

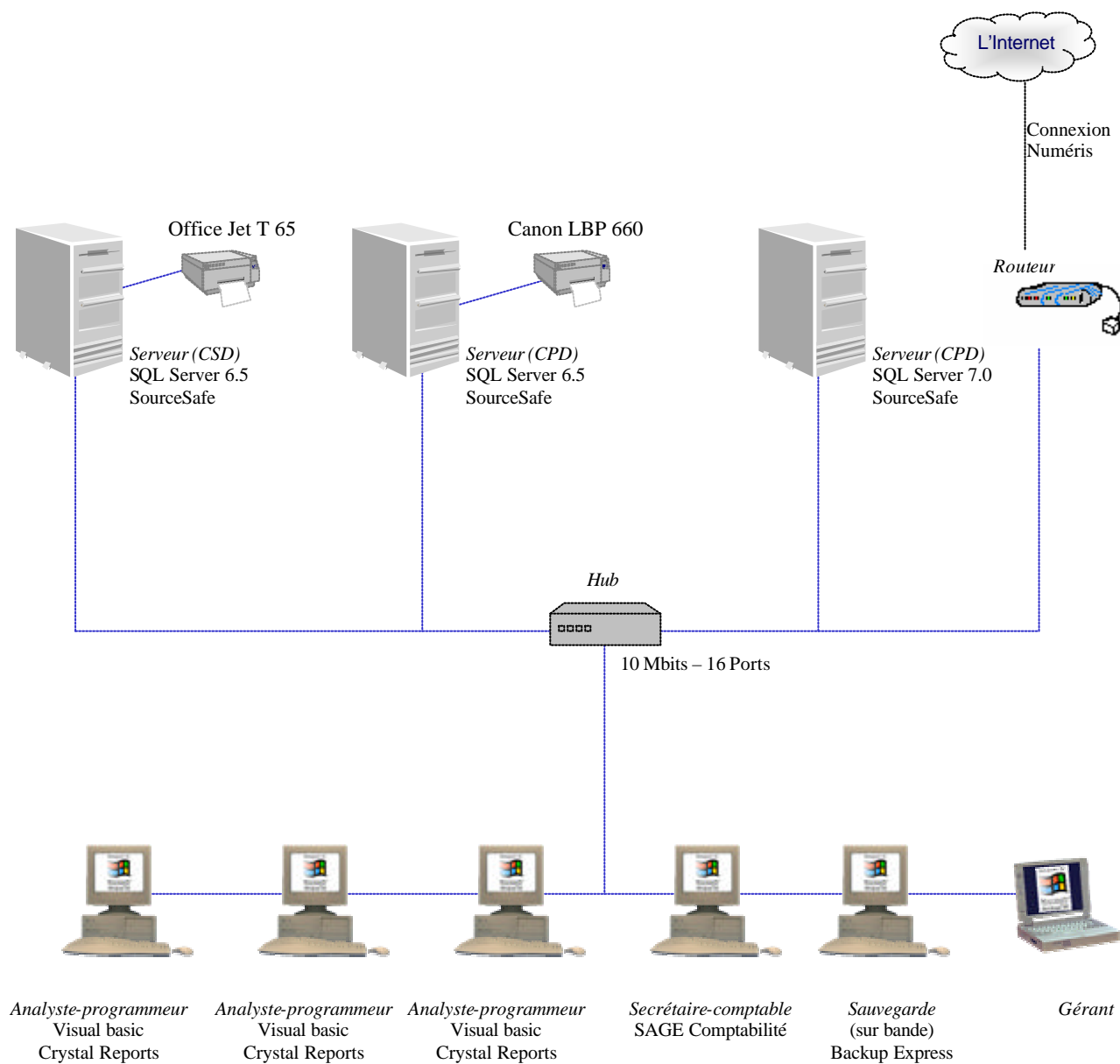
Edité à PAU le lundi 22 décembre 2003.

ANNEXES - SOMMAIRE

<i>I - Annexe 1, RÉSEAU CIMA</i>	<i>3</i>
<i>II - Annexe 2, COM+</i>	<i>4</i>
<i>III - Annexe 3, CONFIGURATION MATÉRIELLE DE LA BILLETTERIE</i>	<i>4</i>
<i>IV - Annexe 4, MIGRATION DE SQL SERVER 6.5 À 7.0</i>	<i>4</i>
<i>V - Annexe 5, DIFFÉRENTES VUES POUR UNE MÊME IDÉE</i>	<i>4</i>
<i>VI - Annexe 6, MCT VENTE SUR PORTABLE</i>	<i>4</i>
<i>VII - Annexe 7, SCHÉMAS VB Miner</i>	<i>4</i>
<i>VIII - Annexe 8, CONTRÔLE ActiveX POUR LE PROJET DE VENTE SUR PORTABLE</i>	<i>4</i>
<i>IX - Annexe 9, SUIVI QUOTIDIEN DU STAGE</i>	<i>4</i>

I - ANNEXE 1, RÉSEAU CIMA

Le réseau informatique dans lequel mon stage s'est déroulé se présente ainsi :



II - ANNEXE 2, COM+

L'annexe 2 présente une note informative écrite pendant ma période de formation sur l'environnement COM+. Je présente le document tel qu'il a été rédigé, avec sa mise en page, sa présentation originelle, ... Je tiens également à signaler qu'un glossaire des termes liés à COM+ est disponible à la fin de cette annexe 2.

Aperçu de COM+**Dresser un rapide tour d’horizon de l’environnement COM+****Aperçu de COM+**

REF. :

VERSION : 1.1

DATE : 04/08/00

	SOCIETE	NOM
REDACTEUR	CIMA	David ROUSSE
INTERLOCUTEUR		
PARTICIPANTS		

REVISIONS

VERSION	DATE	OBJET
1.1	31/08/2000	Ajout de la partie relative à Visual Basic

COM+ - SOMMAIRE

<i>I – INTRODUCTION</i>	4
<i>II - COM+</i>	4
1. Historique	4
2. Définition.....	4
3. Principes fondamentaux	4
4. Services fondamentaux	4
5. Technologies basées sur COM+.....	4
<i>III - LES COMPOSANTS</i>	4
1. Contrôle ActiveX.....	4
2. DLL ActiveX.....	4
3. EXE ActiveX.....	4
4. Document ActiveX.....	4
5. Choix entre un EXE et une DLL	4
<i>IV - LES INTERFACES</i>	4
1. IDL.....	4
2. L'interface IUnknown	4
3. L'interface IClassFactory	4
4. L'interface IDispatch.....	4
5. L'interface IMoniker	4
<i>V – LA COMMUNICATION ENTRE COMPOSANTS</i>	4
1. Proxy et stub.....	4
2. Bibliothèque de types.....	4
3. Points de connexions	4
<i>VI - COM+ ET VISUAL BASIC</i>	4
1. Utilisation de composants COM+ en VB	4
2. Création de composants COM+ en VB	4
3. Communication entre composants en VB.....	4
4. Détail d'un appel Automation en VB	4
<i>VII - CONCLUSION</i>	4
<i>VIII – GLOSSAIRE</i>	4
<i>IX - BIBLIOGRAPHIE</i>	4

I – INTRODUCTION

L'informatique actuelle doit être à l'écoute des utilisateurs et se doit de répondre le plus rapidement possible à leurs besoins.

En conséquence, les applications monolithiques du passé sont progressivement remplacées par des applications distribuées à bases de composants. Une application à base de composants se compose de briques logicielles réutilisables pouvant interagir entre elles.

Ces assemblages de composants logiciels réutilisables et prêts à l'emploi permettent de décomposer les applications stratégiques en sous ensembles de processus coopérant les uns avec les autres : réactivité des architectures informatiques, maîtrise des coûts, évolutivité dans le temps sont alors améliorées.

Pour mettre en place des applications à base de composants, une architecture logicielle est nécessaire : elle définit des spécifications fondamentales que doivent respecter les composants de l'architecture et fournit un ensemble de services. Deux grands modèles d'architectures logicielles sont sur le devant de la scène : CORBA de l'OMG et COM+ de Microsoft, entre lesquelles EJB de Sun se présente comme un outsider. Le présent document va décrire sommairement l'architecture logicielle COM+.

II - COM+

1. Historique

A l'origine, un besoin d'intégration de documents Microsoft Graph dans Microsoft PowerPoint a donné naissance à une technique pour l'incorporation d'objets entre différentes applications. Le principe fondateur d'**OLE** (Object Linking and Embedding) était né. Ce mécanisme de liaison et d'incorporation d'objets, d'abord enrichie à Microsoft Office, a progressivement évolué vers un terme générique regroupant des services parfois indépendants, **COM+** (Component Object Model). Sous cette technologie, Microsoft regroupe un ensemble de services dont ceux fournis par COM, DCOM et MTS.

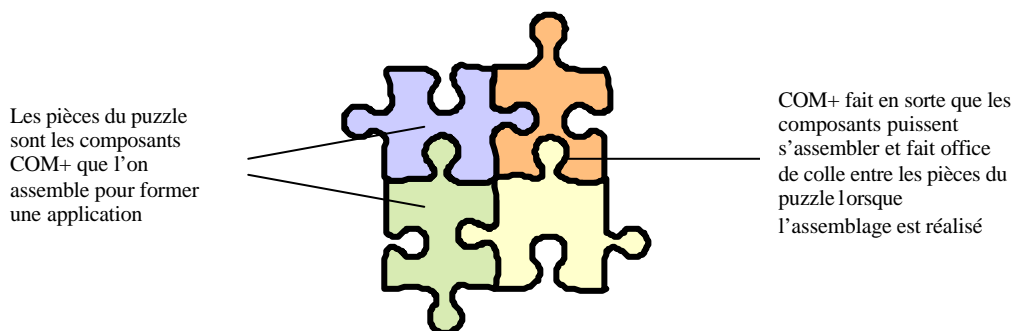
2. Définition

Le modèle COM+ présente deux aspects distincts. D'une part, la spécification du COM fournit une définition (ou modèle) de ce qu'est un objet. En d'autres termes, COM décrit un **format binaire** que doivent respecter les composants. D'autre part, le modèle COM offre un certain nombre de **services** qui facilitent la création d'objets et les communications entre les objets.

Il est possible de faire une analogie avec les fabricants d'automobiles. Un fabricant d'automobiles conçoit l'architecture générale de ses voitures, achètent des pièces détachées chez des fournisseurs spécialisés puis les assemblent pour créer une voiture. La voiture est l'application finale, les pièces détachées constituent les composants logiciels prêts à l'emploi et le fabricant joue le rôle de COM+ en « collant » les composants entre eux.

3. Principes fondamentaux

Le logo de la suite Microsoft Office donne une idée de la philosophie de COM+.



Pour que les pièces du puzzle (les composants) puissent s'assembler, se désassembler et être remplaçables, COM+ édicte quelques principes fondamentaux.

A la base, COM définit le format binaire des objets (attributs et méthodes) tels qu'ils sont représentés physiquement en mémoire. En effet, COM assure une indépendance complète vis à vis du langage de programmation utilisé. Que l'on utilise C, C++, ADA, Visual Basic, ..., tant que le compilateur produit des objets dans le format défini par COM, le standard de compatibilité binaire est assuré. C'est cette indépendance qui est à la base de l'interopérabilité des objets COM.

Les principes suivants sont également à souligner.

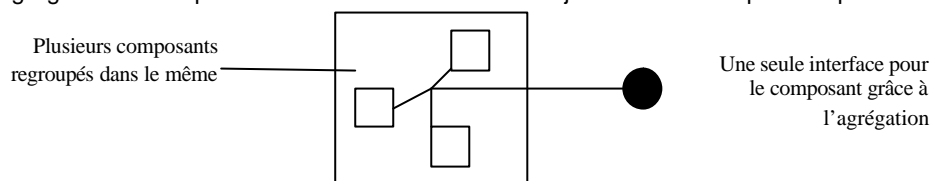
D'une part, il faut que les composants puissent se lier entre eux dynamiquement, c'est à dire au moment de l'exécution.

D'autre part, l'encapsulation doit être réalisée. En d'autres termes, on isole l'interface de l'implémentation. Cette séparation passe par les règles de base suivantes :

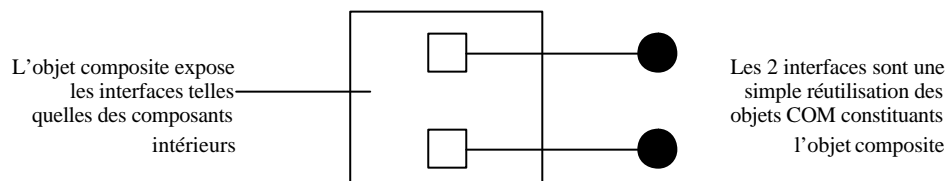
- le langage utilisé pour le codage doit être caché et ne pas influencer sur l'interface
- le composant est livré sous forme binaire (et non pas sous forme de code source). Cela permet une réutilisabilité binaire (entre composants, et non une réutilisabilité au niveau du code source)
- un changement de version du composant ne doit pas remettre en cause les interfaces existantes
- la place du composant dans un réseau importe peu

Enfin, l'enrichissement d'interface ne passe par un héritage classique comme dans CORBA mais par l'agrégation et la délégation.

- l'agrégation simule pour le monde extérieur un seul objet en réalité composé de plusieurs :

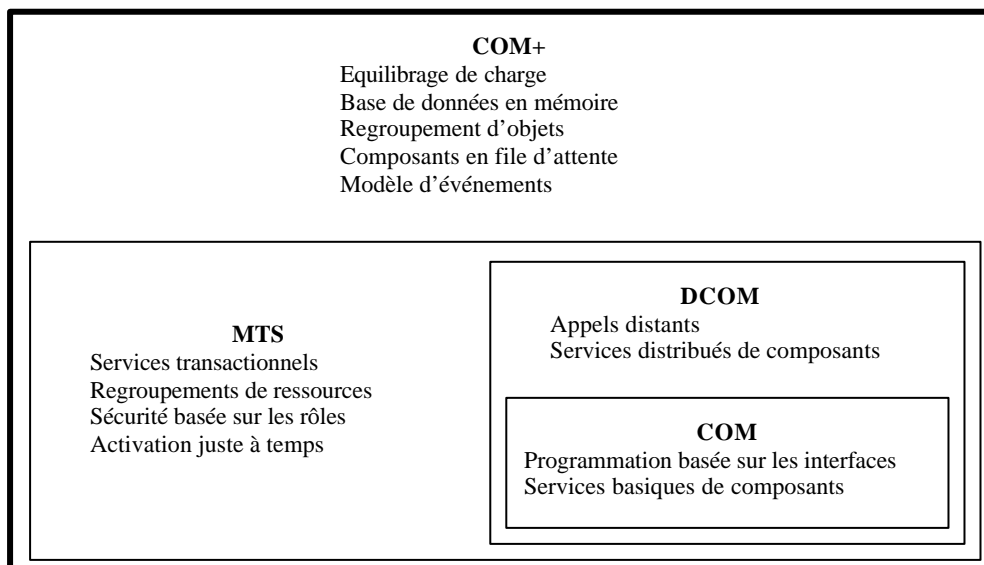


- la délégation consiste en la réutilisation d'une interface d'un objet par un autre objet :



4. Services fondamentaux

COM+ impose aux composants de respecter certaines contraintes. En contrepartie, les programmeurs de composants disposent de services puissants, schématisés ci-dessous :



Les services suivants méritent d'être soulignés :

- activation juste à temps (just-in-time activation): l'instance en mémoire de l'objet est créée lors de l'utilisation et non lors de la demande d'interface. Cela évite une surcharge de l'environnement de travail.

- regroupement des objets (object pooling): COM+ recycle les objets au lieu de les détruire systématiquement quand ils ne sont plus utilisés. Cela accélère un nouvel appel à l'objet.
- équilibrage de charge : lorsqu'un client demande l'exécution d'une application distribuée, le routeur d'équilibrage de charge fourni par COM+ répartit la charge entre les grappes de machines sur lesquelles l'application distribuée est située.
- base de données en mémoire ou IMDB (In Memory Data Base) : cache temporaire qui permet d'améliorer les performances des applications distribuées utilisant des données.
- file d'attente : basée sur MSMQ (Microsoft Message Queue Server), elle permet à un client qui appelle les méthodes d'un composant momentanément inaccessible d'attendre la libération de ce dernier pour renvoyer ensuite la réponse à la demande du client.
- transaction : MS DTC (Microsoft Distributed Transaction Coordinator) permet de gérer les transactions.
- sécurité basée sur les rôles : un rôle est un nom symbolique qui identifie un groupe logique d'utilisateurs. L'administrateur peut, lors du déploiement d'un composant, autoriser tel ou tel rôle à utiliser le composant (sécurité déclarative via la base de registre); on peut également ajouter des conditions d'utilisation du composant au niveau du code (sécurité programmatique).
- événements : le dialogue entre clients et composants est géré par COM+ via des événements, ce qui permet à des entités de communiquer sans à priori se connaître.

5. Technologies basées sur COM+

L'Automation est une partie de la spécification du modèle COM+ qui définit une méthode standard pour la création des composants et l'utilisation des objets. En fait, l'Automation permet à une application d'en contrôler une autre en manipulant les objets COM+ qu'elle contient.

L'Automation fonctionne par l'intermédiaire de l'interface standard IDispatch qui exporte l'ensemble des propriétés et méthodes prises en charge par un objet. En règle générale, un client Automation peut utiliser tout composant qui bénéficie d'une interface IDispatch. Cette interface est présentée en détail dans la suite du document.

Exemple : VBA et la suite Office.

ActiveX (anciennement appelé OLE 2) est un ensemble de normes de conception et d'interfaces communes qui rendent possibles la communication entre composants ActiveX. Les composants ActiveX peuvent dialoguer entre eux par des échanges bidirectionnels de messages. Cette technologie basée sur COM+ permet d'utiliser des contrôles, des documents et des composants ActiveX dans différents environnements de développements. Les contrôles ActiveX sont donc des composants COM+ qui s'intègrent à des EDI (Environnement de Développement Intégré) et qui sont utilisables pour assembler des applications.

Exemple : quand on clique 2 fois sur un tableau Excel dans un document Word, le tableau envoie un message à Word pour lui indiquer de changer la barre d'outils existante en la remplaçant par celle d'Excel.

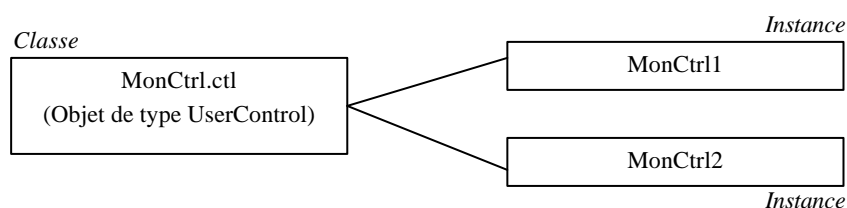
III - LES COMPOSANTS

A la base, un composant est une **unité de code** qui fournit un certain nombre de fonctionnalités. Les composants sont distribués par les serveurs qui sont des fichiers .exe, .dll ou .ocx. Les serveurs peuvent comporter un ou plusieurs composants, et les composants fournissent les modèles à partir desquels les objets sont créés.

1. Contrôle ActiveX

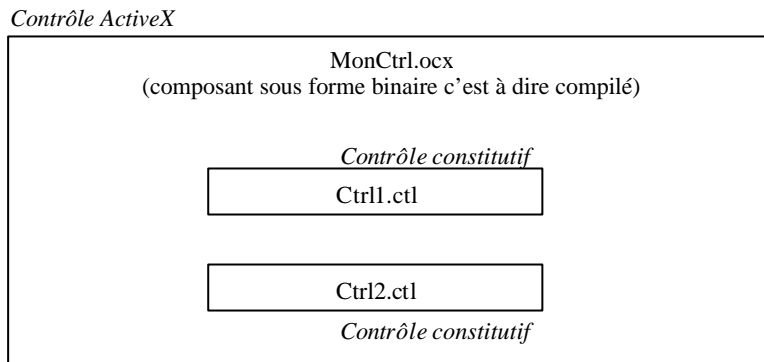
Les contrôles ActiveX (d'extension .ocx pour OLE Control Extension) sont des éléments logiciels indépendants et réutilisables qui incluent des éléments visuels et du code. La boîte à outils de Visual Basic renferme l'ensemble des contrôles intégrés; elle facilite et accélère ainsi considérablement le développement des applications. Les contrôles doivent être placés dans un type de conteneur quelconque, comme une feuille ou une application. Ils communiquent entre eux via un ensemble de normes et d'interfaces communs.

Exemple : tout contrôle créé dans Visual Basic est reconnu comme une classe de contrôle, laquelle lui sert de modèle. Lorsqu'on place un contrôle sur une feuille, une instance de ce contrôle est créée :



Un projet de contrôles ActiveX Visual Basic contient un ou plusieurs fichiers .ctl, définissant chacun une classe de contrôle distincte. Lorsqu'on compile un projet de contrôles, un fichier .ocx est créé pour le composant de type Contrôle.

Par ailleurs, un fichier .ocx unique peut compter plusieurs contrôles, comme illustré ci-dessous.



2. DLL ActiveX

Une DLL ActiveX (d'extension .dll) est un composant de code compilé qui s'exécute dans le même processus que l'application cliente. En d'autres termes, une DLL ActiveX expose à la fois des fonctionnalités à d'autres applications et s'exécute dans le même espace mémoire que l'application cliente.

3. EXE ActiveX

Un EXE ActiveX (d'extension .exe) est une application fournissant des classes d'objets. Elle s'exécute out-of-process. En d'autres termes, un EXE ActiveX expose à la fois des fonctionnalités à d'autres applications et s'exécute comme une application autonome.

Remarque : il convient de différencier les contrôles des composants. Un composant est une application qui met certaines fonctionnalités à la disposition d'autres applications et qui peut être utilisée et réutilisée par ces applications. Les composants de type Contrôle contiennent des éléments visuels capables de générer des événements à partir des actions déclenchées par l'utilisateur.

4. Document ActiveX

Un document ActiveX (d'extension .vbd) est un type spécifique d'objet ActiveX qui peut être placé et activé dans un conteneur de document ActiveX, tel que Microsoft Internet Explorer. Les documents ActiveX constituent une extension de la technologie de documents composés (liaison et incorporation d'objets encore appelé OLE).

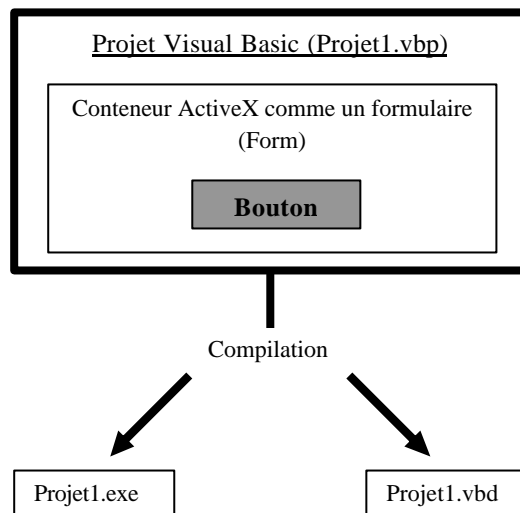
Le principal aspect fonctionnel des documents ActiveX est leur effet sur l'interface utilisateur. Un document ActiveX remplit la zone d'affichage d'un conteneur et donne ainsi à ce dernier la possibilité de se comporter comme l'application serveur à travers une interface utilisateur semblable.

Exemple : lorsqu'on exécute Microsoft Word, on lance en fait l'application Word comme conteneur de son objet document. Lorsqu'un objet document Word est ouvert à partir d'Internet Explorer, Word est lancé comme un serveur fournissant le document à un autre conteneur, Internet Explorer ici.

Il existe donc deux types de documents ActiveX :

- EXE Document ActiveX, pour lequel le serveur est un .exe
- DLL Document ActiveX, pour lequel le serveur est une .dll

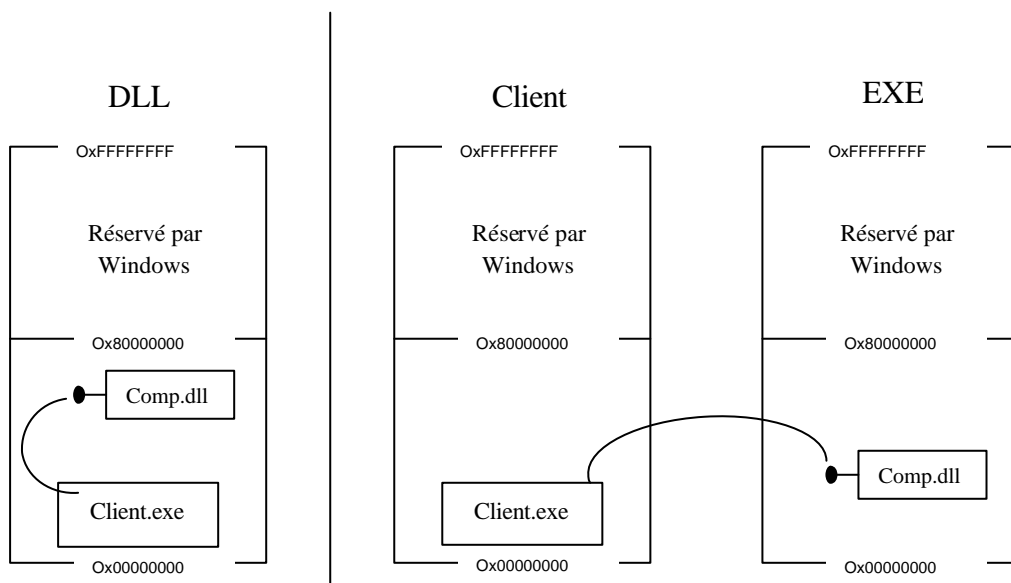
L'illustration suivante montre les fichiers créés par la compilation d'un projet EXE Document ActiveX.



La compilation d'un projet DLL ActiveX engendrerait la création du fichier Projet1.dll.

5. Choix entre un EXE et une DLL

La manière dont s'exécutent les composants EXE ActiveX (out-of-process) et DLL ActiveX (in-process) diffère quelque peu. Le schéma suivant illustre cette différence au niveau de l'organisation de la mémoire :



A partir de cette illustration, il est nécessaire de se poser quelques questions essentielles avant de choisir entre un EXE ou une DLL. Le tableau suivant résume la situation :

Question	DLL in-process	DLL in-process exécutée dans un suppléant (voir la remarque ci-dessous)	EXE out-of-process
Les instances de l'objet seront-elles partagées par plusieurs clients ?	Non	Oui	Oui
Faut-il inscrire l'objet dans la ROT ?	Non	Non	Oui
L'objet doit-il être à l'abri d'une panne du client ?	Non	Oui	Oui
L'objet doit-il s'exécuter dans COM+ ?	Oui	Oui	Non
L'objet doit-il avoir son propre contexte de sécurité ?	Non	Oui	Oui
L'objet est-il responsable de sa durée de vie ?	Non	Non	Oui
Les performances d'exécution de l'objet doivent-elles être importantes ?	Oui	Non	Non
L'objet peut-il s'exécuter de façon autonome ?	Non	Non	Oui
L'objet doit-il exposer ses structures internes au client ?	Oui	Non	Non

Remarques :

- quand une DLL est exécutée à distance, il est nécessaire qu'un processus parent sur l'ordinateur distant gère la DLL. COM+ introduit alors le principe de suppléant (surrogate), dllhost.exe par défaut, programme exécutable qui est chargé de fournir un processus parent et un contexte d'exécution à la DLL. C'est dans la base de registre (HKEY_CLASSES_ROOT\AppID) que la configuration du contexte d'exécution pour un composant est définie : l'utilitaire dcomcnfg.exe permet de configurer les environnements distribués de composants, DCOM en l'occurrence.
- la ROT (Running Object Table) est une table système dans laquelle peuvent s'inscrire les objets en cours d'exécution.

IV - LES INTERFACES

Les interfaces constituent le point de départ de la communication entre composants COM. Une interface est une partie visible du composant depuis le monde extérieur. C'est à partir d'une interface que l'on peut manipuler un composant, c'est à dire appeler les méthodes de l'interface du composant. Dans le monde COM+, un composant peut exposer plusieurs interfaces.

La spécification du modèle COM exige de tous les objets qu'ils remplissent les fonctions suivantes :

- L'objet doit être en mesure de répertorier le nombre de connexions reçues. Lorsqu'il n'est plus utilisé, il doit être capable de s'autodétruire.
- Une interface n'est jamais modifiée.
- Chaque interface a un numéro d'identité unique codé sur 128 bits, le GUID (Globaly Unique ID), fourni par Microsoft.
- Tout client doit avoir la possibilité de demander à accéder à toutes les interfaces complémentaires prises en charge par l'objet.

Toutes les interfaces COM+ dérivent de l'interface de base, IUnknown. Par convention, leurs noms commencent par I. Le langage de définition des interfaces préconisé est IDL (Interface Definition Language).

1. IDL

L'IDL (Interface Definition Language) est une norme qui décrit la syntaxe du langage de spécification des objets (entre d'autres termes la manière de décrire les interfaces des composants) et des projections vers les langages de programmation classiques (c'est à dire comment les instructions IDL seront traduites dans un langage de programmation donné). Microsoft a introduit quelques extensions à la spécification originelle d'IDL pour l'adapter à COM+.

Par exemple, le mot clé **object** signifie que l'on est en présence d'une interface COM+. La définition en IDL d'une interface qui ne commence par COM+ décrit une interface RPC (Remote Procedure Call), ce qui nous ramène à l'origine de l'utilisation d'IDL : dans le cadre de la création de la spécification DCE (Distributed Computing Environment), l'OSF (Open Software Fundation) a créé le protocole RPC en décrivant les interfaces en IDL.

L'interface IUnknown est définie ainsi en IDL :

```
[
    local,
    object,
    uuid(00000000-0000-0000-C000-000000000046),
    pointer_default(unique)
]
interface IUnknown
{
    typedef [unique] IUnknown *LPUNKNOWN ;
    HRESULT QueryInterface( [in] REFIID riid, [out, iid_is(riid)] void ** ppvObject) ;

    ULONG AddRef() ;
    ULONG Release() ;
}
```

Il suffit ensuite d'utiliser un compilateur IDL qui transforme le code IDL en un code relatif à un langage de programmation, comme par exemple MIDL qui, à partir de l'interface IDL, crée la spécification C/C++ correspondante.

2. L'interface IUnknown

Tous les objets disposent d'une interface intégrée appelée IUnknown. L'interface IUnknown est prise en charge par chaque objet du modèle COM. Toute autre interface ajoutée à l'objet doit en outre inclure les fonctionnalités offertes par l'interface IUnknown. En d'autres termes, tout composant logiciel COM+ doit redéfinir l'interface IUnknown.

L'interface IUnknown, de GUID 00000000-0000-0000-C000-000000000046, comprend trois méthodes : **AddRef**, **Release** et **QueryInterface**.

```
interface IUnknown /* déclarée en C++ */
{
    virtual HRESULT QueryInterface() ;
    virtual ULONG AddRef() ;
    virtual ULONG Release() ;
}
```

Les fonctions **AddRef** et **Release** assurent le suivi de la création et de la destruction de l'objet, et la fonction **QueryInterface** permet aux clients de demander un accès aux autres interfaces offertes par l'objet.

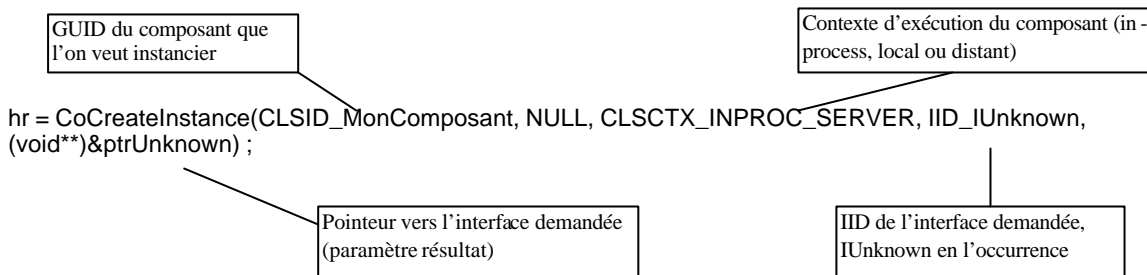
La fonction **AddRef** incrémente le compteur d'utilisations de l'objet lors de l'affectation d'un pointeur d'interface. La fonction **Release** décrémente ce compteur lorsqu'une variable pointant sur l'objet sort de la portée.

Typiquement, on aura le scénario suivant : acquisition de la part du client d'un pointeur vers IUnknown, appel de **QueryInterface** avec en paramètre le GUID de l'interface demandée par le client ; si l'interface demandée est trouvée, **QueryInterface** renvoie un pointeur sur celle-ci.

Pour qu'un composant soit accessible aux clients, il doit s'enregistrer auprès du système d'exploitation. Pour ce faire, sous Microsoft Windows, il procède à la mise à jour de la section HKEY_CLASSES_ROOT de la base de registres système en précisant un certain nombre d'informations le concernant, telles son nom de classe (ProgID), qui est un nom lisible désignant le composant, son identificateur de classe (ClassID), son emplacement et l'emplacement de sa bibliothèque de types (qui constitue une description binaire du composant).

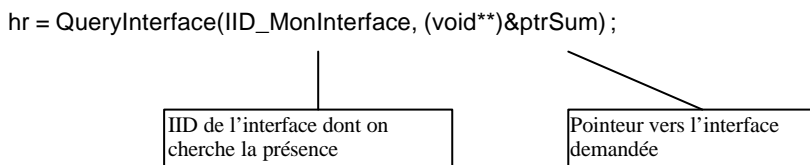
Exemple : l'appel d'un composant COM+ depuis un client en C++ se décompose selon les étapes suivantes.

- appel de CoInitializeEx : `hr = CoInitializeEx(NULL, COINIT_APARTMENTTTRHEADED)` ;
Cet appel de la fonction CoInitializeEx membre de l'API COM+ (toutes les fonctions de l'API COM+ commencent par Co) initialise la bibliothèque COM+.
- demande d'instanciation du composant. Le client doit toujours en premier lieu essayer d'obtenir un pointeur vers l'interface IUnknown du composant dont il a besoin :

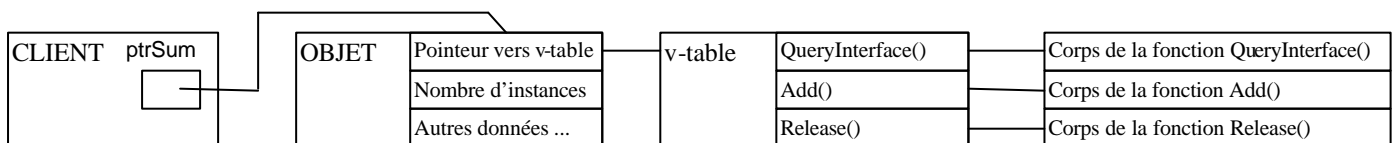


Remarque : la recherche du serveur dans la base de registre qui contient le composant de GUID CLSID_MonComposant se fait en fait via l'interface IClassFactory présentée ultérieurement.

- après l'appel à `CoCreateInstance()`, on dispose d'un pointeur sur l'interface IUnknown. Avec ce pointeur `ptrUnknown`, on peut appeler les méthodes `Add()`, `Release()` et `QueryInterface()` :



- si la phase de découverte réussie, `ptrSum` permettra d'appeler les méthodes de l'interface demandée. On peut souligner qu'en réalité, `ptrSum` est un pointeur vers un pointeur qui référence une v-table (virtual table). On a donc :



- après le travail, il est nécessaire de fermer la bibliothèque COM+ par l'appel de `CoUnInitalize()`.

3. L'interface IClassFactory

Quand un client appelle `CoCreateInstance()`, celle-ci doit retourner un pointeur vers l'interface IUnknown du composant demandé. Le client fournit le GUID du composant et COM+ fournit un moyen de retourner un pointeur vers ce composant.

C'est un composant Factory ou « usine à objet » qui se charge de ce service. Ce composant COM, via son interface IClassFactory, localise le serveur hébergeant l'objet demandé par le client, l'instancie et retourne un pointeur vers l'interface IUnknown au client.

L'interface IClassFactory a deux méthodes : `CreateInstance()` et `LockServer()`. `CreateInstance()` instancie réellement un composant COM+ et `LockServer` empêche le composant d'être déchargé de la mémoire pendant qu'un client est en train de l'utiliser.

Remarque : la fonction `CoCreateInstance()` de l'API COM+ vu précédemment appelle en fait la fonction `CoGetClassObject()` qui appelle la fabrique à objet du composant demandé...

La correspondance entre le GUID donné par le client et le nom du serveur à utiliser est enregistrée dans la base de registre. La clé `HKEY_CLASSES_ROOT\CLSID` contient le GUID et son serveur associé. En fait, l'appel de `CoCreateInstance()` entraîne celui de `CoGetClassObject()`. Un service de COM+, appelé SCM (Service Control Manager), est alors lancé. Le SCM localise et charge le composant demandé par le client puis s'efface lorsque la communication est établie.

4. L'interface IDispatch

L'interface IDispatch permet de piloter les objets qui la porte via des scripts VBA par exemple. Les objets COM+ qui portent cette interface sont appelés des objets automates (Automation Objects) et les clients sont désignés par le terme contrôleur d'automates (Automation Controllers). Elle est à la base de l'Automation présentée précédemment. L'interface IDispatch permet donc une invocation dynamique d'un composant COM+ en établissant un canal de communication et de commande entre le contrôleur et l'automate. La spécification simplifiée de IDispatch en IDL est la suivante :

```
interface IDispatch : IUnknown
{
    HRESULT GetTypeInfoCount(...); /* renvoie 1 s'il existe une librairie de type, 0 sinon */

    HRESULT GetTypeInfo(...); /* renvoie une interface ITypeInfo qui donne la liste des arguments d'une
                               méthode donnée */

    HRESULT GetIdsOfNames(...); /* transforme le nom d'une méthode en un identifiant pour appeler
                                Invoke() */

    HRESULT Invoke(...); /* appelle une méthode à partir de son identifiant */
}
```

5. L'interface IMoniker

Les monikers sont des objets qui identifient d'autres objets. Les monikers permettent à chaque type d'objets de gérer leurs conventions de nommage via une interface commune, IMoniker. Au lieu d'utiliser des CLSID pour identifier des objets, les monikers utilisent des noms en clair (display name) comme C:\MonRep\MonFich.doc ou Excel.Application. La ROT mentionnée précédemment utilise les monikers. Pour en voir une application réelle, l'utilitaire ROTViewer, livré avec Microsoft Visual Studio, permet de visualiser les objets en exécution avec leurs noms en clair.

Exemple : l'appel de la méthode GetObject(« C:\MonRep\MonFich.doc ») en Visual Basic implique un appel interne à la fonction COM+ MkParseDisplayName() (pour obtenir un moniker adéquat) puis une utilisation de la méthode IMoniker::BindToObject() permet d'obtenir un pointeur vers l'objet passé en paramètre de GetObject().

V – LA COMMUNICATION ENTRE COMPOSANTS

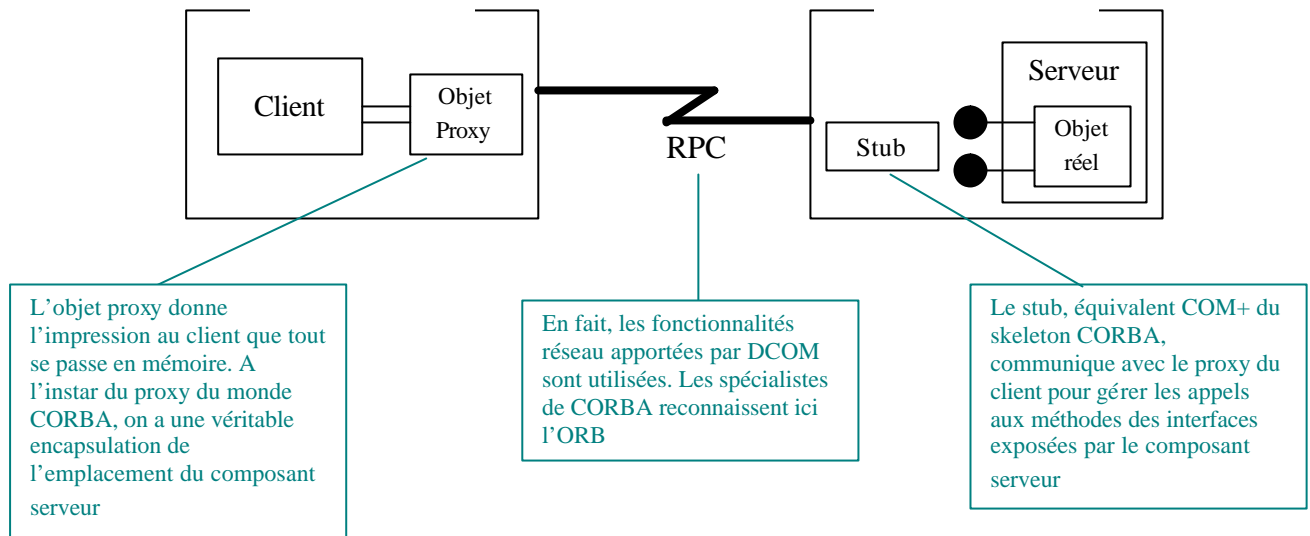
1. Proxy et stub

Comme mentionné précédemment, COM+ fournit des services permettant à des composants de dialoguer entre eux, quel que soit leurs emplacements sur un réseau. Pour que les composants puissent dialoguer, COM+ utilise deux objets spécifiques appelés proxy et stub.

En pratique, un compilateur compatible COM+ insère du code proxy à l'intérieur d'une application cliente afin de gérer les appels du serveur et les valeurs renvoyées. Du côté composant serveur compatible COM+, du code stub est inséré pour gérer les appels à partir du code proxy du client ainsi que toutes les valeurs renvoyées aux clients.

Ce traitement des paramètres et des valeurs renvoyées constitue ce que l'on appelle le **marshalling**. Le marshalling peut donc se définir comme le conditionnement de l'ensemble des paramètres et valeurs renvoyées, et leur va-et-vient entre les processus.

L'illustration ci-après représente la circulation des informations entre un client et un serveur par l'intermédiaire de code proxy et stub.



Remarque : en ce qui concerne le réseau, COM+ se situe au niveau Application. On parle de ORPC (Object Remote Procedure Call) car le protocole réseau de COM+ s'appuie sur RPC, lequel utilisera un protocole de niveau Transport... Cette communication interprocessus est donc indépendante de la plate-forme.

2. Bibliothèque de types

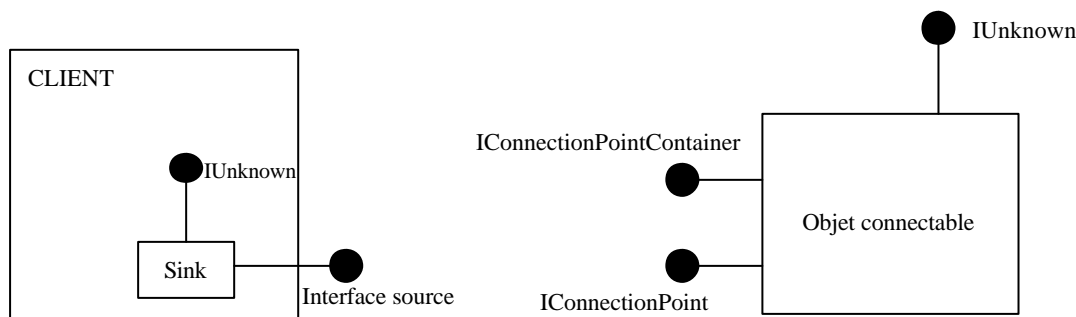
Les bibliothèques de type sont une description dynamique de chaque objet (propriétés et méthodes). En d'autres termes, une bibliothèque de type est une version binaire d'un fichier IDL : elle constitue une description binaire des interfaces exposées par le composant.

Beaucoup d'EDI (Environnement de Développement Intégré) peuvent lire les bibliothèques de types (Visual C++, Visual Basic, ...) : cela facilite l'utilisation des composants. Sous Visual Basic, la liste de suggestion qui s'affiche lors de la frappe du code illustre l'utilisation des bibliothèques de types. De même, l'Explorateur d'objets de Visual Basic permet de s'informer, via les bibliothèques de types, sur les interfaces d'un composant.

Les interfaces COM+ qui sont utilisées pour construire des bibliothèques de types sont ICreateTypeLib et ICreateTypeInfo. La lecture se fait par ITypeLib et ITypeInfo.

3. Points de connexions

La communication bidirectionnelle entre un client et un composant COM+ se fait via les points de connexion. Un point de connexion permet à un objet de parler à son client : on dit que l'objet déclenche un événement que le client traitera. Les objets ayant des points de connexion sont appelés des objets connectables. Un objet connectable va utiliser l'interface source du client pour lui parler. L'interface source du client est portée dans le client par un objet sink; on a en fait la situation suivante :



L'interface principale est IConnectionPoint, elle permet au client de fournir à l'objet connectable un pointeur vers son objet sink. L'objet connectable peut ensuite utiliser ce pointeur pour appeler les méthodes du sink et ainsi émettre des messages vers le client.

Remarque : il existe deux autres interfaces (IEnumConnectionPoints et IEnumConnections) pour les points de connexion non commentées ici.

VI - COM+ ET VISUAL BASIC

Visual Basic permet, à partir de la version 5.0, de créer des composants COM+. Il masque un grand nombre de détails. Cela permet de faciliter la création de composants et d'alléger la charge de travail. Mais le point sur lequel Visual Basic se distingue concerne l'utilisation de composants COM+ : pour assembler des composants, Visual Basic apparaît comme un outil adapté. Il faut donc voir Visual Basic comme un langage de haut niveau par opposition aux langages plus puissants comme le C++, que l'on peut considérer comme des langages de bas niveau. Lorsqu'il est nécessaire de contrôler dans les moindres détails le code du composant, le C++ est recommandé. Par contre, l'assemblage de composants préfabriqués est plus simple et plus accessible à tous en Visual Basic.

Il faut par ailleurs être conscient que le développement d'applications à base de composants, comme tout autre développement d'ailleurs, ne peut se faire sans méthode de conception rigoureuse. De plus, la description des interfaces est capitale pour concevoir un système cohérent. A ce titre, un langage de conception, comme UML par exemple, et une méthode de travail sont indispensables lors de la création de composants logiciels.

1. Utilisation de composants COM+ en VB

Visual Basic (VB) encapsule les services de COM+, ce qui se traduit par la création rapide d'applications à partir de composants préfabriqués. Les lignes qui suivent illustrent les relations que VB entretient avec COM+ pour faciliter la programmation.

En premier lieu, VB initialise automatiquement les services de COM+ par les appels en interne de `CoInitialize()` et `CoUninitialize()`.

De plus, le mot clé `New` remplace l'appel de `CoCreateInstance()` pour les appels locaux seulement. Cependant, il est conseillé d'utiliser `CreateObject()`, fonction de VB qui utilise pleinement les services de COM+.

```
Dim maReference As Object
```

'création d'une référence à partir d'un ProgID, la recherche du CLSID de l'objet dans la base de registre est effectuée de manière transparente

```
Set maReference = CreateObject(« MonComposant », « MonServeur »)
```

```
'...
```

L'interface appelée est celle par défaut. Le code ci-dessus appelle donc `QueryInterface()` avec le GUID de l'interface par défaut du composant.

Cependant, le code de création ci-dessus peut être amélioré. En effet, le fait de déclarer un objet avec `As Object` fait que VB utilise l'interface `IDispatch` pour manipuler l'objet. La déclaration suivante est donc recommandée :

```
Dim maReference As IUnknown
```

```
Set maReference = New MonInstance
```

Enfin, le garbage collector intégré à VB appelle `Release()` quand le moment est venu. Il est possible de forcer cet appel par l'instruction `Set maReference = Nothing`.

2. Création de composants COM+ en VB

On l'a déjà mentionné, les versions de VB postérieures à la release 5.0 permettent de créer des composants COM+. Cette création est simplifiée en VB par rapport au travail que doit fournir le programmeur sous Visual C++ et ses MFC par exemple.

La démarche reste cependant la même, quel que soit l'EDI utilisé. Les principales étapes de la création sont les suivantes :

- Définition du besoin à l'origine du projet, « limites » du composant
- Spécifications de(s) interfaces (méthodes fournies)
- Choix du type de composant selon le contexte d'utilisation (se reporter au tableau III -5)
- Création du composant
- Tests unitaires
- Compilation (p-code, code natif)
- Distribution (licence, ...)

Pour voir ce qu'emmène VB lors de la création d'un composant, il est intéressant de partir de la définition IDL que l'on veut implémenter :

```
import « unknown.idl » ;

[object, uuid(00...001)]
interface MonInterfaceParDefaut : IUnknown
{
    HRESULT MaMethode1() ;
}

[object, uuid(00...002)]
interface MonAutreInterface : IUnknown
{
    HRESULT MaMethode2() ;
}

[uuid(00..003)]
library MonComposant
{
    importlib(« stdole32.tlb ») ;
    interface MonInterfaceParDefaut;
    interface MonAutreInterface ;

    [uuid(00..004)]
    coclass MaClasse
    {
        [default] interface MonInterfaceParDefaut;
        interface MonAutreInterface ;
    }
};
```

La création sous Visual Basic d'un composant ActiveX respectant la spécification ci-dessus peut se faire par exemple en créant un nouveau projet de type DLL ActiveX, dans lequel deux modules de classe de noms MonInterfaceParDefaut et MonAutreInterface seraient créés. Le présent document n'ayant pas pour objectif de décrire comment développer sous Visual Basic un composant ActiveX, les détails de cette création ne sont pas présentés ici.

Le composant créé sera utilisé ainsi par un client VB (les commentaires en pseudo-code expliquent ce que fait COM+) :

```
'IMonInterfaceParDefaut *maRef
'CoCreateInstance(CLSID_MaClasse, NULL, CLSCTX_INPROC_SERVER, IID_IMonInterfaceParDefaut,
'(void**)&maRef) ;
Dim maReference As New MonComposant.MaClasse

'MonAutreInterface *monAutreRef
Dim monAutreReference As MonComposant.MonAutreInterface

'maRef→ MaMethode1() ;
maReference.MaMethode1

'maRef→QueryInterface(IID_MonAutreInterface, (void**)& monAutreRef) ;
Set monAutreReference = maReference

'monAutreRef → MaMethode2() ;
monAutreReference.MaMethode2

'maRef→Release() ;
Set maReference = Nothing

'le garbage collector appelle 'monAutreRef →Release() ;
```

Par défaut, un composant COM+ crée en VB expose les interfaces suivantes :

- IUnknown
- IDispatch
- IProvideClassInfo
- ISupportErrorInfo (garantit la bonne gestion des erreurs)
- IConnectionPoint et IConnectionPointContainer (points de connections des objets connectables)
- IExternalConnection (gère le compteur des références)

La machine virtuelle VB6.0 (msvbvm60.dll) fournit automatiquement un objet qui implémente l'interface IClassFactory.

Un module de classe VB ne définit qu'une seule classe. L'interface de cette classe constitue l'interface COM+ par défaut du composant. Le mot clé Implements permet d'ajouter des interfaces supplémentaires au composant. Le code du module de classe MaClasse suivant illustre ce principe :

```
Implements IMonAutreInterface
Private Function MaMethode()
    '...
End Function
```

L'interface par défaut sera par défaut appelée _MaClasse.

3. Communication entre composants en VB

L'utilisation de l'instruction WithEvents en VB lors de la création d'objets permet de revenir sur les objets connectables présentés préalablement. En VB, la déclaration d'un objet avec l'instruction WithEvents crée automatiquement un objet sink. Chaque fois que l'objet connectable appelle les méthodes de l'interface source du sink, ce même objet sink créé par VB regarde si le programmeur a écrit du code pour traiter l'événement entrant. Le code suivant illustre ce service COM+ encapsulé par VB :

'déclaration d'un objet

```
Dim WithEvents maRef As MonComposant
```

'instanciation du composant

```
Set maRef = New MonComposant
```

'les messages envoyés par le composant sont à traiter dans les procédures événementielles

```
Private Sub maRef_MonEvenement(ByVal MonMessage As Long)
```

```
    'on traite ici un évènement entrant provoqué par l'objet connectable MonComposant
```

```
End Sub
```

4. Détail d'un appel Automation en VB

Ce dernier paragraphe revient sur la fonction CreateObject pour détailler ce que COM+ réalise automatiquement grâce aux appels internes de VB.

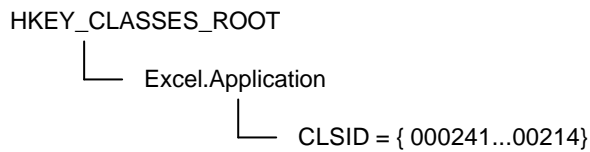
```
Dim Excel As Excel.Application
```

```
Set Excel = CreateObject(« Excel.Application »)
```

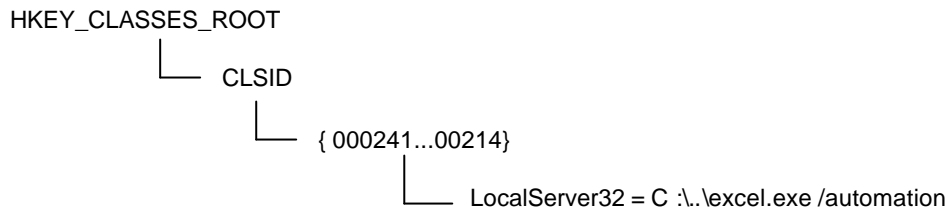
Après cette instruction, voici les détails des actions réalisées par COM+ pour récupérer la référence demandée.

- recherche dans la base de registre, dans HKEY_CLASSES_ROOT, d'une entrée correspondant à l'identifiant du programme Excel.Application (via les monikers).

- si la recherche aboutie, COM+ récupère le GUID de la classe :



- avec la CLSID trouvée, COM+ récupère une autre entrée dans la base de registre qui fournit les informations sur le serveur d'objets à utiliser :



- enfin, il y a une demande de pointeur sur l'interface IDispatch du serveur Automation (Excel ici) et renvoie de ce pointeur à l'application cliente, le projet VB en l'occurrence.

Finalement, on voit apparaître avec cet exemple les services que rend COM+ aux programmeurs. Un véritable environnement d'exécution et un ensemble des services sont fournis pour faciliter la communication des composants. En tant que langage de haut niveau, VB est adapté à la manipulation de composants COM+.

VII - CONCLUSION

La complexité toujours croissante des applications, l'intégration devenue de plus en plus difficile ont propulsé les applications à base de composants sur le devant de la scène. Leurs aptitudes à la réutilisation, à la modularité en feront peut-être les fondements même des entreprises dans le futur.

De plus, la difficulté de conception d'applications stratégiques totalement réparties sur un réseau est considérablement réduite grâce aux architectures logicielles objet. En conséquence, COM+, CORBA et EJB constituent les bases de l'informatique de demain.

VIII – GLOSSAIRE

A

ActiveX.....
Permet à un programme client d'invoquer dynamiquement les méthodes de composants OLE.

Automation.....
Technologie Microsoft permettant à une application d'en contrôler une autre via les objets exposés par cette dernière.

C

COM.....
Component Object Model, norme binaire définissant un modèle objet et un ensemble de règles programmatiques permettant aux objets (composants) d'interagir entre eux via leurs interfaces.

COM+.....
COM+ peut être présenté comme la réunion de DCOM et MTS.

composants.....
Un composant est un bloc de code binaire (par opposition à un bloc de code source) fournissant des services spécifiques via de(s) interface(s).

CORBA.....
Common Request Broker Architecture, spécification normative de l'OMG définissant les systèmes d'objets distribués. Elle comprend un langage de description d'objets, l'IDL, une infrastructure de distribution des composants, l'ORB (Object Request Broker) - dont l'équivalent chez Microsoft est une partie de COM/DCOM -, une description des services de bases, CorbaServices, une description des services haut niveau, CorbaFacilities, une collection de composants métiers, DomainServices et enfin une norme d'interopérabilité entre ORB.

D

DCOM.....
Distributed COM, évolution antérieure à COM+, qui offre des fonctionnalités réseau à COM.

DLL.....
Dynamic Link Library, une librairie de lien dynamique est un ensemble de code stocké dans un fichier .dll. Ces routines sont appelées dynamiquement et chargées une seule fois en mémoire. Un composant in-process est une variété de DLL.

E

EJB.....
Entreprise Java Beans, évolution et enrichissement des composants logiciels Java Beans par Sun pour former un nouveau modèle de composants logiciels basé sur Java. A mettre en parallèle avec CORBA et COM+.

G

grappes.....
Une grappe de machines désigne dans l'environnement COM+ un ensemble de serveurs, permettant la répartition des charges de travail (load balancing) entre plusieurs CPU.

GUID.....
Global Unique Identifier, comme les UUID (Universal Unique Identifier), ce sont des nombres codés sur 128 bits identifiant une certaine interface d'une certaine classe.

I

interopérabilité.....
Capacité des composants à coopérer en échangeant des informations

M

marshaling.....
Action de mettre en ordre le dialogue entre un client et un composant COM+ distant.

MS DTC	Microsoft Distributed Transaction Coordinator, c'est un gestionnaire de transactions qui permet aux applications clientes d'inclure plusieurs sources de données différentes dans une seule transaction. MS DTC coordonne la validation de la transaction distribuée au travers de tous les serveurs inscrits dans la transaction.
MSMQ	Microsoft Message Queue Server, technologie COM+ permettant de placer les appels de méthodes de composants dans des files d'attente.
MTS	Microsoft Transaction Server, système transactionnel de Microsoft permettant de gérer des applications distribuées. Les équivalents de MTS dans les autres modèles d'objets distribués sont OTS (Object Transaction Services) pour CORBA et JTS (Java Transaction Services) pour EJB.
O	
OLE	Object Linking and Embedding, permet d'assembler et d'intégrer des objets. OLE regroupe un ensemble de services basé sur COM, comme le Glisser-Déposer par exemple.
OMG	Object Management Group, fondé en 1989, c'est un consortium de 700 sociétés impliquées dans les évolutions de l'informatique qui a pour objectif de fournir des spécifications adoptées par ses membres. Est à l'origine de la norme CORBA.
P	
proxy	Le proxy est le représentant coté client d'une interface COM+ lorsque un client veut utiliser un composant en dehors de son espace d'adressage. Dialogue avec le stub lors du marshaling.
R	
RAD	Rapid Application Development, technique de développement rapide d'applications basée sur la création de maquettes jetables (des prototypes d'écrans par exemple) pour explorer les besoins utilisateurs tout en préparant le système réel.
rôle	Un rôle permet de gérer la sécurité des composants COM+.
routeur d'équilibrage de charge	Le routeur d'équilibrage de charge est un service fourni par COM+ permettant de répartir la charge de travail entre plusieurs machines.
RPC	Remote Procedure Call, norme définie par l'OSF pour permettre à un processus d'appeler les fonctions d'un autre processus situé sur une machine distante.
S	
stub	Le stub coté serveur dialogue avec le proxy sur le client pour permettre l'appel de méthodes d'une interface COM+ distante.
T	
transaction	Suite d'instructions devant respecter la règle ACID (Atomique, Cohérente, Indivisible, Durable).

IX - BIBLIOGRAPHIE

Les ouvrages spécifiques à Microsoft :

- Au cœur de COM+, G. et H. Heddon, Microsoft Press
- Au cœur de ActiveX et OLE, David Chappell, Microsoft Press
- Atelier Visual Basic 6.0, Microsoft Press
- Atelier Visual C++ 6.0, Microsoft Press

Les ouvrages sur les modèles d'architectures distribuées :

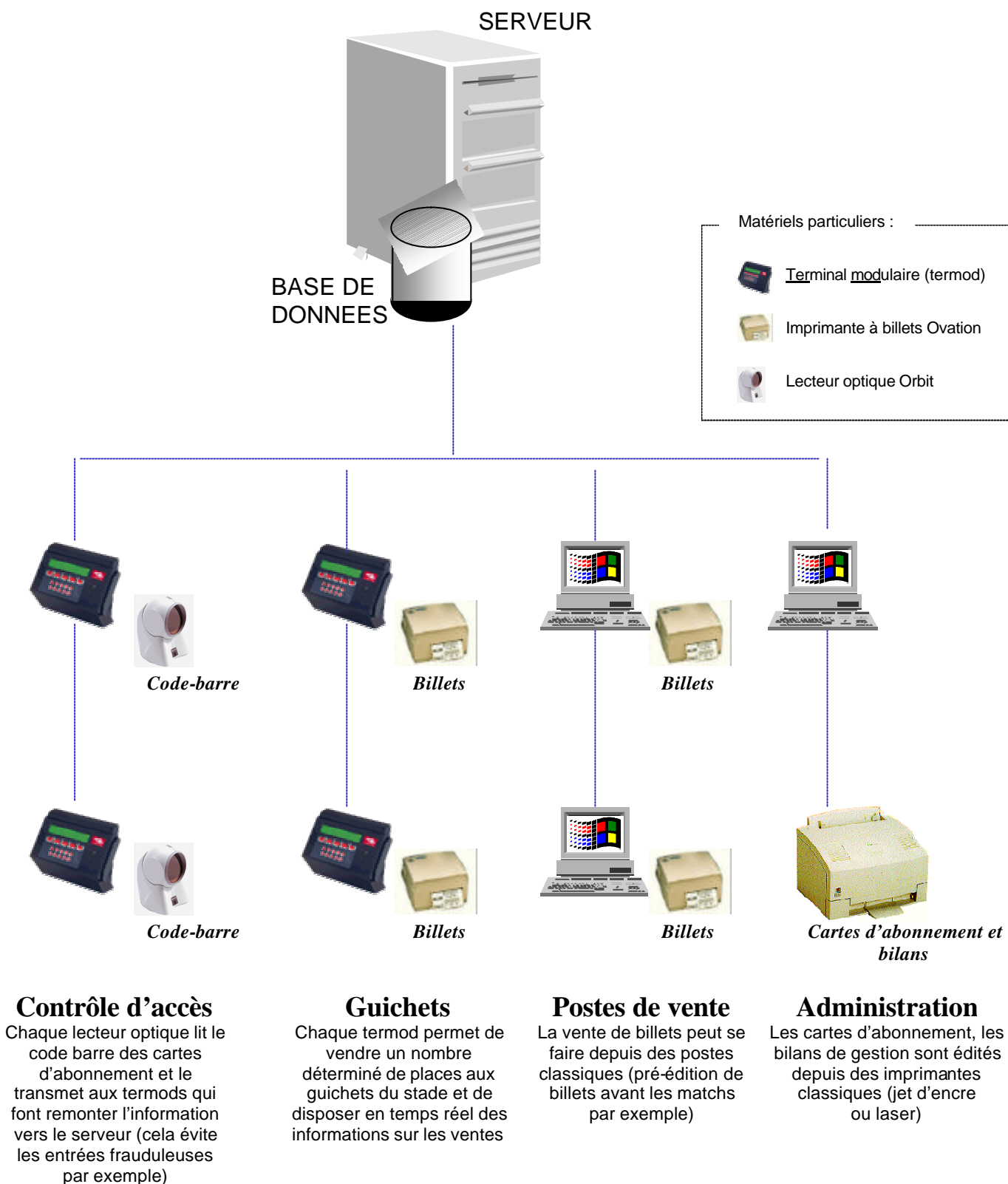
- Corba, ActiveX et Java Beans, J.M. Chauvet, Eyrolles
- Composants et transactions, J.M. Chauvet, Eyrolles
- Objets métier, T. Andro et J.M. Chauvet, Eyrolles

La documentation sur l'Internet :

- Site de l'OMG, www.omg.org/
- Spécifications de CORBA, www.omg.org/technology/documents/formal/
- Spécifications d'EJB, java.sun.com/products/ejb/
- Site dédié à COM+, www.microsoft.com/com/tech/COMPlus.asp
- Spécifications de COM+, www.microsoft.com/Com/resources/comdocs.asp

III - ANNEXE 3, CONFIGURATION MATÉRIELLE DE LA BILLETTERIE

De manière générale, le progiciel de BILLETTERIE de la société CIMA se présente selon la topologie suivante :



IV - ANNEXE 4, MIGRATION DE SQL SERVER 6.5 À 7.0

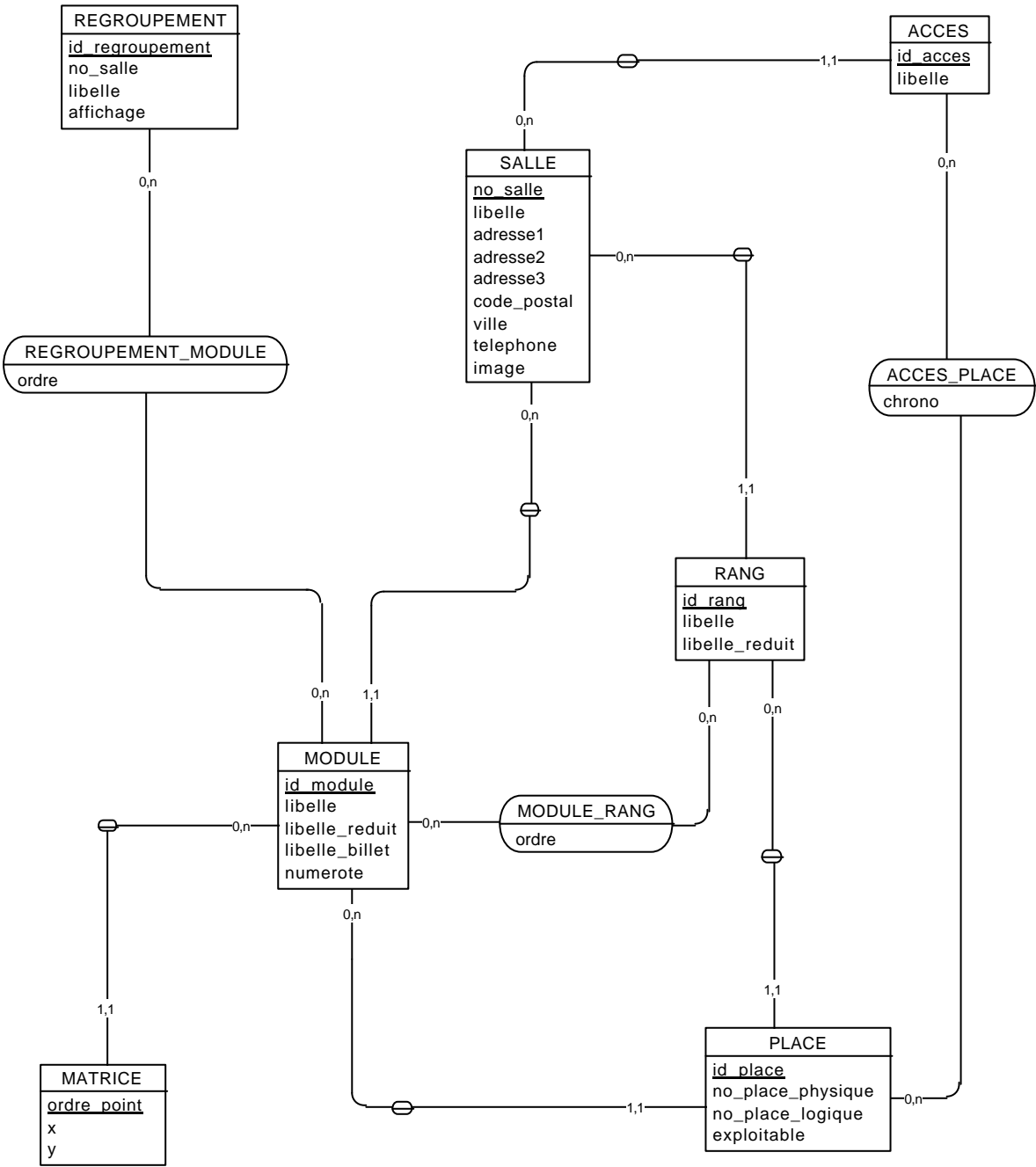
La marche à suivre lors d'une migration de SQL Server 6.5 vers SQL Server 7.0 est synthétisée dans le tableau suivant (seule la partie de préparation est disponible ici). Des références à des documents de l'étude complète sont présentes dans les lignes du tableau. Elles ont été conservées même si l'ensemble des documents de la migration ne fait pas partie du rapport de stage.

Libellé de l'étape	Solution ou remarque	Fait ? (O/N)
Avant la migration		
connexion sur le serveur NT Cimasql du domaine Spdmn	login <i>nomadministrateur/motdepasse</i> . A noter que pour accéder à Dragon via le réseau, il faut donner à l'explorateur le login <i>nomadministrateur/motdepasse</i> .	O
connexion sur le serveur NT Dragon du domaine Cassiopee	login <i>nomadministrateur/motdepasse</i> . A noter que pour accéder à Cimasql via le réseau, il faut donner à l'explorateur le login <i>nomadministrateur/motdepasse</i> .	O
vérifier que l'assistant de migration est installé	si besoin, installer l'assistant	O
informer les utilisateurs que les serveurs impliqués dans la migration seront indisponibles (à priori une matinée maximum)	empêcher l'accès aux machines impliquées dans la migration	O
établir une relation d'approbation entre les 2 domaines	non réalisée, modification du contenu des groupes NT sur Dragon à la place (voir <i>migra_result.doc</i> , II, 3°)	N
sauvegarde des bases sur les 2 systèmes	fait quotidiennement sur bande par Christine, concerne les .bak adap, adapei, bilbcp, ebpo, gemco, minotbcp et les dossiers de comptabilité, de projets et de bureautique	O
Tests de restauration des bases sur les 2 systèmes		O
arrêt de tous les services dépendant de SQL Server	Panneau de Configuration - Services	O
arrêt de l'Observateur d'événements Microsoft Windows NT et de Regedt32.exe.	Panneau de Configuration - Services	O
fermeture de toutes les applications		O
vérifier qu'un compte d'utilisateur de domaine est affecté aux services MSSQLServer, SQLServerAgent et MSDTC sur les 2 machines	utilisation de Entreprise Manager sur les 2 serveurs et vérification que tous les services sont activés	O
vérification du canal d'écoute par défaut \\.\pipe\sql\query	vérification avec les utilitaires de configuration Réseau fournit avec SQL Server	O
vérification (si besoin installation) des derniers Service Packs	<i>sp_server_info</i>	O
contrôle de cohérence de bases de données	[<i>sp_dboption</i> 'base_a_verifier', 'single user', True - DBCC CHECKALLOC ('base_a_verifier') - <i>sp_dboption</i> 'base_a_verifier', 'single user', False] à faire pour toutes les bases - DBCC CHECKCATALOG pour les cohérences inter-bases (voir <i>migra_etapes_info.xls</i>)	O
vérification de l'espace disque serveur cible > 1 fois et demi la taille totale des unités de BD 6.5 source	utilisation de l'Analyseur de performances de SQL, l'expl. Win ou [<i>sp_spaceused</i> @updateusage = 'TRUE' ou <i>sp_helpdb</i>]	O
affectation d'au moins 10 Mo à tempdb dans SQL Server 6.5 (25 Mo sont recommandés).	<i>sp_spaceused</i> tempdb @updateusage = 'TRUE'. Si besoin, augmentation, Entreprise Manager ou [DISK INIT name = 'tempdb1', physname = 'c:\mssql\data\tempdb1.DAT', vdevno = 100, size = 12800 - ALTER DATABASE tempdb ON tempdb1 = 25]	O
vérification que la base master de 6.5 a au moins 3 Mo de libre	use master - <i>sp_spaceused</i> @updateusage = 'True'	O
vérification que les utilisateurs possèdent une connexion dans la table syslogins et sysusers de master	quel que soit le mode de sécurité (standard ou intégré), il faut que les logins soient présents sur le nouveau système pour que les créations des objets réussissent, la migration de master est donc recommandée	N
vérifier que le nom du serveur 6.5 n'est pas NULL (valeur de @@servername)	select @@servername - si valeur NULL, <i>sp_addserver</i> 'Dragon', local	O

Libellé de l'étape	Solution ou remarque	Fait ? (O/N)
désactivation des procédures stockées de démarrage (car l'Assistant Mise à niveau de SQL Server démarre et arrête le serveur SQL Server 6.5 pendant le processus de mise à niveau)	sp_helpstartup pour connaître les sp lancées automatiquement puis sp_unmakestartup <nom_proc> (voir migra_detail.doc)	O
vérification que le journal est vide	DBCC SQLPERF(LOGSPACE) pour avoir des informations sur le journal, DBCC OPENTRAN ou SELECT @@TRANCOUNT pour savoir si des transactions sont en cours, sp_lock ou KILL si besoin pour arrêter des processus	O
vérification que sp_rename n'a pas été utilisée sur 6.5 : le texte original de création de l'objet est gardé intact dans syscomments (sp_helptext permet de le voir) et c'est à partir de ce texte que l'upgrade travaille	consulter le journal d'historiques de travaux dans Entreprise Manager, voir verifname.sql et res_verifname.txt	O
sortir les directives de l'optimiseur éventuellement utilisées en 6.5	voir Entreprise Manager, getoption65.sql et res_getoption65.txt	O
les colonnes utilisées pour la définition de clés étrangères doivent être du même type (il n'y a plus de conversion implicite)	vérifier le code des create table ...	N
compatibilité de niveau entre 6.5 et 7.0 (pour la liste exhaustive, voir Annexe_etapes.doc)	Les points suivants sont à souligner: 1- SQL Server 7.0 ne supporte pas *= ou =* (remplacés par OUTER JOIN), 2- Le GROUP BY n'est plus suivi implicitement d'un ORDER BY. Voir migra_detail.doc et getoption65.sql	N
<i>Le reste de la marche à suivre (paramétrage de l'assistant, exécution de l'assistant, résultats de la migration) n'est pas présentée, par souci de concision ...</i>		

V - ANNEXE 5, DIFFÉRENTES VUES POUR UNE MÊME IDÉE

Le MCD de la Gestion des salles a été réalisé suite à une opération de reverse engineering avec AMC Designer.



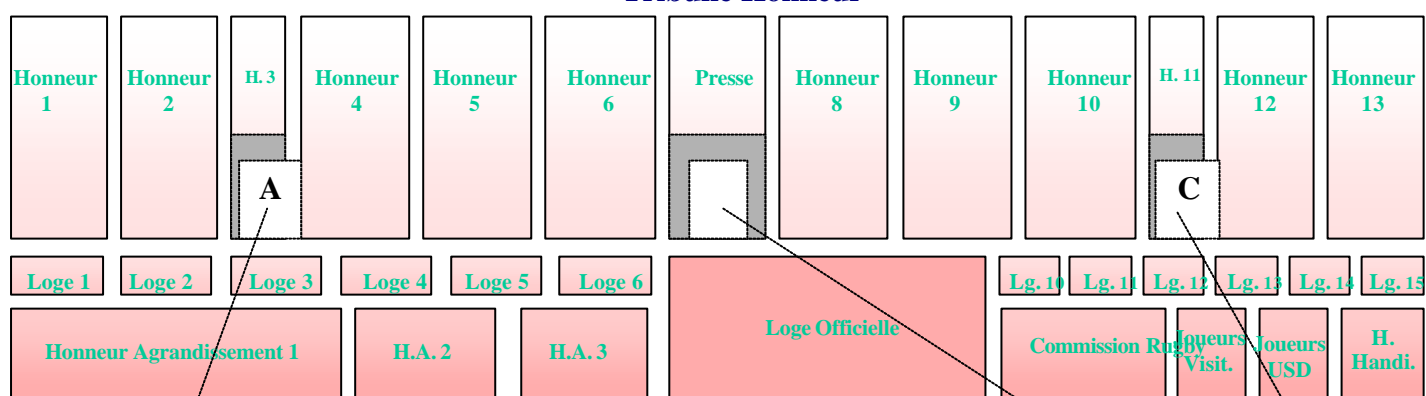
Modèle conceptuel de données		
Projet : BILLETTERIE		
Modèle : MCD Salles		
Auteur : D.R.	Version: 1.0	09/11/100

Les principales règles de gestion relatives aux salles sont les suivantes :

- Les différents libellés (libelle, libelle_billet, libelle_reduit) sont utilisés pour l'édition des cartes d'abonnement et des billets. Il faut donc les renseigner correctement.
- Une place est utilisable si son champ exploitable vaut 1.
- Un module non numéroté permet son affichage sans matrice.
- L'ordre du rang définit la manière d'afficher le rang pour le module.
- Pour chaque module, on définit sa position géographique.
- La place logique concerne le numéro de la place par rapport au module.
- La place physique est le numéro inscrit sur le siège dans les tribunes.
- Le champ image de l'entité salle correspond à la représentation numérique de la salle.

Le client, pour sa part, visualise le découpage de la salle (en l'occurrence le stade Maurice BOYAU de l'U.S. Dax Rugby Landes) sur un simple schéma :

Tribune Honneur



Porte A

Porte B

Porte C

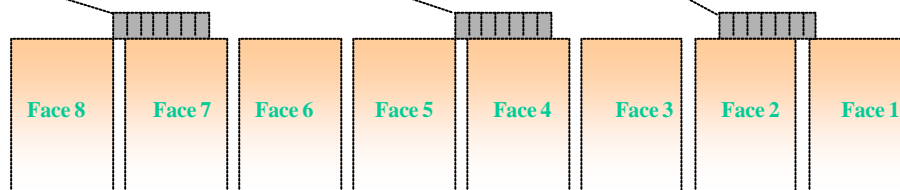
Autres modules :

- Laisser Passer – 400 places
- Debout – 15000 places

Escalier 3

Escalier 2

Escalier 1

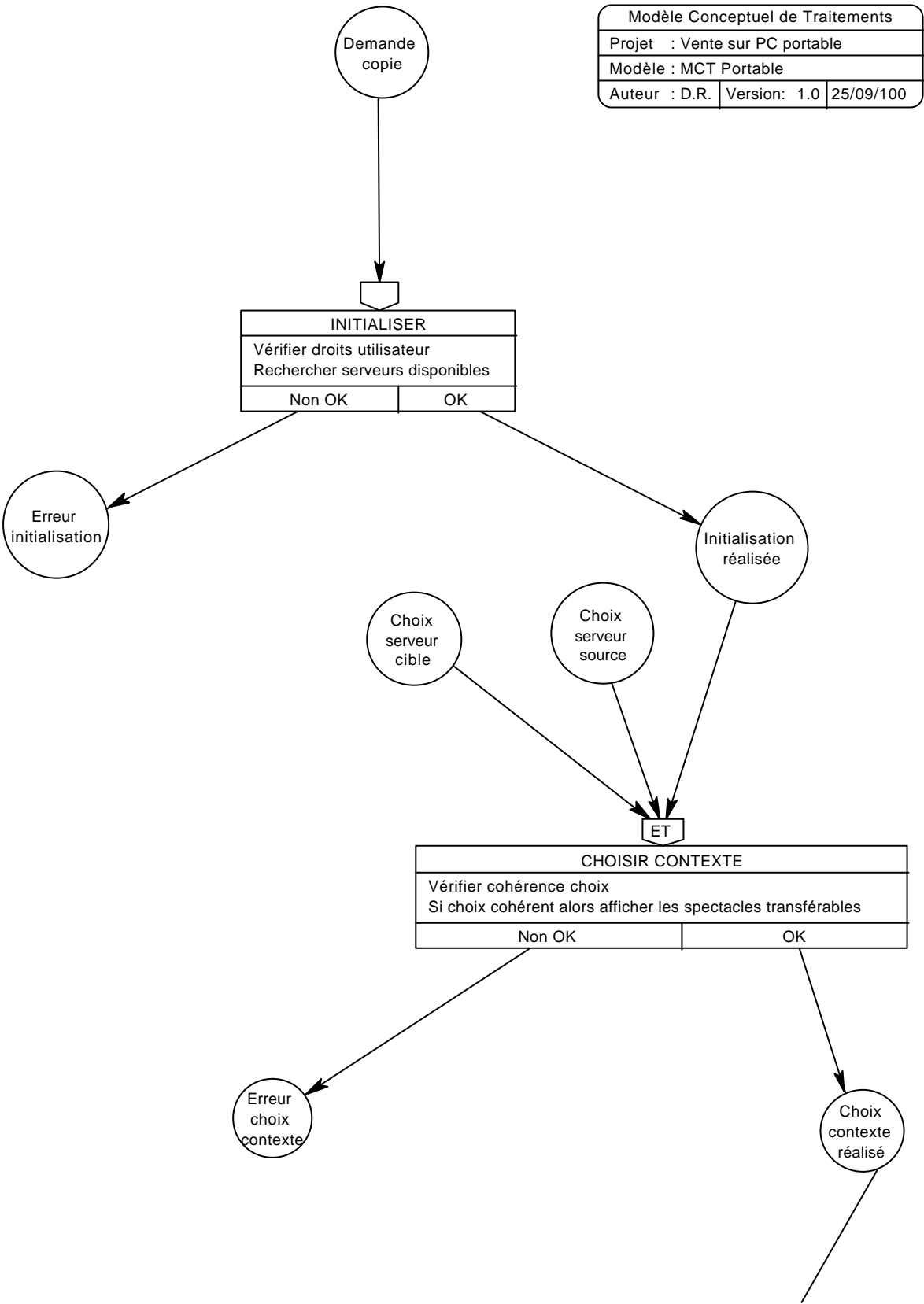


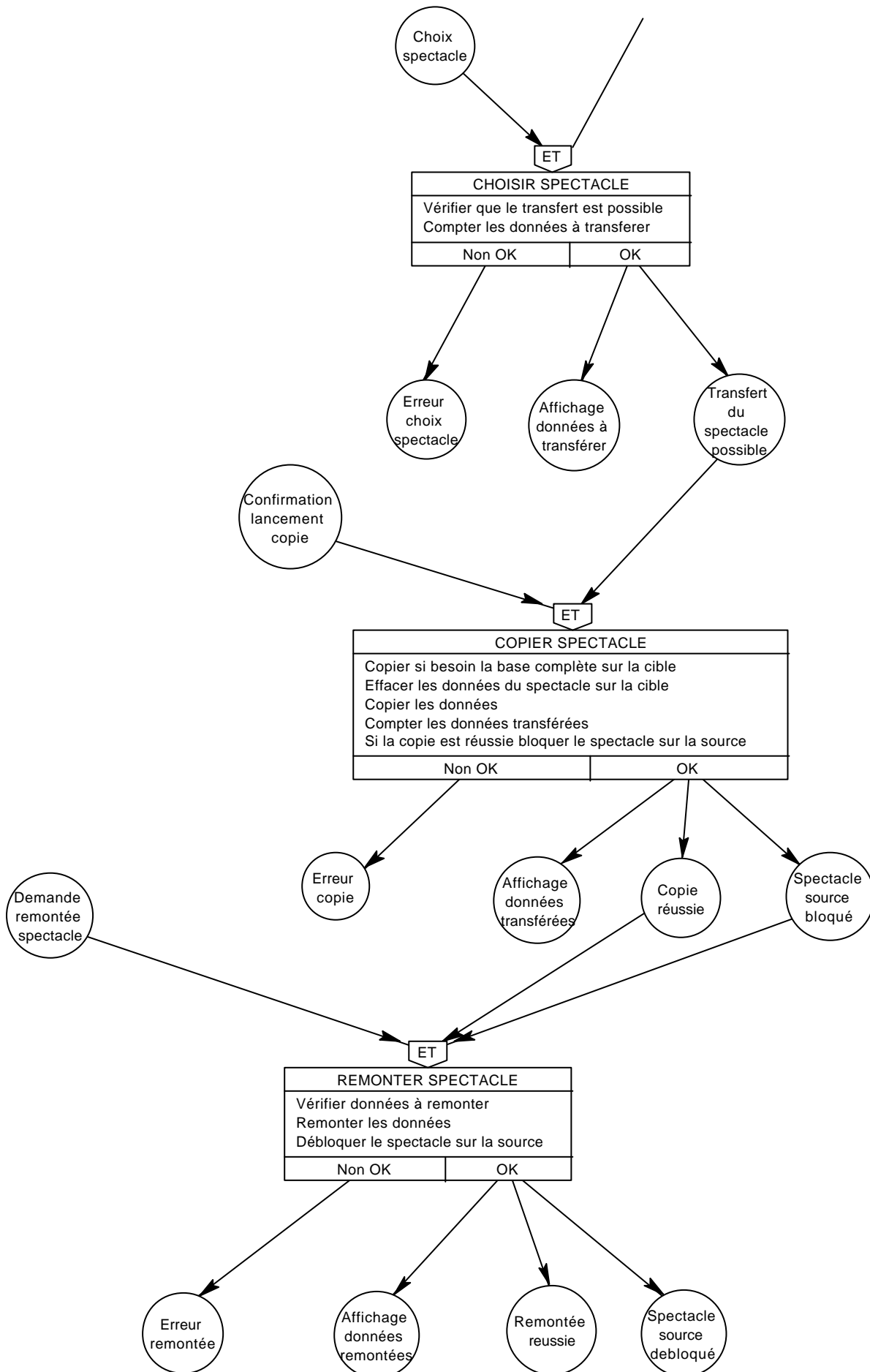
Tribune de Face

Remarque : pour les besoins de la présentation, le schéma a été amputé de ses parties descriptives, qui, pour chaque module, explicitent la numérotation des places, les accès possibles à ces places, ...

VI - ANNEXE 6, MCT VENTE SUR PORTABLE

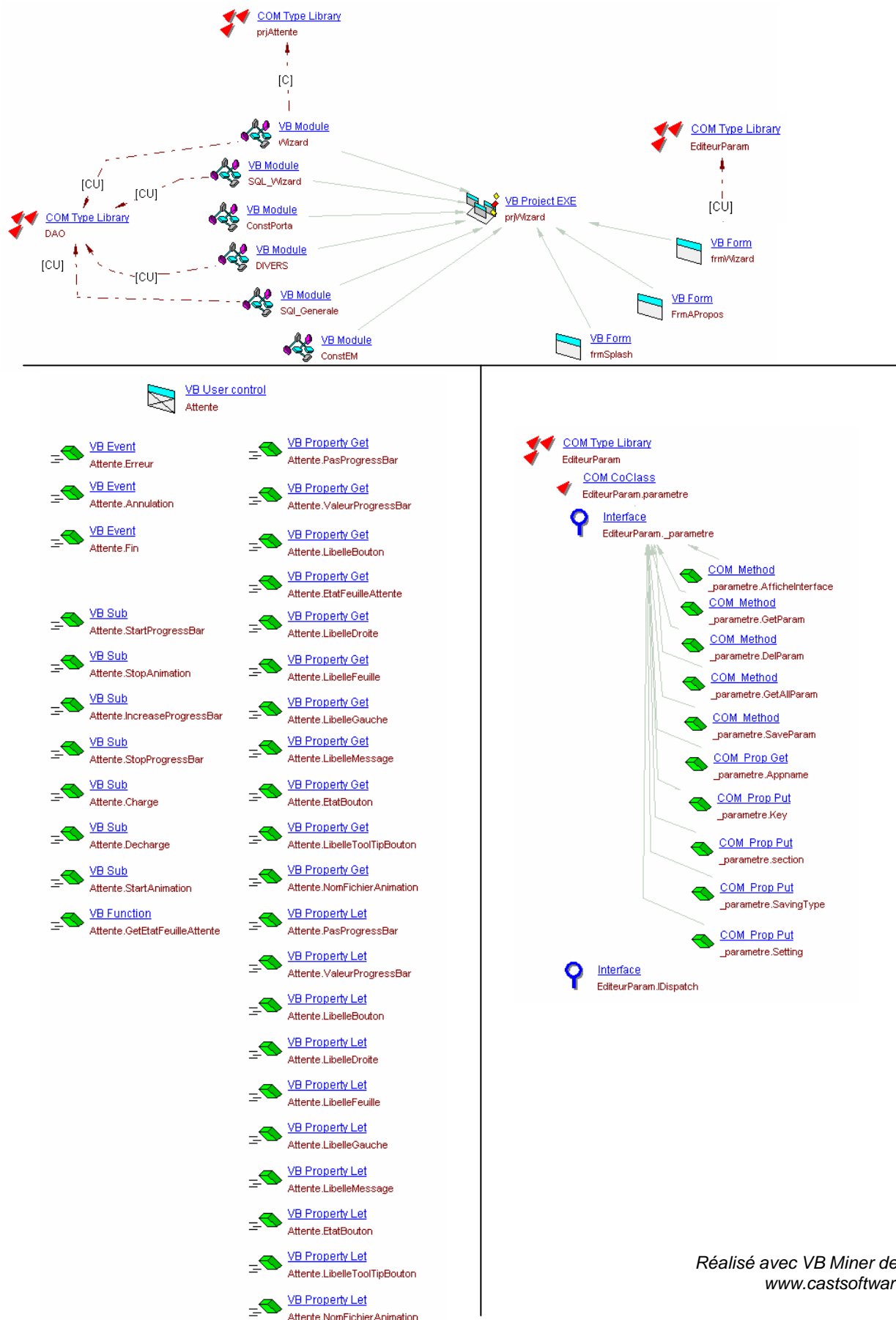
Le MCT du projet de vente sur un ordinateur portable a été réalisé avec AMC Designor. Le même logiciel de modélisation a servi à créer les MCD et MOT du projet qui ne sont pas présentés ici.





VII - ANNEXE 7, SCHÉMAS VB MINER

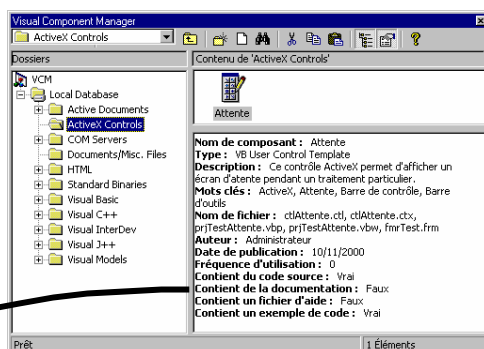
VB Miner de la société Cast permet d'analyser et de suivre la structure d'un projet Visual Basic.



Réalisé avec VB Miner de Cast,
www.castsoftware.com

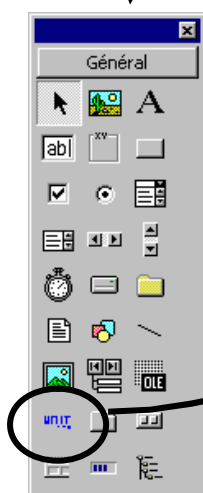
VIII - ANNEXE 8, CONTRÔLE ACTIVEX POUR LE PROJET DE VENTE SUR PORTABLE

Les illustrations présentées permettent de voir comment le contrôle ActiveX Attente.ocx que j'ai créé s'exploite sous Visual Basic 6.0.

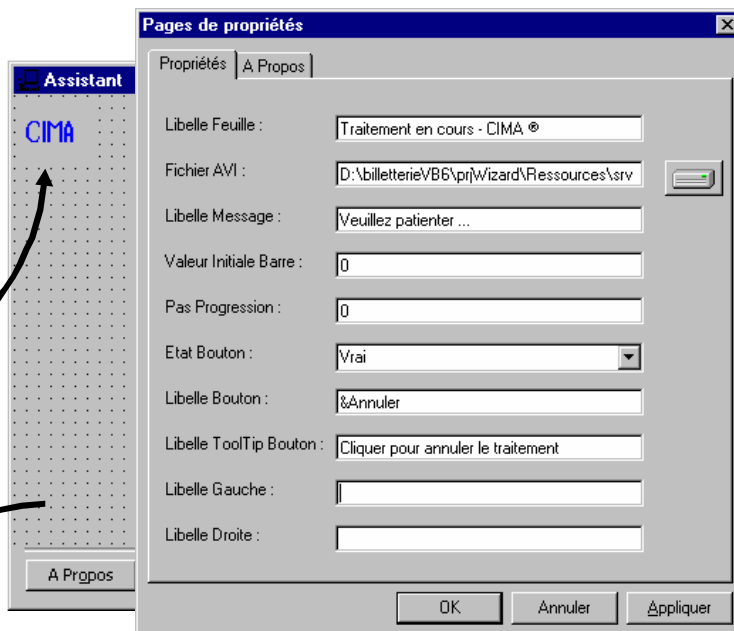


Le Visual Component Manager permet d'explorer les composants disponibles

Le composant s'intègre à la barre d'outils de l'environnement de développement

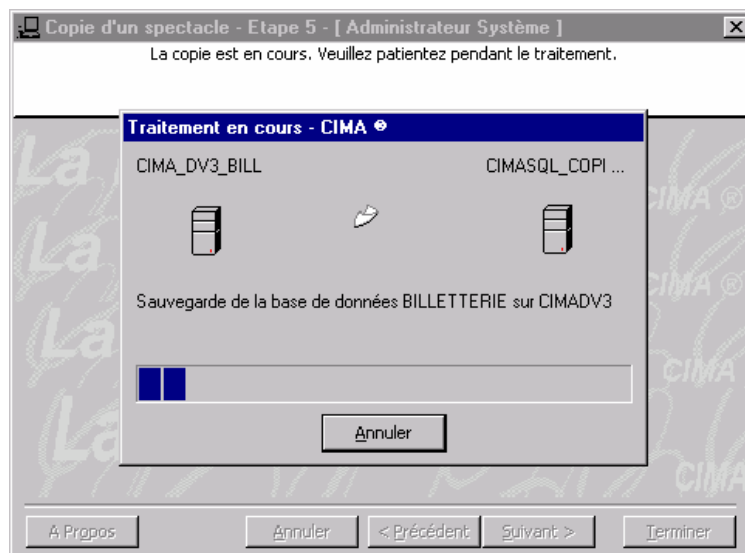


D'un simple « glisser-déposer », on insère le contrôle ActiveX sur une des feuilles du projet dans lequel on veut l'utiliser.



Le paramétrage des propriétés du contrôle est simplifié grâce aux pages de propriétés.

Les efforts pour obtenir une interface riche et fonctionnelle sont considérablement réduits



IX - ANNEXE 9, SUIVI QUOTIDIEN DU STAGE

Le suivi quotidien du stage est présenté jusqu'au 24/11/2000.

Date	Descriptif	Activité
19 juin 2000	Formation sur VB5.0 avec le Mastering MS VB5.0 (chap 1 à 6)	Formation COM+, Visual Basic
"	Création des fichiers notesVB5.doc et com+.doc	
"	Création d'un projet de test pour l'appel d'Excel depuis VB (Automation)	
"	Création d'un projet de test pour l'appel d'objets contenus dans une DLL	
20 juin 2000	Formation sur VB5.0 avec le Mastering MS VB5.0 (chap 7 à 10)	
"	Mise à jour des fichiers notesVB5.doc et com+.doc	
21 juin 2000	Formation sur VB5.0 avec le Mastering MS VB5.0 (chap 10 à 12)	
"	Rédaction finale du fichier notesVB5.doc, continuation de com+.doc	
22 juin 2000	Formation sur SQL Server avec le MacMillan	
"	Formation sur l'accès à SQL Server depuis VB	
23 juin 2000	Formation sur SQL Server avec le MacMillan	
"	Formation sur Crystal Reports	
"	Formation sur VB avec le livre de B. COYDON	
26 juin 2000	Formation sur VB6.0 avec le Mastering MS VB6.0 (ts les chap)	
"	Lecture informations pour la migration de SQL Server 6.5 à 7.0	Formation SQL Server et migration 6.5 vers 7.0
27 juin 2000	Rédaction d'un plan de migration préalable entre SQL Server 6.5 et 7.0	
"	Lecture bugs connus lors la migration de SQL Server 6.5 à 7.0	
28 juin 2000	Rédaction d'un plan de migration préalable entre SQL Server 6.5 et 7.0	
"	Rédaction rapport pour l'utilisation de SQL Mail	
"	Lecture de l'aide en ligne de SQL Server 7.0	
30 juin 2000	Rédaction d'un plan de migration détaillée, bugs et étapes pour la migration entre SQL Server 6.5 et 7.0	
3 juillet 2000	Lecture Mise en œuvre de bases de données SQL Server 7.0 (préparation à la certification 70-029)	
4 juillet 2000	Lecture Mise en œuvre de bases de données SQL Server 7.0 (préparation à la certification 70-029)	
"	Manuel utilisateur ADAPEI 64	ADAPEI 64 (Client)
5 juillet 2000	Manuel utilisateur ADAPEI 64	
"	Lecture Au cœur de COM+	
6 juillet 2000	Manuel utilisateur ADAPEI 64	Forma.
"	Rédaction d'un rapport sur COM+	
7 juillet 2000	Manuel utilisateur ADAPEI 64	ADAPEI 64 (Client)
"	Rédaction d'un rapport sur COM+	
10 juillet 2000	Rédaction d'un rapport sur COM+	
"	Manuel utilisateur ADAPEI 64	
11 juillet 2000	Rédaction d'un rapport sur COM+	Forma.
12 juillet 2000	Analyse de l'existant dans le logiciel de BILLETTERIE	Analyse BILLETTERIE
13 juillet 2000	Analyse de l'existant dans le logiciel de BILLETTERIE	
14 à 21 juillet 2000	Réalisation de la migration effective de 6.5 à 7.0	Passage de la BILLETTERIE de SQL Server 6.5 à 7.0, VB5.0 à 6.0, Crystal Reports 5.0 à 8.0
24 juillet 2000	Tests et corrections sur la base BILLETTERIE pour faire fonctionner les programmes clients sur SQL Server 7.0	
25 juillet 2000	Tests et corrections sur la base BILLETTERIE pour faire fonctionner les programmes clients sur SQL Server 7.0	
26 juillet 2000	Extraction puis recompilation des exécutables de la BILLETTERIE sous VB 6.0	
27 juillet 2000	Création de fichiers de dépendance pour les .EXE de la BILLETTERIE en VB 6.0 et génération d'un nouveau setup	
"	Test du setup et des exe créés en VB 6.0 sur CIMASQL et SQL Server 7.0	
28 juillet 2000	Analyse de l'existant dans le logiciel de BILLETTERIE, partie données	BILL.

Date	Descriptif	Activité
31 juillet 2000	Installation de VB6.0 et Crystal Reports 8.0 sur CIMASQL puis tests de la BILLETTERIE sur cette configuration	BILL.
1 août 2000	Modifications des sources de la BILLETTERIE pour corriger les bugs sur la configuration VB6.0, SQL Server 7.0 et Crystal Reports 8.0	Passage de la BILLETTERIE de SQL Server 6.5 à 7.0, VB5.0 à 6.0, CRW 5.0 à 8.0
2 août 2000	Passage de tous les .rpt en version CRW 8.0	
3 août 1977	Etude des procédures stockées sp_CMBILANJOUR_PV et sp_CMBILANJOUR_Prix qui posent problème sur SQL Server 7.0	
4 août 2000	Création du script de récupération de la table billetterie..module_rang	Assistance technique (Client) Elan béarnais
"	Etude de Anomalies.exe à cause de problème d'exécution à l'EBPO	
7 août 2000	Modification du script de récupération de la table billetterie..module_rang avec Céline par téléphone	
"	Analyse de l'existant dans le logiciel de BILLETTERIE, partie données	US Dax
8 août 2000	Installation de NT4.0 Serveur, SQL Server7.0 sur DAXSRV	
"	Etude de l'impression dans le panneau de contrôle	BILL.
9 août 2000	Etude du découpage en modules du stade de l'US DAX, puis création du bitmap du stade et enfin intégration du stade dans le soft de BILLETTERIE	BILLETTERIE E US Dax
10 août 2000	"	
"	Résolution définitive de la licence du contrôle Sheridan 3D	BILL.
16 août 2000 au 18 août 2000	Récupération des abonnés de l'US Dax et affectation des places (Fichiers texte vers SQL Server 7.0 via DTS)	BILLETTERIE US Dax
21 août 2000	Appel support technique STR pour résoudre le problème de l'affichage des .rpt, finalement résolution seul du problème	BILLETTERIE
22 août 2000	Code barre (2 parmi 5 entrelacé) pour la BILLETTERIE (Section Paloise, US Dax, Pau-Orthez)	
23 août 2000	Préparation du démarrage des abonnements de l'US Dax (Abonnes et Stade)	BILLETTERIE US Dax
24 août 2000	"	
25 août 2000	"	
28 août 2000	"	
29 août 2000	Préparation du démarrage des abonnements de l'US Dax (cartes d'abonnement)	
30 août 2000	"	BILL.
31 août 2000	"	
1 septembre 2000	Rédaction du suivi de la BILLETTERIE pour l'US Dax	-
4 septembre 2000 au 8 septembre 2000	Semaine de regroupement à l'Université	
11 septembre 2000	Analyse et Conception Projet de Vente sur PC portable pour la BILLETTERIE	
"	Résolution de problèmes sur la base de l'US Dax (visualisation de réservations)	BILLETTERIE
12 septembre 2000	Analyse et Conception Projet de Vente sur PC portable pour la BILLETTERIE	
13 septembre 2000	Conception des cartes d'abonnement de la Section Paloise	BILLETTERIE
14 septembre 2000	Création des cartes d'abonnement de la Section Paloise	
15 septembre 2000	Tests des cartes d'abonnement de la Section Paloise	

Date	Descriptif	Activité
18 septembre 2000	Récupération des abonnés de la Section Paloise (Excel vers SQL Server 6.5)	BILLETTERIE
19 septembre 2000	Analyse et Conception Projet de Vente sur PC portable pour la BILLETTERIE	BILLETTERIE
"	Assistant téléphonique pour l'US Dax	US Dax
20 septembre 2000	Conception Projet de Vente sur PC portable pour la BILLETTERIE	BILLETTERIE
"	Conception des cartes d'abonnement (loge officielle) de la Section Paloise	BILLETTERIE
21 septembre 2000	Préparation billets pour l'US Dax	US Dax
22 septembre 2000	Préparation des termods pour la Section	Section
25 septembre 2000	Conception Projet de Vente sur PC portable pour la BILLETTERIE	BILLETTERIE
26 septembre 2000	"	BILLETTERIE
27 septembre 2000	Assistance Section Paloise	Section
28 septembre 2000	Assistance US Dax	US Dax
29 septembre 2000	"	US Dax
2 octobre 2000 au 6 octobre 2000	Vente sur PC portable - interface graphique (prototypes, icônes, images, ...)	BILLETTERIE
9 octobre 2000 au 11 octobre 2000	Vente sur PC portable - création d'une DLL de gestion de paramètres	BILLETTERIE
12 octobre 2000	Vente sur PC portable - développement de la partie traitement Source vers Cible	BILLETTERIE
13 octobre 2000	Vente sur PC portable - développement de la partie traitement Source vers Cible	BILLETTERIE
16 octobre 2000 au 19 octobre 2000	Vente sur PC portable - développement de la partie traitement Source vers Cible	BILLETTERIE
17 octobre 2000	Vente sur PC portable - test d'intégration de la DLL	BILLETTERIE
18 octobre 2000	Vente sur PC portable - test d'intégration de la DLL	BILLETTERIE
23 octobre 2000 au 27 octobre 2000	Modification du programme de gestion des termods	BILLETTERIE
4 octobre 2000 au 6 octobre 2000	Vente sur PC portable - Création d'un composant ActiveX affichant un écran d'attente paramétrable	BILLETTERIE
30 octobre 2000 et 1 novembre 2000	Vente sur PC portable - tests fonctionnels et de performance de la partie Copie	BILLETTERIE
2 novembre 2000	Rapport de stage	-
3 novembre 2000	"	-
6 novembre 2000 au 10 novembre 2000	Vente sur PC portable, tests fonctionnels, d'intégration et de performances	BILLETTERIE

Date	Descriptif	Activité
13 novembre 2000	Préparation de la démonstration de la BILLETTERIE pour la mairie de Mont de Marsan	BILLETTERIE
14 novembre 2000	Rapport de stage - Visite de Patrick DEMEURISSE	
15 novembre 2000	Modification du programme de gestion des termods	BILLETTERIE
16 novembre 2000	Tests du programme de gestion des termods	
17 novembre 2000	"	
20 novembre 2000	Assistance US Dax, maintenance de la base de données (passage des abonnés de Pelouse vers Debout, libération des places non confirmées)	BILLETTERIE
21 novembre 2000	"	
22 novembre 2000	Modification de l'administration du contrôle d'accès pour y intégrer la visualisation graphique	BILLETTERIE
23 novembre 2000	Préparation de l'installation des exécutables pour le CABBG	
24 novembre 2000	Vente sur PC portable, correction suite aux résultats des tests	
...		

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