Distributed

COM

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

L'association des concepts d'objet et de répartition a donné naissance à un nombre impressionnant de langages, de systèmes et de bibliothèques d'objets répartis. Tous les projets actuels d'architectures réparties sont basés sur le concept d'objet. D'une part, les chercheurs issus de la communauté « langage de programmation à objets » étendent les environnements de programmation vers des architectures réparties, et d'autre part, les chercheurs issus de la communauté « système » adoptent le modèle objet pour structurer les concepts de la répartition.

Bien que certains développeurs universitaires associent les objets et la répartition suivants une approche particulière (intégrée ou appliquée), la grande majorité des systèmes à objets répartis, et en particulier, les produits commerciaux, font une synthèse pragmatique des deux approches. Parmi les rejetons du couple objet et répartition, les plus connus ont pour noms CORBA, DCOM, Java/RMI, ou encore ONE.

CORBA

Depuis 1989, une association internationale appelée l'OMG (Object Management Group), définit la spécification de l'architecture d'un système à objets répartis, appelée CORBA (Common Object Request Broker Architecture). Près de 700 sociétés sont actuellement membres de l'OMG, dont aussi bien des sociétés informatiques comme HP, Sun, Microsoft, ou IBM, que des utilisateurs comme Boeing ou Alcatel. A l'inverse d'ODP (Open Distributed Processing) - un autre effort de standardisation d'architectures réparties – CORBA a été défini dans un but très pratique, et de nombreuses mises en oeuvre de la spécification CORBA sont actuellement sur le marché, incluant SOM de IBM, Orbix de Iona, et Visibroker d'Inprise.

CORBA associe les concepts d'objet et de répartition dans une approche à la fois intégrée et appliquée. L'intégration se fait à travers la notion d'invocation à distance d'objet (l'objet est l'unité de répartition), et l'application se fait à travers les services de répartition qui sont organisés sous forme de bibliothèques de classes.

CORBA s'appuie sur le protocole IIOP (Internet Inter-ORB Protocol) pour les objets distants. Toute l'architecture de CORBA dépend d'un ORB (Object Request Broker), qui agit comme un bus à objets central sur lequel chaque objet CORBA interagit de manière transparente avec d'autres objets CORBA, qu'ils soient locaux ou distants. Chaque objet serveur CORBA a une interface (décrite en IDL, Interface Description Langage), et expose un ensemble de méthodes. Pour appeler un service, un client acquiert une référence objet à un objet serveur, puis il peut – au travers de cette référence – effectuer des appels de méthode comme si l'objet serveur résidant dans l'espace d'adressage du client. L'ORB s'occupe de trouver l'implémentation des objets CORBA, de la préparer à recevoir des requêtes, de les lui communiquer, et de répondre au client. Les objets CORBA interagissent avec ORB soit au travers de l'interface de ce dernier, soit au travers d'un adaptateur d'objet (Object Adapter), les plus fréquents étant BOA (Basic Object Adapter), et POA (Portable Object Adapter).

Etant donné que CORBA n'est qu'une spécification, elle peut être portée sur diverses plates-formes comme UNIX ou Windows, aussi longtemps que l'implémentation des ORB sur ces plates-formes existent.

DCOM

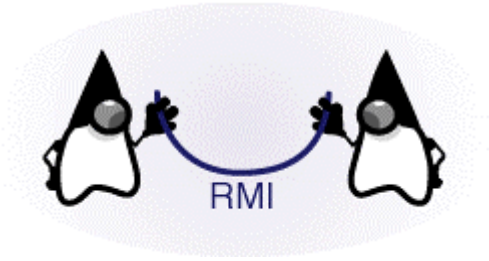
Bien que membre de l'OMG, Microsoft a développé son propre standard d'objets répartis, appelé DCOM (Distributed Component Object Model). Le terme « standard » revêt dans ce contexte un caractère particulier, car il signifie standard pour – et seulement pour – les produits Microsoft.

DCOM supporte les objets distants grâce au protocole ORPC (Object Remote Procedure Call), qui lui, est basé sur RPC (Remote Protocol Call, de DCE : Distributed Computing Environment) et interagit avec les services run-time de COM. Un serveur DCOM est un code capable de servir des objets d'un type particulier en cours d'exécution. Chaque objet serveur DCOM peut supporter des interfaces multiples, chacune d'entre elle représentant un comportement différent de l'objet. Un client DCOM appelle les méthodes exposées d'un serveur DCOM en pointant une des interfaces de l'objet du serveur.
Cet appel, au travers du pointeur vers l'interface, s'effectue comme si l'objet serveur résidait dans l'espace d'adressage du client. Etant donné la spécification binaire de COM, les composants serveurs DCOM peuvent être écrits en C++, Java, Delphi, Visual Basic, voire même COBOL. Aussi longtemps qu'une plate-forme supporte les services COM, DCOM l'est aussi.

DCOM est le modèle distribué de composants le plus utilisé sur plate-forme Windows. Malgré son caractère « standard », Microsoft fournit une implémentation aussi bien pour plate-forme Windows que pour plate-forme Solaris. Notons également que d'autres compagnies fournissent également des implémentations : Software AG fournit une implémentation des services COM et DCOM pour UNIX et Linux avec le produit EntireX, et Digital permet d'opérer sur plate-forme OpenVMS.

Java/RMI et Java Beans

La bibliothèque de classes RMI (Remote Method Invocation), développée par la société JavaSoft (filiale de Sun Microsystems) s'appuie sur le protocole JRMP (Java Remote Method Protocol). Essentiellement basé sur la sérialisation – qui permet aux objets d'être transmis (marshalling) comme un flux – spécifique à Java (Java Object Serialization), les objets serveurs et clients doivent tous deux être écrits en Java. Chaque objet serveur définit une interface permettant d'accéder à l'objet depuis la machine virtuelle Java d'un ordinateur distant[1]. L'interface expose un ensemble de méthodes indiquant les services offerts par l'objet serveur.



RMI dépend du mécanisme de nommage RMIRegistry (s'exécutant sur la machine serveur) et permettant de contenir toutes les informations relatives au serveur à objets. Le client RMI acquiert une référence à un objet serveur en effectuant un lookup sur le serveur à objets, et invoque ensuite les méthodes désirées de la même manière que si l'objet serveur résidait dans son espace d'adressage.
Les objets serveurs RMI sont nommés avec une URL (Uniform Resource Locator); ainsi, un client désirant obtenir une référence sur un objet serveur spécifiera l'URL de ce dernier de la même manière qu'on adresse une page HTML.

[1] Java, du moins dans les premières versions des bibliothèques, ne permet pas l'invocation d'objets distants. Java intègre les objets et la répartition à travers le mécanisme de migration : c'est le code qui se déplace. Lorsqu'un navigateur comme Netscape charge une page web et rencontre dans un code HTML une étiquette désignant une applet Java (soit un programme Java qui peut se télécharger sur le web), il retrouve le code de l'applet et le fait migrer vers le site client (en établissant une connexion TCP/IP). La machine virtuelle Java du navigateur permet ensuite d'exécuter le code Java (en fait le byte code) migré chez le client. Puis le navigateur efface ensuite le code de la mémoire lorsqu'il quitte la page web en question.

Java/RMI fonctionne sur toutes les plate-formes implémentant une machine virtuelle Java.

L'introduction des Java Beans (puis des EJB[2], Entreprise Java Beans) pour le développement d'applications sous forme d'un assemblage de composants s'appuyant sur un canevas a démontré la validité du concept, validité déjà démontrée en partie par les ActiveX, malheureusement tributaires d'une seule plate-forme.
Java Beans est un modèle de composants portables et multi-plate-forme écrit en Java. Il permet aux développeurs d'écrire des composants selon le dogme « *write once – run everywhere* » en bénéficiant de la portabilité et de la puissance de Java. Leur spécification a été définie par Sun et de nombreux autres acteurs, tel IBM, impliqués dans le monde Java, dans le but d'instaurer un véritable marché des composants logiciels.

Les Beans peuvent être manipulés dans des outils de développement graphique (Jbuilder de Borland ou Visual Age d'IBM par exemple). Un Bean est une classe Java classique qui adhère à certaines conventions concernant ses propriétés et sa gestion des événements. C'est grâce au respect de cette norme qu'il peut être analysé par l'environnement qui l'accueille.

1.1 Une comparaison entre Java Beans et ActiveX…

Facilités de maintenance

De ce côté, l'avantage a longtemps été du côté des ActiveX : bien documentés, rapidement intégrés aux outils de développement leaders (Visual Basic, Visual C++, Delphi,…), désormais ils font partie des meubles. Les Java Beans, pour leur part, profitent d'un effet de mode, et bon nombre d'outils (ceux qui supportent le JDK, dès sa version 1.1, Jbuilder, Power J, Visual Age, par exemple) permettent de développer ceux-ci. Ils offrent généralement des assistants de création aux Java Beans aussi simples que ceux connus pour les ActiveX, voire souvent plus sophistiqués (selon le côté de la barrière où l'on se place… ☺).

Performances

En ce qui concerne la performance du composant pris hors de son contexte, l'avantage se situe du côté des ActiveX, qui s'appuient sur du code véritablement compilé. Les Java Beans, en revanche, se contentent des performances de Java, qui sont encore perfectibles.

Etendue du parc

La viabilité d'une solution informatique dépend souvent du parc existant à un instant donné. De ce point de vue, les composants ActiveX vendus dans le commerce ont été pendant longtemps plus nombreux que les bibliothèques de Java Beans. Une tendance qui s'inverse, dans la mesure où Java concerne les éditeurs de toutes les plate-forme serveurs. D'autre part, les acteurs majeurs du marché base de données (Oracle, Informix, Ingres, IBM) ont plutôt jeté leur dévolu sur Java qu'ActiveX, tant il est vrai que le parc installé des grands comptes se situe plutôt sur des architectures à base AS400 ou UNIX que Microsoft. Cette tendance, que l'on croyait un temps devoir disparaître, semble au contraire se renforcer. Tant et si bien que l'architecture Entreprise Java Beans semble

[2] Les EJB sont semblables aux composants MTS/COM+ de Microsoft. Ils répondent principalement aux problématiques de montée en charge et de transactionnel en distinguant les composants métiers dits Session Beans, des composants persistants dits Entity Beans. Les serveurs EJB sont le cœur des serveurs d'applications J2EE.

en mesure de définit un nouveau standard dans la problématique des mécanismes d'accès à l'information.

Portabilité et interopérabilité

A ce sujet, la comparaison est vite faite : hors de la plate-forme Wintel, peu de chances (même si quelques éditeurs indépendants proposent des possibilités de déploiement sur d'autres plate-formes) de voir fleurir des ActiveX. Ces derniers ne sont pas portables, et n'interopèrent qu'avec eux-mêmes. L'interopérabilité est néanmoins techniquement possible par l'utilisation de divers ponts permettant la communication avec des Java Beans ou des objets CORBA. Il reste néanmoins vrai que l'ajout d'un pont est un pis-aller qui augmente la complexité et diminue les performances…

Cette caractéristique rend douteuse, voire improbable la présence des ActiveX avec les serveurs UNIX. Un atout de poids pour les Java Beans qui, au contraire, peuvent être déployés sur toutes les plate-formes serveurs et qui, par l'intermédiaire de CORBA, bénéficient d'une interopérabilité sans faille.

Montée en charge

Côté montée en charge, les technologies Java Beans et ActiveX se tiennent dans un mouchoir de poche : les Java Beans s'appuient sur les médiateurs d'objets de l'architecture de l'OMG (Object Management Group), ces ORB CORBA tels que Visibroker (société Inprise) mettent à disposition des jeux de composants identiques facilement « distribuables » sur plusieurs machines.

De son côté, Microsoft intègre à Windows 2000 son Transaction Server, dont l'avantage principal est d'offrir une interface d'utilisation relativement simple. De cette manière, on obtient les avantages d'un moniteur transactionnel dans les composants ActiveX, en quelques appels de fonctions. Le « faible » coût de cette approche permet de démocratiser l'utilisation des moniteurs transactionnels dans l'entreprise.

Le pont Java Beans / ActiveX

Javasoft propose un outil permettant à un Java Bean d'être vu comme un ActiveX par les applications Windows permettant de les visualiser. Il s'agit du packager, qui produit, à partir d'un Java Bean un fichier destiné à la base de registres Windows, des fichiers amorces Java (constituant le pont entre le Java Bean et l'ActiveX qui l'encapsule), et un fichier binaire OLE décrivant les méthodes, événements et propriétés à insérer dans le Type Library.
Les classes nécessaires sont insérées dans le fichier JAR original. Ce fichier JAR reste portable si les classes du paquetage sun.beans.ole sont disponibles.

Notons aussi que Microsoft propose sa propre version de cette passerelle, et que l'outil de Taligent, WebRunner Toolkit, effectue l'opération inverse : ActiveX en Java Bean.
CORBA quand à lui propose un modèle permettant en principe d'intégrer aussi bien des Java Beans que des composants ActiveX au sein d'un même environnement.

Alors : Java Beans ou ActiveX ?

Le choix d'une architecture n'est pas simple. Pour une solution centrée sur UNIX, le choix des Java Beans semble inévitable.

De même, la sélection des ActiveX en environnement exclusivement Windows semble tout aussi judicieuse. Lorsque les ressources serveurs se partagent les environnements Windows et UNIX. Une solution mixte peut constituer un bon choix. Le

développement Web autorise – par le biais des adresses URL – l'accès à des ressources variées, et cela de manière complètement transparente pour le client. Autant en profiter, en répartissant les traitements côté serveur entre les applications Javascript/Java Beans et ASP/ActiveX.

Une démarche logique d'analyse consiste, dans ce cas, à découper fonctionnellement l'applicatif serveur, à valider pour chacune des fonctions l'adéquation des Java Beans ou ActiveX (en termes de performances et de facilité d'accès à l'information souhaitée) et à implémenter la fonctionnalité avec l'architecture souhaitée.

1.2 Le monde des composants distribués selon Microsoft

Faisons le point sur les multiples abréviations qui parsèment le jargon qui caractérise Microsoft. Les 6 technologies fondamentales de **DNA** (Distributed interNet Applications[3]) sont :

- **COM**, qui comme nous l'avons vu, permet de définir des composants,
- **DCOM**, qui permet de manipuler ceux-ci de manière distribuée,
- **MTS** (Microsoft Transaction Server), un environnement d'exécution dans lequel les composants évoluent, offrant un service de requêtes-réponses en fournissant une sécurité ainsi qu'une gestion des transactions[4],
- **MSDTC** (Microsoft Distributed Transaction Coordinator), un outil traitant de manière automatique les mises à jour sur de grosses applications,
- **MSMQ** (Microsoft Message Queue, nom de code The Falcon), similaire à l'interface d'invocation dynamique de CORBA, de manière asynchrone[5],
- Et **MSCS** (Microsoft Cluster Server, nom de code The Wolf Pack), permettant de regrouper plusieurs serveurs pour les faire travailler ensemble.

[3] DNA a désigné de 1996 à aujourd'hui l'architecture de composants distribués de Microsoft. Avec l'arrivée de .NET, il semblerait que cette appellation soit abandonnée au profit d'une nouvelle…

[4] Les technologies middleware concurrentes de MTS sont les EJB (Entreprise Java Beans) et CCM (CORBA Component Model).

[5] Les technologies « MOM » (Message Oriented Middleware) concurrentes de MSMQ sont JMS (Java Message Service) et MQSeries d'IBM.

2 - DCOM

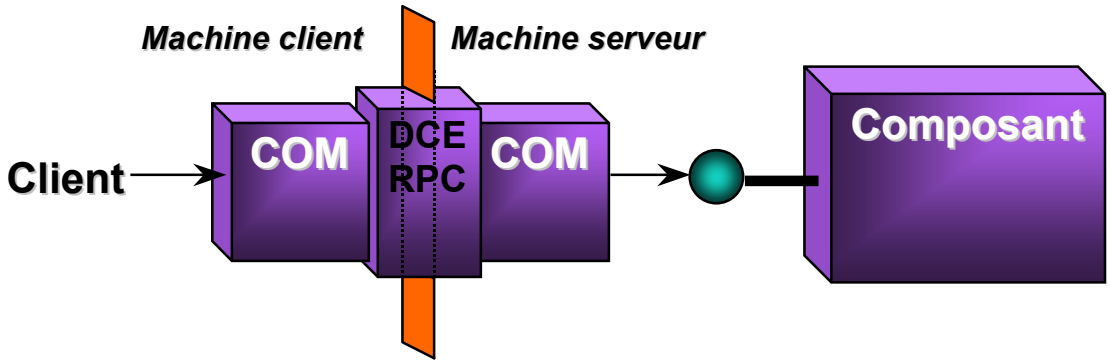
2.1 Introduction

Le modèle de composants distribués de Microsoft, DCOM, étend COM afin de supporter les communications inter-objets sur différents ordinateurs répartis sur un réseau LAN (Local Area Network) ou WAN (Wide Area Network), ou sur Internet.

La distance qui sépare COM de DCOM est très courte : même si COM permet de concevoir des objets s'exécutant sur des machines distantes, il faudra utiliser l'outil de configuration de DCOM pour permettre à votre machine de supporter les objets COM de manière appropriée. En termes de technologie, DCOM ne nécessite pas de programmation plus spécifique que COM (du moins sous VB), soit ce que ne nous avons vu jusque là. En fait, il n'y a pas de différences entre le développement d'un serveur out-of-process COM du développement d'un serveur out-of-process DCOM : les programmes sont identiques.

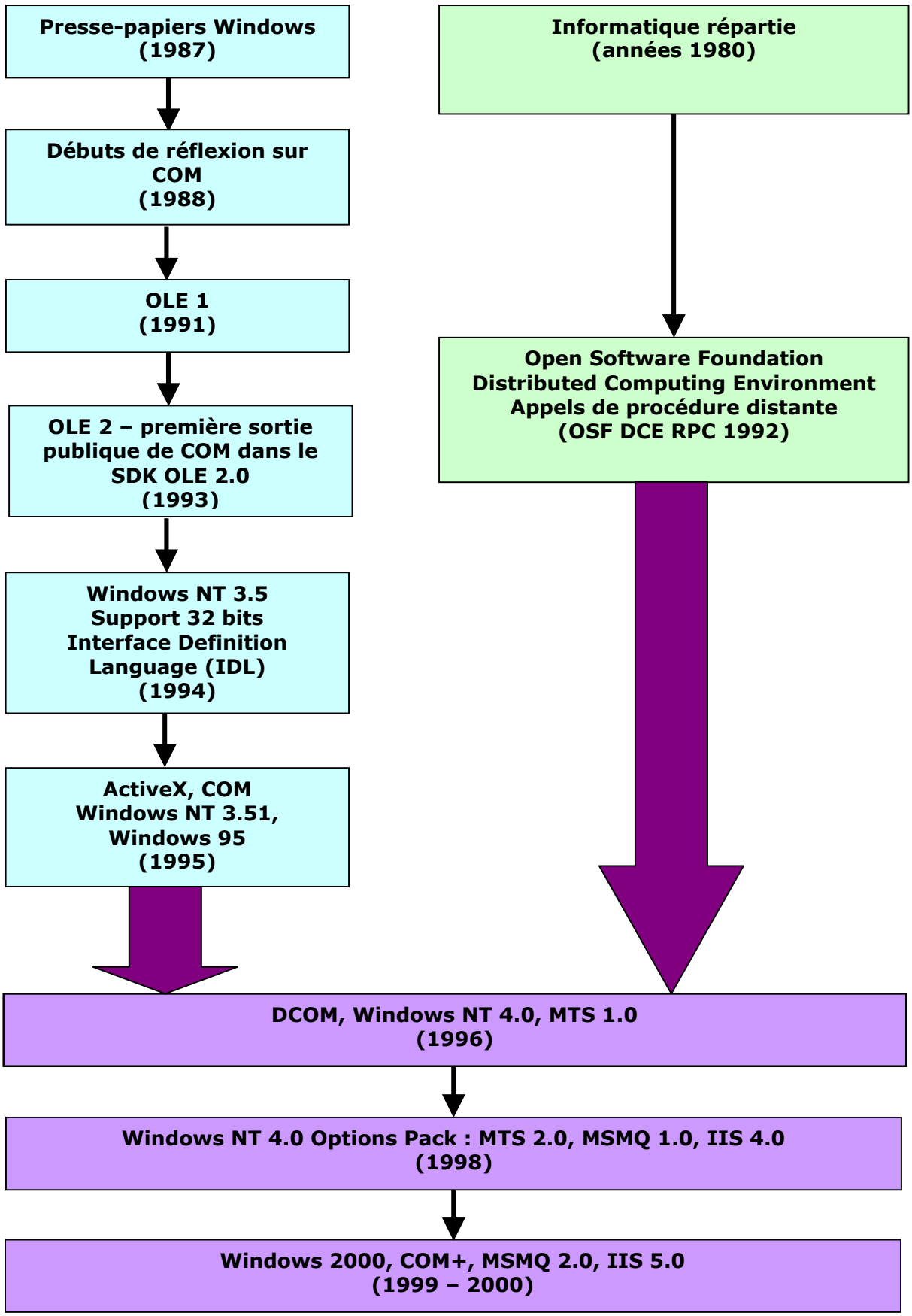
2.2 Concept fondamentaux de DCOM

DCOM se base sur les couches DCE-RPC pour assurer la transparence au niveau du réseau et gérer les appels distants de composants. Il facilite énormément la gestion du réseau pour les programmeurs, en prenant en charge aussi bien les communications que le marshalling, soit l'encapsulation du côté client des paramètres donnés par le client à la fonction appelée, ainsi que la reconstitution de ces paramètres du côté serveur pour l'appel de la fonction. DCOM reste également ouvert et permet au programmeur (s'il en a les compétences) de réaliser son propre marshalling, ou de contrôler son propre protocole de communication.
La figure suivante montre schématiquement la frontière entre le client et le serveur lors de l'appel.



Il n'y a pas de différences fondamentales entre COM et DCOM, puisque ce dernier ne sert finalement « qu'à » faire communiquer des composants distants reliés par un réseau. Nous verrons par la suite comment configurer une machine pour qu'elle puisse utiliser DCOM.

2.3 DCOM dans les technologies de composant de Microsoft



2.4 Considérations de conception propres aux objets DCOM

Bien qu'il soit possible de travailler avec des composants COM de manière distante (en utilisant DCOM), il vous faudra certainement repenser le design de votre composant avant de le mettre en production. Les compromis de conceptions qui étaient acceptables avec un objet COM ne le sont plus dès que celui-ci est utilisé sur un réseau.

Premièrement, vous devez garder à l'esprit que vous aurez deux programmes qui communiqueront au travers d'un réseau. La manière sont ils seront reliés n'a que peu d'importance, il ne sera pas possible d'échanger des données aussi vite qu'il est possible de le faire sur une machine. De plus, si votre réseau est de type 10BaseT ou 100BaseT, il faudra penser à partager la bande passante entre toutes les machines connectées. Même si le réseau est suffisamment rapide pour traiter les communications entre vos deux machines comme sur une seule, le fait que d'autres ordinateurs soient présents sur ce réseau n'arrangera rien à l'affaire. La première règle est donc de minimiser la quantité d'informations déplacée d'un ordinateur à un autre.

La seconde règle découle de la première : chaque fois que vous envoyez un message d'une machine à une autre, une quantité non négligeable d'informations – dont la taille est indépendante de la taille du message - est incluse automatiquement dans celui-ci. Il vaut donc mieux envoyer un gros message qu'une centaine de petits messages. Autrement dit, après avoir minimisé la taille des messages, il faut les envoyer dans des paquets les plus gros possibles.

Relativement à Visual Basic, voici quelques règles qui aident à réaliser ces idées :

→ Minimisez la quantité de données envoyée entre deux ordinateurs en transférant uniquement ce qui est nécessaire.

→ Utilisez le mot-clé `ByVal` pour envoyer l'information dans un sens seulement. Sans ce mot-clé, DCOM devra retourner la valeur même si elle n'a pas été modifiée.

→ Utilisez des objets `Recordset` (déconnectés) d'ADO[6] pour transférer l'information entre machines. ADO met en place un marshalling spécial pour améliorer les performances de communications sur un réseau.

→ Utilisez des méthodes qui mettent à jour les propriétés de manière globale, plutôt que de mettre à jour les propriétés une à une.

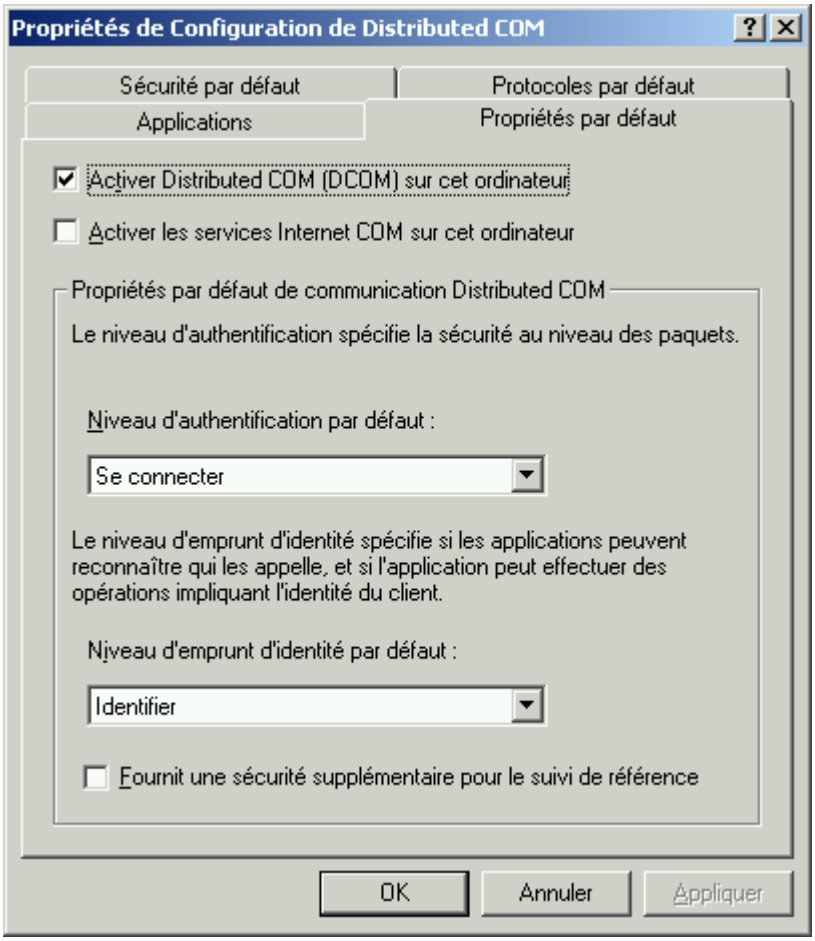
[6] ADO (ActiveX Data Object) : voir glossaire.

3 - Utilitaire de configuration DCOM

Ce qui différencie COM de DCOM est son outil de configuration, qui permet de régler comment et où les serveurs out-of-process sont créés en éditant les entrées de la base de registres associées avec un serveur out-of-process COM « normal ». Techniquement, cela évite ainsi d'avoir a spécifier le nom d'un ordinateur distant dans l'instruction `CreateObject`, mais aussi de pouvoir utiliser le mot-clé `New` dans les instructions `Set` et `Dim`, afin de créer des objets distants.
Cet outil de configuration permet également de contrôler comment l'objet est sécurisé. Il est possible de travailler en relation avec les règles de sécurité de Windows NT pour décider de règles très précises.

Il existe deux versions de l'utilitaire de configurations : une pour Windows 2000 et NT, l'autre pour Windows 95 et 98. Les différences entre ces deux versions se situe essentiellement dans les options de sécurité. Comme les OS Windows 95 et 98 ne sont de loin pas complets en matière de sécurité, les options sécuritaires de DCOM se résument à autoriser ou non l'accès à un composant selon le nom d'utilisateur. Nous détaillerons donc l'utilitaire pour Windows 2000 et NT, dans la mesure où le travail de l'auteur a été effectué sur ces deux OS.

L'utilitaire se lance avec la commande DCOMCNFG dans la boîte de dialogue Exécuter (Menu Démarrer → Exécuter…).



La case Activer Distributed COM… de l'onglet **Propriétés par défaut** est certainement la plus importante : elle soit être cochée afin de pouvoir exécuter DCOM sur la machine. La case Activer les services… permet le support du protocole de transport Tunnelling TCP, sur le port 80, afin de communiquer plus facilement à travers les proxy et firewalls[7].

[7] Cependant, tant que les objets CIS (COM Internet Services) devront être développés sous Visual C++, cette possibilité ne pourra s'appliquer à Visual Basic.

Le niveau d'identification devrait être placé sur Aucun lors du développement et du test des composants, ce qui permettra d'éliminer la plupart des erreurs rencontrées (on se penchera alors plus particulièrement sur les erreurs de COM).
Attention à ne jamais laisser ce niveau à Aucun en phase de production, pour des raisons évidentes de sécurité (infiltrations externes).

Les différentes valeurs du niveau d'identification sont les suivantes :

Aucun	Aucune vérification de sécurité. Devrait être utilisé lorsque le niveau d'emprunt d'identité est fixé à Anonyme.
Appel	Une vérification est effectuée lors de chaque appel.
Confidentialité du paquet	Les données, l'identité et la signatures de l'émetteur sont encryptés pour assurer une sécurité maximale.
Intégrité du paquet	L'identité et la signature de l'émetteur sont encryptés dans chaque paquet pour assurer que ceux-ci arrivent inchangés.
Paquet	L'identité de l'émetteur est encryptée dans chaque paquet.
Par défaut	Pour Windows 2000 et NT, identique à Se connecter.
Se Connecter	Les vérifications de sécurité sont effectuées quand la connection est faite avec le serveur. Cette option ne fonctionne qu'avec des protocoles orientés connections.

Le niveau d'emprunt d'identité décrit comment le nom d'utilisateur sera consigné sur le serveur. La valeur Anonyme ne devrait être utilisée uniquement que dans la phase de test.

Les valeurs sont les suivantes :

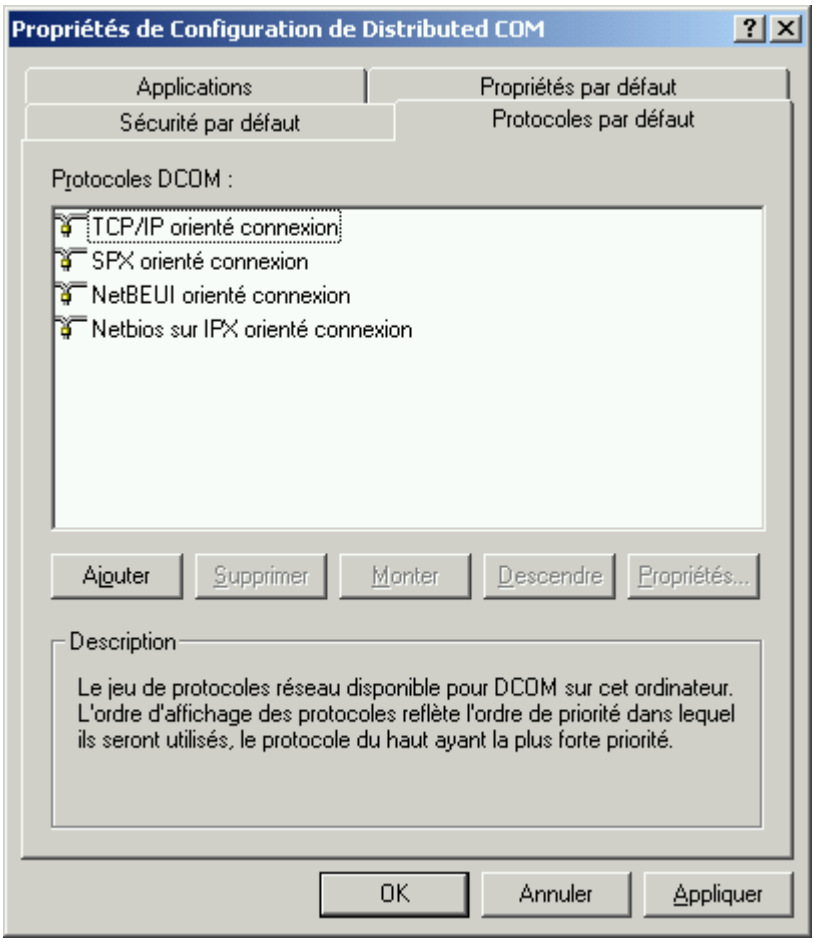
Anonyme	Le serveur d'applications ne vérifie pas l'identité de l'appelant.
Déléguer	Le serveur d'applications accomplit certaines tâches sur l'ordinateur du client. Cette valeur n'a jamais été implémentée par l'utilitaire.
Emprunter l'identité	Le serveur d'applications emprunte l'identité du client. Cette valeur n'est disponible que sur la machine qui fait office de serveur.
Identifier	Le serveur d'applications vérifie le nom d'utilisateur associé avec l'application cliente.

La dernière case à cocher (Fournit une sécurité…) demande au serveur de compter le nombre de clients connectés en plus du nombre de références faites sur l'objet COM. Cocher cette case permet d'éviter au programmeur d'initialiser le compteur de références. Du point de vue de Visual Basic, cette case ne sert à rien, puisqu'il n'est pas possible de modifier ce compteur explicitement.

L'onglet **Sécurité par défaut** est utilisé pour spécifier quel utilisateur ou groupe d'utilisateur peut accéder à un objet, l'exécuter, ou configurer ses permissions. Ces valeurs sont modifiables donc relativement à un objet donné.

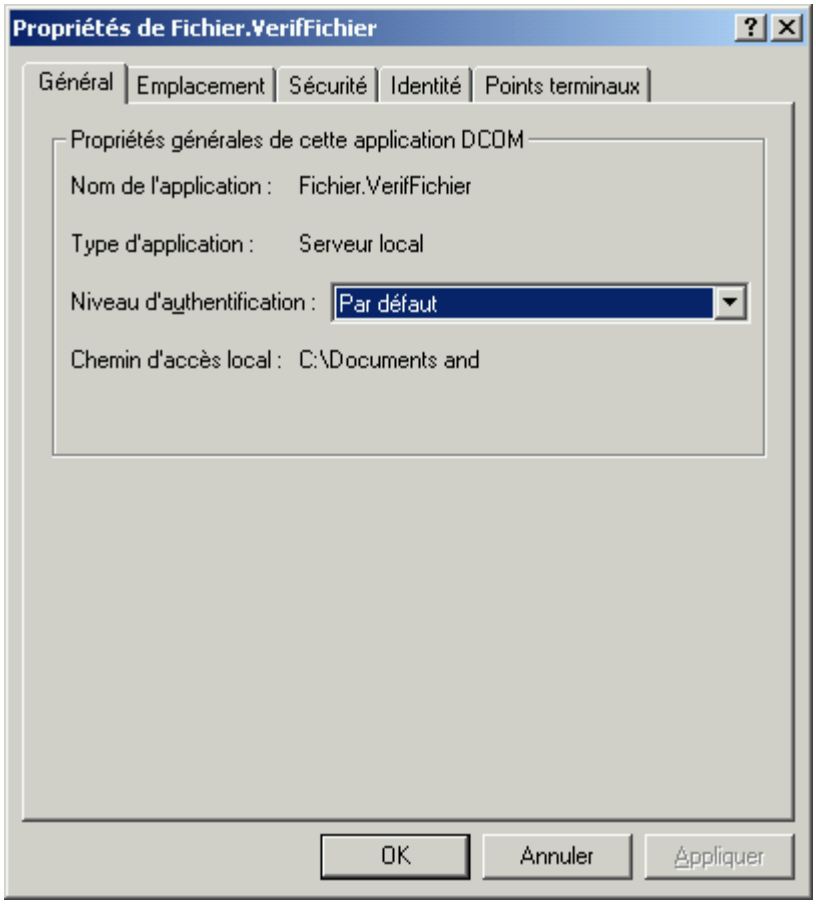
La gestion des utilisateurs étant presque identique à celle de Windows 2000/NT, nous ne l'expliquerons pas ici.

Notons seulement que de manière générale, il est plus pratique (et sûr) de gérer des groupes d'utilisateurs que des individus. Si aucun groupe déjà créé n'est approprié pour vos applications DCOM, vous pouvez en créer avec Active Directory (Menu Démarrer → Programmes → Outils d'administration → Active Directory, ou Gestion de l'ordinateur).
Windows 2000/NT supporte les principaux protocoles réseau travaillant en collaboration avec DCOM. Depuis l'onglet **Protocoles par défaut**, il est possible de les spécifier.



L'ordre dans la liste est important, car DCOM essaiera d'établir une connexion avec le protocole listé en premier. Si ce n'est pas possible, il tentera chaque protocole jusqu'à ce qu'il en trouve un qui fonctionne.

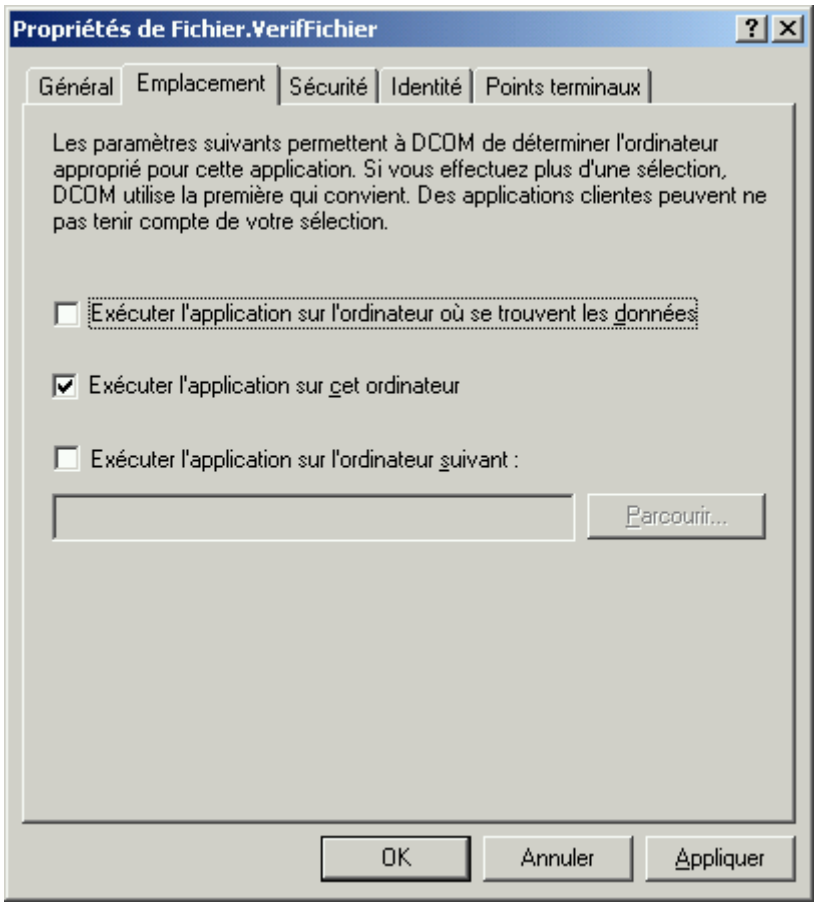
Voyons maintenant le contenu de l'onglet Applications. Sélectionnez `Fichier.VerifFichier`[8] et cliquez sur Propriétés.



Toutes les valeurs que vous saisirez dans cette boîte de dialogue écraseront celles définies par le système. Il est ainsi possible d'ajouter des contraintes sécuritaires sur un objet (par exemple limiter son accès à seulement quelques personnes, ou supprimer toutes les sécurité lors d'une phase de test).

Depuis l'onglet **Général**, on ne peut modifier que le niveau d'authentification (les valeurs – qui sont identiques à celles de l'onglet Propriétés par défaut - sont listées plus haut.
Le reste des informations de cette fenêtre ne sont que des confirmations d'informations qui sont modifiable ailleurs dans l'utilitaire. Par exemple, le type d'application est contrôlé par les valeurs que vous aurez saisies sous l'onglet **Emplacement**, et le chemin d'accès local est défini lors de l'inscription (registration) de l'objet COM sur votre machine.

[8] La présence de cette application nécessite que vous ayez réalisé l'EXE ActiveX « Fichier ». Si ce n'est pas le cas, sélectionnez n'importe quelle autre application. Notez que si l'utilitaire ne peut identifier le nom de l'application, il listera son GUID à la place.



L'onglet suivant vous propose 3 choix :

Le premier permet de travailler en relation avec la fonction `GetObject`[9] et de choisir d'utiliser l'ordinateur spécifié par son nom.

Le second (coché par défaut) exécute l'objet localement.

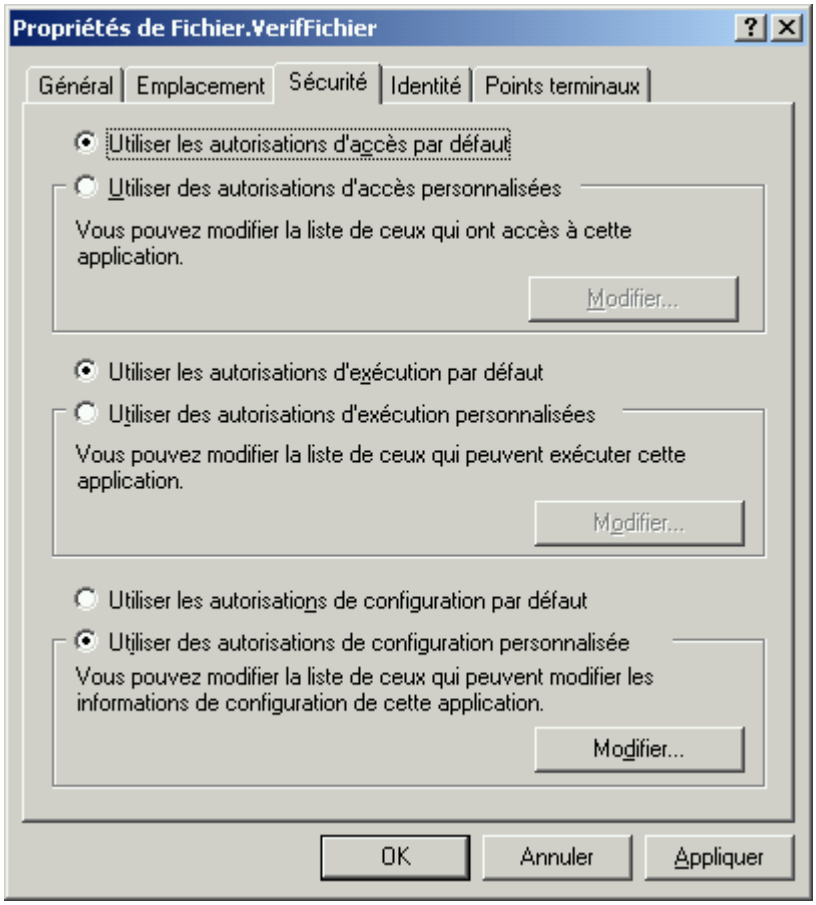
Le troisième vous permet de spécifier l'ordinateur sur lequel exécuter l'objet.

Imaginons que vous avez coché les trois cases. Si vos données sont situées sur un système distant, DCOM va essayer de charger l'objet sur cette machine en question (cette option ne fonctionne que si vous avez créé un objet basé sur des données, comme une feuille Excel ou un document Word). Si l'objet n'est pas présent sur la machine, DCOM va essayer de charger l'objet localement. S'il n'y parvient pas, il va essayer de charger l'objet sur l'ordinateur spécifié. Si cela ne donne rien, DCOM retournera un message d'erreur.

La case Exécuter l'application sur l'ordinateur suivant fait essentiellement la même chose que de spécifier le nom de l'ordinateur dans l'instruction `CreateObject`. L'intérêt de le faire ici et non dans `CreateObject` est qu'il n'y a dès lors plus besoin de se préoccuper de ce que le code contient (codage dur !). Si vous désirez déplacer l'objet d'un ordinateur à un autre, il suffit simplement de le signaler dans l'utilitaire de configuration, et le code du programme reste inchangé. Notez que la valeur indiquée dans le code écrase toute autre valeur.

[9] La fonction `CreateObject` est similaire aux instructions `Set…New` et `Dim…New`. Elle crée une nouvelle instance de l'objet spécifié et retourne une référence sur l'objet que l'on peut assigner à une variable objet avec `Set`. La syntaxe de cette fonction est « `Set varobjet = CreateObject (class [, serveur])` », dans laquelle `varobjet` est le nom de la variable objet qui contiendra un pointeur vers le nouvel objet ainsi créé, `class` est le nom de l'objet à créer, et `serveur` le nom de la machine où l'objet sera créé (la machine locale par défaut). Cette fonction est l'unique moyen de créer un objet sur une autre machine, en utilisant les techniques standards de COM. Si l'ordinateur distant n'existe pas ou que la sécurité empêche d'accéder à l'objet, une erreur d'exécution sera générée, et l'objet ne sera pas créé.

Sous l'onglet **Sécurité**, on trouve des options similaires à celle de l'onglet Sécurité par défaut, si ce n'est qu'ici vous est offerte la possibilité d'utiliser les autorisations par défaut, ou de personnaliser les autorisations par individu ou par groupe d'individus qui peuvent accéder à l'objet.

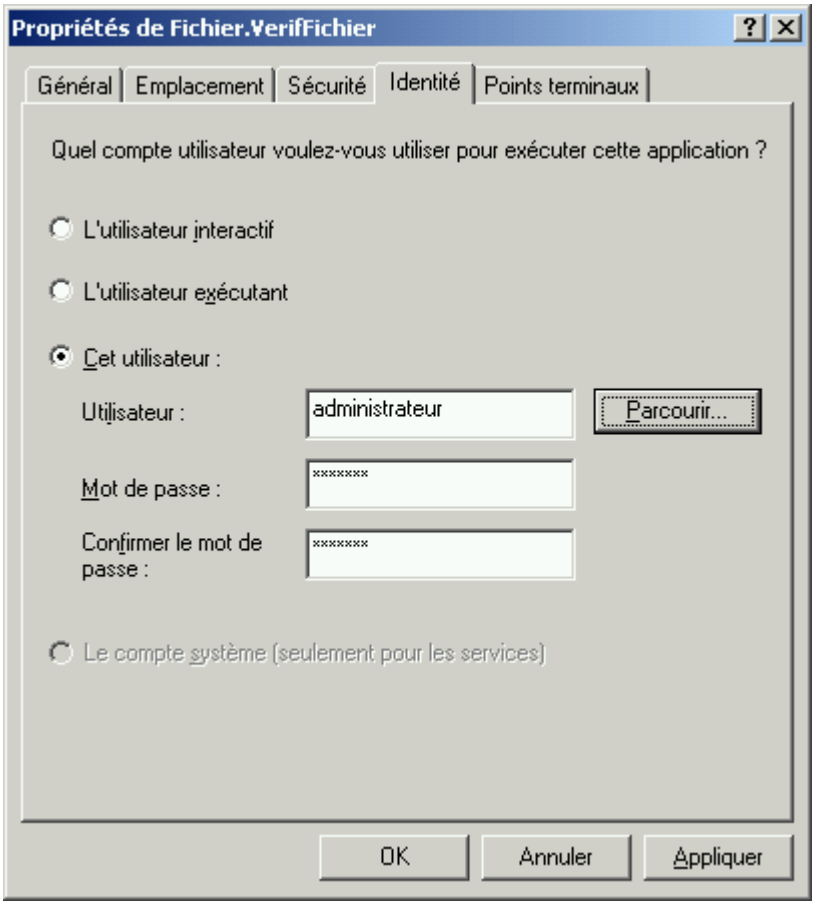


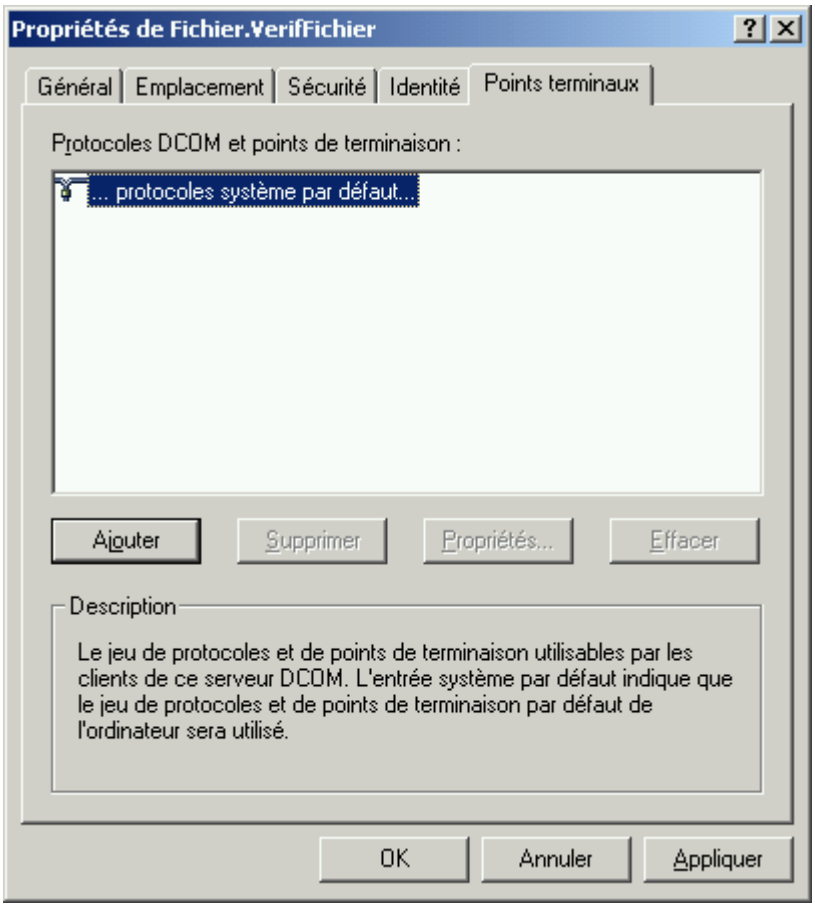
L'onglet **Identité** permet de spécifier quel profil utiliser pour exécuter l'objet :

Utilisateur interactif
Sélectionne le nom (d'utilisateur) de celui qui crée l'instance de l'objet, ce qui signifie que le nom de l'utilisateur qui ouvre une session sur la console du serveur sera exploité pour déterminer les réglages de sécurité. Si aucun utilisateur n'est loggé sur la console, une erreur sera retournée et tout ordinateur distant essayant de créer une instance de l'objet recevra une erreur.

Utilisateur exécutant
Sélectionne le nom de la personne qui a exécuté l'objet en premier. Si vous sélectionnez cette option, le profil de sécurité de cette personne sera appliqué.

Cet utilisateur…
Si vous sélectionnez cette option, le nom d'utilisateur spécifié déterminera les droits de sécurité de l'objet COM. Attention : utiliser cette option peut devenir ennuyeux à la longue si la politique de sécurité de votre système (ou de votre entreprise) nécessite de devoir changer périodiquement le mot de passe.





L'onglet **Points terminaux** vous permet de lister les protocoles de communications pouvant être utilisés avec votre application COM. Il est également possible de régler des options spécifiques pour chaque protocole.

4 - Erreurs de DCOM fréquemment rencontrées

La plupart des erreurs rencontrées – et c'est particulièrement le cas sous un serveur Windows 2000/NT – sont en relation avec la sécurité. Le meilleur moyen de régler ce genre de problème est de fixer des règles de sécurité qui fonctionnent : certains auteurs recommandent de commencer par supprimer toute sécurité, puis de spécifier une règle après l'autre, de tester et de continuer jusqu'à ce l'erreur se produise. A ce moment là, il devient plus simple de comprendre pourquoi le problème apparaît.

Les trois erreurs les plus fréquentes sont les suivantes :

Erreur 70 : Permission refusée

Marche à suivre (entre chaque point vous devrez réessayer d'accéder à l'objet) :

1/ Vérifiez que DCOM est activé aussi bien sur le serveur que sur le client.

2/ Vérifiez que le nom d'utilisateur associé à la machine locale a les permissions pour accéder l'ordinateur distant ainsi que l'objet distant.

3/ Utilisez l'utilitaire de configuration de DCOM pour faire les changements suivants sur le serveur distant DCOM et réessayez :

a/ Modifiez les propriétés de sécurité de l'objet de telle manière que les utilisateurs tout le monde, système et utilisateur interactif ait les autorisations de lancement de l'objet (pour l'accès et pour l'exécution).

b/ Sous le même onglet (Sécurité), régler les autorisations de configuration de telle manière que les utilisateurs créateur propriétaire, système ainsi qu'utilisateur interactif aient le contrôle total, et que l'utilisateur tout le monde ait la permission en lecture seule.

c/ Sous l'onglet Identité, sélectionnez l'utilisateur interactif ou spécifiez un nom d'utilisateur d'un utilisateur appartenant au groupe des administrateurs. Vérifiez qu'un mot de passe n'est pas requis.

4/ Si cela ne fonctionne toujours pas, régler le niveau d'authentification par défaut sur aucun, ainsi que le niveau d'emprunt d'identité par défaut sur emprunter l'identité.

5/ Si cela ne fonctionne toujours pas, redémarrer la machine cliente ainsi que le serveur : certaines modifications peuvent nécessiter le redémarrage de la machine.

Erreur 429 : Le composant ActiveX ne peut pas créer l'objet

Points à vérifier :

- Vérifiez que DCOM est activé aussi bien sur le serveur que sur le client.
- Vérifier que le serveur DCOM est bien inscrit (registered) sur l'ordinateur client.
- Vérifier que l'objet distant existe sur la machine distante et qu'il est bien inscrit. Il est possible que quelqu'un ait désinscrit (unregistered) le serveur DCOM. Il se

peut aussi que lors d'une mise à niveau du serveur DCOM, celui-ci ait acquis un nouveau GUID.

- Si le programme fonctionne, mais ne fait rien et ne répond pas, vérifiez sur l'ordinateur serveur si le processeur ne tourne pas à 100%. Ceci est causé par un problème de sécurité lorsque l'objet distant essaie d'écrire quelque chose dans la base de registres, mais que le nom d'utilisateur n'a pas les permissions pour le faire. Pour y remédier, changez le nom d'utilisateur associé à l'objet distant, ou mettez à jour ses permissions.

Erreur 462 : La machine serveur distante n'existe pas ou est indisponible

Points à vérifier :

- Vérifiez que le nom de la machine distante est correctement orthographié.

- Vérifiez que vous pouvez vous connecter sur la machine distante. Il est possible que celle-ci soit débranchée ou qu'il y ait un problème sur le réseau impliquant les deux machines.

5 - Exemple de composant DCOM

Nous allons réaliser un exemple simple d'application client/serveur utilisant DCOM. Il est nécessaire de travailler avec l'édition Entreprise de Visual Basic 6 pour ce faire. Ce sera également l'occasion d'utiliser l'assistant d'empaquetage et de déploiement. Notre application, lancée depuis la machine cliente, affichera l'heure de la machine sur laquelle s'exécutera le serveur.
Avant de commencer, il faut activer DCOM (avec l'utilitaire Dcomcnfg que nous avons vu auparavant) sur les machines cliente et serveur.

Créez un répertoire nommé DCOMDemo et logez-y deux répertoires nommés Server et Client.

5.1 Création du serveur

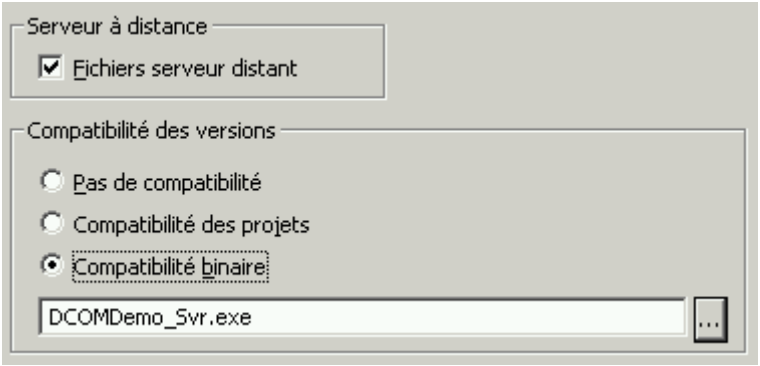
Lancez Visual Basic 6, et sélectionnez un projet ActiveX EXE. Ajoutez le code suivant à la classe créée :

```
Public Function ServerTime() As String
      ServerTime = Time
End Function
```

Dans les propriétés du projet, saisissez DCOMDemo_Svr dans le champ Nom du projet, ainsi que DCOMDemo_Svr – Server dans le champ Description du projet. Cochez l'option Mode d'exécution autonome. Cette option devrait toujours être cochée lorsque le serveur ne possède aucune interface utilisateur, car elle assure qu'aucune boîte de dialogue ne s'affichera dans le processus d'exécution du serveur. Si le serveur est exécuté dans le mode Utilisateur Interactif (dans Dcomcnfg) et qu'effectivement il n'y a aucune interaction de la part de l'utilisateur sur le serveur, celui-ci pourrait utiliser l'intégralité du temps CPU.

Sous l'onglet Composant des propriétés du projet, cochez la case Fichier serveur distant, ce qui aura pour but de générer les fichier VBR et TLB requis pour l'empaquetage du programme client s'adressant au serveur. Ces fichiers contiennent des entrées de la base de registres qui seront inscrites sur la machine cliente.
Validez, sauvegardez le fichier et le projet, et compilez le serveur (Fichier → Créer DCOMDemo_Svr.exe).

Retournez sous les propriétés du projet, et sous l'onglet Composant, cochez l'option Compatibilité binaire, en vérifiant que le programme spécifié dans le champ soit bien DCOMDemo_Svr.exe. Cette manœuvre permet d'assurer que tous les GUID seront les mêmes lors des multiples compilations du serveur.



5.2 Création du client

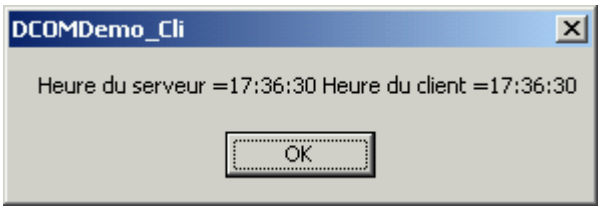
Ouvrez un nouveau projet standard EXE, et allez dans les propriétés du projet. Sous l'onglet Général, saisissez DCOMDemo_Cli comme nom de projet, et DCOMDemo_Cli – Client comme description du projet, et validez.

Dans les références (Projet → Références), sélectionnez DCOMDemo_Svr – Server (en cochant la case correspondante) et validez.

Placez un bouton sur la feuille, nommé Run, et ajoutez le code suivant dans son événement Click :

```
Private Sub Command1_Click()
Dim MonObjet As DCOMDemo_Svr.Class1
On Error GoTo Err

Set MonObjet = CreateObject("DCOMDemo_Svr.Class1")
MsgBox "Heure du serveur =" & MonObjet.ServerTime & _
    "Heure du client =" & Time

Exit Sub

Err:
    MsgBox " la connexion a échoué : Erreur " & _
             Err.Number & "-" & Err.Description

End Sub
```

Sauvegardez le fichier et le projet dans le répertoire Client, puis lancer l'exécution de celui-ci (F5). Les deux heures affichées sont identiques : c'est normal puisque le client est exécuté sur la même machine que le serveur.

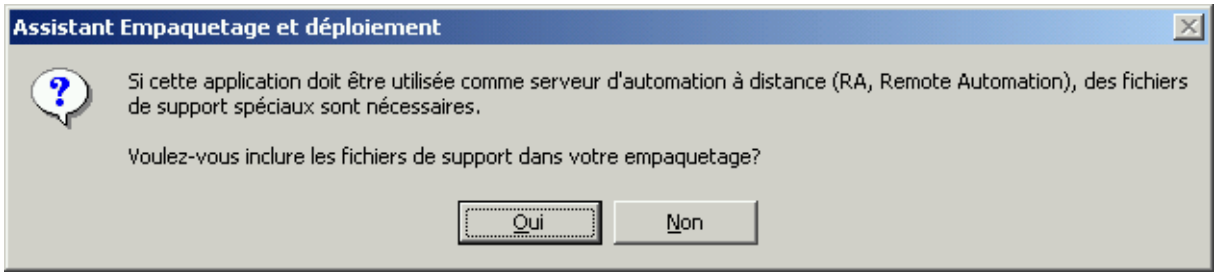


Générez maintenant l'exécutable (Fichier → Créer DCOMDemo_Cli.exe) et fermez Visual Basic 6.

5.3 Empaquetage du serveur

Lancez l'assistant d'empaquetage et de déploiement (Menu Démarrer → Programmes → Microsoft Visual Studio 6.0 → Outils Microsoft Visual Studio 6.0 → Assistant Empaquetage et déploiement).

Sélectionnez le projet DCOMDemo_Svr.vbp, et cliquez sur Empaquetage. Choisissez le type d'empaquetage Logiciel d'installation standard, et validez. Le programme va ensuite afficher la boîte de dialogue suivante :



Cliquez sur Non. DCOM n'est pas Remote Automation, en fait cette dernière technologie a été remplacée par DCOM.

L'écran suivant vous propose une liste de fichiers à empaqueter avec le programme DCOMDemo_Svr. Il vaut mieux, à moins que vous ne sachiez quel est le rôle des fichiers, cliquer sur Suivant en acceptant cette configuration. Vous pouvez ensuite définir les groupes et les éléments du menu Démarrer qui seront créés par le processus d'installation (à partir desquels le programme pourra être lancé). Dans la fenêtre Fichiers partagés, cliquez sur le fichier DCOMDemo_svr.exe si vous désirez obtenir plusieurs instances d'objets depuis plusieurs machines différentes. Validez pour terminez l'empaquetage.

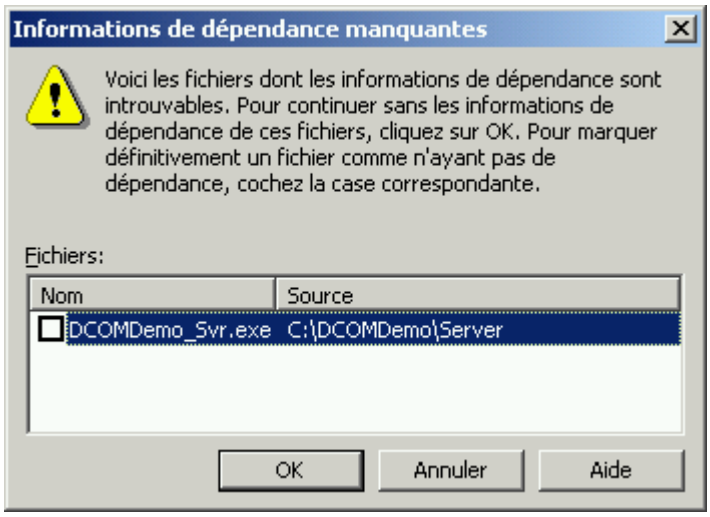
Les fichiers générés sont stockés dans le répertoire Package. Certes, ce répertoire pèse plus de 4 Mo, mais n'oubliez pas qu'il contient toutes les librairies requises pour son installation (il est donc possible d'installer le serveur sur une machine sur laquelle ne figure pas Visual Studio ni Visual Basic).

5.4 Empaquetage du client

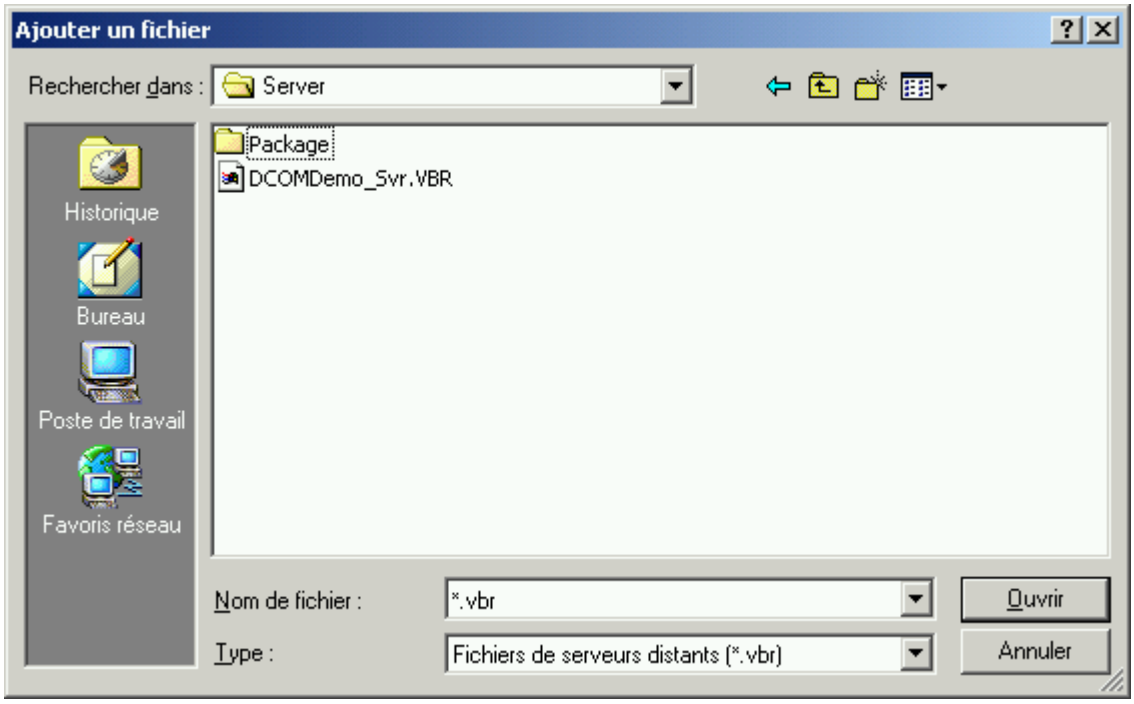
L'opération de l'empaquetage du client est sensiblement différente de celle du serveur, dans la mesure où celui-ci ne tournera pas sur la même machine que le client : ce changement concerne essentiellement l'installation de la librairie de type (fichier .tlb) ainsi que l'inscription de plusieurs entrées dans la base de registres.

La marche à suivre est la suivante :
Après avoir sélectionné le projet DCOMDemo_Cli.exe et cliqué sur Empaqueter, choisissez à nouveau le type d'empaquetage standard, puis l'emplacement où vous désirez enregistrer les fichiers.

La boîte de dialogue suivante apparaît :

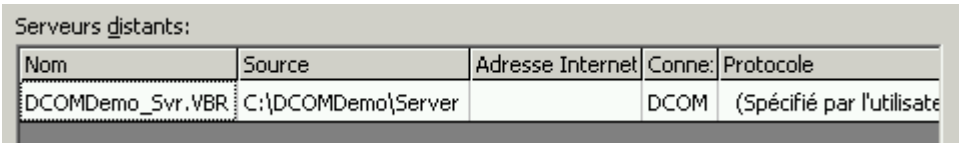


Cliquez sur OK sans cocher la case, le serveur n'ayant aucune dépendances.
Dans la fenêtre Fichiers Inclus, désélectionnez le programme DCOMDemo_Srv.exe, puisque vous ne voulez pas le distribuer avec DCOMDemo_Cli.exe. Cliquez ensuite sur le bouton Ajouter, et recherchez le fichier DCOMDemo_Svr.vbr dans le répertoire du serveur (DCOMDemo\Server). Ce fichier contient les entrées relatives au serveur à ajouter dans la base de registres.



Validez. Vous constaterez que deux fichiers sont apparus dans la liste des fichiers inclus : DCOMDemo_Svr.VBR et DCOMDemo_Svr.TLB.

Dans la fenêtre Serveurs distants, il est possible de spécifier le nom de la machine sur laquelle s'exécutera le serveur.



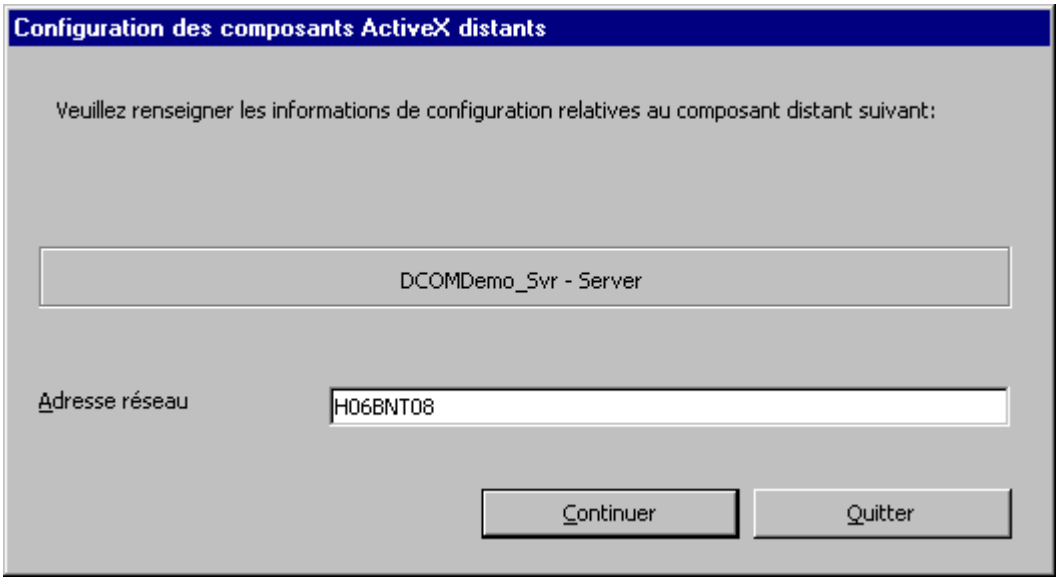
Généralement, on laisse cette case vide, étant donné qu'on ne sait pas forcément à l'avance où le serveur sera installé. Nous verrons plus loin que c'est lors de l'installation que le programme nous demandera de spécifier cette adresse. Cliquez sur Suivant sur toutes les fenêtres suivantes afin de terminer le processus d'empaquetage.

5.5 Installation du serveur

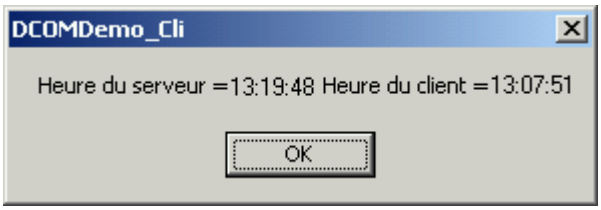
Installer le serveur sur la machine où vous voulez qu'il s'exécute, en utilisant le package que vous venez de créer. Si vous désirez le faire sur la machine utilisée pour le développement, ce n'est pas nécessaire, puisque Visual Basic effectue l'inscription au moment où il a compilé le serveur.

5.6 Installation du client

Installer le client sur la machine où vous voulez qu'il s'exécute. Lors de l'installation, étant donné que le champ de l'adresse de la machine distante a été laissé blanc, le programme vous demande la localisation de celle-ci.



Voilà, il ne vous reste plus qu'à lancer le serveur, puis le client. A l'exécution de celui-ci, la fenêtre indiquant l'heure des deux machines n'est plus la même que lors de la première exécution (notez que dans l'exemple concret – relativement à la figure suivante – les horloges des deux machines étaient particulièrement mal réglées…).



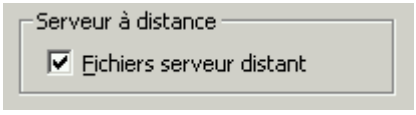
5.7 Réglage de la sécurité du serveur

En utilisant Dcomcnfg, vous pouvez gérer toutes les options de sécurité que nous avons vu auparavant. En phase de test, il est conseillé de désactiver toute barrière qui pourrait rendre l'exécution instable, puis de placer les sécurités les unes après les autres, en prenant soin de tester l'application après chaque manœuvre. Cette phase peut être très longue en raison de la spécificité de chaque réseau.

6 - Cycle de vie d'un composant distant

NB. Attention à ne pas confondre : une machine serveur (ou machine distante) est considérée comme étant l'ordinateur qui fait office de serveur, alors qu'un serveur distant (ou serveur tout court) est considéré comme un composant fournissant des services.

Comme nous l'avons vu lors du cycle de vie d'une EXE ActiveX, un composant distant est inscrit sur une machine cliente à l'aide du programme CliReg32.exe.
Pour ce faire, il faut tout d'abord cocher la case « Fichier serveur distant » sous l'onglet Composant de la fenêtre de propriétés du projet. Puis, recompiler le tout, et lancer CliReg32.exe à la ligne de commande (qui doit normalement se situer dans le répertoire Microsoft Visual Studio\Common\Tools\..).



La compilation du composant distant génère un fichier d'extension .VBR, qui contient des informations à inscrire dans la base de registre. Le programme CliReg32 s'en charge : il faut donc simplement désigner le nom de fichier .VBR après « CliReg32 », et le tour est joué[10].

Au moment de l'inscription (Registration Time)

Bien que le concept théorique de DCOM (ou de Remote Automation) soit relativement simple, il peut s'avérer nettement plus compliqué en pratique. L'idée véritablement importante à retenir est qu'il faut inscrire le composant sur les deux (ou plus) machines impliquées. Il doit être inscrit sur la machine serveur afin que DCOM sache quel serveur EXE charger et comment gérer le marshalling des données du côté du serveur. Mais il doit être également inscrit sur la machine cliente pour que DCOM puisse créer un objet proxy pour le composant et gérer le marshalling des données du côté du client.

Au moment de la conception et de la compilation

Après avoir inscrit le composant distant, on peut l'utiliser de la même manière qu'un composant EXE ActiveX. En effet, Visual Basic ne s'occupe pas de l'emplacement de celui-ci.

Au moment de l'exécution

Visual Basic, grâce au CLSID de l'objet, recherche un serveur EXE implémentant l'objet. Si il ne trouve pas de serveur local, il cherche un serveur distant. Puis, les choses suivantes se mettent en place :

→ Lorsqu'il trouve un serveur distant, il contacte le serveur et lui envoie une requête pour créer l'objet spécifié. Si le serveur accepte sa requête, il lance l'exécutable

[10] Notons qu'une dizaine d'options sont possibles sur la ligne de commande, pour les connaître il suffit de lancer le programme, elles sont indiquées (la MSDN, du moins la version de l'auteur, ne donne aucune information a ce sujet). La façon encore plus simple d'inscrire un composant sur la machine cliente est de l'empaqueter et de l'exécuter ainsi (toutes les manœuvres sont automatisées).

sur la machine distante. En même temps, le sous-système COM crée un objet proxy sur la machine locale.

→ L'objet proxy peut reproduire toutes les interfaces de l'objet « réel »[11]. Les propriétés et méthodes de l'interface de l'objet sont exécutées en appelant les fonctions dans l'interface de l'objet proxy. L'objet proxy utilise le mécanisme d'appel de procédure distante de l'OS pour empaqueter (marshal) les données à travers le réseau vers la machine distante. Du côté du serveur, l'OS appelle l'objet directement[12].

→ Toutes les valeurs ou paramètres passées par référence sont empaquetées (marshalling) en retour vers le système local, à travers l'objet proxy, au client utilisant l'objet.

→ Lorsque le projet est fermé ou que l'objet est référence sur `Nothing`, le compteur de références de l'objet passe à zéro et l'objet proxy est effacé. Lorsque le serveur distant sait que tous les objets qu'ils fournissait sont libérés, il se décharge.

[11] Il est alors possible d'utiliser la liaison précoce, en définissant des variables objet d'un type spécifique donné. Cependant, les bénéfices de cette technique (liaison précoce) dans ce genre de cas seront très minces, étant donné la charge du réseau.

[12] En utilisant Remote Automation en lieu et place de DCOM, on peut considérer une autre couche. Au lieu d'appeler l'objet distant directement, Windows appelle une application nommée Remote Automation Manager, qui gère les objets Remote Automation. Cette application appelle l'objet en utilisant les techniques standards de marshalling sur un composant out-of-process. En d'autres termes, il n'y a plus un canal, mais deux canaux – ou couches – de marshalling. Le premier le fait au travers du réseau, et le second sur la machine serveur. Il s'agit d'une des raisons pour laquelle le protocole DCOM offre de meilleures performances que Remote Automation.

7 - Des DLL distantes ?

Nous avons vu jusqu'ici que lorsqu'on parlait d'objets distribués avec DCOM, on parlait d'EXE ActiveX. Quand est-il des DLL ? Peut-on les déposer sur une machine distante ?
Le problème est le suivant : une DLL ne peut s'exécuter que dans un processus existant. Lorsqu'on crée un objet de manière distante, quel processus chargera le contenu de la DLL (ou les objets implémentés dans la DLL) ? Même si un tel processus existait (et il doit certainement exister des solutions logicielles qui résolvent le problème) et qu'il arrivait à empaqueter les données pour les envoyer sur l'OS de la machine cliente, au sein de quelle thread (processus léger) les objets s'exécuteraient-ils ? En admettant le cas où tous les composants partagent un même thread, des problèmes tels qu'encombrement des appels, ou performance globale sont quasi assurés de se produire.

En fait, DCOM n'est tout simplement pas assez sophistiqué pour résoudre ce genre de situations, il se limite donc à traiter les EXE ActiveX[13].

[13] Pour assurer qu'une solution basée sur serveurs DLL fonctionne, il faudrait un outil qui sache allouer des threads aux objets, gérer leur chargement et déchargement, contrôler la sécurité, etc. Il serait même possible (pour autant que toutes ces techniques soient parfaitement maîtrisées) de développer un EXE ActiveX qui offrirait ce genre d'outil. Ou alors utiliser MTS, dont un des buts est justement de travailler avec les DLL distantes…

Bibliographie critique et liens Internet

- 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 !

- http://msdn.microsoft.com, le lien INCONTOURNABLE pour développer sous plate-forme Windows.

- http://www.microsoft.com/com/tech/com.asp, la page de Microsoft pour tout ce qui concerne la technologie COM.

- http://www.microsoft.com/com/tech/DCOM.asp, la page de Microsoft consacrée à DCOM, on y télécharge entres autres des exemples, des SDK, ainsi que toute la spécification du modèle.

- http://www.componentsource.com, le site d'une entreprise qui vend des composants COM, Java, CORBA,… C'est une vraie mine d'informations, la rubrique consacrée à l'évolution des composants m'a servi dans la rédaction du rapport.