V - MISE À JOUR DU PROGICIEL DE BILLETTERIE	18
1) Généralités	18
2) Système de Gestion de Bases de Données	19
3) Edition d'états	19
VI - PARAMÉTRAGE DU PROGICIEL DE BILLETTERIE	19
1) Généralités	19
2) Les lieux	20
3) Les abonnés	20
4) Les abonnements	21
VII - AJOUT D'UN MODULE AU PROGICIEL DE BILLETTERIE	22
1) Généralités	22
2) Analyse	22
3) Conception	23
4) Développement	24
5) Tests	25

SOMMAIRE

<i>VIII - CONCLUSION</i>	27
<i>IX - DOCUMENTATION</i>	28
1) Livres	28
2) L'Internet	28
<i>X - GLOSSAIRE</i>	28

REMERCIEMENTS

Ce rapport de stage n'existerait pas sans la société CIMA, qui m'a accueilli pendant un semestre. Je tiens donc à remercier l'ensemble des membres de la société pour m'avoir permis de passer un stage agréable.

Je tiens tout particulièrement à remercier Michel CRIADO, mon maître de stage, qui a bien voulu me faire confiance et m'a intégré à l'équipe de travail. Je remercie par ailleurs toute l'équipe pour m'avoir fait partager son expérience professionnelle.

Mes remerciements vont aussi aux clients avec lesquels j'ai été en contact pendant mon stage, à savoir les responsables administratifs de l'US Dax Rugby Landes et de la Section Paloise, respectivement Messieurs Jean Marc DEGOS et Jack LAURENT.

Je tiens également à remercier Patrick DEMEURISSE, mon tuteur de stage, pour sa disponibilité et les conseils qu'il a pu me donner.

Enfin, je remercie toutes les personnes qui ont eu la gentillesse de relire mon rapport et qui m'ont aidé à le réaliser.

AVANT-PROPOS

Le stage de six mois que j'ai effectué chez CIMA est caractérisé par une grande diversité de travaux réalisés. Le rapport présente une synthèse de mes activités. Le lecteur désirant lire un descriptif au jour le jour de mon stage pourra se reporter au suivi quotidien présenté dans la dernière **annexe**.

Je tiens également à souligner qu'un glossaire de certains termes clés mentionnés dans le document est disponible à la fin du rapport ainsi que dans l'**annexe 1**.

Je souhaite au lecteur autant de plaisir à lire les lignes qui suivent que j'en ai eu à les écrire.

I - INTRODUCTION

La conception de logiciels est sans aucun doute un travail complexe. Un espace sans fin attire, la possibilité d'inventer motive, mais ces libertés ne font pas de la conception de logiciels une activité idyllique. Tout n'est pas au mieux dans « le meilleur des mondes possibles ». Le chemin vers le succès d'un projet logiciel est jalonné de multiples pièges, de nombreux paramètres sont à prendre en compte, le principal étant le facteur humain.

De plus, les technologies ne cessent d'évoluer, avec l'apparition de nouveaux langages, de nouvelles plate-formes, les exigences des entreprises sont toujours plus grandes, la capacité d'adaptation devient alors un facteur essentiel de réussite.

Situé entre les utilisateurs et les machines, entre le métier et les systèmes informatiques, l'enseignement de l'IUP MIAGe répond à ces multiples contraintes. Cette formation scientifique et technologique est complétée par une expérience professionnelle de six mois, qui permet de mettre en application dans le contexte professionnel la démarche et les outils de l'ingénierie logicielle.

Pour mieux comprendre le processus de génie logiciel dans le domaine de l'informatique de gestion et pour apprendre des technologies qui feront l'informatique de demain, j'ai choisi de réaliser mon stage au sein de la société CIMA, qui m'a permis de voir l'industrie du logiciel avec les yeux d'un salarié débutant.

Par ailleurs, l'environnement de travail de la société m'a donné la possibilité d'appréhender un ensemble d'objets interagissant dans une structure cohérente, en d'autres termes, a constitué une première approche d'une architecture logicielle.

Dans un premier temps, une période de formation m'a permis d'étudier les technologies utilisées par CIMA. Le but de cet apprentissage a été de s'imprégner de l'essence de l'informatique actuelle, du passage du métier de développeur d'applications à celui d'assembleur de composants.

Une période d'intégration a suivi, me permettant de voir l'implémentation des concepts abordés précédemment. Ce passage de la théorie à la pratique, de la fiction à la réalité fait surgir la contrainte économique dans le jeu de la création logicielle.

Enfin, l'intégration en tant qu'analyste-programmeur à part entière au sein de l'équipe de travail a constitué une première expérience professionnelle significative et a été l'occasion de voir dans les moindres détails les multiples facettes de la vie d'une société de services.

C'est dans cette philosophie d'apprentissage du métier que les lignes qui suivent sont présentées, en prenant soin d'ouvrir les yeux sur la réalité du métier.

II - ENVIRONNEMENT DE TRAVAIL

1) Présentation de la société

La société CIMA (Communications Informatiques Micros Applications) est une SSII, basée à Pau, spécialisée dans l'informatique de gestion. Depuis six ans, elle fournit des prestations de services et conçoit des progiciels de gestion.






Les services proposés sont principalement tournés vers le choix et la mise en place d'infrastructures de communications (réseau informatique, serveur de messagerie, ...) et l'intégration de mainframes dans des environnements client / serveur. A ces prestations techniques s'ajoutent des conseils en gestion de projet. Ces conseils recouvrent l'audit, la définition des besoins et l'expertise.

CIMA conçoit également des progiciels de gestion, c'est à dire des logiciels qui répondent à un besoin spécifique d'une entreprise. Ce créneau de développement se caractérise par des sociétés qui ne trouvent pas sur le marché du logiciel la réponse à la spécificité de leurs métiers. En effet, les grands éditeurs informatiques de logiciels ou de progiciels sectoriels proposent des produits pour lesquels le nombre de clients potentiels est suffisant d'un point de vue économique. Lorsque les entreprises ont besoin d'un travail sur mesure, les SSII comme CIMA répondent à leurs attentes. Cette symbiose entre membres de l'industrie du logiciel est également illustrée par le partenariat de CIMA et de la société ARISTA: cette dernière, spécialisée dans les progiciels de gestion comptable et commerciale, s'adresse à CIMA lorsque les logiciels du marché ne peuvent répondre aux besoins des clients.

En parallèle des développements spécifiques, CIMA dispose d'un progiciel sectoriel de gestion de manifestations sportives et de spectacles. Ce progiciel de BILLETTERIE est un système informatique de réservations et de ventes de billets. Il est constamment amélioré et constitue le logiciel phare de la société.

2) L'équipe de travail

La société est à ce jour composée de cinq personnes, trois analystes-programmeurs, une secrétaire et le gérant.

<p>Michel CRIADO</p>  <ul style="list-style-type: none"> Il est le fondateur et le gérant de la société. Il s'occupe également des relations commerciales, de la gestion de projet et des services informatiques que la société propose. Il a une formation AES et est autodidacte en informatique, il travaille dans ce milieu depuis 1971. 		
		<p>Christine CAZABAN</p>  <ul style="list-style-type: none"> Elle est secrétaire-comptable. Elle s'occupe également de l'accueil des clients, du suivi des affaires et des sauvegardes des données. Elle est titulaire d'un BAC Techniques Administratives, a 3 ans d'expérience professionnelle au sein de la société CIMA.
		<p>Pantxika ARANDIA</p>  <ul style="list-style-type: none"> Elle est analyste-programmeur. Elle s'occupe également du service après-vente du progiciel de BILLETTERIE. Elle est titulaire d'un DESS MASS, a 5 ans d'expérience professionnelle dont 3 ans au sein de la société CIMA.
		<p>Gilles BENOIST</p>  <ul style="list-style-type: none"> Il est analyste-programmeur. Il est régulièrement envoyé en délégation chez des clients. Il est titulaire d'un DUT d'informatique, a 22 ans d'expérience professionnelle dont 3 ans au sein de la société CIMA.
		<p>Céline LACABERATS</p>  <ul style="list-style-type: none"> Elle est analyste-programmeur. Elle est titulaire d'un Destaup (Diplôme d'Etudes Scientifiques et Techniques Appliquées de l'Université de Poitiers), a 1 an d'expérience professionnelle au sein de la société CIMA.

Deux catégories de personnes apparaissent clairement : les analystes-programmeurs, qui s'apparentent le plus souvent à des programmeurs et passent la majorité de leur temps à écrire des lignes de code tandis que la gestion de la société est assurée par Michel CRIADO et sa secrétaire.

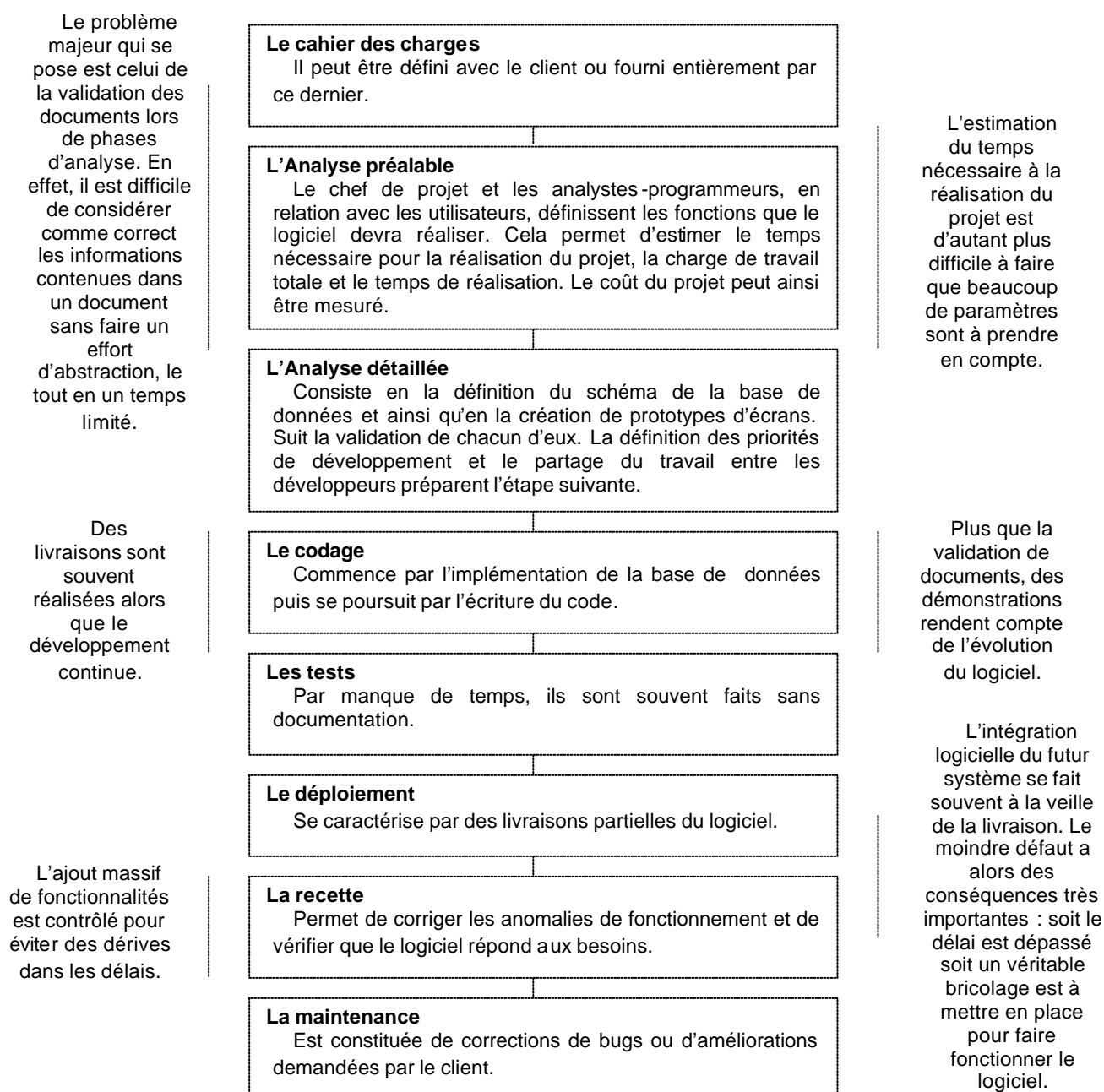
Les joies du métier d'analyste-programmeur se résument en quelques concepts forts, le plaisir de créer, la satisfaction de construire des choses utiles à autrui, le bonheur d'apprendre tous les jours. Cependant, l'activité n'est pas exempte d'inconvénients, la nécessité d'écrire des lignes de code parfaites (dans le sens où une erreur de syntaxe se traduit inévitablement par une erreur d'exécution), la dépendance vis à vis des programmes d'autrui, le débogage des applications peuvent être perçus comme de pénibles impératifs.

Le chef de projet, Monsieur CRIADO, à côté de son activité d'administration de la société dans laquelle Mlle CAZABAN est impliquée, doit intégrer au mieux les différents facteurs qui interviennent dans le génie logiciel, pour faire en sorte que les joies surpassent les peines, que le projet ne se noie pas dans des sables mouvants dont la sortie est coûteuse en temps et en argent.

Pour ma part, l'objectif du stage consiste à réaliser un travail analogue aux trois analystes-programmeurs de la société, en étant positionné plus particulièrement sur la BILLETTERIE.

3) Méthode de travail

La petite structure de l'entreprise n'a pas incité le gérant à la mise en place d'une méthode de travail strictement définie, préférant favoriser la réactivité de la société. Chacun des membres de l'équipe dispose d'une autonomie peu commune, tant au niveau de la démarche de travail que du formalisme utilisé. Une méthode est cependant récurrente lors de chaque projet, qu'il est possible de présenter ainsi :

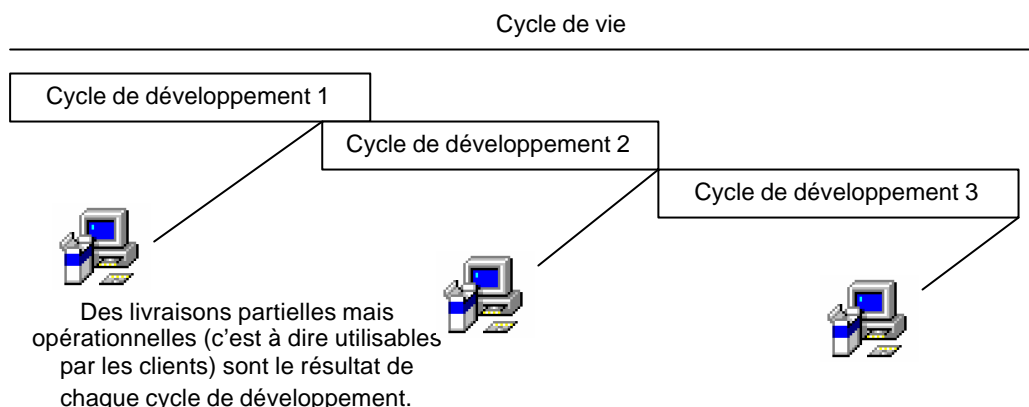


La démarche exposée ci-dessus donne la prépondérance à la phase de codage dans le cycle de développement et, les membres de l'équipe en conviennent, la démarche suivie incite à arriver le plus vite possible à l'écriture du code. En effet, les contraintes financières imposent d'accepter des délais très courts. Le réflexe est alors de boucler rapidement les phases d'analyse (et de conception quand elle existe) pour écrire du code au plus vite. La livraison est la priorité.

Des avatars inhérents à cette méthode de travail sont principalement la spécificité du logiciel (la réutilisation est réduite au minimum, ce qui fait qu'à chaque nouveau projet, il faut « repartir de zéro ») et le peu de documentation réalisée.

Par conséquent, dans une telle démarche, il est nécessaire de se concentrer sur les éléments fondamentaux du logiciel (les 20 % des composants) qui couvrent la plupart du temps les besoins essentiels à satisfaire (80 % des besoins).

Il faut souligner que le cycle de développement présenté se répète souvent plusieurs fois dans le cycle de vie des logiciels créés par CIMA. En fait, le cycle de vie se compose d'une succession (ou d'une superposition) de cycles de développement, selon l'enchaînement suivant :



Dissocier ainsi cycle de vie et cycle de développement permet de disposer d'un logiciel en évolution constante. De plus, la vie de la société en dépend, chaque livraison étant synonyme d'une facturation au client.

Enfin, la phase de codage étant prépondérante, éviter au maximum les erreurs de programmation permet d'améliorer le processus. La mise en place de règles de programmation y contribue grandement. Elles rationalisent le développement. Ces standards de codage interne bénéficient à l'ensemble du projet en cours, mais aussi à tous les projets suivants qui utilisent le code développé lors du projet. Les avantages à long terme sont des développements plus courts et de meilleure qualité, une maintenance plus facile. L'investissement initial en temps et en argent correspondant à la mise en place est rentable au niveau de chaque projet futur.

Cette mise en place de règles de programmation est particulièrement important dans une société comme CIMA qui utilise intensément la réutilisation au niveau du code source. Pour ce faire, des paquetages de fonctions et de données ont été créés. Cela permet de structurer le code et offre un niveau de granularité adéquat pour une réutilisation correcte. En développement, cela permet de répartir le travail entre les développeurs. Mises en place lors de la création de la société, force est de constater qu'elles n'ont pas été toujours respectées par les différents intervenants, ce qui nuit à la qualité de certains nouveaux projets et complique la maintenance d'autres.

4) Environnement technique

La diversité des besoins informatiques impose l'utilisation de différents outils de développement, en d'autres termes, nécessite une boîte à outils adaptée. En l'occurrence, une véritable stratégie doit être mise en place pour le choix des outils.

La société CIMA s'est tournée vers la plate-forme Microsoft qui offre une structure riche et de haut niveau, un intégré qui minimise les risques et simplifie le travail, en adéquation avec le marché sur lequel CIMA est positionnée.

L'autre caractéristique essentielle de l'environnement technique est l'utilisation de l'architecture client / serveur au sein de locaux de l'entreprise. Le schéma en **annexe 1** illustre cette architecture matérielle.

Les produits utilisés sont les systèmes d'exploitation Microsoft Windows NT et 9.x, le SGBRD (Système de Gestion de Bases de Données) Microsoft SQL Server, les EDI (Environnement de Développement Intégré) Microsoft Visual Basic et Microsoft Visual InterDev, le gestionnaire de code source Microsoft Visual SourceSafe et enfin le générateur d'états Seagate Crystal Reports.

Remarque : on peut regretter qu'au niveau des phases d'analyse et de conception, aucun logiciel ne soit utilisé de manière systématique. L'offre de Microsoft dans le domaine est pourtant intéressante, avec notamment Microsoft Visual Modeler, l'équivalent du logiciel Rational Rose inclus à la suite Visual Studio de l'éditeur de Redmond.

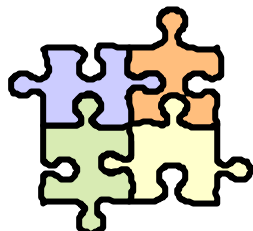
Le choix de cet environnement part d'un constat simple : il améliore la productivité des développeurs en leur offrant des outils faciles à utiliser, il simplifie la création des applications en donnant la possibilité de construire des logiciels riches, complexes et modulaires et enfin (surtout devrait-on dire), il permet de mieux satisfaire le client.

L'intégration des produits utilisés est possible grâce à des fondations communes qui donne à l'ensemble les bases pour une évolutivité et une cohérence de facto. COM+, pour ne pas la nommer, constitue la base de cet ensemble, le modèle qui sert de fondations au système. Le paragraphe qui suit commence par une brève description de ce modèle.

III - FORMATION

1) Généralités

La première partie du stage m'a permis de découvrir les technologies utilisées par CIMA. Cet apprentissage porte en grande partie sur l'architecture logicielle à la base de l'ingénierie du logiciel en place dans l'entreprise.



La philosophie d'approche a consisté à comprendre comment, disposant de multiples pièces de puzzles, il est possible de les assembler en un tout cohérent, et ce à moindre frais.

Cet apprentissage s'est déroulé grâce à la lecture d'ouvrages spécialisés en association avec des formations interactives de Microsoft, en l'occurrence les Mastering Series.

2) COM+

Selon Maarten Boasson (IEEE Software), « la conception de logiciels fait sans aucun doute appel à l'art dans une large mesure. Mais, en l'absence de règles, l'expression artistique se transforme souvent en une conception chaotique. Pour produire un système ouvert, nous devons utiliser quelques règles bien définies pour gouverner les interactions entre les systèmes et les sous-systèmes. »

Cette problématique est à la base des architectures objet qui émergent actuellement. En effet, les applications ont atteint aujourd'hui un niveau de sophistication élevé, suite à des développements longs et successifs. Les utilisateurs sont devenus exigeants, placés au cœur de la démarche, le temps du « programmeur-roi » est bel et bien révolu. Se pose alors le problème de l'évolution des applications monolithiques :

- Elles sont complexes donc difficiles à maintenir et leur évolution engendre un risque certain.
- Elles sont en général peu modulaires.
- La communication entre applications est difficile.

Pour franchir ces limites et répondre aux nouveaux besoins des entreprises (distribution, modularité, réutilisabilité), il est nécessaire de relever les défis suivants :

- Des services génériques sont nécessaires.
- Il faut être capable de prendre « le train de l'objet ».
- La distribution des applications est incontournable.

Mieux que la POO (Programmation Orientée Objet) qui a eu pour résultat la constitution dans les entreprises d'îles d'objets, les architectures objet, et les logiciels à base de composants qui en sont le reflet concret pour les consommateurs d'informatique, permettent de relever ces défis.

Une architecture logicielle peut se définir par un système de base dans lequel des composants coopèrent pour former un tout cohérent. Un composant peut être vu comme une brique élémentaire du système, une pièce du puzzle, qui, grâce à l'architecture logicielle, est capable de communiquer avec d'autres briques, de s'assembler avec d'autres pièces du puzzle.

A ce jour, l'une des architectures objet la plus connue, et sans aucun doute la plus utilisée, s'appelle COM+ (Component Object Model). Pour avoir un aperçu plus complet de cette architecture, le lecteur est invité à consulter l'**annexe 2**, qui dresse un rapide tour d'horizon du modèle objet.

Après la rédaction de cet aperçu de COM+, j'ai pu appréhender les avantages multiples qu'apporte une architecture logicielle, à commencer par la possibilité de communication entre composants. Sans à priori se connaître (liaison dynamique de composants), sans avoir nécessairement la même origine (indépendance vis à vis du langage de programmation utilisé), des composants peuvent communiquer grâce au rôle de facilitation de COM+ : il initie la communication puis, lorsque le dialogue est établi, il se retire laissant les composants se parler dans un langage commun (standard binaire d'interopérabilité).

3) Outils de développement

Le métier de développeur change. Grâce aux architectures à base de composants logiciels, la plupart des développeurs se transforment de plus en plus en colleurs de composants pour assembler des applications spécifiques.

Bien entendu, il faut que certains éditeurs fournissent ces composants métier, ces briques de base, pour que l'assemblage puisse se faire, mais le créneau devient affaire de spécialistes. Pour le reste des éditeurs, la tendance actuelle est de produire vite et bien.

A ces besoins, Microsoft apporte une réponse adaptée : dans son environnement de génie logiciel Visual Studio, l'idée porteuse est de développer aussi simplement que possible, via des « drag et drop » de contrôles, des cadres de travail prédéfinis. D'ailleurs, avec sa stratégie .NET, Microsoft compte pousser jusqu'aux développements Web le « look and feel » de ses produits Visual. Les outils de développement utilisés chez CIMA constituent donc un point de départ idéal. En effet, Visual Basic 6.0 et Visual InterDev 6.0 permettent en quelques lignes de codes de créer des logiciels complexes.

Visual InterDev 6.0 est utile pour les développements tournés vers les technologies de l'Internet. Il combine la fonction d'éditeurs de code HTML, d'éditeurs WYSIWYG, tout en offrant la possibilité d'intégrer des données aux sites Web créés.

Visual Basic 6.0, sur lequel j'ai concentré mon étude, se positionne comme un EDI qui permet de créer une grande variété de types de composants, tous basés sur le modèle COM+. Le point fort de Visual Basic est qu'il simplifie grandement le développement et qu'il permet aux entreprises qui l'utilisent de se concentrer sur l'essentiel, la création rapide de logiciels. Microsoft le présente bien comme un outil d'assemblage, face à Visual C++ par exemple, les logos des deux produits le prouvent :

Visual Basic est un
outil d'assemblage de
composants.



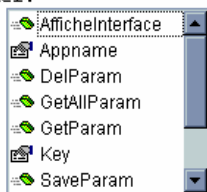
Visual C++ offre un
niveau de granularité
plus fin.



A la base, le langage BASIC (Beginners All-purpose Symbolic Instruction Code) a permis au début des années 80 la démocratisation de la programmation. La simplicité du langage initial est toujours présente, mais le Basic a bien grandi et s'est doté de nouvelles fonctionnalités (collection, module de classe, ...) qui en font un langage professionnel. De plus, le mot Visual a fait son apparition, reprenant évidemment le concept de programmation visuelle, rapide et facile. De plus, la gestion de la programmation événementielle de manière native fait de Visual Basic un outil adapté aux systèmes d'exploitation Windows. S'ajoute à cela une programmation facilitée par la saisie semi-automatique en cours d'écriture du code et la vérification de syntaxe en arrière plan.

```
Dim maRef As New parametre

'on définit le nom de l'application
maRef.Appname = App.Name
'on définit le nom de la section
maRef.section = "CLIENT"
'on affiche l'interface
maRef.
```

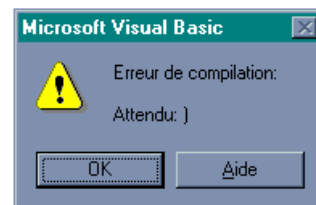


La technologie
Intellisense facilite
l'écriture du code en
dévoilant l'interface
des composants.

```
Dim maRef As New parametre

'on définit le nom de l'application
maRef.Appname = App.Name
'on définit le nom de la section
maRef.section = "CLIENT"
'on affiche l'interface
if (maRef.AfficheInterface
```

En arrière plan, Visual
Basic interprète le code au
fur et à mesure de la
frappe, réduisant ainsi les
erreurs éventuelles de
compilation.



Par ailleurs, son image de langage « lent » née à l'époque des premières éditions (qui ne permettaient de créer que des applications en code interprété) a disparu aujourd'hui. En effet, le compilateur C++ qui génère à présent du code natif offre au langage ses lettres de noblesse.

Remarque : pour vous persuader que les exécutables subissent une compilation classique, il est ludique de visualiser le répertoire du projet courant pendant qu'une compilation est effectuée. Une série de fichiers objets (les .obj du C++) sont visibles, attestant qu'une compilation a effectivement lieu.

Malgré tout, les concepts objets présents actuellement dans Visual Basic 6.0 (encapsulation, polymorphisme, méthodes amies) sont insuffisants pour qualifier le langage d'orienté objet. La version 7.0 du produit va véritablement changer la donne, et l'introduction de l'héritage, de la surcharge, de la redéfinition, des membres de classe, des constructeurs permettra à Visual Basic de passer du statut de langage « basé sur l'utilisation d'objets » à celui « d'orienté objet ».

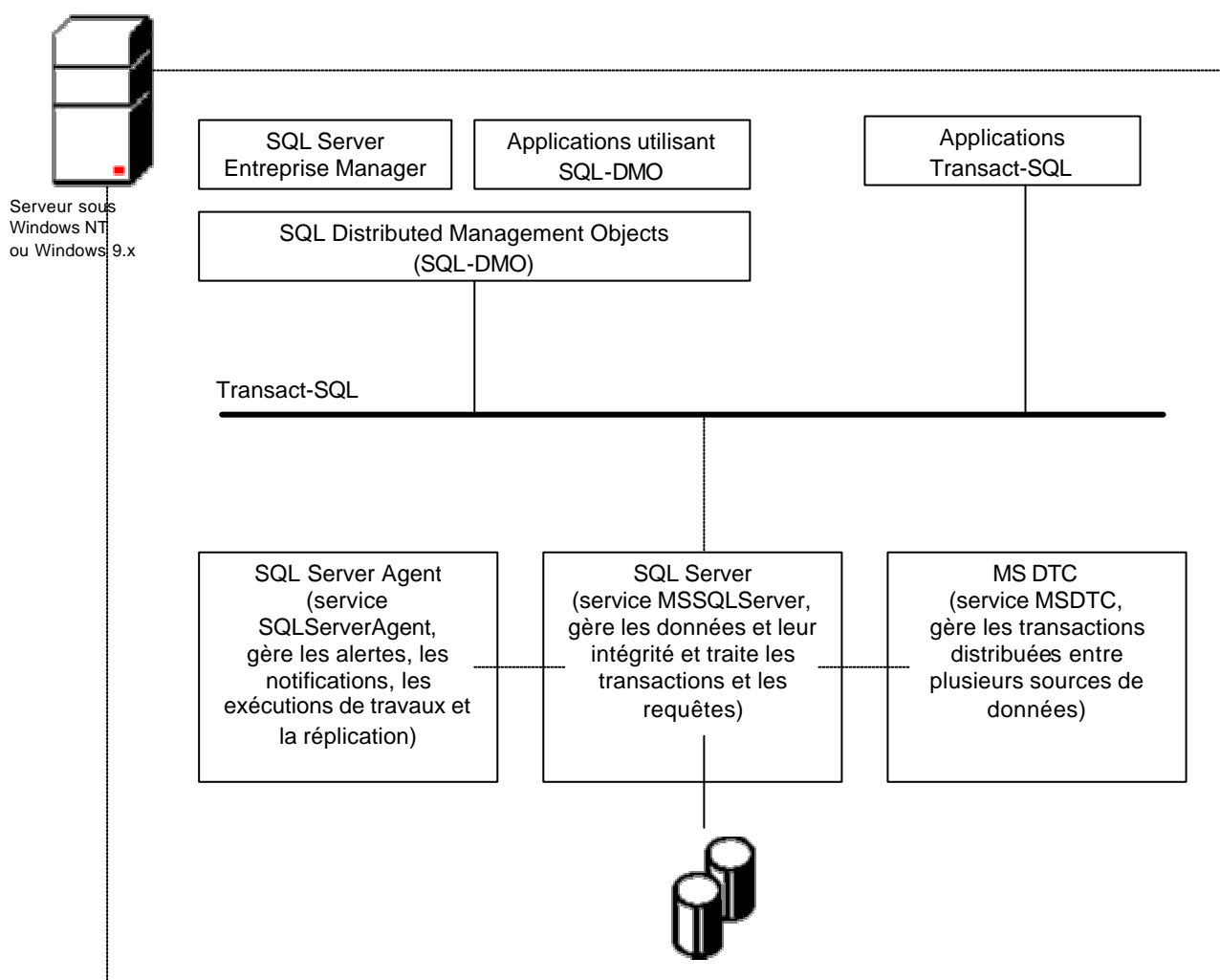
4) Accès aux données

Avec les éditeurs d'outils et d'applications comme Inprise ou IBM et les éditeurs de progiciels sectoriels comme SAP ou Baan, les éditeurs de systèmes de gestion de bases de données constituent l'un des trois acteurs fondamentaux de l'industrie du logiciel. Sur ce marché, l'offre que propose Microsoft avec SQL Server est de premier ordre, en concurrence directe avec Oracle d'Oracle Corporation.

Suite à la découverte de la mise en œuvre de bases de données sous Oracle dans le cadre universitaire, l'initiation à l'administration système sous SQL Server dans l'architecture client / serveur de la société CIMA a été une expérience intéressante.

Les caractéristiques principales de SQL Server sont sa facilité de prise en main, ses outils d'administration et sa parfaite intégration avec Windows NT. Par rapport à Oracle, plus robuste mais plus technique, SQL Server constitue donc une solution très attractive pour les petites et moyennes entreprises.

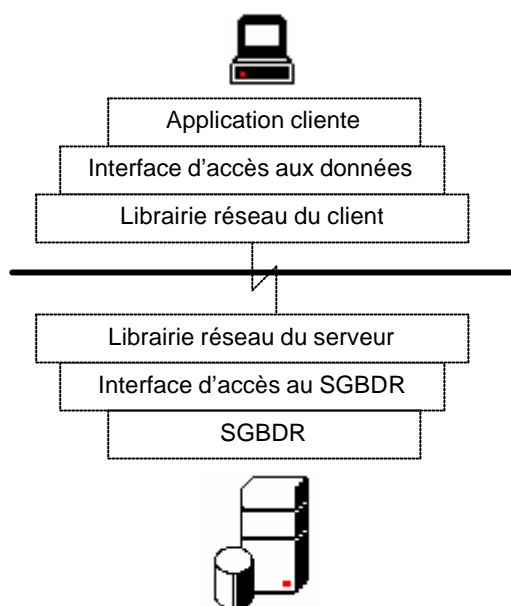
L'architecture d'administration se compose de différents outils accédant via plusieurs interfaces au cœur du SGBDR :



Remarque : il faut bien le reconnaître, l'utilisation d'outils graphiques dans un environnement agréable facilite grandement l'utilisation quotidienne du SGBDR. De plus, le fait que l'utilitaire Enterprise Manager soit un composant de MMC (Microsoft Management Console) favorise la cohérence de l'administration générale des différents serveurs de l'architecture informatique (MMC concerne les serveurs de l'offre Microsoft Back Office, qui inclut par exemple le serveur de messagerie Microsoft Exchange).

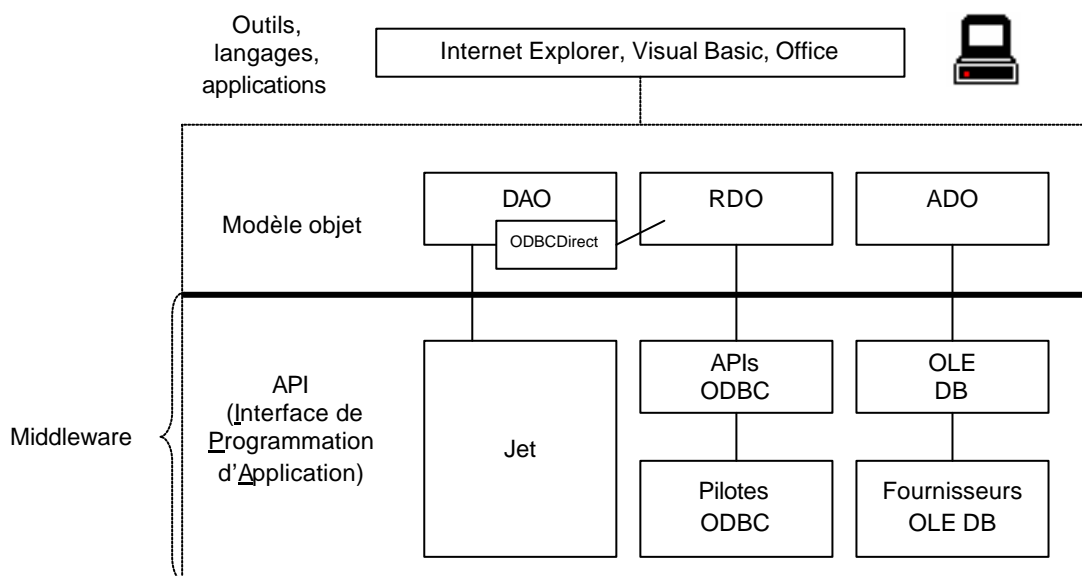
Le fait de disposer de données pour toute organisation n'a de sens que si les données sont utilisées. Par conséquent, les méthodes d'accès aux données ont une place prépondérante dans toute architecture, et plus particulièrement dans le mode client / serveur où les données sont séparées des traitements et de la présentation à l'utilisateur.

L'architecture d'accès aux données se présente en couches selon la topologie suivante :



Dans cette configuration, le choix de l'interface d'accès aux données est crucial. En effet, cette couche cache une complexité induite par une plus grande fluidité de communication pour les applications. Le principe de cette couche appelée aussi middleware (ou médiateur), apparaît clairement ci-dessus : le « logiciel du milieu » constitue une interface d'accès qui permet de dialoguer avec le serveur de données. Le médiateur cache l'hétérogénéité des réseaux traversés, fournissant une même interface quelle que soit le SGBD.

Cette couche peut s'illustrer ainsi :



Les différents composants de cette interface d'accès aux données sont détaillés dans le tableau suivant :

Terminologie	Définition
DAO (Data Access Objects)	DAO fournit une hiérarchie d'objets pour manipuler les données Jet, ISAM ou bases de données relationnelles. Ces objets sont devenus l'interface la plus utilisée en Visual Basic pour accéder aux données. L'extension ODBCDirect de DAO permet la connexion à des bases de données ODBC, sans le chargement du moteur Jet.
RDO (Remote Data Objects)	RDO a été conçu comme complément de DAO, dans le cas de bases de données distantes. Ce modèle objet hiérarchique est une fine couche se situant au-dessus de la couche ODBC, et permet une utilisation d'ODBC grandement simplifiée mais complète.
ADO (ActiveX Data Objects)	Ce modèle objet s'appuie sur les API de la technologie OLE DB et a été conçu dans deux optiques précises : accéder à n'importe quelle source de données et permettre l'utilisation et la montée en charge sur Internet. Le modèle ADO est souple, permettant la création et la manipulation des objets indépendamment les uns des autres.
Jet	Jet est un moteur de bases de données relationnelles. Il a évolué depuis sa première version qui ne permettait l'accès qu'à des bases Microsoft Access. Comme tout moteur de bases de données, il propose des services de gestion de données (définition, stockage, manipulation, intégrité, partage, sécurité des données).
ODBC (Open DataBase Connectivity)	ODBC est un protocole standard d'accès aux bases de données. Il permet à une application de se connecter de la même façon à un ensemble de bases de données relationnelles différentes. Il a été défini par un ensemble de constructeurs de systèmes de bases de données.
Pilote ODBC	Un pilote ODBC est une DLL qui répond à un ensemble de spécifications. Le pilote implémente les accès spécifiques à une base de données particulière. Il existe des pilotes ODBC pour des bases Microsoft SQL Server, Oracle, ... Le pilote s'appuie sur les librairies de bas niveau pour accéder aux bases (DBLib pour Microsoft SQL Server, SQL*Net pour Oracle, ...).
OLE DB	OLE DB est une spécification d'interfaces, exposant des interfaces COM de bas niveau (difficile à appeler depuis des langages tels que Visual Basic) et également des interfaces COM de plus haut niveau.
Fournisseur OLE DB	Un fournisseur OLE DB implémente les interfaces OLE DB. Il permet à un consommateur OLE DB d'accéder à tout type de sources de données d'une façon uniforme. Un fournisseur OLE DB joue le même rôle qu'un pilote ODBC qui fournit un mécanisme uniforme d'accès à des données relationnelles. Les fournisseurs OLE DB fournissent un accès uniforme aux données relationnelles et non relationnelles. Par ailleurs, les fournisseurs OLE DB sont construits sur la base du modèle COM alors que les pilotes ODBC ont été créés sur une spécification d'API C.

IV - ÉTUDE DU PROGICIEL DE BILLETTERIE

1) Généralités

L'étude présentée dans les lignes qui suivent reflète ma démarche pour analyser la BILLETTERIE de la société CIMA. Pour des raisons de confidentialité évidente, seule une partie des documents créés au cours de cette étude est fournie.

Comme dans la plupart des progiciels du marché, l'idée de répartir les fonctions fondamentales du métier entre différents composants s'applique à la BILLETTERIE. Le niveau de granularité présenté (le niveau d'abstraction choisi) reprend cette idée avec une collection de composants métier qui correspondent chacun à des chaînes complètes de processus standards du métier de gestion de manifestations sportives et événementielles.

L'étude de la BILLETTERIE ne consiste pas à faire une rétro-conception détaillée du système mais à en comprendre suffisamment pour s'y insérer. Pour arriver à cette compréhension, il faut pouvoir se ramener à une vision synthétique du système, en suivant un mécanisme d'abstraction pour se concentrer sur l'essentiel en laissant les détails en coulisse.

Le fait est que, pour point de départ, je n'avais que le détail, en d'autres termes, l'implémentation du système en réel, et seulement l'implémentation, avec très peu de documentation sur ce dernier. Pour arriver à modéliser la BILLETTERIE, je me suis basé sur une approche systémique, en partant du système complet en état de marche, en le divisant en sous-systèmes (par processus principaux), en étudiant chaque sous-système et les inter-actions entre ces derniers. En d'autres termes, j'ai réalisé ce que l'on appelle dans la méthode MERISE, l'étude de l'existant : j'ai modélisé de manière textuelle et formelle l'existant afin d'arriver à un niveau de détails suffisant pour comprendre les finalités du système.

2) Données

Le logiciel permet de gérer l'ensemble des données relatives à un spectacle, à savoir :

- Les **types de spectacles** et les **spectacles**.
- Les **lieux** utilisés par les spectacles.
- Les **abonnés** et les **abonnements**.
- Les **pré-réservations**, les **réservations** et les **ventes**.
- Les **billets** et les **cartes**.
- L'**historique** de l'activité.

Pour essayer de comprendre la manière dont les données sont structurées, la documentation actuelle du logiciel est insuffisante. En conséquence, des outils permettant de générer des modèles visuels compréhensibles facilement sont donc nécessaires. Visual InterDev permet de créer des MLD sous forme graphique (SQL Server 7.0 implémente cette fonctionnalité, à l'opposé des versions antérieures de ce produit). Mieux encore, des outils de génie logiciel comme AMC Designor permettent de régénérer des MLD et des MCD en partant des bases de données existantes. A partir de ces formalisations schématiques des données, la compréhension de ces dernières est facilitée.

Le reverse engineering réalisé a donc permis, à partir des bases de données implantées, de générer le MLD puis, à partir du MLD, le MCD, grâce à l'AGL AMC Designor. Etant donné le nombre de relations et le nombre de tables, il est nécessaire de parler au pluriel, plusieurs modèles ayant été en réalité créés. En effet, il est impossible de présenter un seul et même schéma pour le système, la lisibilité en serait nulle. Ce découpage en paquetage (en unités sémantiques), qui améliore la compréhension, simplifie également les évolutions futures.

A partir des modèles conceptuels de données présentés, il devient facile de décrire les données. Pour donner une idée de l'essence des données utilisées et de l'importance du schéma complet, le nombre de tables et d'associations par paquetages est présenté ci-dessous :

- **Gestion de la sécurité** : 8 tables, 4 associations.
- **Gestion des salles et des places** : 7 tables, 3 associations.
- **Gestion des types de spectacles** : 7 tables, 3 associations.
- **Gestion des abonnés et des abonnements** : 17 tables, 5 associations.
- **Gestion des pré-réservations, des réservations et des ventes** : 14 tables, 3 associations.
- **Gestion des billets et des titres** : 9 tables, 2 associations.
- **Gestion des tarifs** : 7 tables, 4 associations.
- **Gestion des règlements** : 5 tables, 1 association.

Les MLD partiels n'apportent pas de nouvelles informations, si ce n'est l'augmentation du nombre de relations présentes dans les schémas.

L'intérêt d'être remonté au niveau conceptuel m'a permis de fournir une spécification du système et ainsi d'en comprendre le fonctionnement général. Cette compréhension est d'autant plus facile que la spécification fournie se compose de descriptions textuelles et de représentations formelles. Le texte a l'avantage du détail mais son interprétation peut varier d'une personne à une autre. Par contre, la représentation formelle a le mérite de la précision mais ne décrit pas le « pourquoi », la philosophie d'ensemble. Par conséquent, l'alliance du texte et d'une représentation formelle m'est apparue importante dans cette étude.

3) Traitements

La modélisation a été faite en partant de ce que l'utilisateur voit du système (les cas d'utilisation du système) pour remonter progressivement vers les bases du progiciel, en d'autres termes, du technique vers le conceptuel, de l'implémentation vers sa représentation abstraite (les adeptes de MERISE voient que l'on « remonte » la courbe du soleil).

D'un point de vue utilisateur, la principale caractéristique est l'interface intuitive de la BILLETTERIE. La plupart des exécutables ne demandent pas de formation particulière et se basent sur une métaphore de la réalité pour certains, comme le clic de souris sur un endroit des tribunes pour en afficher les places, le fait de sélectionner les places pour les réserver. L'illustration au début de la page suivante montre ce principe avec l'interface de vente et de réservation de places. Celle-ci présente l'image schématique de la salle de spectacle (en l'occurrence le stade de l'U.S. Dax Rugby Landes) sur laquelle il suffit de cliquer pour afficher les places, symbolisées par des carrés dans la partie inférieure de l'écran. La souris permet ensuite de sélectionner les places que l'on veut vendre ou réserver. Il n'est donc pas difficile de prendre en main ce processus essentiel de la BILLETTERIE.

Le logiciel est divisé en composants logiciels indépendants qui dialoguent avec les données gérées par un SGBDR, en l'occurrence SQL Server. Cependant, cette structure n'a pas de réalité intrinsèque, le cerveau est donc privé d'une de ses armes favorites, la représentation visuelle. De plus, la documentation sur les traitements réalisés par la BILLETTERIE est encore une fois limitée. Un outil de visualisation graphique comme VB Miner de la société Cast a été très utile.

VB Miner est un outil d'Application Mining, c'est à dire une application qui analyse syntaxiquement et sémantiquement un code source et qui en fournit des vues graphiques. Il permet donc d'explorer les inter-actions entre les objets, facilite la rétro-ingénierie, améliore la compréhension de l'application, tout ce dont j'avais besoin pour faire une meilleure analyse (le lecteur peut se reporter à l'**annexe 7** pour visualiser les schémas que VB Miner permet de réaliser).

Il ne m'a pas paru nécessaire de suivre aveuglement la démarche MERISE et réaliser une analyse organisationnelle puis conceptuelle des traitements car il me fallait simplement comprendre rapidement et dans les grandes lignes le fonctionnement général des traitements de la BILLETTERIE. J'ai donc adapté ma démarche à la situation et choisi de passer directement au niveau conceptuel, sans réaliser des MOT (Modèle Organisationnel de Traitement) ou des MCT (Modèle Conceptuel de Traitement), en donnant directement une description générale et textuelle du progiciel.

Trois phases distinctes permettent de classer les traitements relatifs à la BILLETTERIE : la mise en exploitation du progiciel, réalisée une seule fois à l'installation ou lors d'importantes modifications du processus métier, la définition des conditions d'utilisation, dont la fréquence peut aller de la semaine à l'année et enfin les traitements quotidiens. La compréhension de ces trois phases permet de se familiariser avec les processus métier que le progiciel gère.

La mise en exploitation du logiciel se compose des étapes suivantes :

- **La gestion des lieux** : cette étape consiste en la définition, pour chaque lieu sur lequel des spectacles se dérouleront, de la division du lieu en modules, chaque module regroupant un nombre particulier de places, accessibles par différents accès. Une définition judicieuse de ces paramètres influe grandement sur la simplicité quotidienne d'utilisation.
- **La gestion des exploitants** : il faut définir lors de cette étape les conditions d'utilisation du progiciel, c'est à dire « qui peut utiliser quoi sous quelles conditions ». Définir les utilisateurs, les points de vente et la sécurité d'accès aux fonctions proposées par le progiciel sont les principales tâches réalisées à cette heure.
- **L'intégration des données existantes** : lors de la mise en place du progiciel chez un client, les données existantes, principalement la liste des abonnés et des partenaires du club, sont à « insérer dans le progiciel ».
- **La mise en place du matériel** : cette mise en place consiste en l'installation du matériel nécessaire à la BILLETTERIE. L'**annexe 3** donne une vision schématique des matériels informatiques qui interviennent dans le progiciel.

Après la mise en exploitation vient la définition des conditions d'utilisation. Dans cette étape, on distingue la gestion des spectacles dont la fréquence de réalisation est hebdomadaire ou mensuelle des traitements moins fréquents que sont la gestion des types de spectacles et la gestion des abonnements. Leurs descriptions sont les suivantes :

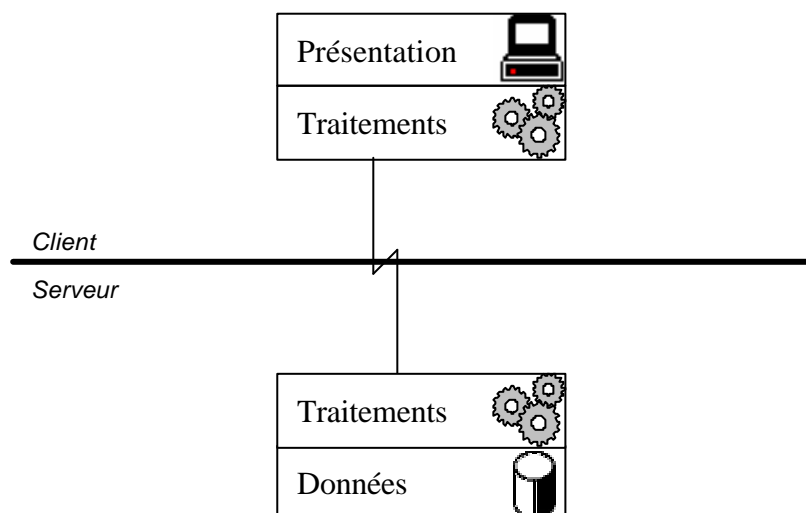
- **Gestion des types de spectacles** : on détermine les prix affectés à chaque module pour les différents types de spectacle (par exemple Championnat de France, Coupe d'Europe, ...) et les modèles de titre (cartes d'abonnement, billets, ...) disponibles.
- **Gestion des abonnements** : ce traitement se caractérise par la gestion des abonnements que l'on veut mettre en place, en paramétrant le prix, la période, le modèle de titre à utiliser.
- **Gestion des spectacles** : cette gestion consiste à mettre en place un spectacle c'est à dire à définir les tarifs par module, les points de vente, le modèle de billet utilisé ainsi que les conditions de vente.

Les traitements quotidiens viennent enfin. On y distingue la préparation du spectacle (qui consiste à une phase de réservation et de vente de places quelques jours avant la manifestation), la phase « d'avant match » (les quelques heures précédant la manifestation) et les traitements qui suivent la clôture du spectacle :

- **La gestion des abonnés** : ce traitement réalisé le plus souvent dans la semaine précédant la manifestation permet de gérer les informations relatives à un abonné, des places qu'il a réservées jusqu'à la manière dont il a payé son abonnement.
- **La gestion administrative** : des traitements spéciaux peuvent être nécessaires à tout moment lors du fonctionnement du système comme l'édition de duplicata de cartes d'abonnement, la réintégration des titres, la visualisation de l'historique d'une place, d'une vente, la gestion des pré-réservations, des titres.
- **La vente et la réservation de places** : ce traitement est le cœur du progiciel car il est utilisé le plus souvent et ses résultats sont directement liés avec la finalité du métier, à savoir la vente de billets et l'encaissement des paiements. Une seule et même interface (celle présentée sur la page précédente) permet de choisir les places, le prix et le mode de paiement.
- **La gestion des termods** : quelques heures avant les manifestations, il est nécessaire de paramétrer les terminaux modulaires (termods) qui sont utilisés pour la vente de billets aux guichets. Chaque guichetier dispose ainsi d'un stock de places à vendre à un certain prix.
- **Le contrôle d'accès** : pour vérifier la régularité des entrées de personnes titulaires d'une carte d'abonnement, une lecture du code-barre présent sur chaque carte est réalisée via des lecteurs optiques reliés au système (comme schématisé dans l'**annexe 3**).
- **L'administration du contrôle d'accès** : après la fermeture des portes, il est possible de visualiser les problèmes relatifs au contrôle de l'accès des abonnés via ce traitement.
- **L'édition des bilans** : en quinze états différents, les informations vitales du système peuvent être visualisées et imprimées. Par exemple, dans le milieu rugbystique, les comptes rendus financiers que les clubs doivent remettre aux délégués de la fédération sont directement générés par le système.

4) Architecture technique

En ce qui concerne la répartition des différents composants de la BILLETTERIE, le schéma suivant illustre la situation :



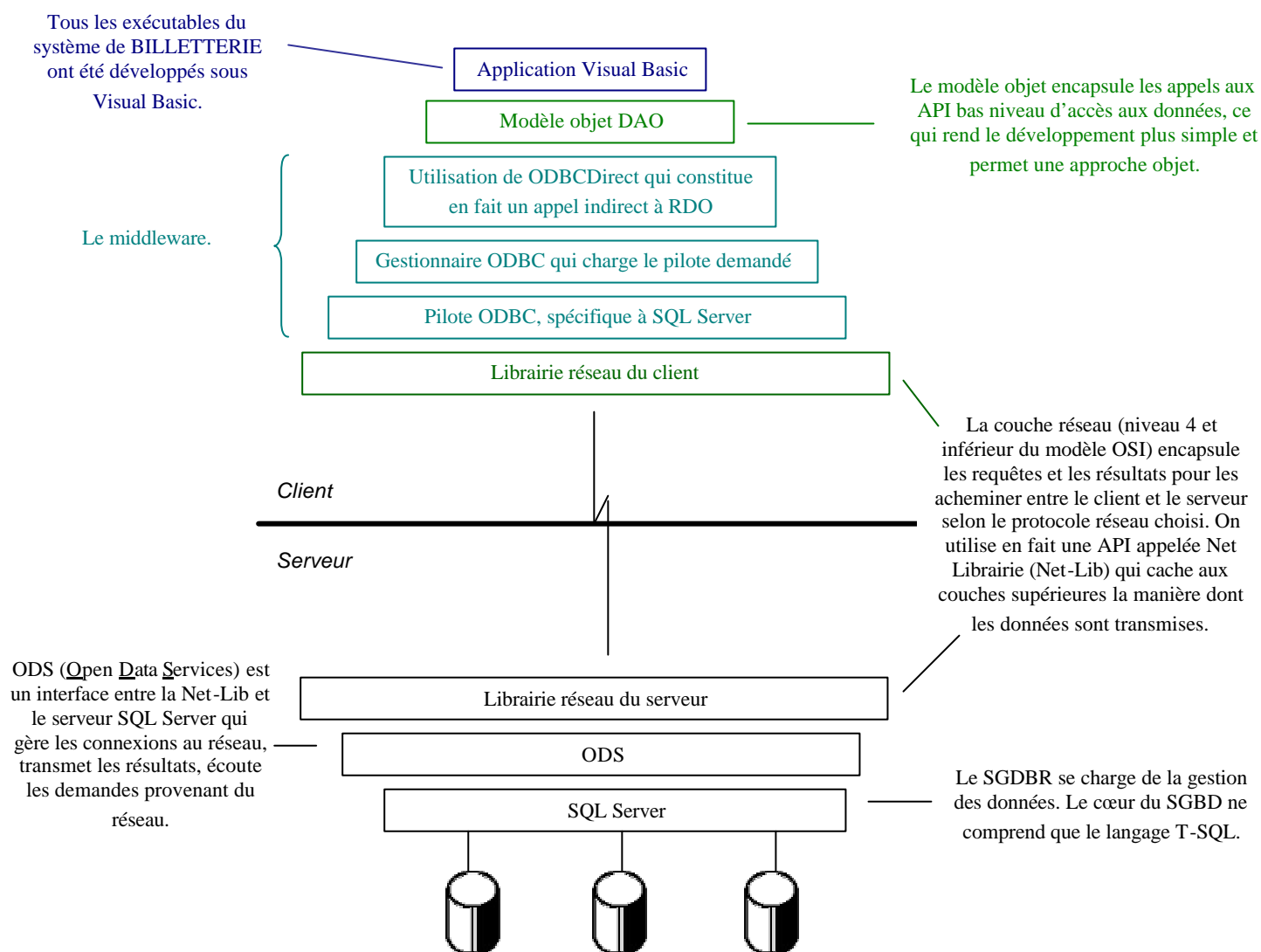
On voit donc que la BILLETTERIE est un logiciel qui s'exécute dans un environnement client / serveur dans lequel les traitements sont distribués entre le poste client et le serveur. Pour reprendre la terminologie du Gartner Group, on se situe dans une architecture client / serveur de traitement de 2^e génération.

La répartition des traitements entre le client et le serveur se situe aux niveaux des procédures stockées qui sont utilisées dans la base de données. Ces procédures stockées sont les seuls traitements effectués coté serveur. Le poste client pour sa part doit exécuter tous les autres traitements (on ne se situe donc pas dans une architecture trois-tiers, comme Windows DNA par exemple).

La présentation des données à l'utilisateur est quant à elle effectuée également sur le poste client.

L'architecture d'accès aux données utilisée dans la BILLETTERIE est intéressante car pensée dans le but d'optimiser les performances du système.

Le modèle objet d'accès aux données (qui encapsule les couches de niveau inférieur comme illustré précédemment), permet une programmation plus facile. Cependant, plus le nombre de couches est important, plus les performances du système sont réduites. Il est donc nécessaire de trouver le bon compromis entre la simplicité d'accès aux données et les performances requises. Le choix de d'ODBCDirect depuis DAO constituait, à l'époque de la conception du progiciel, la meilleure alternative possible. Dans le détail, l'architecture d'accès aux données de la BILLETTERIE se présente ainsi :



V - MISE À JOUR DU PROGICIEL DE BILLETTERIE

1) Généralités

La mise à jour a consisté à faire évoluer de version le SGBDR qui héberge les données, en passant de la version 6.5 de SQL Server à SQL Server 7.0, à passer les sources (les lignes des programmes qui sont à l'origine des exécutables) de Visual Basic 5.0 à Visual Basic 6.0 et enfin à migrer de Crystal Reports 5.0 à 8.0 pour l'édition des états. La recompilation des programmes de Visual Basic 5.0 vers 6.0 ne présente pas d'intérêt particulier, donc ce travail n'est pas détaillé dans les lignes qui suivent.

Le manque de capitalisation des connaissances m'est apparu assez important en étudiant le progiciel. Garder les expériences passées pour en tirer profit par la suite aurait permis d'améliorer plus facilement la version existante. Cela suppose cependant une bonne traçabilité, à laquelle une volonté de réutilisation doit être associée. J'ai donc pris soin, pour ma part, de noter ce que je faisais et les résultats que j'obtenais.

2) Système de Gestion de Bases de Données

La migration en elle-même n'est pas un processus très compliqué si l'on prend soin de bien préparer son travail. Je vais donc m'attarder sur la démarche que j'ai suivie et non sur les détails ou problèmes techniques qui font par exemple que les fichiers de bases de données SQL Server 7.0, à cause des nouvelles fonctionnalités qu'il introduit, sont incompatibles avec les fichiers de bases de données de la version 6.5, rendant la migration des bases de données version 6.5 vers la version 7.0 nécessaire. La migration est facilitée par l'assistant de migration fourni avec SQL Server 7.0 (l'assistant se compose de 2 programmes distincts, upgrade.exe qui collecte les informations pour la mise à jour et crée un script upgrade.ini que le second exécutable scriptin.exe lit pour réaliser la migration effective).

L'idée directrice de ma démarche a été d'éviter les éventuels problèmes de migration avant de lancer celle-ci. De plus, je voulais pouvoir disposer d'une marche à suivre claire de ce que je devais faire pendant la migration, en ayant rapidement la réponse à un éventuel problème. J'ai donc divisé ma réflexion en quatre étapes :

- Une étude préalable consistant en la définition des grandes lignes à prendre en compte lors de la migration, me documentant abondamment sur le Web pour disposer des sources d'informations les plus récentes.
- Après avoir dresser le plan général de la migration, une étude plus détaillée encore a été réalisée pour spécifier de manière exhaustive les points cruciaux de la migration. De plus, tous les problèmes déjà rencontrés lors de migrations effectuées sur d'autres systèmes ont été listés : j'ai vérifié s'ils pouvaient s'appliquer à notre architecture. Parmi les huit documents réalisés lors de cette migration, je vous présente en **annexe 4** une partie de la marche à respecter pendant la migration.
- Après la migration, un résumé a été réalisé pour disposer d'un compte rendu détaillé.
- Enfin, la vérification minutieuse des fichiers journaux générés par l'assistant de migration a été faite. Des tests au niveau applicatif ont également été réalisés pour voir si les résultats étaient les mêmes entre des exécutables connectés à une base sous SQL Server 6.5 et les mêmes exécutables utilisant SQL Server 7.0. Des comparaisons des temps de réponse ont aussi été établies.

La documentation générée peut sembler superflue ou apparaître comme une perte de temps. Ce raisonnement est à revoir car le fait de disposer d'une trace claire de ce qui a été réalisé permet d'envisager des migrations d'autres systèmes avec plus de facilité et permet de vendre un service de migration à un client en se basant sur cette documentation. On voit donc bien ici aussi l'importance de la documentation en tant que capitalisation des connaissances dans l'entreprise.

3) Edition d'états

Pour cette mise à jour de la BILLETTERIE, l'idée porteuse a été de faire cohabiter l'existant avec les nouveautés à apporter à l'édifice, le tout en respectant la philosophie du système.

Pour les états, on utilise le logiciel Crystal Reports de Seagate Software. Cet outil permet de préparer des états à partir de données. On crée un modèle ayant une mise en page particulière, un modèle qui sera utilisé lors de l'impression avec les données de la base. En fait, on a le cycle suivant :

- On génère un état sous Crystal Reports. Cet état, d'extension .rpt, constitue une mise en page des données à imprimer.
- Sous Visual Basic, on demande l'impression de cet état en utilisant des outils de gestion d'impression fournis par Seagate Software.

Pour appeler depuis un programme client l'impression d'un état, plusieurs techniques sont disponibles, les antiques contrôles VBX utilisables en Visual Basic 3.0 et 4.0, le contrôle ActiveX Crystal Report, l'API Crystal Report de la DLL crpe32.dll, le modèle objet du serveur Automation cpeaut16.dll ou cpeaut32.dll et enfin le concepteur ActiveX Report Designer Composant qui s'intègre à l'interface de Visual Basic.

En ce qui concerne la BILLETTERIE, les fonctions API de la DLL crpe32.dll sont utilisées pour contrôler l'impression. Dans la terminologie Seagate, la Crystal Report Engine désigne cette API.

Pour vérifier que la documentation de Seagate était en accord avec la réalité et avant de lancer le changement de version, j'ai utilisé l'utilitaire de Visual Studio appelé Dependency Walker pour visualiser les fonctions disponibles dans cette API, leurs noms et surtout les dépendances éventuelles de la DLL avec d'autres composants.

VI - PARAMÉTRAGE DU PROGICIEL DE BILLETTERIE

1) Généralités

Comme tout progiciel, la BILLETTERIE nécessite un paramétrage lors de son installation, selon les besoins du client. Il impose également une modification de la manière dont le client travaille. Cette modification du processus métier est d'autant plus vrai qu'il est impossible de disposer d'un progiciel qui réponde à toutes les spécificités des professionnels d'un métier. En conséquence, le choix d'un progiciel est une décision stratégique pour l'entreprise, qui doit prendre soin de modéliser son métier (pour définir ses besoins) afin d'évaluer l'impact de l'introduction du progiciel dans son fonctionnement quotidien.

Cette phase est réalisée pour la BILLETTERIE par Michel CRIADO, qui a l'avantage de bien connaître le métier sur lequel le progiciel est positionné. Il peut ainsi évaluer au mieux l'impact des modifications que l'introduction du logiciel va induire.

Remarque : cette étape préliminaire a parfois pour résultat l'abandon pur et simple du projet, les clients se refusant quelquefois à repenser leur manière de travailler.

Le paramétrage dont il est question ici concerne les clubs de rugby de Bègles-Bordeaux, Dax et Pau, respectivement le CABBG (qui a choisi la BILLETTERIE CIMA en décembre 2000), l'U.S. Dax Rugby Landes (qui a mis en place la BILLETTERIE au début de la saison 2000/2001) et la Section Paloise (qui dispose du progiciel depuis un an déjà). J'ai donc pu participer aux différentes étapes de la mise en place, en collaboration avec Michel CRIADO.

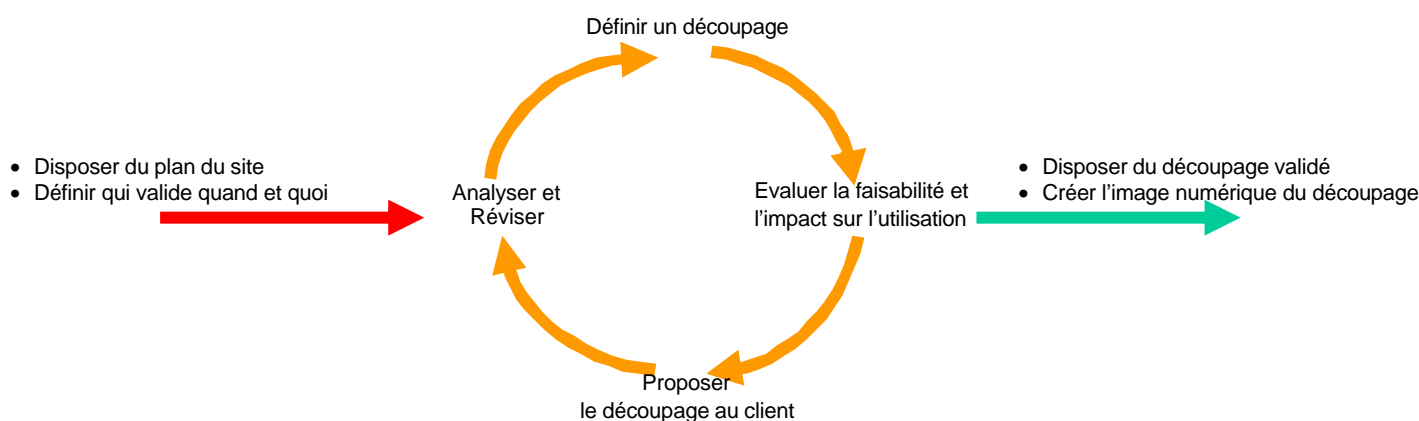
Plus précisément, il m'a été confié le paramétrage de la gestion des lieux. J'ai réalisé aussi l'intégration des abonnés existants dans le système. J'ai participé à la préparation des campagne d'abonnement en créant les cartes des abonnés et enfin j'ai assuré une partie de la gestion de la vie quotidienne des progiciels en exploitation.

2) Les lieux

Le point de départ se situe dans la configuration géographique du lieu sur lequel les spectacles auront lieu. Pour l'exemple, on se basera sur le stade Maurice BOYAU de l'U.S. Dax Rugby Landes. Un plan du stade, des tribunes en particulier, est l'idéal pour mener à bien cette étape.

A partir du plan des tribunes du stade, une discussion avec les responsables administratifs de l'U.S. Dax a eu lieu pour définir un découpage en modules (dans la terminologie de la gestion des lieux de la BILLETTERIE, un module est un regroupement de places d'une partie d'une tribune).

Une démarche itérative a été ici suivie :



Le découpage validé, on dispose d'un plan modulaire du stade avec des informations (nom, nombre de places, numérotation des places, noms des accès) sur chaque module. Il est intéressant de comparer le document qui a été présenté au client et la représentation du découpage en interne (le lecteur est invité à se reporter à l'**annexe 5**) : dans le premier cas, on présente un schéma compréhensible facilement tandis qu'en interne, la vue est tournée vers l'implémentation de la Gestion des lieux dans le progiciel (on utilise un MCD pour visualiser les relations entre les données d'une salle).

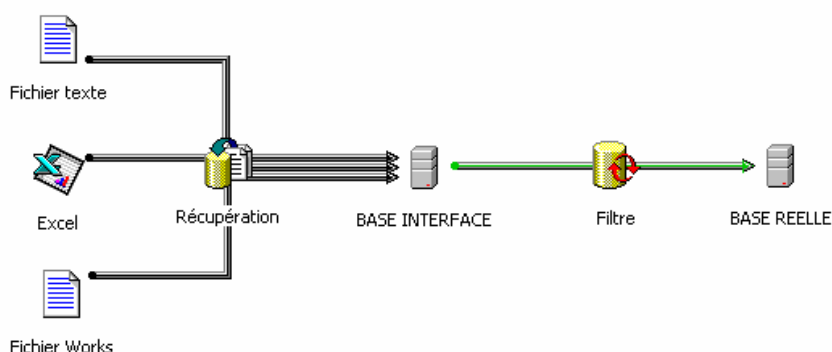
A partir de ces documents et après la réalisation d'une image numérique du stade, il est nécessaire d'implémenter les lieux dans le progiciel, en d'autres termes d'insérer les données dans la base de données. Cette tâche (réalisée via une interface graphique, la Gestion des sites en l'occurrence) doit être réalisée minutieusement car elle constitue une des pierres angulaires du système : si l'on crée des places qui n'existent pas dans la réalité, de graves problèmes risquent de se poser lorsque le titulaire du billet essaiera de s'y installer ...

3) Les abonnés

L'intégration de données dans le progiciel peut se faire par des programmes comme celui de la Gestion des sites utilisé lors de l'étape présentée ci-avant. Une autre voie est cependant à mettre en place pour la récupération de la liste des abonnés du client pour lequel le progiciel doit être paramétré. Cette méthode consiste à insérer dans la base de données sous SQL Server des données provenant de sources hétérogènes (fichiers texte, fichiers Excel, documents Works, ...).

Pour réaliser ce travail, j'ai imaginé une architecture simple et réutilisable, basée sur les outils fournis en standard par SQL Server.

Plus précisément, j'ai mis en place la structure suivante :



La partie spécifique a été réalisée par des lots DTS (Data Transformation Services). L'intérêt des services de transformation de données est qu'il est très rapide de mettre en place cette intégration des données de sources spécifiques vers la base intermédiaire. Ensuite, des scripts TSQL ont servi à copier les données de la base intermédiaire vers la base réelle. La difficulté est de s'assurer que toutes les données sont bien transférées. Pour ce faire, des traces (enregistrement du déroulement d'une action) sont implémentées, permettant ainsi de vérifier les problèmes de copie, les tuples rejetés ayant été insérés dans une table d'erreurs, permettant de comprendre et de relancer le traitement après la correction des problèmes.

4) Les abonnements

J'ai participé à la préparation des périodes d'abonnement de la Section Paloise et de l'U.S. Dax Rugby Landes. L'ensemble des actions effectuées pendant ces périodes comprend la création des abonnements que l'on veut proposer puis la vente des cartes d'abonnement aux personnes désirant supporter le club. A ce titre, j'ai participé à la conception puis à la réalisation des cartes d'abonnement des deux clubs.

La conception des cartes passe par une période de maquettes pour définir quelles informations doivent être inscrites sur chaque carte, à quels endroits les inscrire, quelles sont les couleurs à utiliser, ...

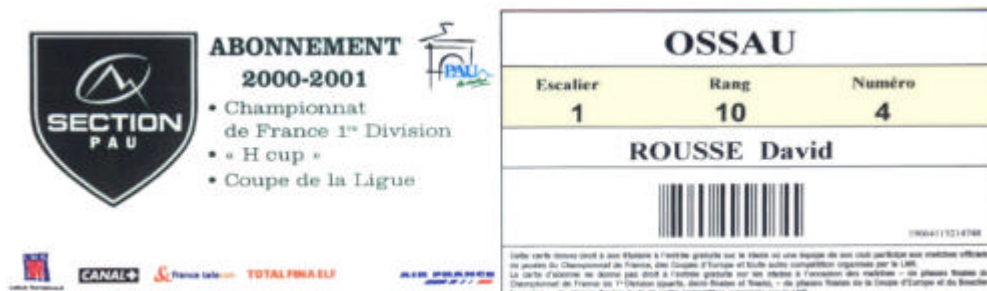
Après la validation de la maquette, la création des cartes commence sur le progiciel. Cette création passe par le logiciel de génération d'états Crystal Reports. S'ajoute la mise en place d'un code-barre qui va permettre de contrôler l'accès au stade lors des matchs.

L'utilisation de l'identification automatique par code-barre permet d'automatiser la saisie des informations, augmentant ainsi la vitesse, permettant un meilleur suivi, une plus grande précision, en un mot une gestion plus efficace.

Parmi les différentes techniques d'authentification automatique utilisées couramment (reconnaissance de caractères, étiquettes magnétiques, ...), la technique du code-barre s'est imposée dans le cadre de la BILLETTERIE. En effet, elle permet de traiter l'information rapidement, en liaison avec un équipement informatique.

On rappelle le principe du contrôle d'accès présenté précédemment : l'abonné présente sa carte à l'entrée du stade, un lecteur optique (lecteur ORBIT) lit le code-barre (en l'occurrence un code-barre au format 2 parmi 5 entrelacé) et envoie l'information lue vers un termod, le termod dialogue ensuite avec un programme fonctionnant sur un PC pour l'informer des données lues. Sur le PC, le programme qui s'éveille dès qu'un termod parle récupère l'information, enregistre les données et accorde ou refuse l'entrée à l'abonné, le tout en quelques secondes.

L'exemple de carte d'abonné présentée ci-dessous est celui utilisé par la Section Paloise pour la saison 2000/2001 :



VII - AJOUT D'UN MODULE AU PROGICIEL DE BILLETTERIE

1) Généralités

Le projet consiste en une application qui permette de réaliser la vente et la réservation de billets du progiciel de BILLETTERIE sur un ordinateur portable. Cet ajout au progiciel est une des améliorations les plus demandées par les clients afin de faciliter la mise en place du matériel les jours de match. Il apparaît donc que l'on doit mettre en place un système qui transfère les données du serveur hébergeant les données de la BILLETTERIE vers un PC portable, qui interdise l'utilisation sur le serveur des données transférées sur le portable et qui permette enfin le retour des données du portable vers le serveur.

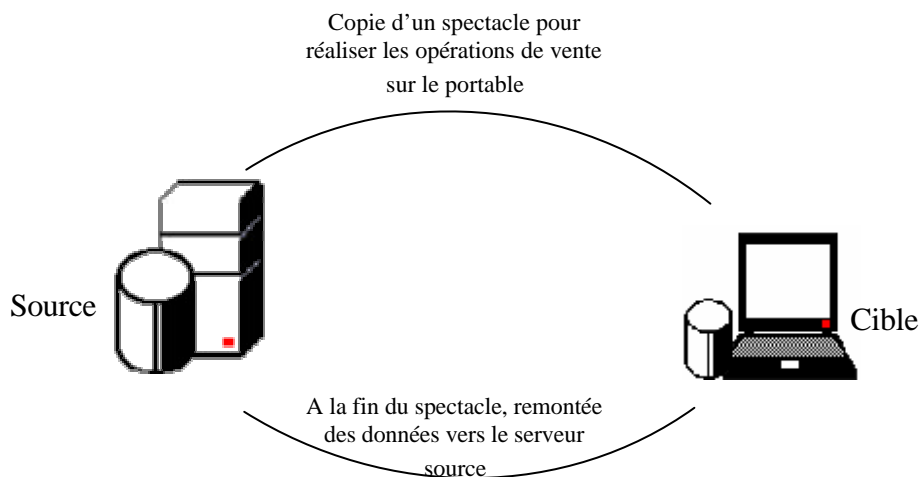
La pression économique impose à CIMA de produire du code aussi rapidement que possible et ensuite de le tester et de le déboguer ; en corrélation avec la démarche suivie par CIMA, cette stratégie pose d'évidents problèmes, notamment le risque de ne pas atteindre les objectifs initiaux (l'effet tunnel), le temps passé à déboguer, les tests qui ne sont pas toujours exhaustifs.

M'inspirant de la philosophie actuelle du développement logiciel, j'ai essayé de construire un module à base de composants. Lorsqu'un architecte doit construire un immeuble, il ne se met pas à fabriquer les moellons, les poutres ou les ardoises, il les achète et les assemble. C'est le même principe que j'ai essayé de suivre. Le problème, c'est que la société CIMA ne dispose pas d'un catalogue de composants préfabriqués. Je me suis donc retrouvé dans la position de l'architecte qui doit fabriquer ses moellons avant de commencer la construction.

En ce qui concerne la démarche suivie, elle est basée sur une séquence étude préalable, étude détaillée, étude technique, développement et tests, en d'autres termes les étapes préconisées en MERISE, le tout en utilisant un formalisme MERISE également. Cependant, pour présenter le travail réalisé, j'ai préféré synthétiser les idées principales du cycle de développement en me basant sur un niveau de granularité plus général que celui des différents documents générés lors de la réalisation du projet. Par conséquent, je présente seulement les grandes lignes du projet, en ce qui concerne l'analyse, la conception, le développement et enfin les tests.

2) Analyse

Suite à un cahier des charges réalisé par Michel CRIADO, une étude de l'existant a été menée pour savoir où le projet était à situer par rapport aux fonctionnalités actuellement présentes dans le progiciel de BILLETTERIE. Cette étape est nécessaire pour éviter de refaire quelque chose qui l'a déjà été et permet de définir les contours du projet.



L'analyse de la fonction de mise en historique des spectacles du progiciel a suivi : cette partie du progiciel présente en effet des spécifications proches de celles écrites dans le cahier des charges du projet.

Ensuite, j'ai continué l'étude préalable pour savoir quelles données sont concernées par le traitement, sans entrer dans le détail des enregistrements à prendre en compte. Un MCD a été réalisé, dans lequel les tables à traiter sont présentées. Dans le même ordre d'idée, les principales opérations à réaliser ont été identifiées :

- Identification des serveurs.
- Affichage des spectacles « transférables ».
- Comptage des données à transférer.
- Transfert de la source vers la cible (avec création en parallèle d'un fichier journal) puis blocage sur la source du spectacle transféré.
- Transfert de la cible vers la source (avec création en parallèle d'un fichier journal) puis déblocage sur la source du spectacle transféré.
- Traçage des tâches et gestion des erreurs (avec notamment le déblocage d'un spectacle mal transféré).

Le MCT de l'**annexe 6** montre l'enchaînement des opérations présentées sur la page précédente.

La phase suivante est celle de l'étude des scénarios de transfert entre les deux machines. Deux techniques différentes apparaissent : copie de données via des lignes T-SQL ou utilisation de la réplication.

La première solution, basée sur le modèle de la mise en historique d'un spectacle, consiste en l'utilisation exclusive de scripts T-SQL pour les transferts des données. Cette technique a le mérite d'être déjà utilisée dans la société et elle permet une relative indépendance par rapport à la version de SQL Server. Par contre, elle nécessite l'écriture de codes T-SQL volumineux voire délicats.

La deuxième solution propose l'utilisation de la réplication. SQL Server fournit un ensemble de services permettant de copier des données entre serveurs. Dans le projet de vente sur un ordinateur portable, deux techniques de réplication paraissent adaptées : d'une part la réplication de capture instantanée avec le serveur source jouant le rôle d'éditeur et de distributeur, la cible étant abonnée à la publication que l'on veut transférer, la réplication de fusion d'autre part avec la cible qui consolide ses données sur la source. Dans ce scénario, la fiabilité des services de réplication et le peu de code à écrire sont les avantages principaux. Par contre, la conception et l'implémentation sont plus compliquées.

On peut noter que les MOT des deux scénarios proposés sont assez proches et n'apportent des informations qu'en terme de répartition entre unités de traitement (c'est seulement lors du passage de chaque tâche au niveau physique que la différence sera notable).

3) Conception

Comme indiqué précédemment, j'ai fait en sorte de minimiser le travail spécifique à ce nouveau module pour éviter de faire un logiciel « jetable ». Pour favoriser la modularité, j'ai créé des composants non spécifiques et réutilisables. Pour arriver à découper le logiciel en parties distinctes, les MLD réalisés facilitent le travail. En l'occurrence, il se dégage les éléments suivants : l'interface visuelle permettant de réaliser le traitement, les paramètres de traitement, l'interface d'attente pendant le traitement et enfin le cœur du traitement, à savoir la copie et la remontée des données.

Le choix ergonomique d'un assistant a pour objectif de pouvoir réaliser les tâches le plus simplement possible. Pour ce faire, le choix des icônes et des couleurs a été pensé pour augmenter la simplicité d'utilisation. Les règles suivantes ont été mises en place :

- Afficher seulement l'information utile.
- Appliquer un style de présentation unique pour l'ensemble de l'assistant.
- Afficher la même information au même endroit.
- Guider l'utilisateur dans le processus d'utilisation.
- Permettre l'interruption et le retour à l'étape précédente.
- Afficher une aide adéquate.
- Afficher un écran d'attente si le traitement est long (plus de 5 secondes).

Après présentation de prototypes, une validation a donné lieu à un modèle d'écran utilisateur. La charte définie peut s'illustrer par l'écran de bienvenue du logiciel :



Les paramètres du traitement sont l'ensemble des informations qui sont dépendantes de l'environnement dans lequel l'exécution aura lieu. Par exemple, on peut vouloir paramétrer le nombre d'essais qui seront réalisés avant de déclarer l'échec du traitement. De même sont paramétrables les noms des bases, des serveurs, ... Cette notion de paramètres d'applications est cependant un concept récurrent, au niveau de tous les modules de la BILLETTERIE, mais également dans la plupart des applications de CIMA. Il est alors intéressant de disposer d'un composant générique qui facilite la manipulation de paramètres applicatifs. J'ai donc pris le parti de mettre en place un tel composant.

Pour réaliser sa spécification, j'ai utilisé Visual Modeler : l'objectif à ce stade est de définir l'interface du composant, en d'autres termes ce que les clients du composant verront de celui-ci et par extension ce qu'ils pourront en faire.

La spécification faite, il arrive le moment particulier de cette phase de conception, à savoir que le choix du type de composant est à réaliser alors que précisément, ce choix relève de l'implémentation. Cependant, dans une architecture objet, la conception est parfois dépendante de l'implémentation. En l'occurrence, il est nécessaire pour notre composant d'en choisir la nature. Le choix est assez simple ici puisque le composant ne doit pas s'exécuter de façon autonome (donc on ne choisit pas un EXE ActiveX) et la finalité du composant n'est pas d'enrichir l'interface graphique (on ne choisit pas un contrôle ActiveX). Le choix d'une DLL ActiveX est donc l'option retenue.

Arrive ensuite la partie de traitement automatique réalisé par le logiciel. A ce stade, la définition de l'architecture du projet, en termes de répartition des parties présentation, traitements et données, m'a poussé à séparer l'interface d'attente des traitements de copie et de remontée proprement dit. Cette séparation a également été motivée par le caractère commun d'une interface d'attente : lors de l'étude de la BILLETTERIE, cette disparité entre plusieurs exécutables qui implémentent chacun une interface d'attente différente constitue un argument supplémentaire en faveur de ce choix de conception.

Etant donné la finalité du composant, un contrôle ActiveX s'imposait, puisque le composant est bien un élément visuel capable de générer des événements déclenchés par l'utilisateur. Comme pour le développement de la DLL de gestion de paramètres applicatifs, la spécification du contrôle a été réalisée en premier lieu.

Pour les traitements, la phase de conception s'est terminée en spécifiant, à partir des MOT réalisés au préalable, les grandes lignes des traitements, avec des algorithmes succincts pour la copie de la source vers la cible, puis pour la remontée.

A ce stade, je disposais d'un cahier des charges de développement, en d'autres termes, de toutes les informations nécessaires pour lancer un développement sans s'y poser de questions fondamentales : les données à copier sont identifiées, les fonctions et procédures stockées qui vont manipuler les données sont listées, l'architecture de l'application est définie.

4) Développement

Lors du développement, j'ai commencé par l'implémentation de la DLL de gestion des paramètres et du contrôle ActiveX d'interface d'attente. Visual Basic simplifie la tâche et permet de créer les composants rapidement. L'**annexe 7** montre la représentation schématique réalisée avec VB Miner des deux composants.

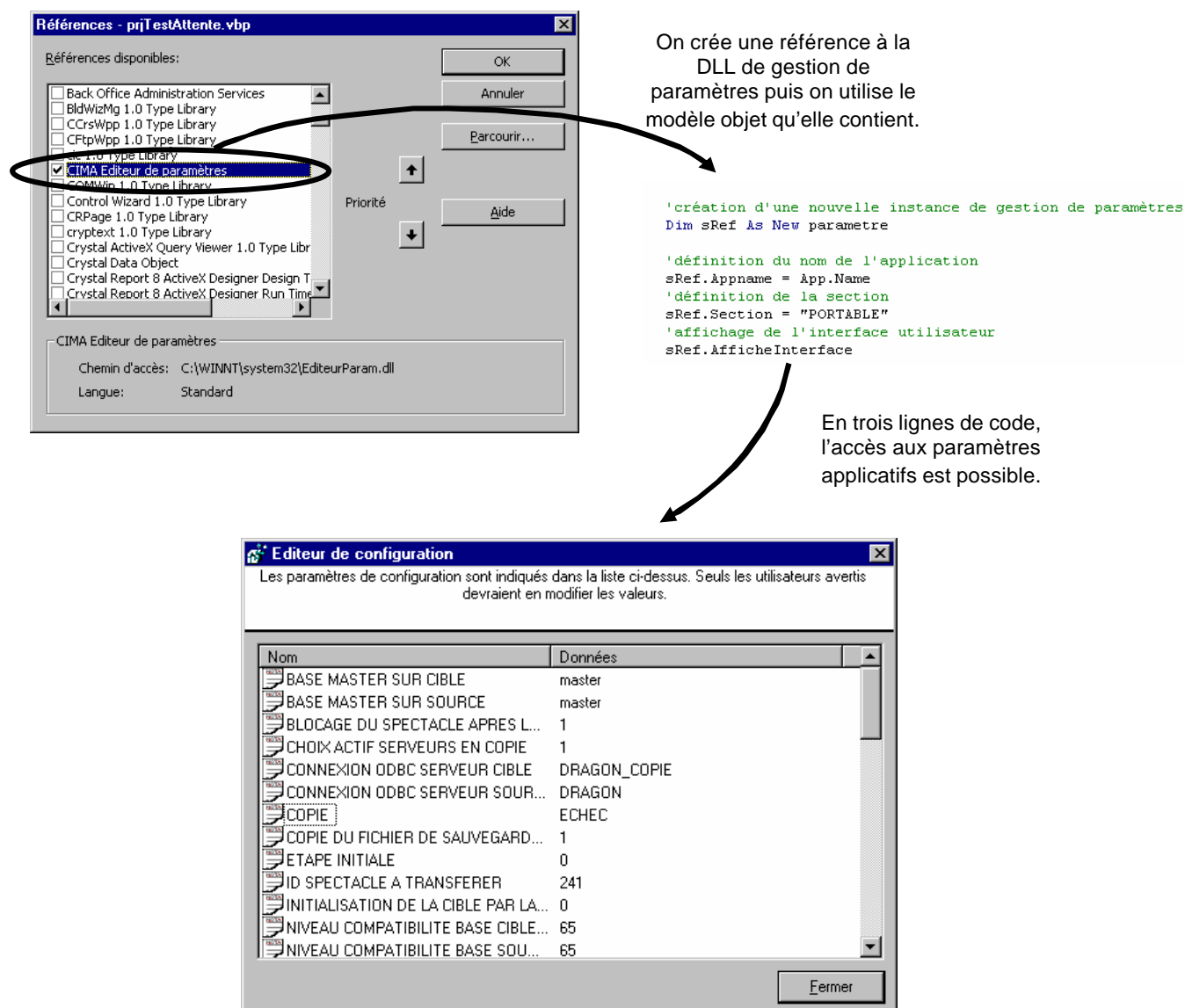
Je tiens également à souligner que j'ai pris soin de respecter l'intégrité de construction des programmes de la BILLETTERIE. Le style de programmation mis en œuvre est composé de formulaires et de modules exclusivement. J'ai donc préservé cette intégrité du système, et la création d'un contrôle ActiveX et d'une DLL ne remet pas en cause celle-ci. L'utilisation de ces deux composants est transparente et se fait au même titre que les composants standards de Visual Basic.

Ce maintien de l'intégrité de la philosophie de conception du système me paraît important pour éviter que chaque intervenant sur le projet ne mette en place ses propres idées, son propre style de programmation. Il faut savoir sacrifier certaines de ses convictions pour sauvegarder l'intégrité du logiciel, et éviter ainsi de concevoir un système hétérogène et finalement incohérent.

L'intérêt de la création de composants arrive lors du développement du reste du projet. Il suffit de choisir le contrôle dans la barre d'outils Visual Basic pour l'intégrer au projet : l'**annexe 8** donne un exemple de cette programmation via des « drag and drop ».

De même, une référence à la DLL de gestion de paramètres applicatifs est nécessaire pour utiliser ensuite les services de celle-ci.

Cette programmation visuelle est illustrée par le schéma suivant :



Le second point à mettre en lumière est le suivi du développement réalisé, qui, malgré son aspect contraignant, permet de valoriser les actions réalisées et de savoir ce qui est fait, comment, ... Par exemple, pour les traitement de copie et de remontée, j'ai mis en place un suivi permettant de savoir quelle fonction Visual Basic appelle quelle procédure stockée, et ce que la procédure stockée réalise sur quelles données.

5) Tests

« Les tests sont une façon efficace de démontrer la présence d'erreurs mais ils sont complètement inutiles pour démontrer leur absence. » (E. Dijkstra).

A ce titre, la phase de tests, malgré son caractère ingrat, constitue une part importante du projet et est indispensable pour fournir un produit fonctionnel.

Pour ma part, j'ai en réalité intégré cette étape au développement, dans le sens où les tests unitaires ont été réalisés avant la fin du développement. Plus précisément, j'ai adopté pendant le développement et les tests une démarche itérative, en partant du général pour affiner progressivement les programmes. Dans cette technique d'affinages successifs, on définit un algorithme de traitement, on le code puis on le teste. L'intégration des tests au développement est donc naturelle. De plus, ce type de développement incrémental me paraît intéressant car il permet de voir grandir progressivement le projet tout en diminuant la difficulté de débogage des modules.

Remarque : aucun automate de tests (logiciel qui capture les instructions générées par l'application dans des scripts puis qui les rejoue) n'était à ma disposition, tous les tests réalisés l'ont donc été manuellement.

La définition des procédures de tests passe par l'identification de ce que l'on veut tester, comment on veut le tester et quelles sont les parties à tester en priorité. De plus, j'ai essayé de mettre en place des tests paramétrables, en dissociant les données de tests de la procédure.

Lors du processus, les tests réalisés sont de plusieurs types :

- Des tests au niveau du code source, pour corriger les erreurs de syntaxe, les cas non gérés, les fuites de mémoire, les problèmes techniques.
- Des tests fonctionnels pour vérifier si l'application se comporte comme prévue. Par exemple, on vérifie que les résultats obtenus sur la machine source et sur la machine cible sont les mêmes pour un même traitement.
- Des tests d'intégration, pour vérifier que les composants du projet s'assemblent correctement (l'intégration de la DLL au reste du projet par exemple).
- Des tests de performance ont également été réalisés pour vérifier si l'exécution dans un environnement de production était conforme aux besoins exprimés dans le cahier des charges.

J'ai pu apprécier pendant cette étape de tests l'apport d'un langage de haut niveau tel que Visual Basic. En effet, la langage minimise les bogues de programmation et facilite les corrections grâce au débogage interactif. Par ailleurs, l'utilisation des fichiers journaux mis en place dans l'application m'a servi à vérifier le bon fonctionnement des programmes.

En résumé de ce projet de vente sur un ordinateur portable, j'ai pu vérifier la véracité de la maxime suivante : quand on veut créer une voiture, on l'imagine, on la dessine puis on la construit ; quand on bâtit un immeuble, on en fait les plans auparavant.

L'informatique n'échappe pas à ce processus : une méthode de travail est nécessaire pour faire d'un projet une réussite.

VIII - CONCLUSION

Les six mois passés au sein de la société CIMA m'ont ouvert les yeux sur la difficulté de mise en œuvre du processus d'ingénierie logicielle dans un contexte économique agressif. Face à une pression financière de tous les instants, la conception de logiciels se transforme en une course effrénée pour respecter les délais et satisfaire les clients.

Le réflexe de travailler plus ne résout que ponctuellement le problème. La difficulté est alors de trouver le moyen d'améliorer le processus d'ingénierie.

La réutilisation binaire de composants préfabriqués me semble une réponse adéquate à cette problématique qui se pose dans les sociétés de services de petites tailles. En se basant sur une méthode de travail adaptée, en choisissant une architecture logicielle qui facilite la mise en œuvre et en utilisant un langage de haut niveau simple et performant, des bases solides pour une évolution naturelle de la société sont posées.

Personnellement, l'analyse, le paramétrage, l'utilisation quotidienne du progiciel de BILLETTERIE m'a permis de faire un parallèle entre la vision informatique et le métier qu'elle sert. Travailler sur un progiciel sectoriel revient à apprendre le métier pour lequel le progiciel est conçu mais requiert aussi la maîtrise des technologies informatiques qui ont servi à la création du progiciel.

L'ingénierie du logiciel a donc le mérite de mettre en place constamment de nouveaux terrains de jeux, sur lesquels la faculté d'adaptation aux nouvelles pensées, aux nouvelles technologies est essentielle. Il faut savoir cultiver l'espace conquis pour survivre et grandir.

Dans l'avenir, j'espère pouvoir suivre l'évolution actuelle du monde informatique et explorer en détail la modélisation objet et son application aux systèmes de composants distribués.

IX - DOCUMENTATION

1) Livres

La documentation sur l'environnement de travail :

- Au cœur de COM+, G. et H. Heddon, Microsoft Press

La documentation sur les outils de développement :

- Atelier Visual Basic 6.0, John Clark Craig et Jeff Webb, Microsoft Press
- Les 1000 fonctions de programmation de Windows 95, Bernard Frala, Marabout

La documentation sur SQL Server :

- Microsoft SQL Server 7.0, Administration système, Microsoft Corporation, Microsoft Press
- Microsoft SQL Server 7.0, Mise en œuvre de bases de données, Microsoft Corporation, Microsoft Press
- SQL Server 7.0, Stephen Wynkoop, Le Macmillan Campus Press

2) L'Internet

La documentation sur les logiciels utilisés :

- Site officiel de Microsoft, www.microsoft.com
- Site de MSDN, www.microsoft.com/france/msdn/
- Site de Technet, www.microsoft.com/france/technet/

La documentation sur le modèle objet COM+ :

- Site dédié à COM+, www.microsoft.com/com/tech/COMPlus.asp
- Spécifications de COM+, www.microsoft.com/Com/resources/comdocs.asp

Les sources de documentations diverses :

- IEEE Software, www.computer.org
- Forum sur Visual Basic, microsoft.public.fr.vb
- Forum sur les codes-barre, comp.fonts
- Forum sur les tests logiciels, comp.software.testing

X - GLOSSAIRE

Un glossaire concernant les termes relatifs à l'environnement COM+ est également disponible dans l'**annexe 1**.

2

2 parmi 5 entrelacé
Ce terme désigne un format de code-barre dans lequel 2 traits sont toujours larges sur un total de cinq. De plus, les caractères, toujours numériques (0 à 9) sont codés en utilisant les « barres noires » et les « espaces blancs » entre celles-ci, d'où le qualificatif entrelacé. La conséquence est qu'un code-barre au format 2 parmi 5 entrelacé code toujours un nombre de chiffres pairs.

A

abstraction
L'abstraction est le fait de se concentrer sur l'essentiel et d'oublier les détails, en d'autres termes, c'est le fait de mettre en évidence les caractéristiques fondamentales d'une entité au sens premier du terme.

AGL
Un **A**telier de **G**énie **L**ogiciel est un programme ou un ensemble de programmes permettant de concevoir des logiciels.

API
Cet acronyme d'**A**pplication **P**rogramming **I**nterface désigne un ensemble de fonctions de bas niveau (proches du système) facilitant la création d'applications de plus haut niveau (les programmes utilisés par les utilisateurs finaux). Sous Microsoft Windows, les DLL (**D**ynamic **L**ink **L**ibrary) contiennent typiquement des API.

B**Boasson Maarten**

Maarten Boasson est un consultant de l'organisation IEEE (Institute of Electrical and Electronics Engineers) qui fournit une source d'informations techniques aux professionnels de l'informatique du monde entier (notamment par la parution du journal IEEE Software).

C**classe**

Une classe (un module de classe) en Visual Basic constitue l'équivalent d'une classe C++ dans laquelle on aurait regroupé la spécification et l'implémentation. Un module de classe constitue le modèle, le moule à partir duquel les instances de la classe (les objets) sont créées. Avec Visual Basic 7.0, on disposera également des membres de classe, c'est à dire que l'on pourra manipuler des données ou des méthodes communes à toutes les instances d'une classe.

client / serveur

Modèle d'architecture applicative, mode de fonctionnement dans lequel un poste client demande des services distants à un serveur par le biais de requêtes et de réponses associées. Le serveur met à disposition des services (accès à l'Internet, messagerie, ...) et des ressources (périphériques, applications, ...) que le poste client utilise.

code natif

Le code natif, par opposition à du code interprété, est directement exécuté par le processeur. Il ne faut cependant pas confondre le code natif et un exécutable compilé en liaison statique. L'exécutable créé en mode de liaison statique contient tout ce dont il a besoin pour fonctionner seul. A l'inverse, les exécutables créés par le compilateur de Visual Basic 6.0 en code natif ont besoin de la DLL msvbvm60.dll pour fonctionner. Cette DLL d'exécution lie dynamiquement à l'exécution le code natif et les composants de base (feuilles, classes, ...) nécessaires à l'application.

collection

Une collection permet de regrouper des objets dans un ensemble hiérarchique auquel on peut faire facilement référence. Par analogie avec le C++, une collection peut être présentée comme un « tableau » contenant l'ensemble des instances d'une classe, « tableau » que l'on peut manipuler aisément avec des méthodes standards telles que Add, Remove ou Get.

courbe du soleil

Ce terme désigne en MERISE le déroulement du processus de conception d'un logiciel à travers les niveaux du cycle d'abstraction.

compilation

Le processus de compilation au sens Visual Basic du terme regroupe en réalité les étapes de compilation et d'édition des liens (quoique l'exécutable final ne soit pas utilisable directement par le processeur). Typiquement, lors de la création d'un programme, on écrit le code source du programme (les lignes en langage de programmation que les développeurs d'applications doivent sans relâche produire), le code source de chaque module est ensuite compilé en un fichier objet (le .obj) puis arrive l'édition des liens (la mise en commun des différents modules compilés et l'intégration des éléments annexes utilisés par ces modules) pour créer le programme exécutable final.

constructeur

Le constructeur est une méthode particulière qui permet de créer une instance d'une classe.

cycle de développement

Le cycle de développement constitue l'ensemble des étapes nécessaires à l'élaboration d'un logiciel, ce qui en MERISE se traduit par la séquence Schéma directeur, Etude préalable, Etude détaillée, Développement, Tests, Recette, Déploiement, Maintenance. Pour chaque occurrence du cycle de développement, les trois cycles de MERISE, le cycle de décision (décrit les points de validation qui permettront de gérer l'enchaînement des étapes d'un projet comme par exemple la planification), le cycle de vie et le cycle d'abstraction (processus visant à se concentrer sur l'essentiel, en distinguant le niveau physique, le niveau organisationnel et le niveau conceptuel) sont parcourus.

cycle de vie

Le cycle de vie constitue la séquence de vie d'un logiciel, basée sur une évolution naturelle, à savoir la gestation, la naissance, la maturité et la mort.

D

Dijkstra
 Dijkstra Edsger est un mathématicien ayant eu un rôle essentiel dans le développement du langage ALGOL à la fin des années 1950, et ayant ensuite développé « la science et l'art des langages de programmation en général, contribuant grandement à notre compréhension de leur structure, de leur représentation et de leur implémentation » selon l'ACM (Association for Computer Machinery).

distribution
 Une seule vision du système pour l'utilisateur alors qu'en réalité, ce dernier est situé à plusieurs endroits.

E

éditeurs WYSIWYG
 Un éditeur WYSIWYG (What You See Is What You Get, ce qui en français peut se traduire par « Ce que vous voyez à l'écran, c'est ce que vous obtiendrez ») permet de faciliter l'utilisation de logiciels. Dans le développement Web par exemple, on dispose d'éditeurs WYSIWYG qui permettent créer des pages au format HTML sans en connaître la syntaxe et de visualiser directement les pages telles qu'elles apparaîtront dans la fenêtre du navigateur de l'internaute.

effet tunnel
 Ce terme est la caractéristique de développements réalisés sans point de validation ce qui a pour résultat de ne pas atteindre les objectifs, de « rater la cible initiale », à cause d'un manque de communication (le confinement dans le tunnel).

F

fichiers de bases de données
 Ce terme désigne l'ensemble des fichiers dans lesquels les données d'une base particulière sont stockées. Sous SQL Server 7.0, on distingue pour une même base de données le fichier de données primaire (.mdf), le fichier journal (.ldf) et les fichiers secondaires (.ndf). Ceci constitue une amélioration par rapport aux fichiers d'unités de la version 6.5 (.dat) qui permettaient de stocker plusieurs bases dans un même fichier.

fichiers journaux
 Un fichier journal (encore appelé fichier log) contient l'enregistrement de l'activité d'un système, au même titre que le journal de bord d'un commandant de bateau. Il contient des informations diverses, comme par exemple les erreurs de connexion, les erreurs d'exécution, ...

G

Gartner Group
 Société d'analystes, réputée dans le milieu informatique pour ses études et ses audits.

générateur d'états
 Logiciel offrant un ensemble d'outils permettant de générer des états (c'est à dire une représentation visuelle, sous forme papier par exemple) de données provenant de sources diverses.

gestionnaire de code source
 Utilitaire de contrôle de code source permettant de sauvegarder des fichiers source, de partager ces derniers entre plusieurs utilisateurs. Il facilite la gestion des projets et le travail en équipe.

H

héritage
 L'héritage consiste à spécialiser une classe existante par ajout de nouvelles caractéristiques (propriétés et méthodes) dans les sous-classes.

I

ingénierie
L'ingénierie est l'étude globale d'un projet sur tous ses aspects, ce qui comprend en réalité plusieurs sous-études particulières. Dans le milieu du génie logiciel, ce terme désigne les processus permettant de concevoir proprement des logiciels.

ISAM.....
ISAM (Index Sequential Access Methods) est une méthode d'accès à des données stockées dans des fichiers, méthode basée sur des index.

M

mainframe.....
Un mainframe (littéralement la cadre principal sur lequel d'autres éléments sont montés) est un gros ordinateur entouré de terminaux ayant peu ou pas de capacité de traitements (terminaux passifs).

MERISE
MERISE est une méthode qui fournit une démarche (Etude préalable, Etude détaillée, ...) et des outils (MCD, ...) permettant d'atteindre un but fixé (la stricte satisfaction du besoin validé dans le respect des coûts et des délais).

modularité
Se définit par la division d'un système en sous-systèmes relativement indépendants les uns des autres.

P

polymorphisme
Le polymorphisme permet de donner une interface commune à plusieurs objets. A l'inverse du C++ qui utilise l'héritage et les fonctions virtuelles pour mettre en œuvre le polymorphisme, Visual Basic 6.0 aborde le sujet en passant par la notion de classe abstraite (classe ne contenant que des spécifications d'interfaces) que l'on implémente dans les modules de classe voulant signer le contrat de l'interface. Avec Visual Basic 7.0, on pourra utiliser le polymorphisme avec l'héritage et travailler véritablement en objet.

progiciel.....
Contraction de Produit et Logiciel, un progiciel est un logiciel spécifique à un métier ou à un client particulier. On parle parfois de logiciel sur étagère pour désigner un progiciel c'est à dire un logiciel personnalisable.

programmation événementielle
Dans une application événementielle, le chemin qui sera suivi dans le code de l'application n'est pas figé. Il est fonction des événements reçus, eux même étant déclenchés par une source externe, un clic de souris sur un bouton par exemple. Par opposition, une programmation procédurale définit un ordre d'exécution stricte.

R

redéfinition
La redéfinition consiste à créer une nouvelle implémentation d'une méthode d'une sur-classe dans une sous-classe.

réplication
La réplication consiste à copier des données d'une base source vers une base cible, la base cible se trouvant généralement sur un serveur distant. La réplication diffère des transactions distribuées dans le sens où les copies des données entre les serveurs ne sont pas forcément identiques à tout instant, alors qu'avec les transactions distribuées, la cohérence transactionnelle complète et immédiate est assurée. On distingue plusieurs types de réplication sous SQL Server, la réplication de capture instantanée (simple copie de données à intervalles réguliers), la réplication de fusion (les données répliquées sur les cibles peuvent être modifiées puis consolidées sous certaines conditions sur la source) et la réplication transactionnelle (les mises à jour de la base source sont répercutées quasi-instantanément sur les bases cibles).

reverse engineering
Le reverse engineering (ingénierie inverse en français) consiste à comprendre ce qui existe pour en reconstituer un modèle montrant les principales caractéristiques du système.

réutilisabilité.....
La réutilisabilité (la réutilisation devrait-on dire) consiste à réutiliser une entité (un composant par exemple) dans des contextes différents.

S

SQL-DMO
 Modèle objet reposant sur COM, à partir duquel tous les outils graphiques livrés avec SQL Server sont construits. Avec SQL NameSpace (objets encapsulant l'interface utilisateur d'Entreprise Manager) et DTS (services de transformations de données), il constitue l'une des trois API permettant l'administration de SQL Server.

SSII
 Société de Services et d'Ingénierie Informatique.

surcharge
 La surcharge consiste à implémenter plusieurs comportements pour un même nom.

T

Transact-SQL
 Transact-SQL constitue l'implémentation mise en œuvre par Microsoft de la norme ANSI SQL-92. T-SQL, l'équivalent pour SQL Server de PL/SQL pour Oracle, possède comme son rival des éléments de langage supplémentaires qui lui sont propres.

tuple
 Synonyme d'enregistrement, le terme désigne une ligne d'une table d'une base de données.

W

Windows DNA
 Windows DNA (Distributed interNet Applications Architecture) est une plate-forme qui implémente la division entre la présentation, les traitements et les données pour améliorer la sécurité, la fiabilité et l'évolutivité des applications de commerce distribuées et ouvertes sur l'Internet. Cette plate-forme permet de se concentrer sur les problèmes métier et non sur les problèmes d'infrastructure.

Pour toutes informations complémentaires,
les coordonnées des parties prenantes du stage sont présentées ci-dessous.

- **David ROUSSE, rédacteur**

Adresse : 30, rue Ronsard,
Rés. TAO, Bât. A, Appt. 14
64000 PAU

Téléphone : 05.59.80.30.23

E-mail : d.rousse@wanadoo.fr

- **Michel CRIADO, maître de stage**

Adresse : CIMA
Centre Activa,
allée Catherine de Bourbon,
64000 PAU

Téléphone : 05.59.30.97.98

E-mail : michel.criado@cima.fr

- **Patrick DEMEURISSE, tuteur de stage**

Adresse : TMIC,
Parc Technologique du Canal,
26, rue Hermès,
31520 RAMONVILLE

Téléphone : 05.61.75.85.23

E-mail : ellipses.tmic@wanadoo.fr

- **IUP MIAGe, Toulouse I**

Adresse : Université des Sciences Sociales,
Manufacture des Tabacs
Département I.U.P.,
21, allée de Brienne,
31042 TOULOUSE

Téléphone : 05.61.12.86.53

E-mail : iupmiage@univ-tlse1.fr

- **IUP MIAGe, Toulouse III**

Adresse : Université Paul Sabatier
Département Inter-UFR,
118, route de Narbonne,
31062 TOULOUSE Cedex 04

Téléphone : 05.61.55.67.68

E-mail : depereti@cict.fr