Edité le lundi 22 décembre 2003.

COM+ - SOMMAIRE

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 prés 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.

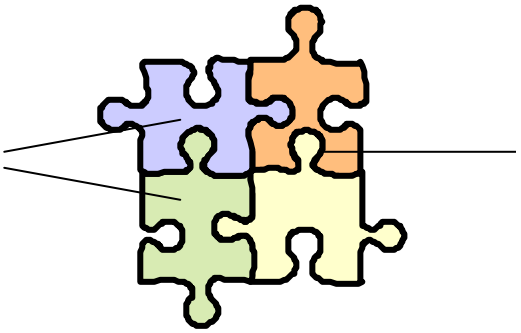
2. Définition

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 prés à l'emploi et le fabriquant 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.

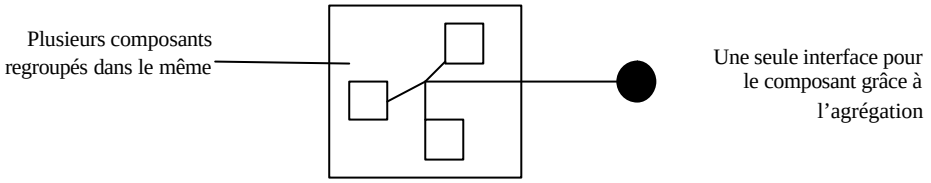
D'une part, il faut que les composants puissent se lier entre eux dynamiquement, c'est à dire au moment de l'exécution.

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 :

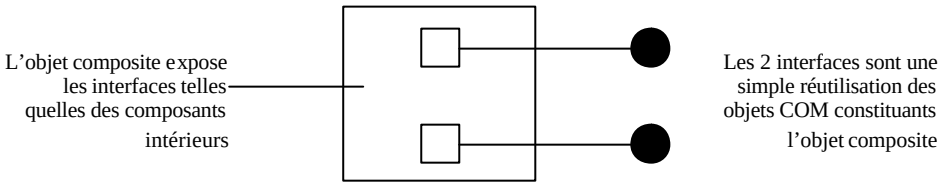
- le langage utilisé pour le codage doit être caché et ne pas influer sur l'interface
- le composant est livré sous forme binaire (et non pas sous forme de code source). Cela permet une réutilisabilité binaire (entre composants, et non une réutilisabilité au niveau du code source)
- un changement de 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 :

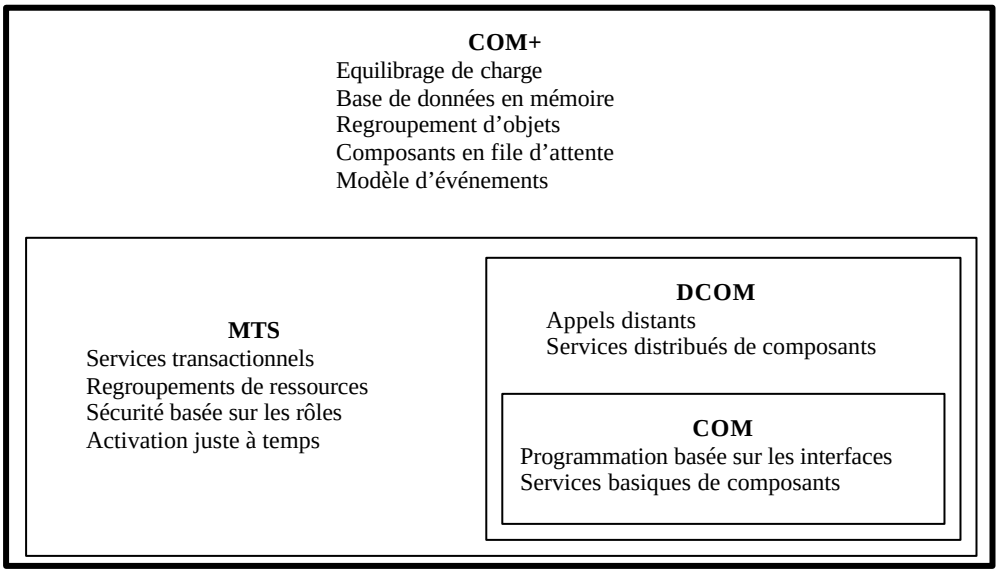


- la délégation consiste en la réutilisation d'une interface d'un objet par un autre objet :



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 polling): 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+ repartit 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 programmative).
- 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.

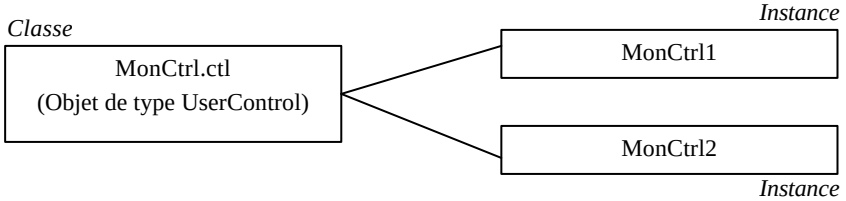
III - LES COMPOSANTS

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 s ont créés.

1. Contrôle ActiveX

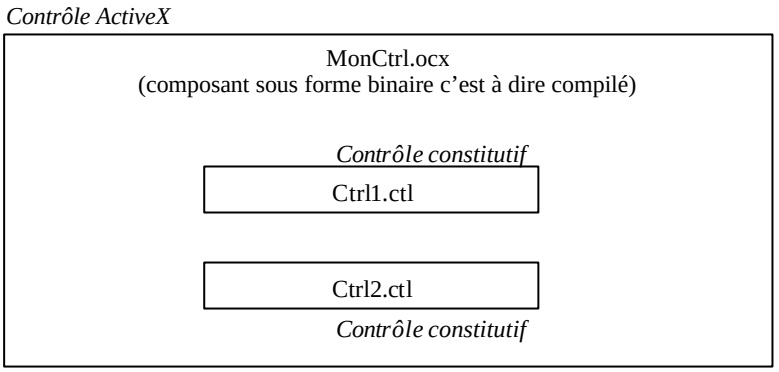
Les contrôles ActiveX (d'extension .ocx pour OLE Control Extension) sont des éléments logiciels indépendants et réutilisables qui incluent des éléments visuels et du code. La boîte à outils de Visual Basic renferme l'ensemble des contrôles intégrés; elle facilite et accélère ainsi considérablement le 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).

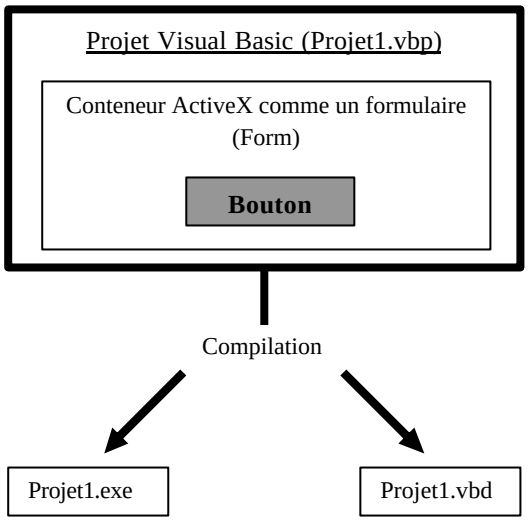
Le principal aspect fonctionnel des documents ActiveX est leur effet sur l'interface utilisateur. Un document ActiveX remplit la zone d'affichage d'un conteneur et donne ainsi à ce dernier la possibilité de se comporter comme l'application serveur à travers une interface utilisateur semblable.

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

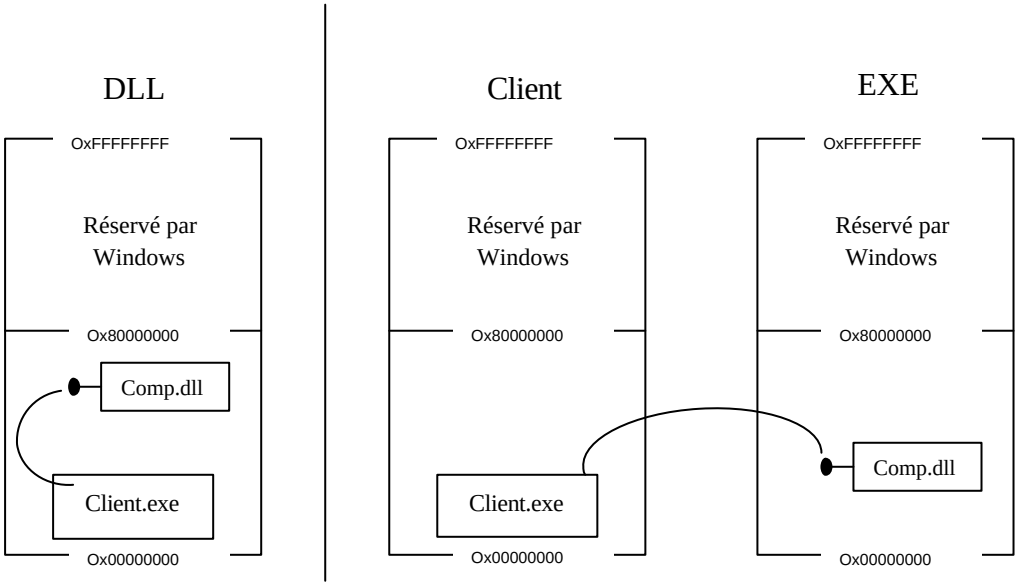
L'illustration suivante montre les fichiers créés par la compilation d'un projet EXE Document ActiveX.



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.

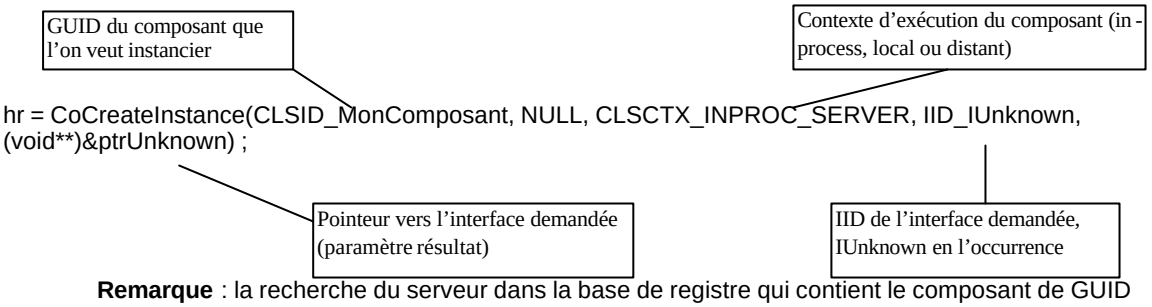
La fonction **AddRef** incrémente le compteur d'utilisations de l'objet lors de l'affectation d'un pointeur d'interface. La fonction **Release** décrémente ce compteur lorsqu'une variable pointant sur l'objet sort de la portée.

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).

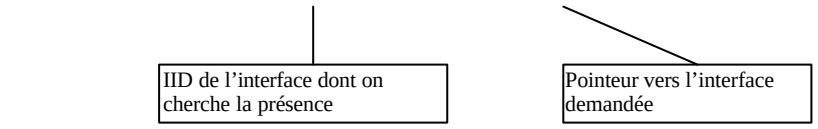
Exemple : l'appel d'un composant COM+ depuis un client en C++ se décompose selon les étapes suivantes.

- appel de CoInitializeEx : hr = CoInitializeEx (NULL, COINIT_APARTMENTTRHEADED) ;
  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 :

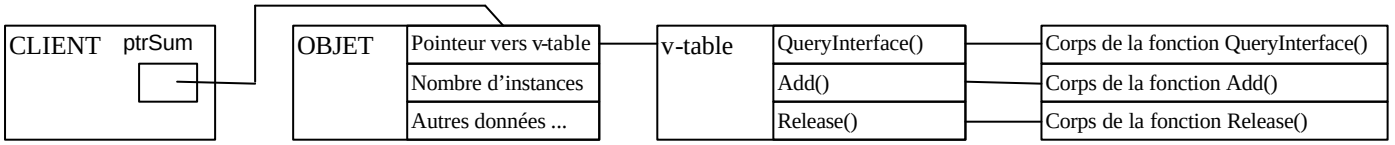


Remarque : la recherche du serveur dans la base de registre qui contient le composant de GUID CLSID_MonComposant se fait en fait via l'interface IClassFactory présentée ultérieurement.

- 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 CoUnInintalize().

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 GetObjet().

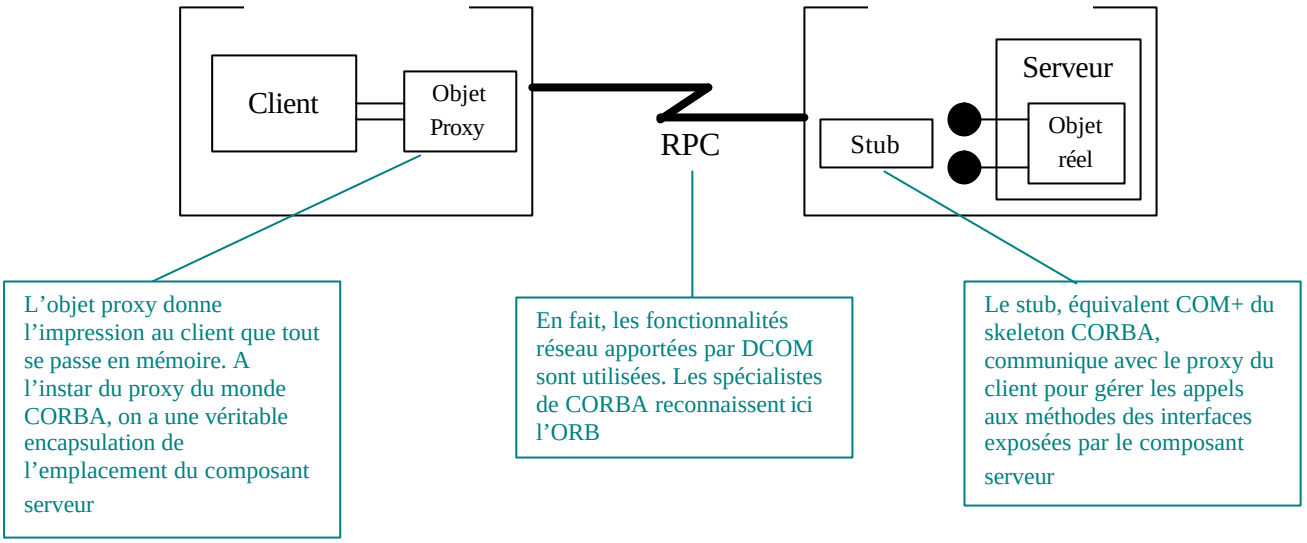
V – LA COMMUNICATION ENTRE COMPOSANTS

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 coté 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

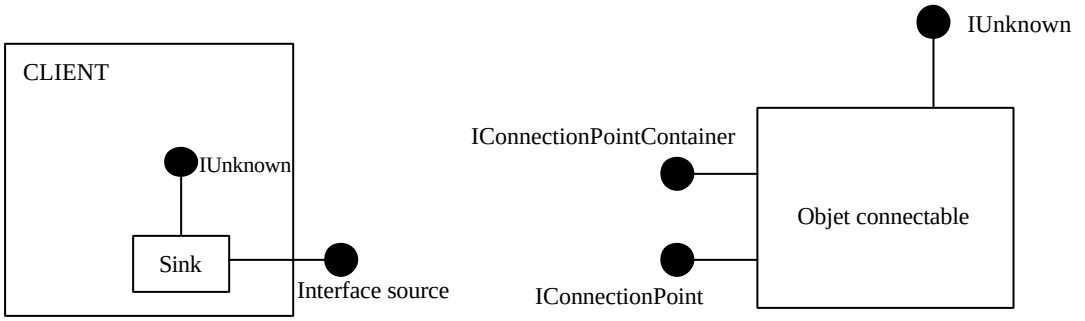
Les bibliothèques de type sont une description dynamique de chaque objet (propriétés et méthodes). En d'autres termes, une bibliothèque de type est une version binaire d'un fichier IDL : elle constitue une description binaire des interfaces exposées par le composant.

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 appel 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 « unknwn.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 évènementielles
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.

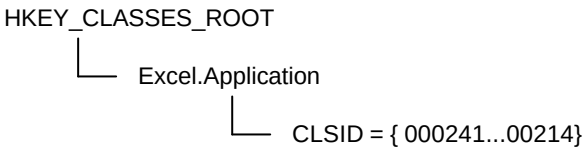
```
Dim Excel As Excel.Application

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

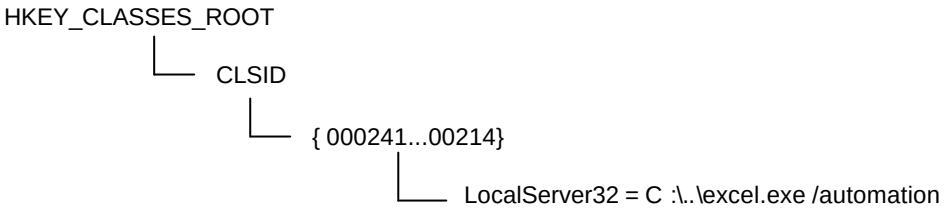
Après cette instruction, voici les détails des actions réalisées par COM+ pour récupérer la référence demandée.

- 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'envirronement 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-Déposser 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 repartir 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