Plan

Introduction

Generalité

Historiqe

Definition

Difference entre COM et DCOM

Mode de fonctionnment de COM et DCOM

Les services de COM et DCOM

Les composants

Contrôle ActiveX

DLL activeX

EXE activeX

Document activeX

Choix entre un EXE et un DLL

Les interfces

IDL

I nterface Unknown

Interface IclanFactory

Interface IDispatc

Interface IMoniker

Architecture des COM et DCOM

Architecture de COM

Architecture de DCOM

Avantage et inconvenients

Avantage de COM/DCOM

Inconvenients de COM/DCOM

Codage

COM  
 DCOM

Conclusion

Introduction

Generalité

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.

**Definition de com/DCOM**

***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 (Dynamic Link Libraries).

1. **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.

1. **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 platesformes 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.

**Différence entre COM et DCOM :**

| Base de | COM | DCOM |
| --- | --- | --- |
| Abréviation | Modèle d'objet composant. | Modèle d'objet de composant distribué. |
| Exécution sur | Environnement côté client. | Environnement serveur. |
| Installation | Requis sur la machine où il est utilisé. | Requis sur le même réseau. |
| Type de modèle | COM est une norme d'interface. | DCOM un modèle conçu pour les applications distribuées. |
| Réutilisabilité des objets | Il permet. | Cela ne permet pas. |
| Autoriser distribué | Non, une telle capacité d'objets distribués. | Oui, il est capable d'objets distribués. |
| Utilisation de la mémoire | Il ne fournit pas une meilleure utilisation. | Il dispose d'un collecteur d'objets distribué qui améliore l'utilisation de la mémoire. |

Les composants

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 (Dynamic Link Libraries).

**VI – 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.

**1 – IDL**

**IDL, acronyme d'*Interactive Data Language*, est un**[**langage de programmation**](https://fr.wikipedia.org/wiki/Langage_de_programmation)[**propriétaire**](https://fr.wikipedia.org/wiki/Logiciel_propriétaire)**apparu à la fin des années 1970, et qui est rapidement monté en puissance dans les domaines de la  télédétection et de l'**[**astronomie**](https://fr.wikipedia.org/wiki/Astronomie)**. IDL est un langage vectoriel de traitement de données et de visualisation très répandu dans l'industrie et dans la recherche.**

**Avantages :**

- une syntaxe claire

* un apprentissage très rapide en venant de [Pascal](https://fr.wikipedia.org/wiki/Pascal_(langage)), [Fortran](https://fr.wikipedia.org/wiki/Fortran), [Caml](https://fr.wikipedia.org/wiki/Caml) ou [C](https://fr.wikipedia.org/wiki/C_(langage))
* une rapidité de programmation
* la main sur les données (la commande « help » donnant accès à toutes les variables connues)

**Inconvénients :**

- un langage propriétaire dont les licences sont coûteuses

* une évolution un peu chaotique des Widgets
* un délicat passage à une approche objet *artificielle* (non demandée par les utilisateurs)

**2-Interface Unknown**

l'interface IUnknown est l'interface fondamentale dans le Component Object Model (COM). La spécification COM stipule que les objets COM doivent implémenter cette interface. De plus, toutes les autres interfaces COM doivent être dérivées de IUnknown. IUnknown expose deux fonctionnalités essentielles de tous les objets COM : la gestion de la durée de vie des objets via le comptage des références et l'accès aux fonctionnalités des objets via d'autres interfaces.

Une interface IUnknown (ou dérivée de IUnknown) consiste en un pointeur vers une table de méthodes virtuelles qui contient une liste de pointeurs vers les fonctions qui implémentent les fonctions déclarées dans l'interface, dans l'ordre dans lequel elles sont déclarées dans l'interface. La surcharge d'appel d'invocation in-process est donc identique aux appels de méthode virtuelle en C++.

**3-Interface IClassFactory**

Tous les serveurs COM/DCOM suivent le modèle de conception d'usine. La classe IClassFactory est la norme utilisée dans COM/DCOM pour réaliser cette conception. L'interface IClassFactory contient deux méthodes destinées à traiter une classe entière d'objets, et est donc implémentée sur l'objet de classe pour une classe spécifique d'objets (identifiée par un CLSID).

**4-Interfaces IDispatch**

IDispatch est l'interface qui expose le protocole OLE Automation.[1] En étendant IUnknown, c'est l'une des interfaces standard qui peuvent être exposées par les objets COM. COM distingue trois types d'interface : les interfaces personnalisées IUnknown basées sur VTABLE, les interfaces IDispatch prenant en charge l'introspection et les interfaces doubles prenant en charge les deux types.

L'interface Automation (IDispatch) permet à une application cliente de savoir quelles propriétés et méthodes sont prises en charge par un objet au moment de l'exécution, c'est-à-dire implémente le concept de RTTI. Il fournit également les informations nécessaires pour appeler ces propriétés et méthodes. Les applications clientes n'ont pas besoin de connaître les membres de l'objet lorsqu'elles sont compilées. Cela permet aux objets COM et ActiveX d'être appelés par des plates-formes de programmes de script telles que le serveur ASP et JavaScript sur Internet Explorer, où les conventions d'appel n'étaient pas connues au moment de la construction d'IIS ou d'IE. En revanche, une simple bibliothèque d'objets est compilée et liée dans un programme, par ex. un appel DLL doit connaître un nom de fonction et des paramètres au moment de la compilation.

**5-Interface Imoniker**

L'interface IMoniker est utilisée pour nommer les objets COM. Un objet qui implémente IMoniker est appelé un « moniker », qui est un nom qui identifie de manière unique un objet COM. Vous pouvez considérer un

surnom comme une généralisation d'un nom de chemin ; de la même manière qu'un nom de chemin identifie un fichier dans le système de fichiers, un surnom identifie un objet COM. Par exemple, supposons que vous ayez un objet représentant une plage de cellules dans une feuille de calcul qui est elle-même incorporée dans un document texte.Les monikers sont plus largement applicables que les noms de chemin ; alors que les noms de chemin ne peuvent identifier que des fichiers, les surnoms peuvent identifier n'importe quel type d'objet, qu'il

s'agisse d'un fichier, d'une incorporation, d'une sélection dans un document ou autre chose.

**VII-La communication entre composants**

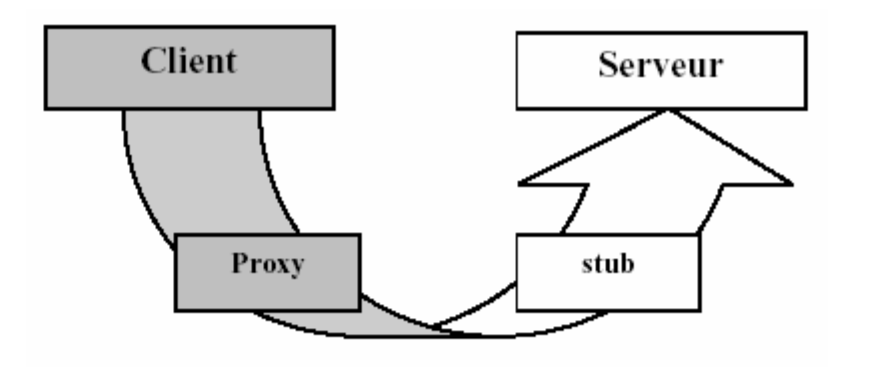
**1-Proxy et Sturb**

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

**3-Point de Connexion**