CORBA vs COM

Urbanisation des SI – NFE107

Fiche de lecture

Y. Durand-Poudret

CORBA vs COM *1*
CORBA, COM ? : Intergiciels (Middleware) *3*
CORBA, COM ? : Intergiciels (Middleware) *3*
CORBA : Définition *3*
CORBA : Introduction *3*
I. Introduction **3**
CORBA : L'ORB (Object Request Broker) *4*
II. L'ORB **4**
A. Object Request Broker 4
B. Interface Definition Language 4
C. Dynamic Invocation Interface 4
D. Interface Repository 5
E. Basic Object Adapter 5
CORBA : exemple de l'ORB ORBIX *5*
How Orbix Implements CORBA ? **5**
COM : Définition *6*
COM : Introduction et Définitions *6*
Qu'est-ce qu'OLE ? 6
Qu'est-ce que COM ? 6
Qu'est-ce que DCOM ? 6
COM : Protocole de communication DCE/RCP *7*
Coté Client (cf Figure 3 : RPC côté client) 7
Coté Serveur (cf Figure 4 : RPC côté serveur) 7
COM : IDL et type library *8*
Qu'est-ce qu'une type library ? 8
Comment écrire un fichier IDL ? 8
CORBA vs COM *9*
Introduction 9
Comparaison IDL 11
Résumé 11
CORBA vs COM : FAQ *11*
CORBA vs COM : Principaux Inconvénients *12*
CORBA vs COM : Comparison of Web Services, RMI, CORBA, DCOM *12*
Bibliographie *13*
CORBA **13**
COM **13**
CORBA, COM et les autres **13**
Table des illustrations *13*

CORBA, COM ? : Intergiciels (Middleware)

- **Wikipedia** : En informatique, un intergiciel (en anglais middleware) est un logiciel servant d'intermédiaire de communication entre plusieurs applications, généralement complexes ou distribuées sur un réseau informatique.

- **Middleware et Internet de Daniel Serain** : Le middleware est une Couche de logiciel située entre le système d'exploitation et les applications, permettant l'échange d'informations entre celles-ci.

CORBA : Définition

- **OMG (92)**: CORBA est l'acronyme de Common Object Request Broker Architecture, ce fournisseur indépendant offre une architecture et une infrastructure permettant aux applications informatiques de travailler ensemble au travers du réseau. CORBA utilise le protocole standard IIOP qui permet l'interopérabilité[1] de composants pouvant provenir de serveurs, de systèmes d'exploitation, de réseaux ou de langages de programmation similaires ou différents.

- **Wikipedia** : CORBA, acronyme de Common Object Request Broker Architecture, est une **architecture logicielle**, pour le développement de **composants** et **d'Object Request Broker ou ORB**. Ces composants, qui sont assemblés afin de construire des applications complètes, peuvent être écrits dans des langages de programmation distincts, être exécutés dans des processus séparés, voire être déployés sur des machines distinctes. Corba est un standard maintenu par **l'Object Management Group**.

CORBA : Introduction

I. Introduction

L'OMG (Object Management Group) est un consortium regroupant :

- Des fournisseurs de systèmes
- Des fournisseurs de logiciels
- Des utilisateurs finaux

Il a pour but le développement d'applications distribuées dont les composants collaborent avec :

- Efficacité
- Fiabilité
- Transparence
- *Scalability, i.e.,* une capacité d'évolution importante

L'OMG a défini un modèle de référence pour des applications distribuées utilisant des techniques orientées objet. Ce modèle comprend quatre points de standardisation. (cf Figure 1 : Architecture CORBA)

- Object Model: c'est un modèle générique pour assurer la communication avec des systèmes orientés objet conformes au modèle de l'OMG
- Object Request Broker (ORB): c'est l'élément clé de communication, il assure la distribution des messages
- ObjectServices (ou encore CORBAServices): ces services fournissent les principales fonctions de base nécessaires à la gestion des objets (nommage, persistance, gestion d'évènements...)
- CommonFacilities (ou encore CORBAFacilities): ce sont des utilitaires destinés aux applications

[1] L'interopérabilité est la capacité que possède un produit ou un système, dont les interfaces sont intégralement connues, à fonctionner avec d'autres produits ou systèmes existants ou futurs et ce sans restriction d'accès ou de mise en œuvre. Il convient de distinguer « interopérabilité » et « compatibilité ». Pour être simple, on peut dire qu'il y a compatibilité quand deux produits ou systèmes peuvent fonctionner ensemble et interopérabilité quand on sait pourquoi et comment ils peuvent fonctionner ensemble. Autrement dit, on ne peut parler d'interopérabilité d'un produit ou d'un système que si on en connaît intégralement toutes ses interfaces.

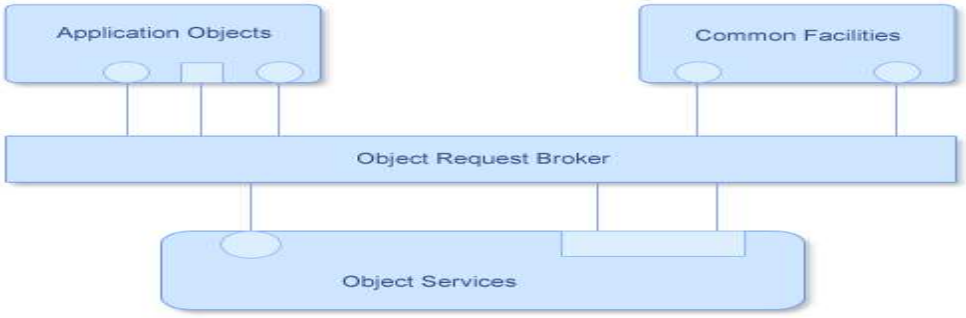

Figure 1 : Architecture CORBA

L'OMG a donc défini CORBA (Common Object Request Broker Architecture), une architecture respectant la standardisation ci-dessus. Les principes de CORBA sont :

- Une séparation stricte Interface/Implémentation
- La transparence de la localisation des objets
- La transparence de l'accès aux objets
- Le typage des Object References par les interfaces
- L'héritage multiple d'interfaces

CORBA : L'ORB (Object Request Broker)

II. L'ORB

A. Object Request Broker

L'ORB fournit les mécanismes par lesquels des objets font des requêtes et reçoivent des réponses, et ce de manière transparente. Il fournit également l'interopérabilité entre des applications sur différentes machines dans des environnements distribués hétérogènes et il interconnecte *sans coutures* de multiples systèmes objets. D'une façon simplifiée, on peut définir l'ORB comme une entité qui fournit des mécanismes d'interrogations permettant de récupérer des objets, des procédures qui constituent une application.

B. Interface Definition Language

Développer des applications distribuées flexibles sur des plateformes hétérogènes nécessite une séparation stricte interface/implémentation(s). IDL aide à accomplir cette séparation. En effet, IDL est un langage de définition d'interface orienté objet. Il définit les types des objets en spécifiant leurs interfaces. Une interface consiste en un jeu d'opérations et de paramètres pour ces opérations. IDL est le moyen par lequel une implémentation d'un objet indique à ses clients potentiels quelles opérations sont disponibles et comment elles doivent être invoquées.

IDL d'un service minimal de gestion du compte d'un client

```
// CompteClient.idl
interface CompteClient {
      void credit ( in unsigned long montantFRF );
      void debit ( in unsigned long montantFRF );
      long solde ( );
};
```

C. Dynamic Invocation Interface

L'interface d'invocation dynamique d'un ORB autorise la création et l'invocation dynamiques de requêtes. Un client utilisant cette interface pour envoyer une requête à un objet obtient la même sémantique qu'un client utilisant l'opération *stub* générée à partir de la spécification de type IDL.

D. Interface Repository

L'IR est le composant de l'ORB qui fournit un stockage persistant des définitions d'interfaces, il gère et permet l'accès à une collection de définitions d'objets spécifiés en IDL.

E. Basic Object Adapter

Un Object Adapter est l'interface principale pour une implémentation objet pour accéder aux services fournis par un ORB. On s'attend à ce qu'il y ait peu d'OA disponibles, avec des interfaces appropriées à des types spécifiques d'objets. Le BOA est donc l'interface qui permet aux implémentations objet de pouvoir communiquer avec l'ORB et d'accéder à ses services (cf. Figure 2 : Structure et scénario d'un BOA)

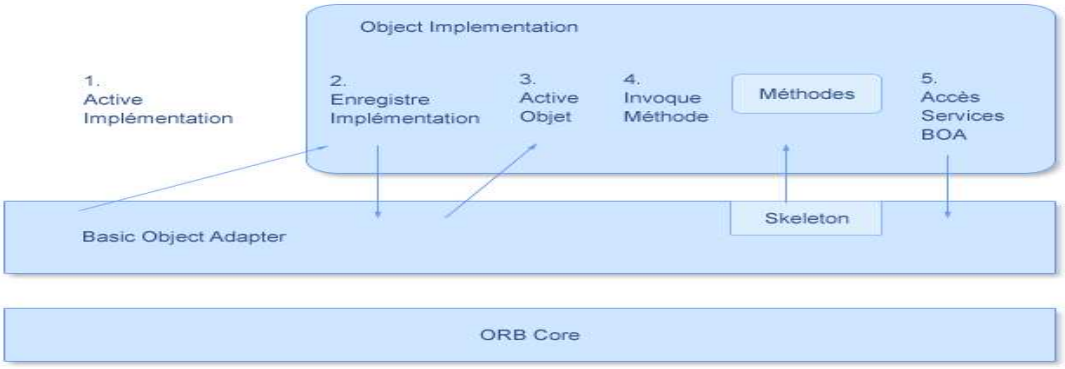

Figure 2 : Structure et scénario d'un BOA

1. Le BOA démarre un programme pour fournir l'implémentation objet
2. L'implémentation objet notifie le BOA que son initialisation est terminée et qu'elle est prête à manipuler des requêtes.
3. Quand la première requête pour un objet particulier arrive, l'implémentation est avertie qu'elle doit activer l'objet.
4. Dans les requêtes suivantes, le BOA appelle la méthode appropriée en utilisant les *skeletons.*
5. A certains moments, l'implémentation objet peut accéder aux services du BOA tels que création d'un objet, désactivation...

CORBA : exemple de l'ORB ORBIX

ORBIX est un produit commercial phare basé sur l'architecture CORBA produit par IONA

Comment Orbix implémente CORBA ?

- Orbix est un ORB qui implémente complètement la spécification CORBA 2. Par défaut, tous les composants et les applications Orbix communique en utilisant le protocole standard IIOP CORBA
- Les composants d'Orbix sont les suivants :
  - Le compilateur IDL analyse les définitions IDL et produit du code C++ qui permet le développement de programmes client/serveur
  - La librairie Orbix est lié à toutes les programmes Orbix et implémente plusieurs composants de l'ORB, incluant la DII, la DSI, et la fonctionnalité de base de l'ORB.
  - Le démon Orbix est un processus qui tourne sur chaque serveur et implémente plusieurs composants ORB, incluant l'« Implementation Repository ».
  - L'« Orbix Interface Repository server » est un processus qui implémente l'« Interface Repository ».
- Orbix inclue aussi plusieurs fonctionnalités de programmation qui étendent les capacités de l'ORB.
- De plus, Orbix est une solution ORB qui combine les fonctionnalités standards de CORBA avec une suite de services intégrés : OrbixNames, OrbixEvents, OrbixOTS, and OrbixSSL.

COM : Définition

- Microsoft : La technologie COM (Component Object Model) dans la famille Microsoft Windows des systèmes d'exploitation permet à des composants logiciels de communiquer. COM est utilisé, par les développeurs, pour créer des composants logiciels ré-utilisables, pour créer des applications réparties à partir de composants liés entre eux, et de tirer profit des services Windows. La famille des technologies COM comprend COM, COM+, Distributed COM (DCOM) et des contrôles ActiveX.

- Wikipedia : Component Object Model (COM) est une interface standard pour les composants logiciels mis en place par Microsoft en 1993. Il est utilisé pour permettre la communication interne et la dynamique de création de l'objet dans un large éventail de langages de programmation. Le terme COM est souvent utilisé dans les industries du développement logiciel comme un terme général qui englobe la OLE, OLE Automation, ActiveX, COM + et DCOM technologies. L'essence de COM est l'implémentation d'objets qui peuvent être utilisés dans des environnements différents de celui sur lequel ils ont été créés. COM permet de réutiliser ces objets sans connaissance de leur implémentation, comme il force l'utilisation des composants au travers de leur interface, celle-ci étant distincte de son implémentation.

COM : Introduction et Définitions

Qu'est-ce qu'OLE ?

OLE est l'abréviation de *Object Linking and Embedding*, ce qui signifie *Intégration d'objets et Lien sur des objets*. OLE est une technologie qui a été développée par Microsoft, initialement dans le but de permettre la programmation d'objets capables d'être insérés dans des applications réceptacles soit par intégration complète, soit par référence (ce que l'on appelle une liaison). Ce but a été atteint pour la plupart des applications et apparaît à présent dans le menu *Coller | Collage spécial*. Les objets intégrés ou liés sont capables de s'afficher dans l'application qui les contient. Ils sont également capables de fournir un certain nombre de services standards permettant leur manipulation (ces services peuvent être la sauvegarde de leur état, la capacité à être édité, etc...). OLE est donc un remplacement efficace des liaisons DDE.

Qu'est-ce que COM ?

Pour réaliser cet objectif, Microsoft a dû fournir un standard de communication entre les différentes applications qui voulaient être OLE. Ce standard de communication, définit la méthode à employer pour accéder aux fonctionnalités des objets OLE, ainsi que les principaux services qui peuvent être nécessaires lors de l'intégration d'objets. Les programmeurs désirant réaliser une application OLE devaient se conformer au protocole d'appel du standard et fournir un certain nombre des services définis dans OLE. Ce standard a été conçu de manière ouverte, c'est à dire qu'il n'est pas nécessaire de fournir tous les services définis dans OLE pour être fonctionnel. Cependant, plus un objet OLE offre de services, meilleure son intégration est. De même, plus une application est capable d'utiliser des services, plus elle est fonctionnelle avec les objets qui fournissent ces services. Par ailleurs, il est possible de définir des services différents de ceux définis dans OLE, le système est donc également extensible. Le standard de communication qui a été défini se nomme COM, ce qui signifie *Component Object Model*, ou *Modèle Objet de Composants*. La plupart des services sont définis dans OLE, cependant, COM lui-même utilise un certain nombre de services. La limite entre ce qui est défini par COM et ce qui est défini par OLE est donc floue, mais en principe, les services COM sont tous des services systèmes. Comme on l'a dit, initialement, OLE devait permettre l'intégration des objets entre applications. En fait, il s'est avéré qu'OLE faisait beaucoup plus que cela : grâce à COM, il permettait l'écriture de composants logiciels réutilisables par différentes applications. Il s'agit donc bien d'une technologie à composants.

Qu'est-ce que DCOM ?

Microsoft a ensuite complété la technologie COM afin de permettre la répartition des composants sur un réseau. L'aspect distribué des composants COM ainsi répartis a donné le nom à cette extension de COM, que l'on appelle simplement DCOM (pour *Distributed COM*). Dans la suite du document, on considérera que COM est distribué, on utilisera donc systématiquement le terme DCOM.

COM : Protocole de communication DCE/RCP

DCE (Distributed Computing Environment, de l'OSF (Open Software Foundation) consortium de fabricants d'ordinateur (IBM, DEC, HP, …). DCE est un ensemble d'outils, qui tournent sur un OS, servant à la création et au déroulement d'applications distribuées. L'un de ces composants est RPC (Remote Procedure Call) basé sur le protocole ORPC (Object RPC). RPC est la base de toute communication dans DCE. RPC-DCE utilisent la génération automatique de stubs

Coté Client (cf Figure 3 : RPC côté client)

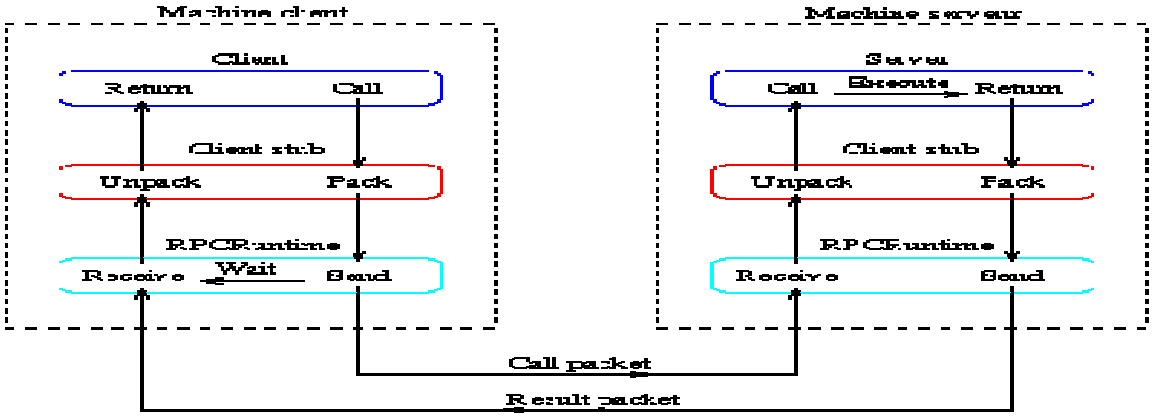

Figure 3 : RPC côté client

- Le « Stub » client :
  - Call request
    - Reçoit un « call request » du client,
    - Forme un message (pack) avec les spécifications de la procédure distante et les paramètres,
    - Demande au RPCRuntime de les transmettre au stub serveur (Dans l'architecture COM c'est le runtime de COM qui est utilisé).
  - Result
    - Reçoit le résultat de l'exécution de la procédure du RPCRuntime (Dans l'architecture COM c'est le runtime de COM qui est utilisé),
    - Décode le message (unpack),
    - Transmet les données au client

Coté Serveur (cf Figure 4 : RPC côté serveur)

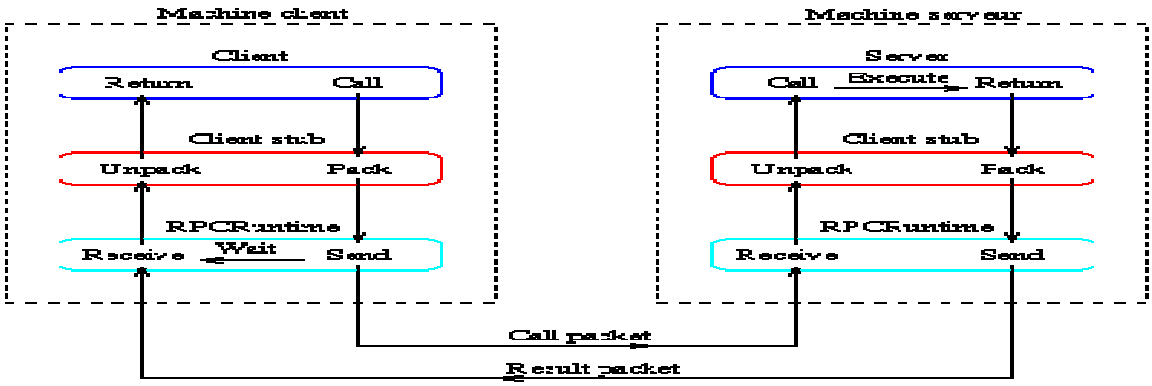

Figure 4 : RPC côté serveur

- Le « Stub » serveur :
  - Call request
    - Reçoit un « call request » du RPCRuntime (Dans l'architecture COM c'est le runtime de COM qui est utilisé),

- Décode le message (unpack),
- Exécute l'appel de procédure locale (normalement) dans le serveur.

- Result
  - Reçoit le résultat de l'exécution de la procédure de la procédure du serveur,
  - Forme un message (pack) avec les résultats,
  - Transmet les données au RPCRuntime (Dans l'architecture COM c'est le runtime de COM qui est utilisé),

COM : IDL et type library

Qu'est-ce qu'une type library ?

Une *type library* est un fichier binaire (portant l'extension *.tlb*) contenant la description des méthodes et propriétés d'objets DCOM. Cette description permet à une application de connaître dynamiquement les différentes interfaces supportées par un objet, ainsi que leurs méthodes, leurs paramètres et leurs rôles.

Initialement, les type libraries étaient générées à l'aide du langage ODL (*Object Description Language*), qui était très proche du langage IDL. Un compilateur spécifique prenait en entrée les fichiers écrits en ODL et produisaient des fichiers portant l'extension *.TLB* (pour *Type LiBrary*). Depuis, le langage ODL a été inclus dans le langage IDL. Il est donc possible à présent d'écrire un seul fichier IDL qui décrit les interfaces pour la génération automatique des facelets et des stublets, et qui permet la génération des type libraries.

Comment écrire un fichier IDL ?

Le compilateur utilisé pour traiter les fichiers IDL est nommé MIDL.EXE (pour Microsoft IDL). MIDL est un outil complexe, qui permet de générer non seulement les stublets et les facelets, mais également les type libraries.
Exemple IDL pour un composant Calculator :

```
import "unknwn.idl";
// Définit l'interface IMultiplier :
[
    uuid(e3261622-0ded-11d2-86cc-444553540000), object,
    helpstring("Definition of the IMultiplier interface")
]
interface IMultiplier : IUnknown
{
    [
        helpstring("Multiplies two integers and returns the result")
    ]
    HRESULT Mul([in] long i, [in] long j, [out] long *pResult);
};
// Importe la définition des interfaces de Adder :
import "..\..\Adder\AdderPrx\adder.idl";
[
    uuid(128abb81-0e9a-11d2-86cc-444553540000),
    helpstring("Calculator Type Library"),
    lcid(0x0000),
    version(1.0)
]
library CalculatorTypeLibrary
{
    [
        uuid(91e132a1-0df1-11d2-86cc-444553540000),
        helpstring("Description of the Calculator Component")
    ]
    coclass Calculator
    {
        interface IAdder;
```

```
        interface IOpposite;
        interface IMultiplier;
    }
};
```

CORBA vs COM

DCOM (Distributed Component Object Model) et CORBA (Common Object Request Broker Architecture) sont deux modèles populaires d'architecture logicielle distribuée objet. Dans ce document, nous faisons la comparaison des modèles architectures DCOM et CORBA.

Introduction

La croissance explosive du Web, la popularité croissante des ordinateurs et les progrès de la haute vitesse d'accès au réseau ont mis l'informatique distribuée au premier plan. Pour simplifier la programmation réseau et de réaliser une architecture logiciel à base de composants, deux modèles objets distribués sont apparus comme des normes, à savoir, DCOM (Distributed Component Object Model) et CORBA (Common Object Request Broker Architecture).

DCOM est l'extension distribuée de **COM (Component Object Model)** [COM 95], qui construit une « object remote procedure call » (ORPC) sur-couche de DCE RPC [DCE 95] supporte l'appel d'objets distants. Un serveur COM peut créer des instances d'objet de multiples classes d'objet. Un objet COM peut supporter de multiples interfaces, chacune représentant un point de vue différent ou le comportement de l'objet. Une interface est composée d'un ensemble de fonctionnalités correspondant aux méthodes. Un client COM interagit avec un objet COM par l'acquisition d'un pointeur sur une des interfaces de l'objet et l'invocation des méthodes par l'intermédiaire de ce pointeur, comme si l'objet réside dans l'espace d'adressage du client. COM précise que toutes les interfaces doit suivre un standard de mise en page mémoire, qui est le même que la table de fonction virtuelle en C++ [Rogerson 96]. Depuis la spécification est au niveau binaire, elle permet l'intégration de composants binaires pouvant être écrits dans divers langages de programmation tels que C++, Java et Visual Basic.

CORBA est un cadriciel d'objet distribué proposé par un consortium de 700 entreprises appelé l'Object Management Group (OMG) [CORBA 95]. Le cœur de l'architecture CORBA est l'**Object Request Broker (ORB)** qui agit comme un bus d'objet sur lequel les objets interagissent de manière transparente avec d'autres objets situés localement ou à distance [Vinoski 97]. Un objet CORBA est représenté à l'extérieur par son interface avec l'ensemble des méthodes qu'il propose. Une instance particulière d'un objet est identifiée par une référence de l'objet. Le client d'un objet CORBA acquiert sa référence d'objet et l'utilise comme un pointeur pour faire des appels aux méthodes, comme si l'objet est localisé sur l'espace d'adressage du client. L'ORB est responsable de tous les mécanismes nécessaires pour trouver l'implémentation de l'objet, pour préparer à recevoir la demande, pour communiquer la demande à celui-ci, et pour apporter la réponse (le cas échéant) au client. L'implémentation de l'objet interagit au travers de l'ORB soit par le biais de l'Object Adapter (OA) ou par l'ORB interface.
Les deux cadriciels DCOM et CORBA fournissent de type de communications client-serveur. Pour demander un service, un client appelle une méthode d'implémentée par un objet distant, qui agit comme le serveur dans le modèle client-serveur. Le service fourni par le serveur est encapsulé dans un objet et l'interface d'un objet est décrit dans une **Interface Definition Language (IDL)**. Les interfaces définies dans un fichier IDL servent comme un contrat entre un serveur et ses clients. Les clients interagissent avec un serveur en invoquant les méthodes décrites dans les IDL. L'implémentation réelle de l'objet est cachée du client. Certains des concepts de programmation orienté objet sont présents au niveau de l'IDL, tels que l'encapsulation des données, le polymorphisme et l'héritage simple. CORBA supporte également l'héritage multiple au niveau IDL, contrairement à DCOM. En revanche, la notion d'un objet ayant plusieurs interfaces est utilisée pour répondre au même but en DCOM. CORBA IDL peut également définir des exceptions.
Dans DCOM et CORBA, les interactions entre un processus client et un objet serveur sont implémenté comme objet orienté RPC-style communications [Birrell 84]. Figure 5 Structure du RPC présente une structure typique RPC. Pour invoquer une fonction distante, le client effectue un appel au « stub » client. Le « stub » compresse les paramètres de l'appel dans un « request » message, et invoque le protocole réseau pour expédier le message au serveur. Sur le serveur, le protocole réseau fournit le message au « stub » serveur, qui décompresse le « request » message, et appel la fonction correspondante de l'objet. En DCOM, le « stub » client est considéré comme un « proxy » et le « stub » serveur est dénommé le « stub ». En revanche, le « stub » client dans CORBA

est appelé « stub » et le « stub » serveur est appelé le « skeleton ». Parfois, le terme de « proxy » est aussi utilisé pour se référer à une instance en cours d'exécution du « stub » dans CORBA.

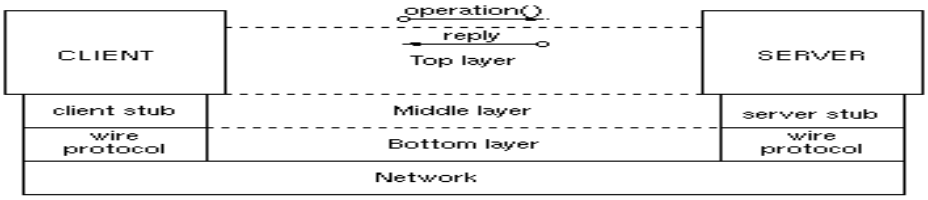

Figure 5 : RPC Structure

L'ensemble des architectures de DCOM et CORBA sont illustrés respectivement dans la Figure 6 et Figure 7. Les schémas suivant décrivent l'invocation des méthodes au niveau de trois différentes couches. La couche supérieure est l'architecture de la base de la programmation, ce qui est visible par les développeurs côté client et les programmes objets côté serveur. La couche intermédiaire est l'architecture distante, qui, par transparence, fait l'interface ou les références d'objet sont définies aux travers des différents processus. La couche inférieure est le protocole de l'architecture réseau, qui étend l'architecture distante pour travailler sur différentes machines.

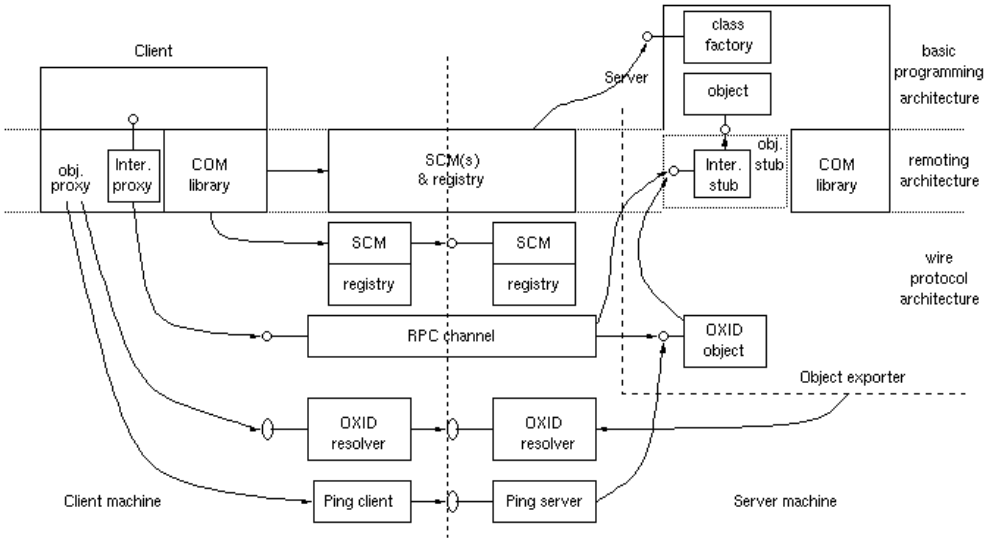

Figure 6 : DCOM overall architecture

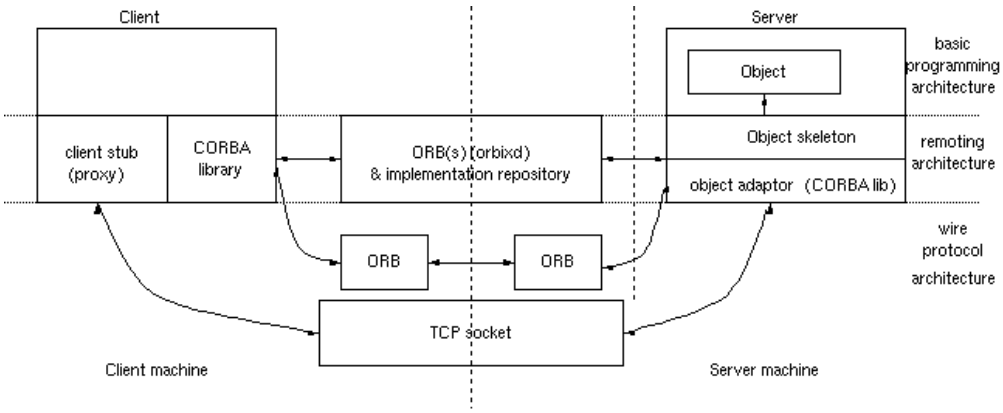

Figure 7 : CORBA overall architecture

Comparaison IDL

DCOM IDL

```
// uuid and definition of IGrid1
    [  object,
       uuid(3CFDB283-CCC5-11D0-BA0B-00A0C90DF8BC),
       helpstring("IGrid1 Interface"),
       pointer_default(unique)
    ]
    interface IGrid1 : IUnknown      {
       import "unknwn.idl";
       HRESULT get([in] SHORT n, [in] SHORT m, [out]
LONG *value);
       HRESULT set([in] SHORT n, [in] SHORT m, [in]
LONG value);
    };

// uuid and definition of IGrid2
    [     object,
          uuid(3CFDB284-CCC5-11D0-BA0B-00A0C90DF8BC),
        helpstring("IGrid2 Interface"),
        pointer_default(unique)
    ]
    interface IGrid2 : IUnknown      {
        import "unknwn.idl";
        HRESULT reset([in] LONG value);
    };

// uuid and definition of type library
[   uuid(3CFDB281-CCC5-11D0-BA0B-00A0C90DF8BC),
    version(1.0),
    helpstring("grid 1.0 Type Library)
]
library GRIDLib
{
    importlib("stdole32.tlb");
    // uuid and definition of class
    [     uuid(3CFDB287-CCC5-11D0-BA0B-00A0C90DF8BC),
        helpstring("Grid Class")
    ]
    // multiple interfaces
    coclass CGrid
    {     [default] interface IGrid1;
        interface IGrid2;
    };
};
```

CORBA IDL

```
interface grid1
{
        long get(in short n, in short m);
        void set(in short n, in short m, in
long value);
};

interface grid2
{
        void reset(in long value);
};

// multiple inheritance of interfaces
interface grid: grid1, grid2
{
};
```

Résumé

Les descriptions ont montré que les architectures DCOM et CORBA/Orbix sont fondamentalement similaires. Ils fournissent tous les deux, une infrastructure d'objets distribués accédée, par transparence, par des objets distants. Leurs principales différences sont résumées ci-dessous.
Tout d'abord, DCOM supporte les objets avec de multiples interfaces. Cela introduit aussi la notion d'un objet proxy / stub dynamiquement l'interface de chargement de plusieurs proxy / stubs dans la couche d'accès distant. Ces concepts n'existent pas dans CORBA.
Deuxièmement, chaque interface CORBA hérite de CORBA::Object, le constructeur qui exécute implicitement des tâches communes telles que l'« object registration », l'« object reference generation », le « skeleton instanciation », etc. Dans DCOM, ces tâches sont effectuées soit explicitement par le serveur de programmes ou manipulés dynamiquement DCOM par le « run-time system ».
Troisièmement, le protocole réseau de DCOM est fortement lié au RPC, contrairement à CORBA.
Enfin, nous tenons à souligner que la spécification DCOM contient de nombreux détails qui sont considérés comme des problèmes d'implémentation et non spécifié par CORBA.

CORBA vs COM : FAQ

Que Choisir entre CORBA et DCOM ?

- J'utilise les produits Microsoft ; Pourquoi devrais-je choisir CORBA plutôt que DCOM ?
  - DCOM est une solution spécifique de distribution Microsoft.
  - Les produits CORBA sont disponibles auprès de plus de 20 fournisseurs différents.
  - Les produits CORBA supportent Microsoft mais pas l'OS Windows.
  - CORBA est un excellent mécanisme de ponts entre les « Windows Desktops » et les serveurs Unix.

- Dois-je choisir entre DCOM et CORBA ?
  - Non. Les applications distribuées peuvent être développées en utilisant CORBA ou DCOM. Par exemple des objets COM peuvent accéder à des objets CORBA en cours d'exécution sur un serveur Unix. L'OMG a défini une sorte de « passerelle » qui standardise les échanges entre COM et CORBA.

- Est-ce que CORBA est plus mature que DCOM ?
  - CORBA existe depuis 1990. Des solutions commerciales sont disponibles depuis 1992.
  - DCOM était disponible en beta version en 1996.
  - CORBA a eu plus de temps pour mûrir et il y a un grand nombre de compagnies qui ont développé en CORBA ORB. Dans l'ensemble c'est cette concurrence qui a certainement contribué à rendre CORBA plus robuste.

- Quels avantages a DCOM de plus que CORBA ?
  - DCOM est bien adapté à des développements d'application de type « front-end ».
  - Si toutes les applications distribuées fonctionnent sous les plateformes Microsoft, DCOM peut être le bon choix.
  - DCOM peut également être utilisé avec CORBA. La question n'est pas toujours devrais-je utiliser CORBA ou DCOM ?

CORBA vs COM : Principaux Inconvénients

- Difficultés pour les faire marcher à travers des firewalls et sur des machines inconnues et non sécurisées.
- Plus difficile à mettre en œuvre que d'autres solutions comme les web services par exemple.

CORBA vs COM : Comparaison de Web Services, RMI, CORBA, DCOM

	Web Service	RMI	CORBA	DCOM
Transport Protocol	Http	JRMP	IIOP	ORPC
Data Binding	XML schema (allow customized data type)	Primitive, Serialized objects	IDL (primitive and structure)	MS IDL
Compliant prog. Langs.	Any (with XML parser, SOAP composition)	java	Any (with IDL mapping standards)	Many (C++, Java, VB, etc.)
Interface Description	WSDL	Interface of server objects	IDL	MS IDL
Remote Call	By SOAP message	Get references of server objects	Get reference of server object	Get Pointer of server objects
Routine	Stub Proxy DII	Client: stub or proxy Server: skeleton	Client: stub or proxy Server: skeleton	Client: proxy Server: stub

Bibliographie

CORBA

- **http://www.omg.org/gettingstarted/corbafaq.htm**
- **http://www.wikipedia.com**
- **http://corba.developpez.com**
- **http://www.sylbarth.com/corba/corba.html**
- **http://www.iona.com/support/docs/orbix/gen3/33/html/orbix33cxx_intro/Chapter_01.html**

COM

- **http://www.microsoft.com/COM/**
- **http://windows.developpez.com/faq/dcom/**

CORBA, COM et les autres

- **Middleware et Internet de Daniel Serain**
- **http://research.microsoft.com/en-us/um/people/ymwang/papers/html/dcomncorba/s.html**
- ***http://www.flydragontech.com/ebcourseWinter2008/9_comparison.ppt***

Table des illustrations

Figure 1 : Architecture CORBA 4
Figure 2 : Structure et scénarion d'un BOA..... 5
Figure 3 : RPC côté client 7
Figure 4 : RPC côté serveur 7
Figure 5 : RPC Structure 10
Figure 6 : DCOM overall architecture 10
Figure 7 : CORBA overall architecture..... 10