Présentation COM / DCOM / COM+

1. Historique

DDE

Tout d'abord cela à commencer par le presse-papiers (1987), qui permettait aux utilisateurs de copier des portions de données d'une application à l'autre. Cela fonctionnait bien pour des documents simples, mais lorsque ceux-ci étaient plus complexes, le presse-papiers atteignait ses limites. Plus tard, Microsoft développa la technologie DDE, Dynamic Data Exchange ou Echange dynamique de donnée, une « nouvelle génération » de presse-papiers, qui permet d'insérer des données à partir d'un document d'une autre application. Ces données restaient liées à l'application d'origine. Cette technologie ne fonctionnait pour les applications de Microsoft. A la différence de OLE, où on pourra lancer l'application d'origine en cliquant dans l'objet : feuille Excel dans l'application Word.

OLE1

OLE « Object Linking and Embedding », signifie « Intégration d'objets et Lien sur des objets » développé en 1991, le but était de permettre l'insertion des données provenant de plusieurs applications sur un seul document. Et cela soit par intégration complète, soit par référence (ce que l'on appelle une liaison). On peut voir cela dans la plupart des applications où apparaît dans le menu « Coller | Collage spécial ». Les objets intégrés ou liés sont capables de s'afficher dans l'application qui les contient.

OLE2

OLE1 était fondé sur des messages Windows mais, cela 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.

COM

COM « Component Object Model » ou Modèle d'objet composant est né avec OLE2. C'est une spécification de Microsoft qui décrit comment créer les objets réutilisables. Il a aussi pour but de définir un standard de communication (en fait des méthodes) afin d'accéder aux objets OLE localement. Nous détaillerons cette partie plus loin.

DCOM

Microsoft a ensuite fait évolué la technologie COM afin de permettre la répartition des composants sur un réseau. L'aspect distribué a donné le préfixe D pour Distributed COM, qui spécifie toujours les composants, mais cette fois distants et interopérables avec aux systèmes d'exploitation hétérogènes.

COM+

Arrivé sur le marché avec Windows 2000, COM+ est l'évolution de COM, qui incorpore 2 technologies MTS (Microsoft Transaction Server) et MSMQ (Microsoft Message Queues), détaillé plus loin. COM+ facilite le développement d'applications distribuées, en supprimant la complexité associée à la programmation, au débogage, au déploiement et à la maintenance d'une application qui repose sur COM pour certains services et sur MTS pour d'autres.

2. Que sont COM/DCOM/COM+ ?

Avant de rentrer dans le vif du sujet, nous allons expliciter la notion de composant.

QU'EST CE QU'UN COMPOSANT ?

Et bien on peut dire qu'un composant est un fournisseur de service qui permet à une application de faire appel à des opérations déjà existantes. Par exemple, un composant peut être un bouton qui a certaines caractéristiques, rond et rouge etc. Ainsi, la portion de code qui gère cet objet est regroupée dans un fichier qui une fois compilé peut s'inclure rapidement dans un autre programme. Ce peut être une DLL (**D**ynamic **L**ink **L**ibraries).

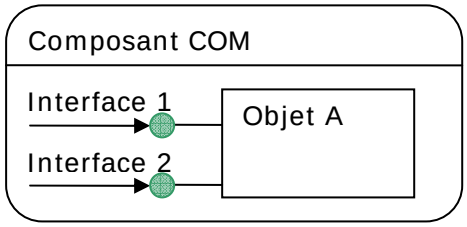
COM

COM « Component Object Model » ou Modèle d'objet composant est une spécification de Microsoft qui décrit comment créer les objets réutilisables. Pour bien comprendre la notion de COM, il est intéressant de se pencher sur les interfaces.

Les Interfaces

Une interface permet d'utiliser un composant, c'est une collection de fonctions (services) qui définissent le comportement des objets. Une Interface n'est ni une classe, ni un composant. C'est une liste qui contient les signatures des méthodes et non pas leur implémentation.
Les objets ne sont vus que par leur interface. Les objets clients invoquent les interfaces des objets serveur (qui fournissent des services). Et pour accéder à un objet, il faut avoir accès à une interface associée à cet objet, autre remarque : un objet peut avoir plusieurs interfaces.

Objet A supportant les interfaces 1 et 2



L'interface est l'unique partie du système qui est visible du client.

Pour être considérée comme une interface COM valide, celle-ci doit respecter les règles suivantes :

1. Une interface doit être signée par un identifiant unique GUI.

Plusieurs interfaces peuvent avoir le même nom, mais chacune est identifiée par son GUID Globally Unique Identifiers, identificateur unique qui est un entier de 128 bits et garanti être unique (temps+espace).

2. Une fois publiée, une interface est immuable (elle ne doit plus changer).

Les Interfaces sont au niveau binaire, c'est à dire des .exe, ce qui permet d'écrire un composant dans n'importe quel langage du moment qu'il peut fournir des pointeurs de fonctions. Les Interfaces n'évoluent jamais. Toute modification est obligatoire une nouvelle Interface unique (GUID).

3. Une interface doit dériver au moins de interface Unknown.

IUnknown est l'Interface de base que tout component COM doit implémenter, sinon ce n'est pas un composant COM, autrement dit sans interface, on ne pourrait interagir avec lui. Toutes les interfaces dérivent au moins de IUnknown. Les interfaces sont décrites en **IDL** (Interface Description Language qui est différente de IDL Corba) basé sur OSF (Open Software Foundation).

Voici le code IDL correspondant :

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

On remarque que l'interface IUnknown fournit 3 méthodes :

* AddRef et Release sont des méthodes qui gèrent le compteur de référence.

* QueryInterface met en place le mécanisme qui permet à un client de connaître toutes les interfaces supportées par le composant, retourne un pointeur vers les interfaces supportées
* AddRef est appelée à chaque fois que le composant COM est utilisé par un client, autrement dit, quand un nouveau pointeur est fourni pour une interface. Incrémente le compteur de référence
Release est appelée quand on cesse d'utiliser un pointeur vers une interface. Décrémente compteur de références.

Remarque : Lorsque le compteur de référence passe à zéro, le composant se détruit lui-même.

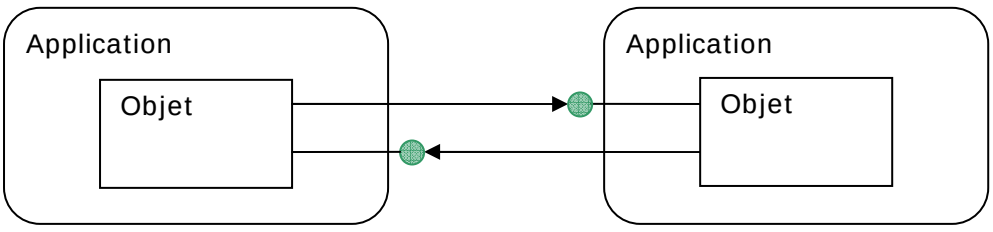
Conclusion : L'interface IUnknown joue 2 rôles, LE contrôle le cycle de vie d'un objet avec les méthodes AddRef() et Release(), ainsi que la découverte dynamique des Interfaces supportées par l'objet grâce à QueryInterface().

Les Objectifs de COM :

Voici les principaux objectifs de COM (Cf. Spécifications de COM) :

- Définir un protocole pour créer des composants.
- Réutilisation des composants
- Microsoft affirme que l'on peut accéder et développer les composants indépendamment du langage de programmation pourvu que le langage permette d'accéder aux pointeurs de fonctions. Il est possible d'utiliser par exemple : C, C++, Visual Basic Etc

Deux applications échangeant des services fournis par des objets en exposant des Interfaces



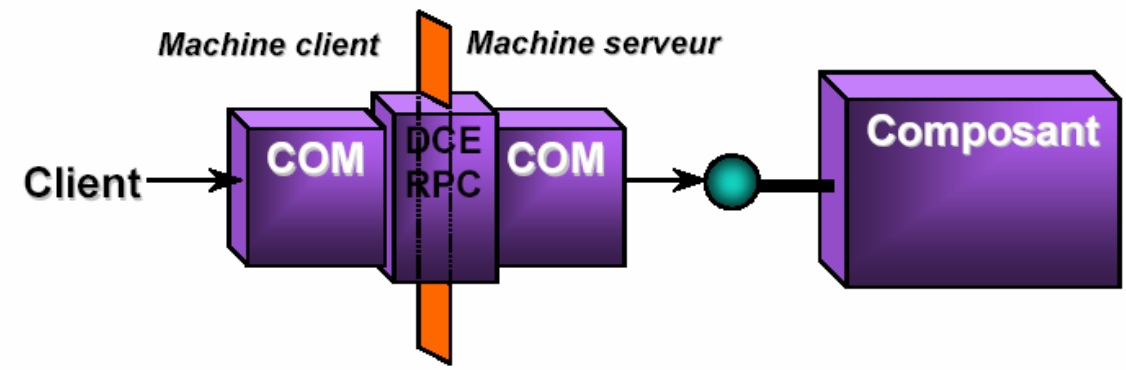
DCOM

Le modèle DCOM permet la communication entre des objets s'exécutant sur la **même machine**. Le modèle de composants distribués DCOM, étend COM afin de supporter les communications avec des objets situés sur des machines distantes, et sur des plates-formes différentes. Initialement développé pour Windows, DCOM est maintenant disponible sur des plates-formes UNIX, dont Solaris, Linux et HP/UX.

DCOM = COM + RPC (Remote Protocol Call)

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). Le fonctionnement est détaillé dans la partie architecture globale.

Schéma



Les Objectifs de DCOM :

- Les composants DCOM peuvent fonctionner dans des processus différents et sur des machines différentes.
- Possibilité de faire évoluer à tout moment les fonctionnalités d'une application (client, serveur ou composant COM). Cela se fait par QueryInterface que tous les composants COM supportent et qui permet de proposer de nouvelles fonctions à de nouveaux clients sans altérer la compatibilité avec les anciens. Les évolutions se font par de nouvelles Interfaces et un composant COM supporte toujours les vieilles Interfaces.
- Interactions simples et rapides. Les appels in-process se font par pointeurs.
- Réutilisation d'Interfaces.
- Transparence Local/Distant. Le standard binaire autorise le détournement des appels aux Interfaces pour les rediriger vers un objet distant de façon transparente pour l'appelant.
- Microsoft affirme que les composants DCOM sont indépendants du langage de programmation utilisé. A condition, que le langage ait accès aux pointeurs.

COM+

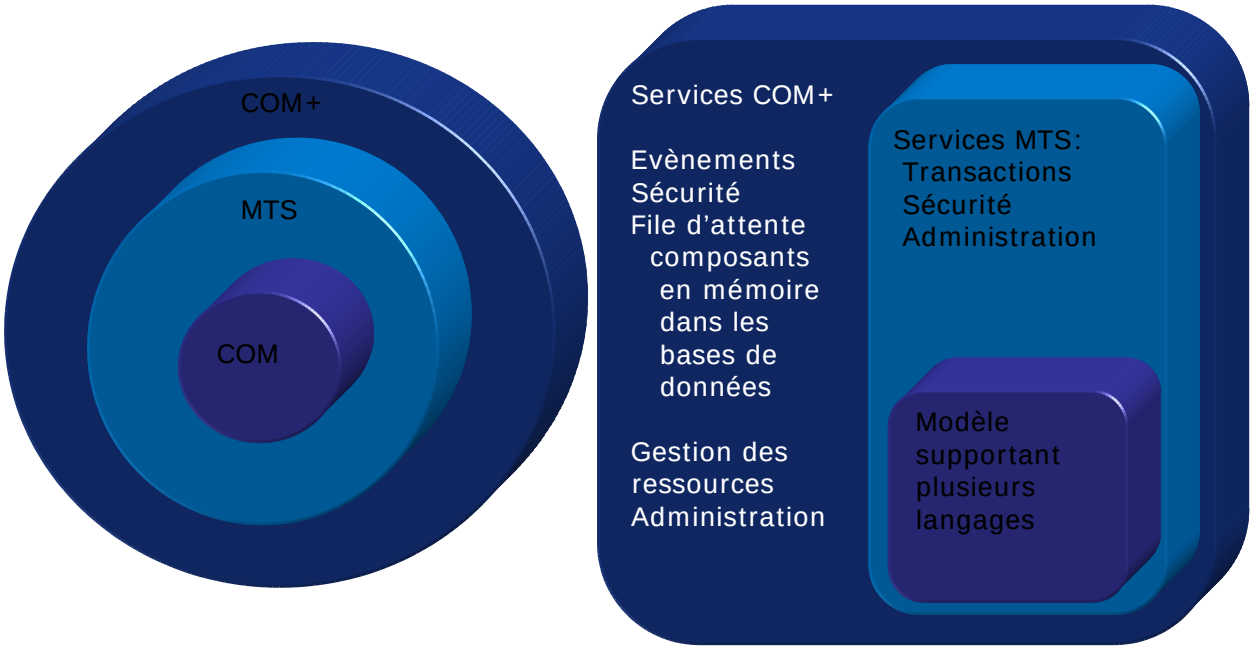
COM+ MTS = COM+

COM+ est apparu avec Windows 2000, qui apporte des services dédiés aux applications distribuées de Windows, à savoir : MTS [1], MSMQ [2] et autres. En plus de cela COM+ apporte de nouvelles fonctionnalités comme les queued components, un mécanisme de gestion des évènements et la répartition de charge.

COM+ incorpore :

- [1] MTS (Microsoft Transaction Server), permet au développeur de créer et utiliser des transactions
- [2] MSMQ (Microsoft Message Queues), un ensemble d'objets qui assure une exécution différée en mode asynchrone lorsque les composants travaillant en coopération sont déconnectés. Cette fonctionnalité vient s'ajouter au modèle de programmation client-serveur synchrone.
- IMDB (In-Memory Database system), qui permet de gérer des informations permanentes et des informations temporaires. In-Memory DataBase est un système de base de données entièrement transactionnel conçu pour fournir un accès rapide aux données.

COM+ facilite le développement d'applications distribuées, en supprimant la complexité associée au développement, au débogage, au déploiement et à la maintenance d'une application qui repose sur COM pour certains services et sur MTS pour d'autres.



Source Microsoft

Objectifs de COM+ :

- Répartition
- Exécution de composants dans un environnement réparti composé de machines hétérogènes
- Utilisation transparente des mécanismes de communication à distance
- Gestion transparente de la concurrence et de la synchronisation

Réutilisation

Interfaces multiples d'accès
Intégration au niveau du binaire (vs.du source dans CORBA)

Evolution

Remplacement du binaire par un binaire équivalent
Ajout de nouvelles interfaces

Homogénéité

Architecture commune à tous les sous-systèmes de Windows 2000 (DirectX, Active Directory, accès aux données…)

3. Architecture d'une application répartie utilisant COM/DCOM/COM+

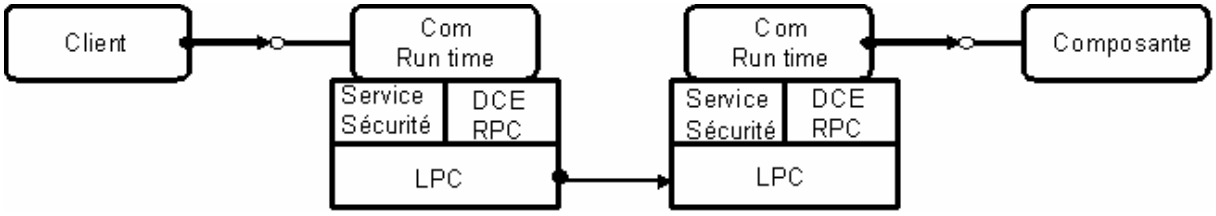
Qu'est ce qu'une application répartie ?

Il s'agit d'une application découpée en plusieurs unités :

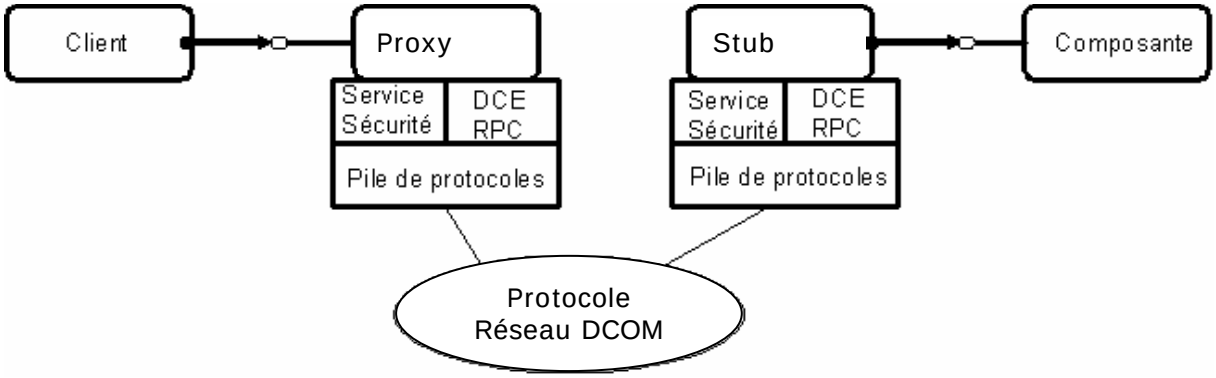
- Chaque unité peut être placée sur une machine différente
- Chaque unité peut s'exécuter sur un système différent
- Chaque unité peut être programmée dans un langage différent

Avantages des application réparties	**Inconvénients**
Décentralisation des responsabilités Découpage en unité - Modularité Fiabilité et disponibilité Individualisation des défaillances Duplication des composants de l'application Performance Partage de la charge Maintenance et évolution	Une mise en œuvre plus délicate Gestion des erreurs Suivi des exécutions Pas de vision globale instantanée Délais des transmissions Administration plus lourde Installation Configuration Surveillance Coût Formation Achat des environnements

Architecture d'une application non répartie COM :



Architecture d'une application répartie DCOM :

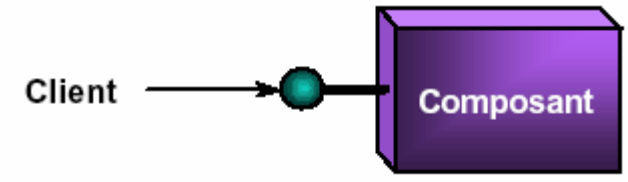


4. Architecture & Explication de COM/DCOM/COM+

En COM/DCOM on distingue trois types de composants :

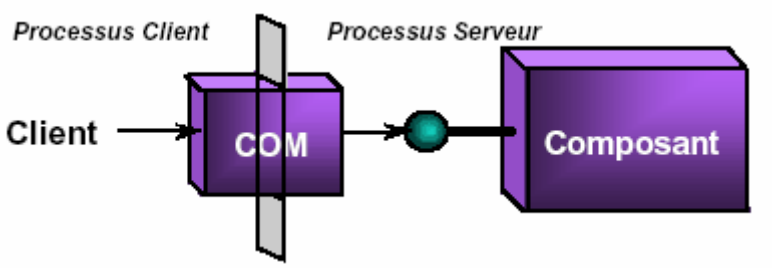
1. Serveur « In-Process » : il s'exécute dans le processus de l'appelant. Tous les composants de ce type sont des DLL (ou des contrôles ActiveX).

Même Processus
→ Appel de fonction



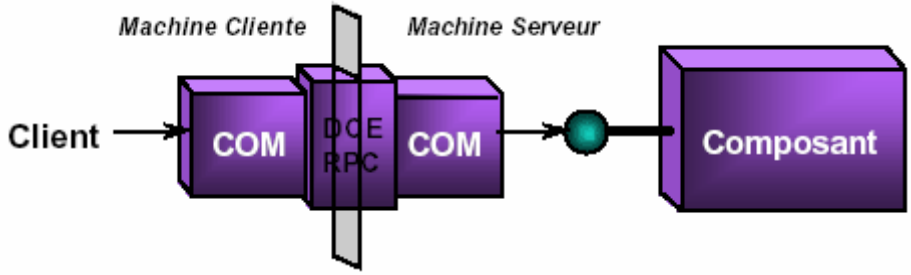
2. Serveur « Local » (ou « Out-of-process) : il s'exécute dans un processus séparé et sur la machine de l'appelant. Un composant de ce type peut être un service ou un exécutable. L'appel atteint d'abord l'objet *proxy* (voir récapitulatif des architectures) fourni par le modèle COM, qui génère l'appel de procédure à distance vers l'autre processus ou l'autre machine.

Même Machine
→ LPC Local Procedure Call



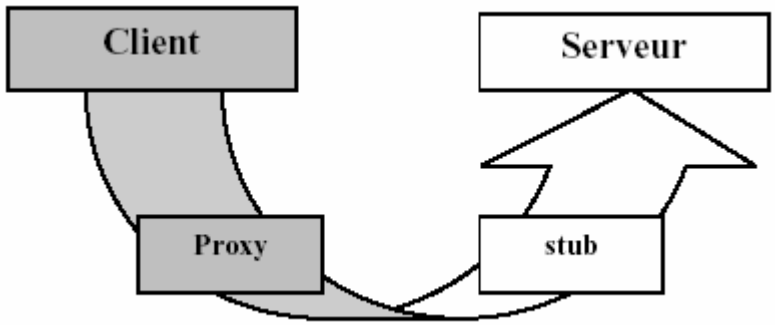
3. Serveur « Remote » : s'exécute dans un processus séparé, sur une machine distance. Un composant de ce type peut être un service ou un exécutable.

Machines Distantes
→ Protocole DCOM reposant sur DCE-RPC



Stub, Proxy : Le marshalling

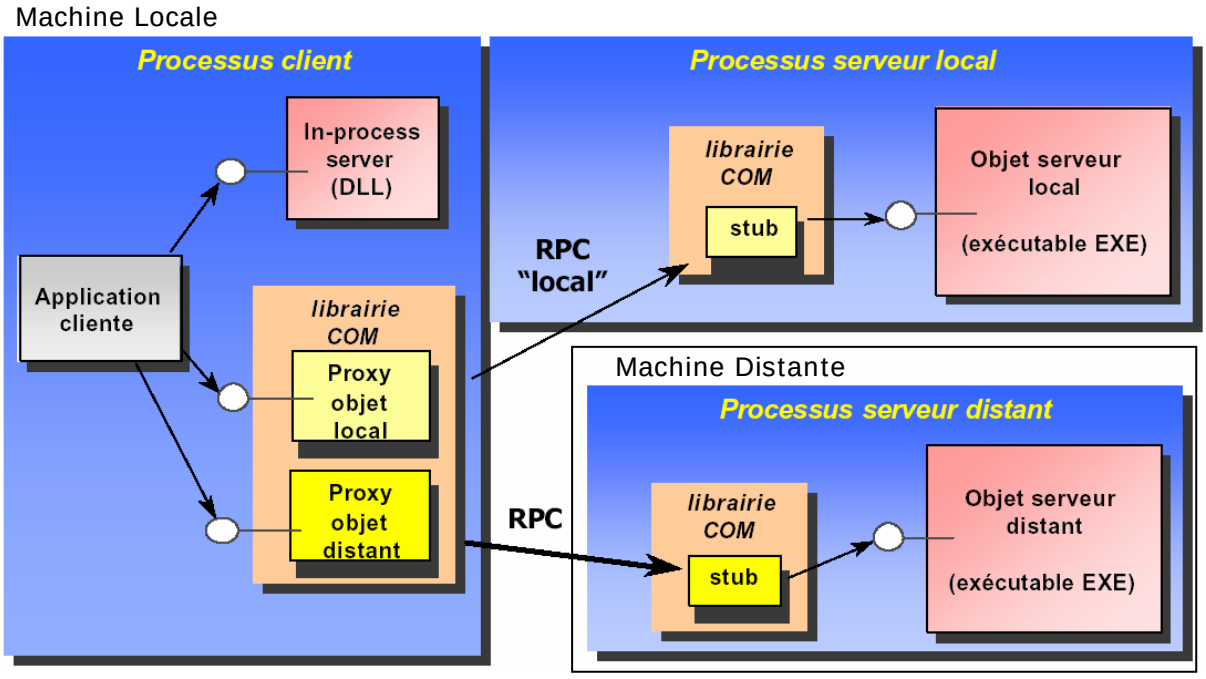
Le marshalling est une opération qui consiste à transformer et à transférer des données (pointeurs, arguments) entre le client et le serveur. Le marshalling fait intervenir deux modules, le proxy et le stub. Le proxy est chargé de passer les arguments au stub qui les passe (au bon format) au serveur.



Les rôles du marshalling sont de !

- Résoudre l'Interopérabilité : Unifier l'accès à des machines distantes
- Résoudre l'Hétérogénéité : Etre indépendant des systèmes d'exploitations et du langage de programmation des applications

RECAPITULATIF DES ARCHITECTURES :



RPC = Remote Procedure Call

FONCTIONNEMENT COM/DCOM ET COM+ :

DCOM

Le serveur est passif, il ne fait que répondre aux requêtes du client et ce, d'une manière synchrone. Les étapes du côté client sont donc les suivantes:

L'application cliente démarre le serveur. Elle fait ensuite une requête d'accès à l'interface DCOM en spécifiant le GUID. Si le serveur n'est pas en fonction, il est lancé. Côté serveur, le composant DCOM est créé, puis son interface et Reference Count (compteur de référence) de l'interface est incrémentée par AddRef().
L'application cliente lance ensuite les méthodes distantes de ces composants DCOM. Du côté serveur, les méthodes sont appelées.
Lorsque l'application cliente cesse d'utiliser l'interface du serveur. Reference Count (compteur de référence) de l'interface est décrémenté avec Release(). Si Reference Count est égal à 0, le composant DCOM est détruit.

Les serveurs DCOM sont donc des « véhicules » pour rendre disponibles les composants serveurs à des clients distants.

COM

Fonctionnement de COM : idem, à la différence que le serveur se trouve dans la même machine et le même processus. Appel d'une DLL.

COM+

Bibliothèque de services en plus.

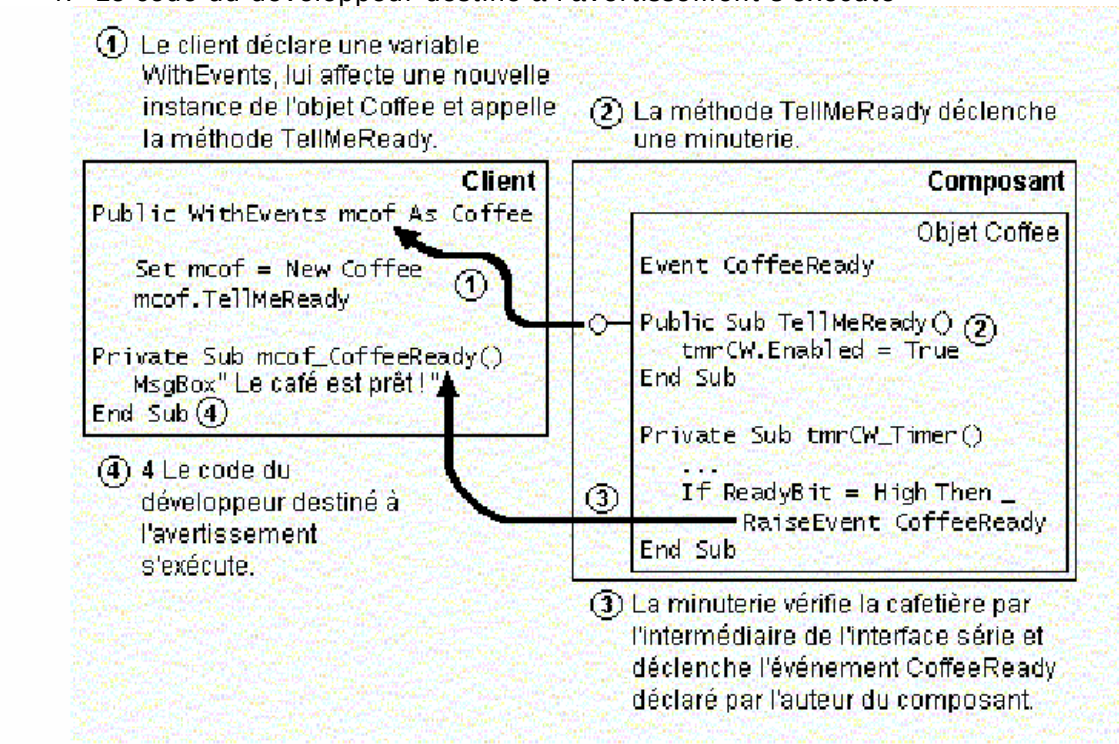
5. Processus de développement d'une application utilisant COM/DCOM/COM+

(à chaque étape : dire ce qu'il faut faire)

6. Un exemple de développement

Client	**Composant** Objet Cafe
Public WithEvents moncaf As Cafe Set moncaf = New Cafe Moncaf.EstPret Private Sub moncaf_CafePret() MsgBox « Le Cafe est prêt !» End	Event CafePret Public Sub EstPret() minuterie.Enabled = True End Sub Private Sub minuterie_Timer() ... If niveauCafe = haut Then CafePret End

1. Le client déclare une variable WithEvents, lui affecte une nouvelle instance de l'objet Cafe et appelle la méthode EstPret
2. La méthode EstPret déclenche une minuterie
3. La minuterie vérifie la cafetière par l'intermédiaire de l'interface série et déclenche l'événement CafePret déclaré par l'auteur du composant
4. Le code du développeur destiné à l'avertissement s'exécute



7. Comparaison entre COM/DCOM/COM+, CORBA et JavaRMI

Ne pas oublier l'interopérabilité entre ces 3 techno

Les critères de comparaison entre les 3 technologies :

- Concept objet ?
- Communication synchrone ou asynchrone ?
- Description via des interfaces ou des messages ?
- Communication directe ou indirecte ?
- Spécifique ou indépendant langage ?
- Possibilité de transformation de messages ou non ?
- Protocole de communication binaire ou textuelle ?

Comparaison DCOM/CORBA

	DCOM	CORBA
Couche de haut niveau		
Classe de base commune	`IUnknown`	`CORBA::Object`
Identifiant de classe d'objet	`CLSID`	nom d'interface
Identifiant d'interface	`IID`	nom d'interface
Activation d'objet côté client	`CoCreateInstance()`	appel méthode/`bind()`
Manipulateur d'objet	pointeur sur interface	référence sur objet
Couche intermédiaire		
Passage du nom à l'implémentation	Registre	Dépôt d'implémentations
Type d'information pour méthodes	Librairie de types	Dépôt d'interfaces
Localiser implémentation	SCM	ORB
Activer implémentation	SCM	OA
Stub côté client	proxy	stub/proxy
Stub côté serveur	stub	skeleton

Comparaison entre les 3 technologies réparties : CORBA, DCOM, RMI :

Concepts	CORBA (IIOP)	DCOM	RMI
Standard	Oui, créée par l'OMG	Non	Non
Fournisseurs	Privés + universités	Microsoft	Sun
Approche objet	Oui	Oui	Oui
Plates-formes supportées	Toutes ou presque	Windows principalement	Toutes ou presque
Langages utilisables pour la création d'objets	C++. Java, Pascal, COBOL, Ada, Smalltalk	C++, Pascal, VB, VC++	Java et autres avec Jini
Protocole de comm.	IIOP	RPC/DCE étendu	JRMP
Langage de définition d'interface	IDL, standardisé par l'OMG	IDL, créé par Microsoft, dérivé de IDL/DCE	Aucun
Coût du déploiement	Payant sauf si on utilise la version TAO-ACE	Gratuit	Gratuit
Complexité	Moyenne	Moyenne à élevée	Simple
Support des applets	Oui	Oui, limité à J++	Oui
Gestion sécurité	Oui (SSL)	Oui, limité à J++	Oui
Tolérance pannes	Oui	Non	
Répartition charge	Oui	Non	
Annuaire d'objets	Oui, service de noms	Non	`rmiregistry`
Appels statiques et dynamiques	Oui	Oui	Oui (JDK 1.2)
Héritage d'interface	Oui, multiple	Oui, simple	Oui
Identification objets	IOR	GUID	Nom et adresse
Gestion du cycle de vie des objets	Oui, simplifiée et complète	Comptage de référence et ping périodiques	Oui
Activation automat. des objets	Oui (OAD)	Oui (SCM)	Oui
Multithreading	Automatique par un pool de threads	Manuel ou automatique	Oui
Appels asynchrones	Oui	Non	Oui
Marshaling	Direct	Via RPC	Oui

DCE : Distributed Computing Environment	MSMQ : Microsoft Message Queue Server
DLL : Dynamic Link Library	MTS : Microsoft Transaction Server
IDL : Interface Definition Language	ODL : Object Definition Language
LPC : Local Procedure Call	OLE : Object Linking and Embedding
MIDL : Microsoft Interface Definition Language	RPC : Remote Procedure Call
	SCM : Service Control Manager

Pour garantir l'interopérabilité de CORBA, l'OMG a définit des protocoles réseaux de communication entre les objets. Le plus utilisé se nomme Internet Inter-ORB Protocol IIOP.Grace à RMI over IIOP, il est possible de créer des EJB compatibles CORBA

8. Des applications à objets répartis aux composants répartis

9. En quoi la proposition COM/DCOM/COM+ réponds aux spécifications des applications Clients/Serveurs et en particulier aux accès concurrents et la gestion des transactions.

APPLICATIONS CLIENTS/SERVEURS

Nous avons vu dans la partie concernant les architectures qu'il existe 3 types de relation « Client Serveur » dans DCOM :

Même processus
Appel direct – DLL

Processus différent, même machine
Appel via schéma LPC ("Local Procedure Call") ou RPC Local

Processus distant
Appel via schéma RPC ("Remote Procedure Call")

GESTION DES TRANSACTIONS

La notion de transaction MTS (Microsoft Transaction Service) a été intégrée avec l'arrivé de COM+. MTS est dédié aux applications transactionnels, il prend en charge le multi-threading (multi-tâches), la sécurité, traitement des transactions.

MTS apporte essentiellement les caractéristiques suivantes :

- Connexion aux bases de données.
- Applications MTS sécurisées.
- Gestion des composants.
- Monitoring des transactions.
- Propriétés partagées (spécifiques aux objets MTS).
- Sécurité améliorée (spécifique aux objets MTS).
- Activation JIT (spécifique aux objets MTS).

Ce dernier point n'est pas très clair.

La gestion des transactions

Lorsqu'un objet est utilisé dans une transaction, il doit signaler l'état de son traitement à l'aide des méthodes *SetComplete* ou *SetAbort*. Ceci peut être réalisé soit par l'objet lui-même, soit par le programme client. Dans ce dernier cas, l'objet MTS doit disposer d'entrée permettant l'appel de ces méthodes.
Ces méthodes positionnent un flag pour la transaction. Si à la sortie d'une méthode de l'objet MTS le flag reste positionné à SetAbort, une exception est soulevée dans l'application ayant appelée l'entrée. Dans ce cas, l'exception soulevée doit être gérée afin d'annuler la transaction avec l'appel de l'entrée Abort de l'objet transaction. Si aucune exception n'est soulevée, alors la transaction peut être validée avec l'appel de la méthode *Commit*. Voici un exemple d'une transaction faisant intervenir deux objets MTS :

```
procedure TForm1.Button15Click(Sender: TObject);
        // Déclaration de l'objet transaction
        var Transaction : ITransactionContextEx;
        begin
                // Création de l'objet transaction
                Transaction:=CreateTransactionContextEx;
                OleCheck(Transaction.CreateInstance(CLASS_ObjectFirst,
                IObjectFirst, ObjectFirst));
                OleCheck(Transaction.CreateInstance(CLASS_ObjectSecond,
                IObjectSecond, ObjectSecond));
        try
                ObjectFirst.DoWork;
                ObjectSecond.DoWork;
                Transaction.Commit;
        Except
                // RollBack
                Transaction.Abort;
                raise;
        end;
end;
```

Dans un premier temps, on créé l'objet transaction, puis les deux objets MTS en protégeant leur appel par la routine OleCheck.
Dans un bloc protégé de type *try..except* , les traitements de chaque objet sont appelé. En cas de réussite des traitements, la transaction est validée par un *Commit* . Si un des traitements déclenche l'appel de *SetAbort* , alors une exception est soulevée provoquant l'annulation de la transaction.

Le rollback des opérations

Lorsqu'une transaction est annulée, il est nécessaire de défaire toutes les opérations ayant abouties avant l'échec.

La méthode *IsInTransaction* permet à un objet MTS de tester s'il est utilisé dans une transaction. Ceci est pratique pour par exemple n'autoriser l'utilisation d'un objet que si celui-ci participe à une transaction

10. Conclusion

COM/DCOM/COM+ sont :

- Des produits “propriétaire” liés à l’environnement Microsoft, mais l’interconnexion est possible avec CORBA
- Outils puissants, mais concepts difficiles à appréhender
- Un héritage de OLE
- Peuvent fonctionner dans des processus différents et sur des ordinateurs différents
- Microsoft affirme que les composantes DCOM sont indépendantes du langage de programmation utilisé.
- Pas d’accès au code source
- Evolution dynamique :
    - Remplacement de composants (sans recompilation ou édition de liens)
    - Association de nouvelles interfaces
    - Capacité d’auto description

La plupart des applications Windows sont des objets COM