COM

&

ActiveX

NB : ce fascicule fait partie d'un travail de diplôme sur le sujet « Technologie ActiveX & Visual Basic 6 ».
Les autres fascicules peuvent être demandés à fcomte@caramail.com.

Critiques, remarques, questions ? fcomte@caramail.com

1 - Introduction

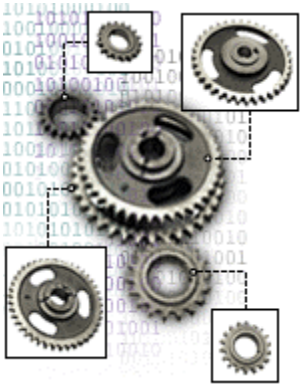
1.1 Introduction aux composants

Au fil du temps, les paradigmes de programmation et la façon de concevoir des applications a progressé. Peut-on imaginer à l'heure actuelle la réalisation entière d'une application développée en code machine ? C'est pourtant ainsi que la programmation a débuté, il y a une trentaines d'années. Puis vinrent des langages plus évolués, destinés à des tâches spécifiques (FORTRAN, COBOL). Et ensuite les centaines de langages différents que nous connaissons aujourd'hui.
Vers la fin des années 80, une première ébauche de librairies pré-compilées vit le jour, notamment avec les produits Quick Basic de Microsoft et Turbo Pascal de Borland. Dès lors, des tierces compagnies se mirent à satisfaire un nouveau marché en vendant des librairies d'algorithmes.

En 1991, Microsoft Visual Basic changea profondément le monde de la programmation, avec son outil de développement Visual Basic. Comme il était construit sur Microsoft Windows, on pouvait appeler des DLL (Dynamic Link Librairies) - un « module » préfigurant les composants - ou des VBX (Visual Basic Extensions), nantis de fonctionnalités pré-compilées, et, pour la première fois, utiliser ces blocs directement dans l'environnement de développement, de façon simple. Ce nouveau paradigme permettait alors de développer des applications en utilisant aussi bien du code écrit au clavier que des blocs qui s'intégraient parfaitement comme des briques de LEGO dans le projet. Telle une ville construite entièrement en LEGO, les applications pouvaient alors être entièrement construites avec des composants logiciels.

1.2 Qu'est-ce qu'un composant ?

Selon le Petit Robert :
Composant : Elément qui rentre dans la composition de quelque chose, qui remplit une fonction particulière. Voir Ingrédient.



Cette définition convient parfaitement – comme nous l'avons vu plus haut – aux composants logiciels, car ceux-ci sont réellement les ingrédients d'une application. Une définition plus technique d'un composant logiciel serait « tout bloc de code définissant des interfaces qui peuvent être appelées pour fournir les fonctionnalités encapsulées au sein du composant ». Généralement, ces composants sont « emballés » dans des normes industrielles standardisées, afin qu'ils puissent être appelés indépendamment du langage, voire de la plate-forme utilisée.
Ces composants sont crées selon le modèle COM (Microsoft Component Object Model), JavaBeans ou Entreprise JavaBeans, VCL (Borland Delphi), ainsi que selon plusieurs autres architectures moins connues. Les composants COM et Java étant de loin les plus utilisés sur le marché.

Le CBD (Component-Based Development) est l'acronyme désignant le paradigme de développement par composants. Le développeur, durant la phase de conception et de spécification, utilise aussi bien ses propres composants (ou ceux de son entreprise) que des composants du marché libre (gratuits, disponibles en majeur partie depuis Internet). Par la suite, il ne lui reste (dans le meilleur des cas !) qu'à coder les composants dont il a besoin, ainsi que la « colle » permettant d'imbriquer les composants entre eux, afin de réaliser son application. Une des idées de ce paradigme étant la réutilisabilité, le développeur peut ensuite déposer son ou ses composant(s) à un endroit où d'autres (que ce soient ses collègues ou Monsieur Tout-le-monde) pourront le(s) charger. On arrive alors à une baisse des coûts de développements non négligeables. Par ailleurs, plus les composants seront utilisés dans des projets de nature différente, plus ils seront testés, et donc gagneront en fiabilité.

Dès le début des années 90, les composants ont commencé à être vendus dans le but d'être utilisés comme des modules « plug'n play », et à l'heure actuelle, on compte plusieurs milliers de ces composants sur le marché.

Services Web

Les services web sont une extension de l'environnement des composants distribués dans laquelle le réseau local est substitué par Internet. Au lieu d'être situé sur un serveur connu du réseau, les composants se trouvent quelque part sur le web. La plupart des acteurs informatiques ont proposé depuis juin 2000 leur solution relative aux services web, notamment Microsoft avec .NET, Sun avec Sun ONE, HP avec les eServices, Oracle avec eSpeak et IBM avec Websphere.

1.3 Le composant et son marché

« Ce ne sont pas les espèces les plus fortes qui survivent, ni les plus intelligentes, mais celles qui réagissent le mieux aux changements ». Charles R. Darwin (1809 – 1882)

La « nouvelle économie » d'aujourd'hui, dans laquelle Internet crée entre autre une compétition globale sur les marchés IT (Information Technology : technologies de l'information), force les entreprises présentes sur ces secteurs à pouvoir s'adapter facilement aux nouvelles tendances, et ce, à moindre coût. Pour rester compétitifs et pour gérer efficacement les évolutions et les changements dans les procédés de fabrication, ces compagnies doivent passer du développement d'applications lourdes et monolithiques à des architectures basées (notamment, mais pas seulement !) sur le développement par composants, où la réutilisabilité et la flexibilité sont deux buts importants d'une stratégie globale de développement d'applications.

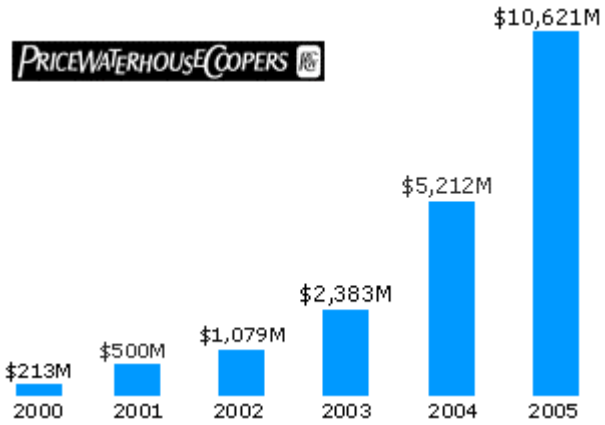
« Nos recherches montrent que la réutilisation de code diminue le nombre de défauts dans ce même code de 25 %; les composants réutilisables permettent de déployer les produits avec un gain de temps de 40 %. » (META Group – Reuse Productivity par Donn DiNunno, septembre 2000).

Le marché des serveurs d'applications est également influencé par le développement par composants : à l'heure actuelle, bon nombre de plates-formes supportent ce type de développement, et comptent déployer la plupart de leurs services en utilisant cette technologie. Le concept des services Web notamment, et plus généralement celui du logiciel comme un ensemble de services, permettra dans un avenir très proche d'utiliser les composants logiciels comme des services pouvant être appelés depuis Internet. Dès lors, que ce soient les stratégies de Microsoft, Sun ou IBM, l'importance des composants logiciels dans le développement d'applications est clairement défini.

« Les composants logiciels jouent un rôle majeur dans les plates-formes eBusiness. Les revenus globaux dans ce marché vont augmenter à un taux d'intérêt composé annuel de 40 %, de $516 millions en 1999 à $2,7 milliards en 2004. »
(IDC, The Software Construction Components Market, juin 2000).

« En 2003, 70 % des nouvelles applications seront déployées avec une combinaison de composants neufs et pré-assemblés, intégrés pour former des systèmes complexes. »
(Gartner Group, Component-Based Development : The Next Wave, avril 1999).

Une recherche de PriceWaterhouseCoopers quantifia en 1999 la valeur moyenne du marché des composants logiciels au cours du temps : le marché dans ce secteur devrait valoir plus d'un milliard de US$ en 2002.
(Etude Franchee Harmon, PwC, 1999).



1.4 L'évolution des composants

Les développeurs, depuis qu'ils ont commencé à concevoir des applications logicielles complexes, regardaient toujours avec envie les autres industries d'équipement, comme la fabrication de matériel informatique, l'automobile ou la construction industrielle et se demandaient "pourquoi le logiciel est donc si difficile ?". Ils ont dès lors largement emprunté les idées et les principes de ces industries et ont essayé de les étendre au logiciel. C'est ainsi que l'application des principes techniques à la conception de logiciels a été au cœur de l'évolution de la technologie des composants.

Les défis fondamentaux

Il y a plusieurs défis fondamentaux dans le développement de logiciels que la technologie des composants permet de résoudre. Nous nous arrêterons sur ces défis à un niveau général d'abord et progresserons ensuite par des étapes diverses dans l'évolution de la technologie des composants afin de mieux la comprendre.

Ces défis fondamentaux peuvent être généralisés en 3 points :

1 réutilisation de logiciel
2 gestion des changements
3 isolation de la technologie.

1 Réutilisation de logiciel

Le but de la réutilisation de logiciel est d'être capable de démolir un problème en morceaux afin de ne pas avoir à réinventer la roue lors de chaque nouveau développement. Des problèmes spécifiques peuvent être décomposés en un certain nombre de sous-problèmes qui sont arrivés et ont été résolus auparavant. Vous pouvez résoudre votre problème spécifique plus rapidement en réutilisant des solutions existantes de ces sous-problèmes. Le truc est donc de démolir votre problème dans des sous-problèmes pour lesquels les solutions existent. Par exemple, si votre problème est de voyager d'une ville à une autre, vous le résolvez en termes d'options disponibles de transport (la voiture, l'autobus, le train, l'avion, etc.). Vous pouvez être capable de résoudre une grande proportion du problème en employant une solution "fixée" (comme prendre un avion) et ajouter ensuite les aspects spécifiques de votre problème en employant une solution plus "personnalisée", comme prendre un taxi. Il serait inutile de fonder votre propre ligne aérienne ou une société de taxi, ou inventer des nouveaux moyens de transport, à moins que le problème ne puisse être résolu qu'ainsi.

2 Gestion des changements

Résoudre un problème rapidement est bien sûr agréable, mais une fois que vous avez votre solution spécifique, vous allez devoir certainement la corriger (sous entendu la maintenir). Les exigences changeront peut-être, ou il peut y avoir des problèmes avec la solution actuelle, ou encore des améliorations devront être apportées. Pendant le cycle de vie d'une application logicielle, la plupart du temps et de l'effort est dépensé dans la maintenance.

L'idée clé ici des changements tient dans le changement localisé - la minimisation de sa portée et de son impact afin que vous ne propagiez d'autres changements dans la solution complète. Par exemple, si votre vol est annulé vous devrez changer vos réservations d'hôtels. Ceci parce que ces parties de la solution ont une dépendance. La minimisation de dépendances entre les éléments de votre solution est la clé dans la gestion des changements.

3 Isolation de la technologie

Les applications, autant que possible, doivent être isolées de la technologie sous-jacente pour que d'une part les solutions mises en place soient affectées de manière minimale par le choix de cette technologie (et n'importe quels changements de celle-ci), et d'autre part l'exigence de devoir comprendre la technologie mise en œuvre soit réduite au minimum. Ainsi, par exemple, la location d'une voiture - une solution d'une partie de votre problème de voyage - ne doit pas être touchée par le genre ou le modèle de la voiture.

Principes d'objet et gestion des dépendances

Le logiciel, à son niveau le plus atomique, est composé de fonctions et de données. Il est probable que l'idée la plus importante (et plus malheureuse) dans l'évolution des technologies logicielles fut la séparation et la spécialisation de ces deux éléments. Cela a mené à des efforts de conceptions indépendantes sur les fonctions et les données, provoquant une variété de technologies de base de données et structurant ainsi diverses approches de programmation. Nous avons essayé de les remettre ensemble depuis.

Un des principes de l'ingénierie du logiciel pendant longtemps a été "la haute cohésion, le couplage faible". Que signifie-t-il ? L'idée est que les éléments logiciels qui ont un haut degré de dépendance mutuelle doivent être groupés ensemble dans des blocs "à haute cohésion[1]", avec aussi peu de dépendances que possible entre ces blocs - "le couplage[2] faible". La gestion des dépendances est importante parce qu'elle garantit la capacité de démolir un problème en pièces, de travailler sur celles-ci, de réunir ces pièces à nouveau et de gérer les changements.

Les techniques de conception orientées données ont appliqué ces principes pour fournir des conceptions de base de données avec une bonne gestion des dépendances entre les données. Les approches de conception structurées ont appliqué ces principes pour fournir des conceptions orientées fonctions avec une bonne gestion des dépendances entre fonctions. Malheureusement, les dépendances entre les fonctions et les données ont été délaissées, menant ainsi à un très haut couplage entre fonctions et données. Souvent, un changement effectué dans la partie de conception des données menait à un impact potentiellement très important sur les fonctions. Et ce n'était pas en plaçant une interface agréable (une API) entre les fonctions et les données qu'on arrivait à réduire la granularité de cette dépendance.

L'orientation objet (OO) a offert une approche de conception différente. Elle est d'abord apparue avec des langages de programmation orientés objet (OOPLs, Object-Oriented Programming Languages), notamment Simula 67, avant que n'apparaisse la programmation structurée. Cependant, les principes OO passèrent une décennie relativement calme. Avec l'arrivée de diverses versions de Smalltalk pendant les années 70 et au début des années 80 - avec Smalltalk-80 – les concepts OO commencèrent à intéresser de plus en plus de développeurs. Au même moment, la programmation structurée et les systèmes de gestion de bases de données avaient pris de l'avance et étaient déjà bien établis.

Les principes de langages objets ont dès lors offert un nouveau moyen de penser la structuration du logiciel. Différents langages avaient des particularités spécifiques, mais ils partageaient un certain nombre de caractéristiques clés ou "principes objet" qui supportent la gestion des dépendances :

1. Unification des données et des fonctions

L'orienté objet combine les fonctions avec les données dans un concept logiciel appelé "objet". Beaucoup de langages créent de tels objets en employant un patron d'objet appelé "classe". Cette approche diffère radicalement de celle de la programmation structurée. Elle applique "la haute cohésion et le couplage bas".

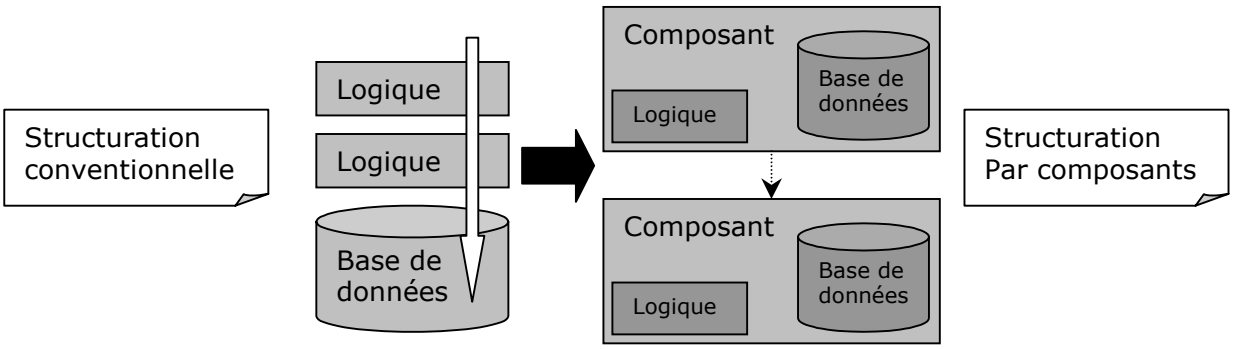
[1] La cohésion peut être vue comme une mesure définissant à quel point les éléments internes d'un module de code sont étroitement liés ou reliés entre eux.
[2] Le couplage peut être vu comme une mesure de la force d'interconnexion entre modules.

2. Encapsulation

Le logiciel qui appelle les fonctions d'un objet logiciel est isolé par rapport à la manière dont ces fonctions sont mises en oeuvre et dont les données qu'il emploie sont structurées et stockées. C'est un développement très important dans la gestion des dépendances - la spécification externe d'une fonction est séparée de l'implémentation de celle-ci, et le logiciel l'appelant dépend seulement de la spécification. Cela signifie que si la spécification reste la même, l'implémentation peut changer sans affecter le logiciel appelant.

3. Identité

Chaque objet logiciel peut être référencé de manière unique, indépendamment de son état.



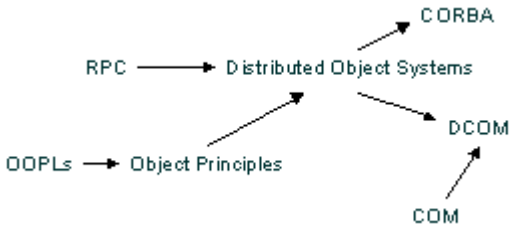
Structuration conventionnelle vs structuration par composants (OO)

Certains langages avaient d'autres particularités aussi, comme l'héritage et le polymorphisme, mais ces caractéristiques ont eu tendance à être spécifiques à des langages plutôt qu'à d'autres, donc nous les traiterons comme des caractéristiques de mise en oeuvre plutôt que des principes objet. Comme nous le verrons, l'évolution des composants se fonde sur ces principes d'objet, et modifie significativement certaines des autres caractéristiques OO.

Des langages objet aux systèmes d'objets distribués

Une fois que les concepts d'objet furent établis, une nouvelle demande importante a commencé à entrer en vigueur - le besoin de permettre aux applications logicielles de s'exécuter sur des plates-formes différentes et de communiquer les unes avec les autres. C'était le début de la notion de systèmes distribués. Les systèmes construits orientés objet comme Smalltalk pouvaient contenir des objets encapsulés bien structurés, interagissant les uns avec les autres, mais ils restaient des systèmes monolithiques, s'exécutant dans un espace d'adresses unique sur une unique machine. Comment cette fonctionnalité pouvait-elle être réalisée de manière plus indépendante du processus et de l'architecture du matériel sur laquelle elle s'exécutait ?

Au milieu des années 80, l'OSF (Open Software Foundation) a développé un mécanisme pour des appels de procédure éloignés (RPC, Remote Procedure Call) appelé DCE (Distributed Computing Environment) et a introduit pour la première fois le concept d'un mécanisme neutre de langage pour définir et révéler les fonctionnalités de codes distribués. Ce mécanisme était l'IDL (Interface Definition Language) et fut à l'origine des variantes des IDL du modèle COM de Microsoft ainsi que de CORBA (Common Object Request Broker Architecture) de l'OMG (Object Management Group).



Des principes objets aux systèmes distribués

Tandis que CORBA était une initiative de l'OMG avec pour but explicite de créer un système d'objets distribués, COM à l'origine est apparu pour des raisons différentes, que nous discuterons plus tard.

L'idée d'un IDL est de définir, de façon neutre, à un langage de décrire les fonctions ou les opérations d'un objet. Le système d'objets distribués fournit un mécanisme pour prendre un appel de fonction et ses paramètres dans un langage donné, les empaqueter dans un flux, les injecter le long d'un canal, les déballer dans leurs équivalents dans un autre langage et appeler l'objet désiré de l'autre côté. Ainsi, par exemple, un objet Smalltalk peut faire un appel à un objet C++.

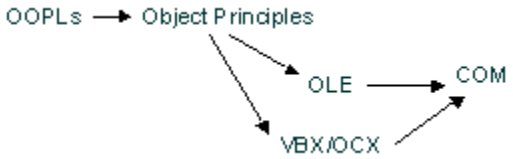
Cette approche applique des principes d'objet – les objets parlent seulement par leurs interfaces et sont connus par des références uniques. Mais il n'y a aucune exigence pour l'objet à être implémenté dans un langage orienté objet - le système d'objets distribués a seulement besoin d'un mécanisme pour transformer une opération décrite en IDL dans un langage d'appel spécifique. Ainsi, non seulement un objet C++ peut-il communiquer avec un objet Smalltalk, mais un objet C peut aussi le faire avec un objet COBOL.

L'introduction de systèmes d'objets distribués était donc un événement marquant important dans l'évolution des technologies logicielles. Dès lors, tant qu'un morceau de code contenant des fonctions et des données pouvait être traité comme un objet de l'extérieur, alors c'était un objet. Il n'importait plus de savoir comment il était implémenté.

Ce changement a au commencement causé bon nombre de critiques de la part des communautés OO. Beaucoup de personnes estimaient qu'il n'était pas possible de créer un objet sans passer par un langage de programmation orienté objet. Mais Microsoft, particulièrement, montra dans sa première documentation sur COM comment on pouvait écrire des objets COM en C. Et cela fonctionnait.

Tandis que COM et CORBA ont adopté une architecture semblable pour la communication d'objets distribués, ils ont utilisé des approches différentes quant au problème de configuration entre les langages employant un IDL neutre. Sous COM, l'approche était d'adopter une norme binaire pour la disposition d'interfaces en mémoire. Ainsi bien que COM ait fourni la neutralité du langage, le système était toujours dépendant de la plate-forme. CORBA, de son côté, était indépendant tant du point de vue du langage que de la plate-forme.

Parce que COM est apparu pour des raisons différentes que la création d'un système d'objets distribués, il a au commencement fourni seulement la distribution "locale" - c'est-à-dire la distribution entre des processus différents sur la même machine. Ainsi, peu après avoir été présenté, COM a été étendu avec la capacité de distribution à travers des machines et DCOM (Distributed COM) a été créé. Le terme COM a continué à être employé pour s'appliquer au modèle composant lui-même, tandis que DCOM est devenu la plomberie du système d'objets distribués. Nous parlerons plus en détails de l'histoire des composants selon Microsoft dans le chapitre suivant.



L'émergence de COM

Internet et Java

Si les développements des systèmes distribués du début des années 90 permettaient d'exécuter des applications distribuées sur un réseau local (LAN), l'apparition de l'Internet étendit massivement ce concept de distribution, le concept du client web apparu, et avec lui les serveurs Web.

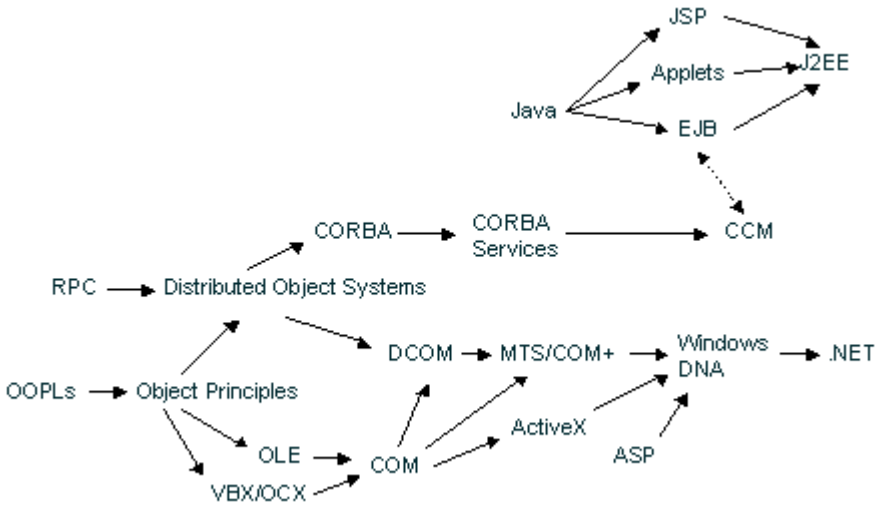
Ces changements menèrent à un certain nombre d'événements, dont peut-être le plus important fut l'apparition de Java de Sun Microsystems. Java était une bête étrange, nouvelle. C'était un langage de programmation objet avec une syntaxe à la C++, mais aussi avec le concept Smalltalk d'une machine virtuelle. Cela signifiait que le code Java pouvait s'exécuter n'importe où, du moins en théorie. Au commencement, Java a été vu comme un langage parfait pour écrire des petites applications, téléchargeables, appelées des applets, s'exécutant sur ces nouveaux clients web, à l'intérieur des pages HTML. Mais bien que Java possédait des particularités semblables au composant, comme des interfaces, c'était toujours un langage et pas un modèle de composant. Les JavaBeans ont été alors créés pour fournir les standards complémentaires nécessaires pour permettre à Java d'être employé comme un environnement de développement de composant.

En parallèle, Microsoft développait ActiveX comme une technologie composante alternative pour le client, mais toujours basée sur COM, pour que le même modèle composant sous-jacent soit répandu dans l'architecture de Microsoft pour Internet.

Sun comprit que l'indépendance de plate-forme de Java le rendait particulièrement applicable comme technologie d'objets distribués et Java se « transforma » d'un langage de programmation à une plate-forme de développement d'application (connu sous le nom de Java 2), ajoutant l'invocation de méthodes éloignées (RMI, Remote Method Invocation) en tant que mécanisme RPC, JDBC (Java DataBase Connectivity) et une liste en croissance d'autres services apparentés à ceux fournis par CORBA et DCOM.

Nous n'irons pas plus loin dans cette petite introduction des composants, la suite du rapport revenant en détails dans les percées technologiques de Microsoft et de Sun.

Le concept entier de ce qu'une application est, change donc. Il est parti de la notion d'un système de logiciel monolithique à l'assemblage de composants distribués. L'approche actuelle des services Web prévoit un monde où les applications sont des entités dynamiques, entrelaçant des services depuis le Web.



L'évolution des composants

2 – Microsoft et les composants : historique

Comme nous allons le voir, une bonne partie de la technologie de COM provient d'anciennes technologies de Microsoft, telles que DDE, VBX, ou OLE. Nous allons entrer un peu plus dans les détails de ces concepts afin de mieux comprendre l'architecture de COM.

2.1 DDE

Au début, il avait le presse-papiers (1987). Cet outil permettait aux utilisateurs de copier des portions de documents d'un programme à un autre. Cela fonctionnait bien pour des documents simples, mais lorsque ceux-ci devenaient plus complexes, le presse-papiers atteignait ses limites. Plus tard, Microsoft développa la technologie DDE (Dynamic Data Exchange), qui permettait aux logiciels de simplifier le partage de données en utilisant un presse-papiers « nouvelle génération ». Cette solution n'était de loin pas universelle, car elle ne fonctionnait que pour les applications de Microsoft.

2.2 OLE 1

En 1991, Microsoft développa une solution plus générale au problème du partage de données appelée OLE (Object Linking and Embedding). Cette technologie permettait aux utilisateurs de combiner des données provenant de plusieurs applications sur un seul document. La figure 2.1 montre un document Word dans lequel a été inséré une feuille de calcul Excel. Cette feuille a été insérée depuis l'article Objet du menu Insérer. Pour éditer cette feuille, il suffit de double-cliquer dessus, et d'effectuer les modifications en utilisant les outils d'Excel, qui s'insèrent dans l'environnement de travail de Word. Ainsi, le document Word fonctionne comme un conteneur pour la feuille de calcul d'Excel, ce qui constitue la fondation de la technologie de OLE.

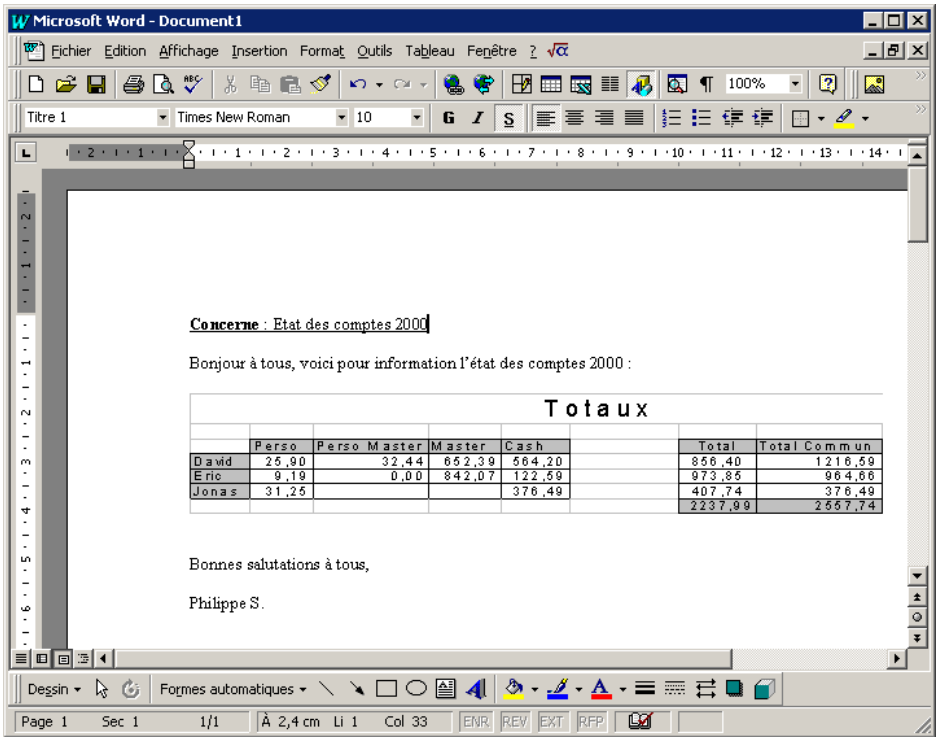


Figure 2.1 – Une feuille Excel est insérée dans un document Word

La figure 2.2 montre comment se comporte Word lorsque l'utilisateur ouvre la feuille de calcul en double-cliquant dessus : les barres d'outils d'Excel sont chargées dans Word et l'édition de la feuille s'opère ainsi directement depuis Word.

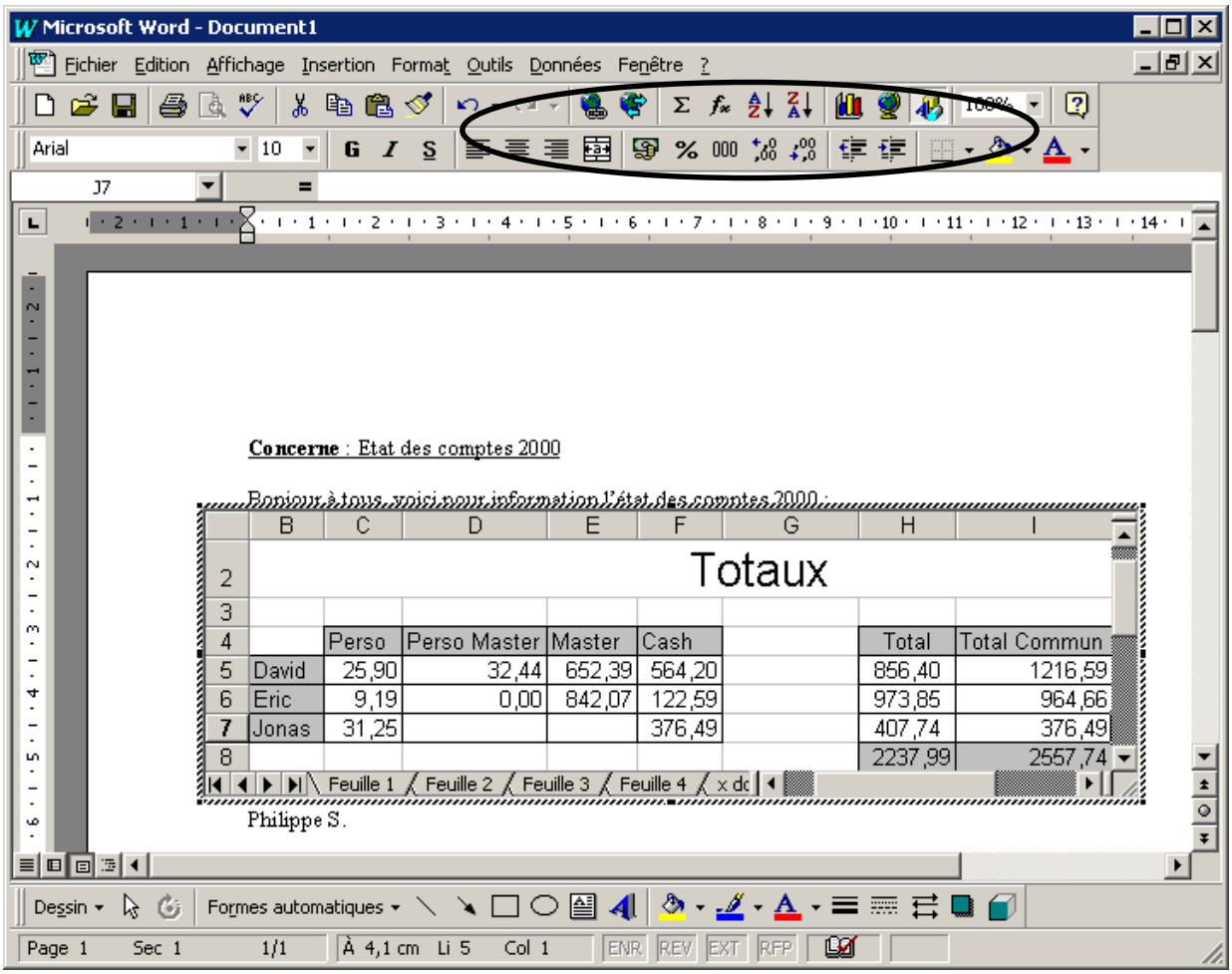


Figure 2.2 – Le double-clic sur la feuille Excel permet l'édition de cette dernière avec les outils d'Excel chargés dans Word.

La première implémentation de OLE, appelée aujourd'hui OLE 1, laissait encore à désirer. En effet, les liaisons inter-applications étaient très lentes, et l'implémentation souffrait de plusieurs bugs. Alors que les développeurs de Microsoft planchaient sur une nouvelle version d'OLE, une autre équipe s'occupait de revoir leur langage de programmation phare : Basic. Le but de ce développement était de transformer l'environnement de programmation Visual Basic en un RAD (Rapid Application Development), permettant aux développeurs de construire rapidement leur application en utilisant des blocs de fonctions couramment utilisées, sans avoir à les coder eux-mêmes. C'est de cette manière que Visual Basic, dès sa version 3, devint un véritable outil RAD.

2.3 VBX

Visual Basic inclus dès lors un système de création de modules de haut niveau appelés contrôles. En glissant un contrôle sur une feuille du projet, on avait alors accès à une fonction complexe. Prenons par exemple le contrôle `TextBox`, qui permet d'afficher une ou plusieurs lignes de texte, de séparer un mot en deux lorsqu'il n'y a plus de place pour l'afficher complètement sur une ligne, d'afficher des ascenseurs lorsque cela est nécessaire, etc.
Ce contrôle a été fourni par Microsoft dès les premières versions de Visual Basic, et il permettait de simplifier la vie des développeurs. Néanmoins, tous les contrôles fournis par Microsoft ne permettaient pas de résoudre tous les problèmes. Ainsi naquit VBX (Visual Basic eXtensions), qui autorisaient les programmeurs à créer leur propres contrôles. Ces contrôles interagissaient avec Visual Basic de trois manières (qui n'ont pas changé, le lecteur pourra donc lire le passage suivant au présent) :

- → Les propriétés permettaient de lire ou d'écrire des valeurs dans le contrôle. Les propriétés communes déterminaient si le contrôle était actif (`Enabled`), si il était visible sur la feuille (`Visible`), ainsi que sa position relative sur la feuille (`Left` et `Top`), et sa taille (`Height` et `Width`).

- → Les méthodes étaient utilisées pour appeler des routines ou des fonctions appartenant au contrôle. Par exemple, une des méthodes typiques permettait de repositionner le contrôle sur la feuille (`Move`).

- → Les événements permettaient au contrôle, lorsque celui-ci rencontrait une condition spécifique, d'appeler une routine de l'application dans laquelle il était contenu. Par exemple, l'événement `Click` était déclenché lorsque l'utilisateur cliquait sur le contrôle, ou encore l'événement `KeyPressed` réagissait lors de l'activation d'une touche du clavier.

Microsoft fourni dès lors une quantité de contrôle avec Visual Basic 3. Par exemple, cette version incluait le contrôle `Data`, qui permettait d'utiliser un accès simple à une base de données, le contrôle `MAPI` (Mail API), qui permettait d'utiliser l'API gestionnaire de courrier de Microsoft, ainsi que d'autres contrôles permettant d'accéder aux fonctions de Windows comme le contrôle `CommonDialog`.

Mais plus important, un grand nombre de compagnies se mirent à créer des contrôles très utiles, afin de travailler sur des graphiques d'entreprises, de faciliter des rapports, etc. Ceci était très pratique étant donné que l'architecture de VBX autorisait la distribution du contrôle sous forme binaire. Les développeurs pouvaient donc garder le code source privé, tout en distribuant leurs contrôles. Ce concept d'extension des fonctions du langage de programmation par ajout de contrôles sous forme binaire fut alors incorporé dans OLE 2.

2.4 OLE 2

Pendant le développement d'OLE 2, les ingénieurs de Microsoft réalisèrent qu'une des grandes difficultés dans la création de documents composites provenait de la façon dont les composants communiquaient entre eux. Avant OLE 2, plusieurs techniques étaient utilisées : quelquefois, il s'agissait uniquement d'appeler des routines. Mais il fallait également dans d'autres circonstances utiliser des systèmes de communication beaucoup plus complexes. Les programmeurs devaient donc souvent utiliser plusieurs techniques à la fois pour permettre à leurs composants de communiquer, ce qui causait de grosses difficultés d'intégration dans la mesure où toutes les applications n'utilisaient pas les mêmes technologies. C'est donc pendant le développement d'OLE 2 que Microsoft inclus la technologie COM dans son modèle de composant.

Comme nous l'avons vu, COM crée une couche d'abstraction entre les composants logiciels. Chaque objet COM est conçu pour travailler comme serveur d'une application cliente ou simplement comme module dans une application. L'objet COM est une instance d'une classe spécifique qui définit une ou plusieurs interfaces aux fonctions qu'il fournit, et chaque interface contient une ou plusieurs méthodes permettant de résoudre des fonctions spécifiques. Bien que COM détermine comment deux composants doivent communiquer entre eux, OLE 2 nécessitait d'autres méthodes standards que celles fournies par COM, comme le fait pour un objet de présenter sa propre interface utilisateur, de pouvoir envoyer des événements au conteneur du contrôle, ainsi que de laisser la possibilité au conteneur de modifier les propriétés du contrôle qu'il contient.

OLE 2 a été supporté d'abord par Visual Basic 4, qui était la première version permettant la programmation 32 bits (bien que deux versions étaient fournies avec VB4, une version 16 bits, et une 32 bits). Notons encore que la technologie OLE 2 était aussi connue à l'époque comme celle permettant de développer des contrôles OCX.

Par la suite, comme les principes de bases n'évoluèrent guère (un objet contient des propriétés, des méthodes et des événements), Microsoft pu changer le nom des technologies sans que cela ne modifie grandement les habitudes des programmeurs.

Voyons de façon plus théorique quels étaient les fonctionnalités des serveurs OLE (nous ne les noterons plus par OLE 1 ou OLE 2 par la suite), car – comme nous le verrons par la suite – la technologie ActiveX est fortement liée à OLE et à la création de contrôles d'extensions .ocx.

Les serveurs OLE gèrent un certain nombre de protocoles et fournissent un certain nombre de services :

→ Les objets OLE sont censés fournir une interface pour leurs commandes internes, afin que d'autres objets puissent faire exécuter au serveur objet les opérations spécifiées sur ses données. Par exemple, un objet Excel possède une méthode qui permet à un client externe de lui faire charger une feuille de calcul.

→ Les objets OLE sont censés gérer le glisser-déposer. Si l'objet est une fenêtre, il doit répondre de façon appropriée aux données ou objets qui sont déposés sur lui avec la souris.

→ Les objets OLE gèrent le protocole UDT (Uniform Data Transfer), qui est un mécanisme de gestion de l'échange de structures de données formatées entre applications. Les transferts UDT sont réalisés en envoyant des informations sur des pointeurs plutôt que les données elles-mêmes, afin d'éviter d'avoir à lire en mémoire de grandes quantités de données.

→ Les objets OLE sont censés participer à une architecture répondant à la définition OLE, nommée structure de stockage, et utilisant un service OLE, les fichiers composites, qui constituent un moyen de partager le contenu d'un fichier entre divers composants en utilisant un mécanisme que l'on peut se représenter comme un « système de fichiers dans un système de fichiers ».

Dans le contexte de documents composites, les objets OLE sont censés :

→ S'incorporer eux-mêmes de manière appropriée à l'intérieur d'un document conteneur, et restituer leurs données de manière appropriée sur l'écran ou l'imprimante. Par exemple, un document Word peut contenir un objet incorporé qui soit une feuille de calcul Excel.

→ Maintenir des liens entre objets incorporés (de manière comparable aux liens automatiques de DDE) afin que les données puissent être automatiquement mises à jour.

→ Gérer l'activation et l'édition en situation, en réponse aux actions de l'utilisateur. Cela signifie fournir une interface visuelle d'édition appropriée dans le contexte du conteneur. Par exemple, un objet Excel incorporé dans un document Word peut être édité en utilisant les outils d'Excel.

3 – Concepts fondamentaux de COM

3.1 Introduction

COM est la norme de Microsoft pour le développement d'objets réutilisables. Ces objets sont indépendants de n'importe quel langage de programmation particulier.

Cela signifie que les programmes Visual Basic peuvent facilement accéder à des objets de COM écrits dans Visual C++ ou Visual J++. Cela signifie également que nous pouvons écrire nos propres objets en Visual Basic, qui peuvent être employés par des programmes de Visual C++.

COM signifie Component Object Model. C'est une spécification de Microsoft qui décrit comment créer les objets réutilisables pour des programmeurs travaillant dans un environnement de programmation Win32. COM est également une norme binaire, qui signifie que n'importe quel langage de programmation (avec les équipements appropriés, naturellement) peut créer des objets de COM.

Puisque les objets de COM sont binaires, ils peuvent être contenus dans leurs propres fichiers exécutables. Ceci facilite le développement d'objets qui peuvent être distribués indépendamment d'un programme d'application. Le développeur COM peut donc assembler son application à partir des composants sans avoir à se préoccuper de détails d'implémentation tels que l'emplacement physique des composants, COM étant conçu sur l'idée que les applications et les composants doivent pouvoir évoluer de façon indépendante tout en continuant à fonctionner ensemble.

Aujourd'hui, COM est un standard utilisé de façon courante (et presque transparente) dans le développement d'applications ; Microsoft l'a adopté pour la grande majorité de ses produits. C'est également une technique incontournable pour la construction des applications trois-tiers conformément à Windows DNA.

Les spécifications de base de ce standard ont été définies à partir des contraintes suivantes :

- **Architecture basée sur l'utilisation de composants.**

  Maintenir et améliorer du code dans une grosse application monolithique est extrêmement complexe. Les systèmes basés sur l'assemblage de composants binaires sont plus simples à développer, maintenir, et étendre. Pour construire un tel système, les composants doivent pouvoir être identifiés de façon unique, et doivent offrir une garantie dans la compatibilité afin de pouvoir être modifiés tout en continuant à fonctionner avec les applications existantes.

- **Orienté objet.**

  La réutilisation dans le monde de Windows est fondée sur l'utilisation de DLL traditionnelles, or cette technique ne permet pas d'exporter les définitions de classes qui sont à la base des concepts orientés objets. COM doit définir une nomenclature de réutilisation et d'interaction entre composants à un niveau binaire, tout en exposant les bénéfices liés à la conception basée sur les classes tels que l'encapsulation et le polymorphisme.

- **Indépendance du langage.**

  Chaque langage possède des avantages et inconvénients ; les développeurs doivent être en mesure d'utiliser le langage et l'outil de développement de leur choix pour créer et utiliser les composants. La possibilité de choisir entre critères de productivité, flexibilité et performance pour chaque composant offre des avantages considérables.
  Cette contrainte stratégique est particulièrement difficile à respecter ; en effet tous les détails d'implémentation d'objets – allocation de mémoire, noms de méthodes, types de paramètres, convention d'appels, etc. – doivent être normalisés pour respecter cette contrainte.

- **Communication inter-processus.**

  Il est indispensable de concevoir un système permettant de composer une architecture où les composants clients et serveurs peuvent s'exécuter dans des processus sur la même machine ou sur des machines différentes. Dans le cas idéal, le mécanisme d'interaction entre objets doit supporter l'exécution distante de façon transparente. Les développeurs ne doivent pas avoir à se soucier de l'emplacement physique des objets. Ce concept est à la base des technologies distribuées.

3.2 Terminologie et historique

COM, OLE et ActiveX sont des termes utilisés couramment dans le monde de Windows, souvent de façon inexacte et confuse. Il devient alors difficile d'associer chacun de ces termes avec leur fondement technique. Le paragraphe qui suit tente de mettre un peu d'ordre dans cette confusion.

- COM (Component Object Model) est un standard et une architecture qui régit la réutilisation binaire de composants. Les spécifications de COM sont disponibles sur le site web de Microsoft (http://www.microsoft.com/com/resources/comdocs.asp).

- OLE (Object Linking and Embedding) s'appuie sur COM. C'est le mécanisme utilisé pour construire des documents composites. Par exemple, l'insertion d'un classeur Microsoft Excel dans un document Microsoft Word utilise OLE.

- Automation, également appelé OLE Automation, est un concept bâti au dessus de OLE. Il désigne le pilotage d'une application programmable par un client. Aujourd'hui, le terme Automation est plus souvent utilisé pour désigner les clients de type scripting (qui communiquent avec les objets par l'interface `IDispatch`).

- ActiveX est apparu avec les technologies Internet basées sur COM. Ce label marketing était essentiellement utilisé à cette époque pour identifier ce type de technologie. Ensuite les choses se sont compliquées et tout est tombé sous la coupe de l'appellation ActiveX, ce qui apporte beaucoup de confusion. Aujourd'hui, ce terme est surtout utilisé pour désigner les « contrôles ActiveX », une technologie spécifique aux contrôles programmables.

COM prend racine dans le projet OLE2 lorsque différentes équipes de développement Microsoft ont commencé à concevoir des infrastructures orientés objets basées sur les composants, destinées à fournir un degré important de réutilisation.

Depuis, un grand nombre d'évolutions ont contribué à améliorer ce modèle :

Date	Fonctionnalités
1988	Début de réflexion sur COM - basé sur les travaux réalisés en SmallTalk, C++, et ceux de Open Software Foundation (OSF) Distributed Computing Environment (DCE)
1993	Première sortie publique de COM dans le SDK OLE 2.0 - communications locales - support de thread unique
1994	Windows NT 3.5 - support 32 bits - Interface Definition Language (IDL)
1995	Windows NT 3.51, Windows 95 - performances - support multi-thread
1996	Windows NT 4.0 - Distributed COM (base sur DCE) - amélioration du support multi-thread - amélioration de la sécurité
1996	MTS 1.0 - infrastructure de développement d'applications distribuées basée sur la collaboration de composants
1998	Windows NT 4.0 Option Pack - MTS 2.0 - MSMQ 1.0 - IIS 4.0
1999 - 2000	Windows 2000 - COM+ - MSMQ 2.0 - IIS 5.0

3.3 Interface

COM repose sur un modèle de programmation fondé sur l'utilisation d'interfaces, c'est un style de développement qui favorise la séparation du prototypage de l'implémentation. L'interface étant le point de convergence de tous les concepts définis par COM, il est indispensable de bien comprendre cette notion pour comprendre COM dans son ensemble.

Une interface COM est un ensemble d'opérations qui définissent un comportement. Lorsqu'on déclare une interface, on ne fournit que le prototypage des différentes opérations supportées, pas leur implémentation. Par convention, on préfixe le nom de l'interface par « I ».
Par exemple, une interface `IAnimal` supporte l'opération `Marcher` qui prend en paramètre une valeur correspondant à la distance de déplacement. Plusieurs classes peuvent implémenter cette opération différemment, la seule chose qui importe pour le client est que la classe implémente `IAnimal` et que `IAnimal` supporte la méthode `Marcher`.
Concrètement, l'interface est le seul point de jonction entre l'application cliente et les différentes classes du composant. A ce titre, l'interface représente un contrat immuable

entre le client et les composants COM, car ces derniers ne peuvent communiquer que par le biais des interfaces.
Pour être considérée comme une interface COM valide, cette dernière devra respecter les règles suivantes :

- Une interface doit être signée par un identifiant unique.
- Une interface doit dériver (au moins) d'une interface spéciale nommée `IUnknown`.
- Une fois publiée, une interface est immuable (elle ne doit plus changer).

Afin de permettre le dialogue entre clients et composants COM, et ceci quel que soit le langage d'implémentation, il a été nécessaire de définir un langage commun de description des interfaces - Interface Definition Language (IDL). L'IDL de COM est basé sur l'IDL de l'Open Software Foundation (OSF) Distributed Computing Environment (DCE).
Voici un extrait de l'interface `IAnimal` en code IDL :

```
interface IAnimal : IDispatch
{
   HRESULT Marcher([in] long nDistance);
};
```

Ceci est donc le premier pas vers l'indépendance du langage de programmation, chaque outil ou langage qui supporte COM est capable de construire des interfaces conformes à IDL. C'est également la raison pour laquelle COM est appelé un standard binaire ; il définit la représentation binaire exacte d'une interface.
Visual Basic offre un niveau d'abstraction qui cache les détails relatifs à IDL ; il n'est donc pas indispensable de comprendre IDL pour développer des composants COM à l'aide de cet outil.

En pratique, une interface définie avec Visual Basic est une classe pour laquelle la propriété Instancing est égale à 2 - PublicNotCreatable. Le code qui suit est l'équivalent Visual Basic de l'extrait IDL précédent, l'interface est nommée `IAnimal` dans la fenêtre de propriétés du module de classe à la rubrique Name.

```
Public Sub Marcher(ByVal nDistance As Long)

End Sub
```

Identifiant unique

Conformément aux spécifications de COM, les interfaces sont immuables et uniques. Il est donc indispensable de pouvoir les identifier de façon unique dans l'espace et dans le temps. Un développeur qui conçoit une interface doit pouvoir garantir que l'identifiant utilisé pour son interface n'est pas déjà utilisé, et ne le sera jamais par une autre interface…
Une approche consiste à utiliser un identifiant basé sur une chaîne de caractères, composée de divers paramètres tels que le nom de la société et autres détails. Dans ce cas, il faudrait définir une organisation mondiale habilitée à livrer des noms de sociétés : cette solution est trop complexe.

COM résout ce problème en définissant un « Globally Unique Identifier » (GUID prononcé "gwid"). Le GUID est un identifiant entier codé sur 128-bits (16 octets). L'algorithme utilisé pour le générer garantit statistiquement que le code résultant est unique dans l'espace et le temps. Cet algorithme est tiré d'un standard de l'Open Software Foundation (OSF) Distributed Computing Environment (DCE).
De par sa complexité, le GUID est souvent représenté par une chaîne de 32 caractères formatée comme suit :

```
{99704F63-5BF4-11d3-B574-0000E8E5FC8D}
```

Les interfaces COM sont donc identifiées par un GUID appelé « Interface Identifier » (IID). C'est ce IID qui est utilisé à chaque fois qu'une interface est requise. Afin d'augmenter le confort des développeurs, le IID est substitué par une constante nommée dans un code source (par exemple `IID_IUnknown`). Il est vrai que ces constantes nommées n'ont aucune garantie d'être uniques à travers le monde, mais il est peu probable qu'elles soient dupliquées au sein d'un même code source. Notons que la souplesse de Visual Basic rend le IID invisible au développeur.

Chacun peut créer autant de GUID que nécessaire. Les critères retenus pour l'implémentation de l'algorithme spécifié par OSF DCE le montrent :

- Date et heure système (à 1/10 micro-secondes près).
- Incrémentation forcée de l'identifiant pour les créations de GUID à fréquence élevée.
- Adresse MAC de la carte réseau, réellement unique car déterminé par IEEE (si la machine ne possède pas de carte réseau, un code aléatoire est généré à partir de l'état de la machine – RAM, disque, etc.).

Le GUID est également appelé « Universally Unique Identifier » (UUID) selon la dénomination initiale par OSF DCE.
La majorité des outils (et c'est le cas de Visual Basic) ont la capacité de construire un squelette de composant COM. Ces outils génèrent alors automatiquement un GUID pour les différents éléments du composant lorsque qu'un identifiant est requis.
Il est toutefois possible de générer manuellement un GUID en utilisant un outil du Platform Software Development Kit (SDK) appelé GUIDGEN (ou bien par l'utilisation des APIs du système).

IUnknown

Pour être conforme aux spécifications de COM, chaque interface doit dériver (au moins) de l'interface prédéfinie `IUnknown`. Cette interface décrit le comportement fondamental des interfaces COM. Voici le code IDL correspondant :

```
interface IUnknown
{
   HRESULT QueryInterface(…);
   ULONG AddRef();
   ULONG Release();
}
```

Notons que Visual Basic implémente automatiquement `IUnknown` ; il n'est donc pas nécessaire de maîtriser ce concept pour développer des composants COM avec ce langage.

`IUnknown` fournit deux caractéristiques majeures aux interfaces dérivées : la navigation entre les interfaces d'une même classe et la gestion de la durée de vie de la classe.
La navigation entre interfaces est fournie par la méthode `QueryInterface`. Comme nous l'avons décrit, une classe COM peut supporter plusieurs interfaces. Pour profiter des opérations définies par une des interfaces, le programmeur doit être en mesure d'obtenir une référence sur celle-ci. Il suffit alors de demander cette interface à l'objet, ceci est réalisé par appel à la méthode `QueryInterface`.
`QueryInterface` est appelée automatiquement par Visual Basic au moment de la création d'un objet, ou en référençant une interface particulière par affectation :

```
' Créer un objet
Set objInterfaceDefaut = New MonProjet.MaClasse
```

```
Set objInterfaceDefaut = CreateObject("MonProjet.MaClasse")
' Référencer une interface par affectation
Set objInterfaceA = objInterfaceDefaut
Set objInterfaceB = objInterfaceDefaut
```

La durée de vie d'un objet est implémentée par les méthodes `AddRef` et `Release`. Tous les objets maintiennent un compteur interne de référence ; `AddRef` incrémente ce compteur et `Release` le décrémente: l'objet est détruit lorsque le compteur interne atteint la valeur zéro.
Cette technique permet entre autres à plusieurs clients de se partager l'utilisation d'un même objet. Aucun n'est responsable de la destruction de l'objet, qui est détruit lorsqu'il n'est plus utilisé. Il est également important de noter que `QueryInterface` appelle `AddRef` en interne.
`AddRef` et `Release` sont appelées automatiquement par Visual Basic. `AddRef` est lancée dans le cadre de l'utilisation de `QueryInterface`, et `Release` est appelée lorsqu'une variable objet est détruite (implicitement si elle est allouée sur la pile ou explicitement avec le mot clé `Nothing`) :

```
'Incrémenter le compteur de référence par QueryInterface
Set objInterfaceA = objInterfaceB
'Décrémenter le compteur de référence
Set objInterfaceA = Nothing
```

Un grand nombre de détails sur l'implémentation et l'utilisation de `IUnknown` sont intentionnellement exclus du présent document.

3.4 Classe et objet

L'objet est probablement le terme utilisé de façon la plus abusive. Comme dans la majorité des concepts orientés objets, les objets dans COM représentent une entité concrète ; ce sont des instances de classes. Le terme objet n'est donc significatif qu'à l'exécution ; il possède alors des caractéristiques tels que identité, état et comportement. Au contraire la classe est le modèle inerte sur lequel l'objet est calqué ; elle inclus également l'implémentation de toutes les méthodes et propriétés des interfaces supportées.

Par exemple, la classe `CChien` est un modèle qui ne possède qu'une valeur conceptuelle. L'objet `objBrutus` est une instance de la classe `CChien`, il possède des caractéristiques propres telles qu'un nom, age, etc.…
Concrètement une classe COM est le nom donné à la déclaration et l'implémentation d'un ensemble d'interfaces. Chaque classe est identifiée par un numéro unique appelé « Class Identifier » (CLSID), qui est un GUID. Comme les IID, les CLSID garantissent l'unicité d'un identifiant de classe dans l'espace et le temps.

Dans le but d'augmenter le confort des développeurs, le CLSID est substitué par une chaîne de caractères dans un code source.
Pour les rendre aisément identifiables par l'homme, les classes sont également nommées par une chaîne appelée « Programmatic Identifier » (ProgID) composée comme suit : *<vendor>.<component>.<version>*. En pratique, le ProgID est également de la forme *<component>.<class>.<version>* (la composante *<version>* est également souvent omise).

La déclaration de la classe `CChien` qui implémente `IAnimal` s'effectue comme suit en IDL (le code d'implémentation de la méthode `Marcher` est omis) :

```
coclass CChien :
{
   [default] dispinterface IAnimal;
};
```

Voici la version équivalente en Visual Basic. Le module de classe est nommé `CChien` dans la fenêtre de propriété en modifiant la rubrique Name.

```
Implements IAnimal

Private Sub IAnimal_Marcher(ByVal nDistance As Long)
   MsgBox "Je me déplace de " & nDistance & " mètres..."
End Sub
```

Dans cet exemple, nous supposerons que le projet est nommé ComDemo. Selon les conventions définies par Visual Basic, le ProgID de la classe `CChien` sera donc ComDemo.CChien.

Notons que l'approche COM est légèrement différente du C++ ou d'autres modèles objets. De la même façon qu'une classe C++ peut dériver de multiples autres classes, une classe COM peut implémenter de nombreuses interfaces.
Une bonne compréhension des interfaces est essentielle à la compréhension de COM dans son ensemble.

En reprenant l'exemple du composant ComDemo et de la classe `CChien`, l'objet `objBrutus` est instancié et utilisé en Visual Basic comme suit :

```
Dim objBrutus As ComDemo.IAnimal

Set objBrutus = New ComDemo.CChien
objBrutus.Marcher 10
Set objBrutus = Nothing
```

En pratique, nous nous assurerons que le composant ComDemo est bien présent dans la liste des références du projet Visual Basic. Ceci est fait en sélectionnant le composant requis dans la fenêtre affichée par le menu Projet → Références.
Notons que le nom de la librairie dans la liste des références disponibles n'est pas forcément identique à celui du composant (il correspond à la description du projet pour un composant développé en Visual Basic). Il est également possible de sélectionner directement le fichier (*.DLL, *.TLB, *.OLB) du composant en cliquant sur le bouton Parcourir.

Attention : Visual Basic permet également de créer des composants sans l'utilisation du mot clé `Implements`. Cette pratique, décrite par l'extrait de code qui suit, ne permet pas de mettre en évidence la séparation entre interface et implémentation. En réalité, Visual Basic crée une interface cachée qu'il implémente automatiquement afin de respecter les règles définies par COM. Ainsi, la classe `CChat` implémente en réalité l'interface cachée _CChat (qui est également l'interface par défaut de la classe).

```
Public Sub Marcher(ByVal nDistance As Long)
   MsgBox "Je me déplace de " & nDistance & " mètres..."
End Sub
```

L'instanciation d'un objet de la classe `CChat` peut se faire sans l'utilisation d'une interface particulière comme le montre le code suivant. Lorsque qu'une variable est typée par un nom de classe plutôt qu'un nom d'interface, Visual Basic utilise automatiquement

l'interface par défaut (dans ce cas l'interface cachée). Ainsi, la variable d'objet `objMinou` est en réalité de type _CChat.

```
Dim objMinou As ComDemo.CChat

Set objMinou = New ComDemo.CChat
objMinou.Marcher 10
Set objMinou = Nothing
```

Bien que les deux techniques soient supportées, nous privilégierons la méthode qui consiste à définir des interfaces séparées de l'implémentation dans une classe. En effet, elle est complètement conforme aux spécifications de COM et ne nécessite pas l'utilisation des astuces qu'apporte Visual Basic.

Il est également important de noter que les clients Automation (scripting par exemple) ne peuvent utiliser que l'interface par défaut d'une classe : ils ne peuvent pas appeler directement `QueryInterface` pour référencer une interface personnalisée, comme le fait Visual Basic avec le mot clé `Set`.
Il existe toutefois des solutions pour remédier à cette limitation des clients de type scripting ; nous en étudierons une dans le chapitre dédié à Automation.

3.5 Composant

Le composant est un autre terme de COM souvent utilisé à mauvais escient.
Ce terme représente en réalité le fichier dans lequel sont implémentées les différentes classes COM. Le composant contient donc le code de création de chaque classe qui le compose. Ainsi chaque classe est incorporée dans un composant (plusieurs classes peuvent être implémentés dans un composant unique). Les composants sont également appelés des serveurs COM.

Les plates-formes Microsoft Windows définissent trois catégories de composants COM :

- Services Windows NT : les composants qui s'exécutent en tant que services sont des serveurs qui peuvent être lancés avec des privilèges particuliers (tel que ceux de l'utilisateur système – LocalSystem Account).
- Exécutables : ce sont souvent des composants qui implémentent une interface utilisateur (tel que Microsoft Word), ou alors des serveurs qui doivent s'exécuter sur une machine distante.
- DLL : ce sont des composants conçus pour être utilisés dans une architecture trois-tiers, ou pour implémenter un service utilisé de façon courante.

Les composants peuvent s'exécuter selon trois modes différents :

- Serveur « In-Process » : s'exécute dans le processus de l'appelant. Tous les composants de ce type sont des DLL (ou des contrôles ActiveX).
- Serveur « Local » (ou « out-of-process) : s'exécute dans un processus séparé et sur la machine de l'appelant. Un composant de ce type peut être un service ou un exécutable.
- Serveur « Remote » : s'exécute dans un processus séparé mais sur une machine distance. Un composant de ce type peut être un service ou un exécutable. Il peut également être une DLL exécutée dans un processus subrogé (surrogate process) tels que dllhost.exe ou Microsoft Transaction Server (MTS).

3.6 Automation

Dans ses débuts, COM n'était accessible qu'aux programmeurs C et C++. En effet, la définition d'une interface est similaire (sinon identique) à la définition d'une classe abstraite en C++, en particulier pour ce qui concerne l'adressage au niveau binaire des procédures et fonctions (structure de pointeurs sur fonction, ou vtable). Malheureusement de part leur niveau d'abstraction élevé, certains langages de programmation (plus particulièrement les langages non-compilés) ne permettent pas la manipulation de ces éléments.

L'introduction de la Type Library (librairie de type) augmente l'accessibilité de COM aux outils autres que les langages de programmation bas-niveau. Cependant, certaines astuces comme `IDispatch` (définit plus loin) sont encore requises pour englober le cas particulier des langages interprétés tels que les langages de macro et de scripting.
Les différents mécanismes décrits dans ce chapitre offrent la possibilité à un client de déterminer, à l'exécution, l'ensemble des méthodes et propriétés implémentées par chacune des classes utilisées ; cela peut même s'étendre jusqu'à la découverte des interfaces implémentées dans certains cas. C'est cette capacité à déterminer la description d'un composant au moment de l'exécution qui est désignée sous l'appellation « Automation ».

Type Library

Pour qu'un client puisse instancier et utiliser des objets à l'exécution, il est nécessaire de connaître certaines informations sur les classes au moment de la compilation. En particulier les éléments suivants doivent être connus :

- Le type d'objet à instancier et à utiliser.
- Les différentes interfaces qui vont être utilisées.
- Les conventions d'appel et le prototypage des méthodes de chaque interface.

IDL apporte les réponses à toutes ces interrogations. Dans son utilisation courante, il est compilé en fichier binaire par le compilateur « Microsoft IDL » (MIDL). Ce fichier, appelé « Type Library », est donc un catalogue qui recense les interfaces, les classes et autres ressources définies dans un composant. Une Type Library peut avoir différentes extensions tels que *.TLB et *.OLB (TLB compilée avec le prédécesseur de MIDL - MkTypeLib), mais peut également être intégrée directement dans les fichiers *.OCX, *.DLL et *.EXE.

Le programmeur peut activer l'utilisation d'une Type Library dans Visual Basic, dans son développement d'application cliente, en désignant explicitement le fichier dans les références du projet, ou en ajoutant un contrôle ActiveX dans le projet. Les informations retenues sont utilisées à la compilation pour la vérification de type, conventions d'appels, mais également pour améliorer les performances de la communication avec les objets instanciés (par une technique appelée « Binding »).
Il est également possible de créer un fichier Type Library à l'aide de Visual Basic, sans utiliser MIDL. Pour cela, il suffit de recompiler le composant après avoir activé l'option Fichier Server Distant dans les propriétés du projet (onglet Composant).

IDispatch

Certains clients ne peuvent pas avoir recours à une Type Library ou ne peuvent pas accéder directement aux interfaces personnalisées. Ces outils sont appelés les clients Automation. Les langages de scripting font partie de cette catégorie, car ils ne permettent pas de typer les variables. COM s'ouvre à tous ces outils en définissant une interface particulière, dérivée de `IUnknown`, appelée `IDispatch` :

```
interface IDispatch : IUnknown
{
   HRESULT GetTypeInfoCount(…);
   HRESULT GetTypeInfo(…);
   HRESULT GetIDsOfNames(…);
   HRESULT Invoke([in] DISPID dispIdMember, …);
}
```

Visual Basic implémente automatiquement `IDispatch`, il n'est donc pas nécessaire de maîtriser ce concept pour développer des composants COM accessibles par scripting avec ce langage.

`IDispatch` (et sa version plus récente `IDispatchEx`) permet aux clients Automation d'utiliser des objets COM qui implémentent cette interface, en mettant en oeuvre une technique de découverte à l'exécution. Ce processus peut être décrit comme suit :

1. Le client instancie l'objet de façon usuelle, mais parce qu'il ne connaît pas les interfaces implémentées, il va tenter d'obtenir une référence sur l'interface connue `IDispatch`.
2. Lorsqu'il doit accéder aux fonctionnalités de l'objet nouvellement créé, le client appelle `GetIDsOfNames` en passant en paramètre le nom de la méthode demandée. Si cette méthode est supportée par la classe, `GetIDsOfNames` renvoie le numéro de fonction correspondant : ce code est appelé « Dispatch ID » (DISPID).
3. Ensuite, le client Automation appelle `Invoke` en passant le DISPID obtenu précédemment. L'implémentation de `IDispatch` redirige alors l'appel vers la fonction demandée.

Cette technique est relativement complexe, mais les moteurs de scripting l'implémentent automatiquement. L'auteur du script peut alors instancier et utiliser les objets COM en suivant cet exemple VBScript :

```
Dim objBrutus

Set objBrutus = CreateObject("ComDemo.Cchien")
objBrutus.Marcher 10
Set objBrutus = Nothing
```

Dans certains cas, Visual Basic peut utiliser l'interface `IDispatch` plutôt qu'une interface personnalisée. Cela s'applique à toutes les variables objets de type Object :

```
Dim objBrutus As Object

Set objBrutus = CreateObject("ComDemo.Cchien")
objBrutus.Marcher 10
Set objBrutus = Nothing
```

Le code qui précède paraît identique au code utilisant une variable de type `ComDemo.Ianimal` (décrit page au chapitre 3.4 : Classe et objet), pourtant le processus mis en œuvre est bien plus complexe : il est donc moins performant. Ce processus est appelé « Late-binding ».

Binding

Le « Binding », ou liaison de type, est un terme qui décrit le processus mis en œuvre dans les outils de programmation pour appeler une méthode d'objet COM. Trois techniques sont utilisées à ce jour :

- Late-binding est utilisé par les clients Automation ou par Visual Basic avec les variables de type Object : c'est la mise en œuvre complète de `IDispatch`.

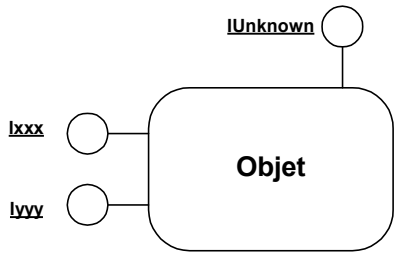
- Early-binding est utilisé par Visual Basic lorsqu'une Type Library est référencée, avec les objets qui n'exposent que l'interface `IDispatch` (par exemple les extensions appelées « Designer »). Visual Basic examine la Type Library à la compilation et détermine tous les DISPID utilisés, l'appel à `GetIDsOfNames` n'est donc pas requis au moment de l'exécution : seul `Invoke` est utilisé.

- VTable-binding est la technique la plus performante, elle consiste à appeler les fonctions des interfaces personnalisées directement par adresse. Elle est mise en œuvre par Visual Basic lorsque qu'une Type Library est référencée, avec les variables typées et lorsque l'objet expose l'interface personnalisée demandée.

3.7 Architecture

Représentation schématique

Certains ouvrages de référence de COM exposent les concepts à l'aide de diagrammes appelées « lollipop diagram » ou « box-spoon diagram ». Cette notation est souvent utilisée pour représenter les classes ainsi que les interfaces exposées. Par convention, l'interface qui pointe vers le haut est toujours `IUnknown`.

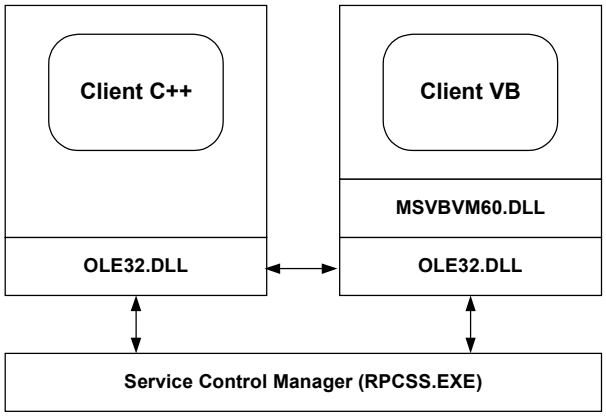


Représentation d'une classe

Infrastructure

Un des concepts de base de COM est qu'un client et un objet communiquent par l'intermédiaire des interfaces, et que le client ignore où s'exécute l'objet. Pour satisfaire à cette contrainte, il est indispensable de se reposer sur les services fournis par le système d'exploitation : l'infrastructure de COM est conçue pour répondre à ces exigences.
Les librairies COM sont un ensemble de DLL et EXE installés sur tous les systèmes Windows. Ces composants de base font actuellement partie intégrante du système d'exploitation, ainsi chaque mise à jour du système est susceptible d'apporter des améliorations à COM. Cela est particulièrement vrai pour les Service Packs de Windows NT.
Les applications clientes interagissent avec tous les composants COM à travers ces librairies. La majorité de ces API sont implémentées dans OLE32.DLL, elles sont appelées

directement par un client C++ ou par la machine virtuelle pour Visual Basic comme le montre le schéma suivant :



Infrastructure COM

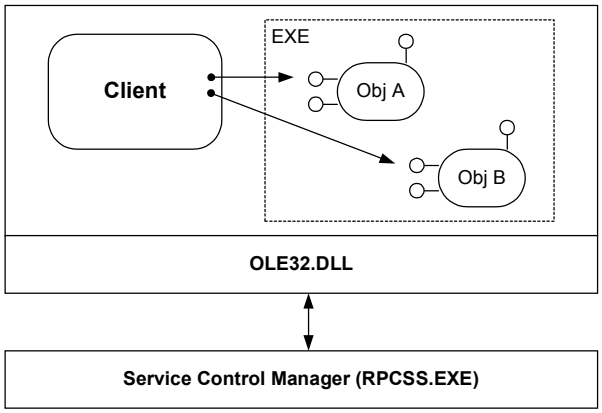
La liste qui suit décrit la correspondance entre les versions les plus courantes de OLE32.DLL avec les mises à jour du système d'exploitation. Cette liste n'est pas exhaustive, et n'est fournie qu'à titre indicatif : COM est installé avec un ensemble de composants qui doivent être homogènes en terme de numéro de version. Il est donc absolument déconseillé de remplacer cette DLL en espérant mettre à jour COM.

OLE32.DLL	Windows 95	Windows 98	Windows NT 4.0
2.10.35.35	Win 95		
2.10.35.36	Win 95 (SP1 / OSR2)		
4.0.1373.1			NT 4.0
4.0.1381.3			NT 4.0 SP2
4.0.1381.4			NT 4.0 SP3
4.0.1381.117			NT 4.0 SP4
4.0.1381.190			NT 4.0 SP5
4.0.71.1817	IE 4.01 (SP1 / SP2)	IE 4.01 (SP1 / SP2)	IE 4.01 (SP1 / SP2)
4.71.1719.0		Win 98	
4.71.2612.0		DCOM 98 (VS6)	
4.71.2618.0	DCOM 95 1.2		
4.71.2900.0	IE 5.0	IE 5.0	IE 5.0
4.71.3328.0	DCOM 95 1.3	DCOM 98 1.3	

Différentes versions de OLE32.DLL

Activation

Dans le processus d'instanciation d'un objet, le client fait appel aux services du système d'exploitation. Ce dernier est alors responsable du chargement du composant et retourne une interface de la classe demandée au client : ce processus est appelé « Activation ».
L'activation d'un objet est mise en œuvre par l'infrastructure du système, qui utilise les services de OLE32.DLL et d'un autre composant appelé « Service Control Manager » (SCM prononcé « scum »). Le SCM de COM est implémenté dans RPCSS.EXE ; il n'est sollicité que pour les composants autres que « In-Process », et ne doit pas être confondu avec « Windows NT Service Control Manager » utilisé pour démarrer et arrêter les services de Windows NT.
Lorsque le développeur instancie un objet par son ProgID, Visual Basic fait appel à OLE32.DLL pour rechercher le CLSID correspondant dans la base de registre (selon un mécanisme décrit dans le chapitre suivant), et pour activer le composant désigné. A ce point, le SCM localise l'emplacement physique du composant et s'occupe de son chargement. Il communique ensuite avec le composant par des interfaces prédéfinies, et retourne enfin l'interface demandée initialement par Visual Basic.

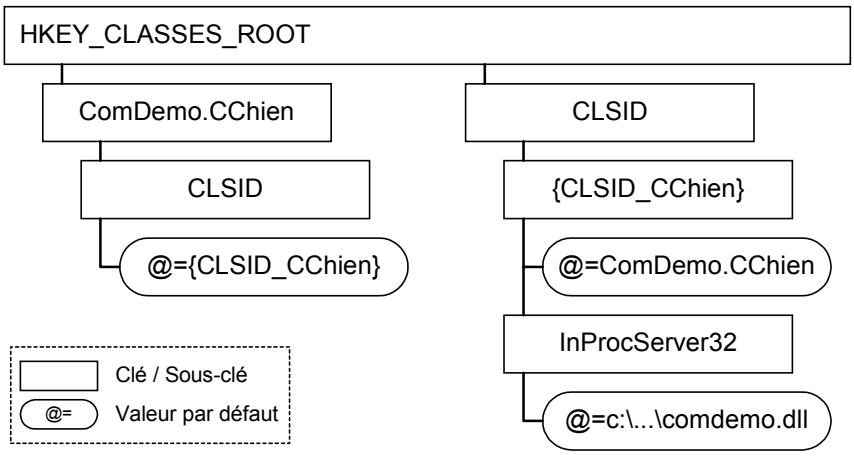


Activation d'un composant COM

Registration (inscription)

Afin de permettre aux composants du système tels que OLE32.DLL (et RPCSS.EXE pour les composants serveurs autres que « In-Process ») d'obtenir les informations nécessaires à l'activation des composants COM, il est nécessaire de stocker les éléments de configuration dans un emplacement bien déterminé : cet emplacement est la base de registre du système.
Le composant est le seul à connaître les détails relatifs aux CLSID et IID qu'il implémente. A ce titre, il est le seul responsable des éléments à inscrire dans la base de registre. La procédure d'écriture des informations de configuration du composant s'appelle « Registration » (ou inscription).

Les informations inscrites dans la base de registre les plus déterminantes sont décrites dans le schéma qui suit. La clé *InProcServer32* est réservée pour un composant DLL et la clé *LocalServer32* ne s'applique qu'à un composant EXE. Pour simplifier, nous ne décrivons pas le cas du composant activé sur une machine distante.



Principales clés de registre pour l'activation d'un composant

La registration s'effectue différemment selon que le composant est implémenté dans une DLL ou dans un EXE :

```
DLL : RegSvr32 <composant.dll>
EXE : composant.exe /RegServer
```

La dé-registration, ou suppression des informations de la base de registre, est réalisée comme suit :

```
DLL : RegSvr32 /u <composant.dll>
EXE : composant.exe /UnregServer
```

Note : /RegServer et /UnregServer sont des paramètres en ligne de commande pour un composant serveur EXE. Ces arguments sont fixés par convention, mais il est évident que l'implémentation est laissée au développeur du composant : certains développeurs « oublient » la convention pour privilégier une syntaxe propriétaire. Les composants ActiveX EXE développés avec Visual Basic respectent cette convention et fonctionnent selon la syntaxe décrite.

4 – Concepts fondamentaux d'ActiveX

ActiveX est avant tout une dénomination marketing, basée sur les contrôles anciennement appelés contrôles OCX (provenant d'OLE). Cet avec l'apparition de Windows 95 et de l'API de jeux DirectX, que l'introduction de la lettre « X » est apparue dans les noms des technologies de Microsoft.

Mais ActiveX est également un standard permettant à des composants logiciels d'interagir, que ce soit sur la même machine ou entre deux machines distantes reliées par un réseau, et quel que soit le langage utilisé pour les développer. Cela vous rappelle-t-il quelque chose ? Hé oui, les composants ActiveX sont des composants COM, ni plus ni moins. Ainsi, lorsqu'on parle d'ActiveX, on parle de COM. Le standard ActiveX n'est pas contrôlé par Microsoft, mais par une organisation indépendante, constituée d'utilisateurs, d'éditeurs de logiciels et de fournisseurs de systèmes d'exploitation, dans laquelle Microsoft devrait intervenir au même titre que les autres membres, avec le même pouvoir de vote et de décision.

La majorité des utilisateurs et des développeurs voient dans le terme ActiveX un raccourci pour « contrôles ActiveX[3] ». Le standard ActiveX ne se limite pas à la création de tels contrôles, mais également au développement de DLL, d'exécutables, de documents, d'exécutables documents, de DLL documents, etc. Il est vrai que les contrôles ActiveX ont été le principal cheval de bataille de Microsoft lorsque le standard est apparu sur le marché. Les arguments en faveur de ce type d'outils étaient à l'époque la dimension interactive (un mot passé de mode actuellement, tant l'interactivité est omniprésente…) et attrayante que les ActiveX apportaient au web. Dans la réalité, même si à une époque tout le monde en parlait, bien peu de sites web ont utilisé cette technologie. Peut-être en raison des trous de sécurité, peut-être en raison de la lenteur des composants, peut-être en raison de la difficulté à devoir apprendre un langage de programmation pour développer de tels contrôles ? Le fait est que finalement, en se baladant sur le net, l'auteur de ce rapport ne s'est trouvé nez à nez que très rarement avec des contrôles ActiveX inclus dans des pages HTML.

L'arrivée des composants ActiveX sur le marché du logiciel était également la réponse de Microsoft aux applets Java. Les composants ActiveX, contrairement aux applets, avaient l'avantage de n'être chargés qu'une fois et exécutés en mode natif, et non par le biais d'un interpréteur ou d'une machine virtuelle.

4.1 Spécifications

Un objet ActiveX est un objet conforme au modèle COM et à les caractéristiques suivantes :

→ Il est implémenté sous forme binaire, et peut donc être écrit dans n'importe quel langage source.

→ Il est encapsulé dans un fichier exécutable (soit .exe pour les applications et .ocx pour les contrôles) ou dans une bibliothèque de liens dynamiques (.dll).

[3] D'ailleurs même Microsoft donne l'impression de limiter la technologie ActiveX aux contrôles. Le titre de la homepage d'ActiveX (http://www.microsoft.com/com/tech/activex.asp) n'indique-t-il pas « ActiveX controls » ?

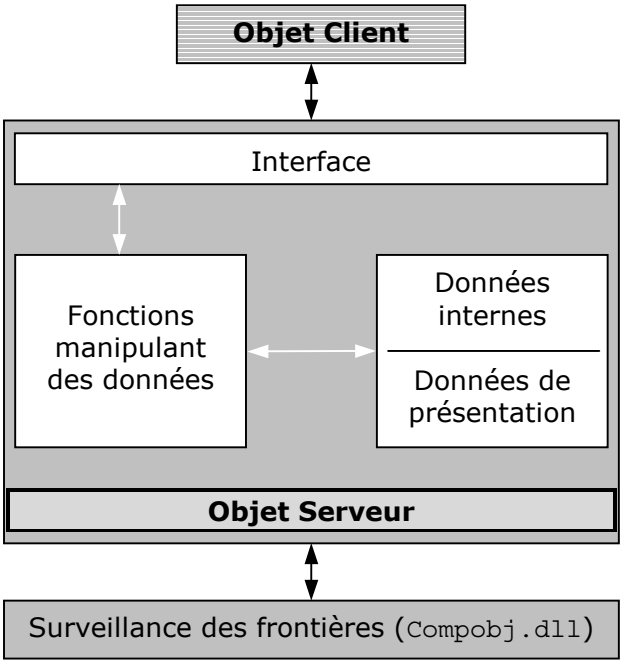
→ Il contient des données de deux types, qui peuvent être représentées comme des propriétés appartenant en privé à l'objet :

- données de présentation, qui sont nécessaires pour l'affichage à l'écran ou pour l'impression, et
- données internes.

→ Il contient certaines fonctions permettant de manipuler les données précitées.

→ Il fournit une interface standardisée afin que d'autres objets puissent communiquer avec lui.

→ Il participe (grâce aux routines de la bibliothèque système `Compobj.dll`) à la surveillance des frontières, soit au processus de passage d'arguments à des fonctions et de retour de valeurs entre processus et machines.



Structure interne d'un objet serveur ActiveX

4.2 Différences entre un contrôle OCX et un contrôle ActiveX

La technologie utilisée dans les contrôles ActiveX englobe celle qui est utilisée dans les contrôles OLE (fichiers ayant l'extension OCX). Les deux types de contrôles s'appuient sur le modèle COM et par conséquent sur les mécanismes de communications inter-processus standard DCE, et sont indépendants du langage de programmation, car ce sont des bibliothèques de fonctions qui appellent l'API Windows.
Les contrôles ActiveX sont les descendants, plus rapides et plus compacts, des contrôles OLE.

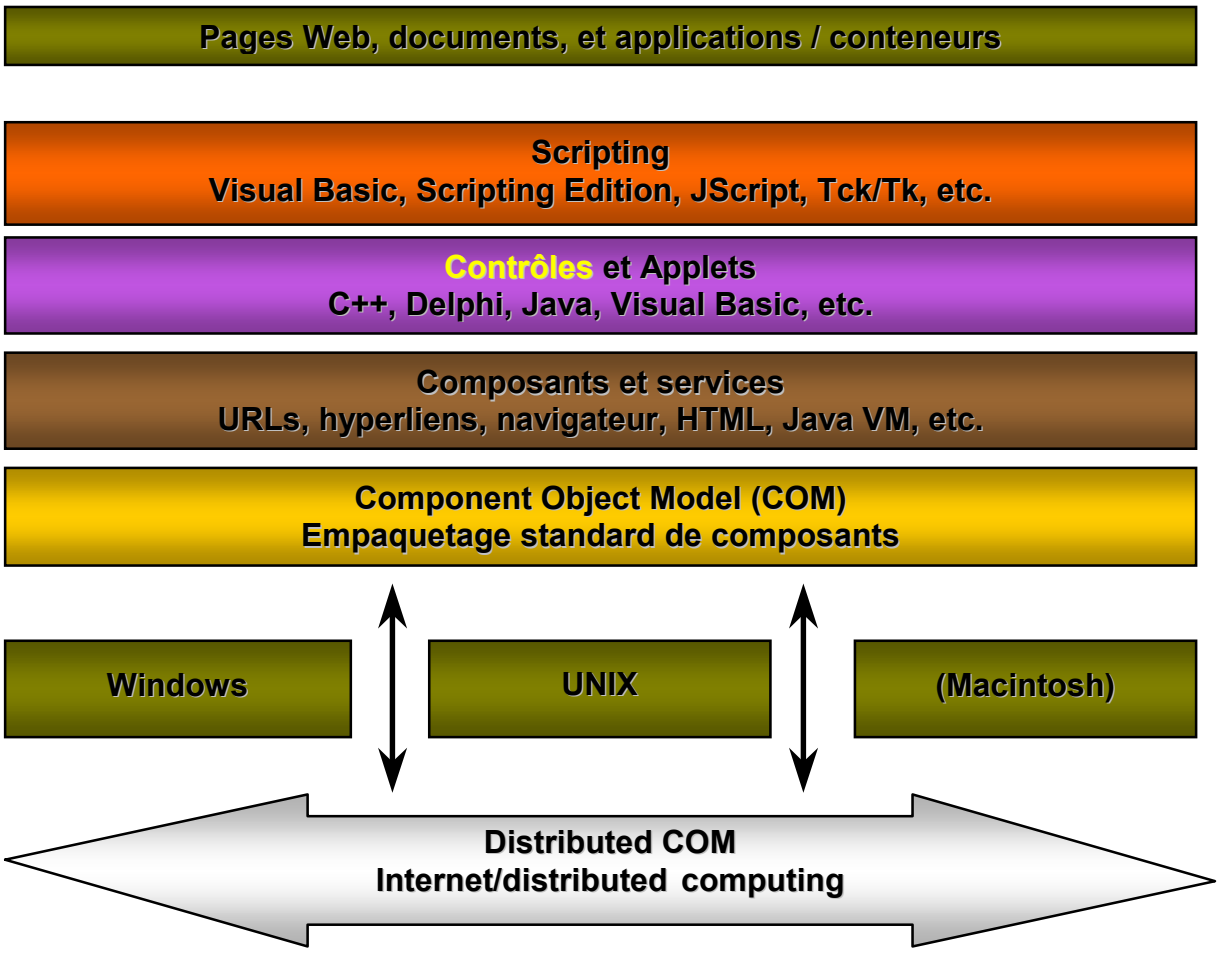
4.3 Outils utilisés pour la conception des composants ActiveX

Les composants ActiveX peuvent être réalisés en C++ à partir d'une nouvelle bibliothèque de modèles de classes, l'ActiveX Template Library ou ATL, mais également en langage Visual Basic à partir de la version 5.0 de cet environnement de développement, ou en Java, en Pascal, en FORTRAN ou encore en COBOL.

Par rapport à une implémentation de contrôle OLE à partir du SDK OLE ou des MFC (Microsoft Foundation Classes), ATL[4] n'apporte que le strict nécessaire pour permettre au contrôle ActiveX d'intéropérer efficacement avec d'autres composants logiciels (les interfaces minimales: `IUnknown`, `IClassFactory`, `IDispatch`, `IEnumXXX`, `connectionPoint`, le fonctionnement monothread ou multi-thread, etc.).

4.4 Domaine d'utilisation

Mis à part l'interactivité et le dynamisme des contrôles ActiveX sur l'Internet, on retrouve ces composants dans des architectures serveurs, en relation avec ASP (Active Server Pages) ou MTS (Microsoft Transaction Server), ainsi que dans d'autres applications client/serveur 3-tiers ou multi-tiers.



[4] Puisqu'il s'agit bien d'une bibliothèque de modèles, par opposition à une bibliothèque de classes telles que les MFC, ATL ne nécessite pas de DLL de run-time, et n'utilise que des fonctions standard des systèmes d'exploitation supportant l'interface de programmation Win32. Le développeur de contrôles ActiveX n'a donc pas le souci de redistribuer un certain nombre de fichiers complémentaires avec son produit.

Bibliographie critique

- Visual Basic developer's guide to COM and COM+, Wayne S. Freeze, Sybex 2000, 460 pages.
  L'ouvrage qui permet le mieux (parmi ceux cités dans cette bibliographie) de comprendre les technologies de composants de Microsoft, et met en relation simplement et efficacement les différents concepts. La partie consacrée à DCOM est malheureusement très courte, et n'offre qu'une petite introduction au protocole. L'auteur traite également COM+, MSMQ, et IMDB.

- Developping COM/ActiveX components with Visual Basic 6, Dan Appleman, Sams 1999, 860 pages.
  Cet ouvrage s'adresse avant tout aux développeurs professionnels sur VB, et montre les meilleures techniques pour développer des composants COM et DCOM. D'un niveau très nettement plus élevé que le titre précédent, cette « bible » (presque 900 pages !) passe en revue la conception d'objets COM mais aussi et surtout les mécanismes internes lors de l'exécution des composants.

- Au cœur de Distributed COM, Guy & Henry Eddon, Microsoft Press 1998, 560 pages.
  Ce livre permet au développeur d'utiliser efficacement le modèle DCOM. S'adresse à un public de connaisseur (Automation, Multithreading, Marshalling,…). Il est capital d'avoir une solide expérience en la matière pour se lancer dans cet ouvrage !

- Technopoche COM, Stéphane Crozatier. Il s'agit d'un fascicule créé par un collaborateur de Microsoft qui m'a été extrêmement utile pour saisir le fonctionnement et l'architecture de COM. Notons qu'il existe également des technopoches sur ADO et ADOX, tout aussi complet que celui-ci.