République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abou-Bekr Belkaid - Tlemcen
Faculté des Sciences de l'Ingénieur
Département d'Informatique

Laboratoire de Télécommunications

MEMOIRE

Projet de Fin d'Etudes pour l'Obtention du Diplôme
d'Ingénieur d'Etat en Informatique
Option: Informatique Industrielle

THEME

Développement d'une Application Distribuée par la norme CORBA

Présenté par: AHMAT HISSEINE OUSMANE

Soutenu publiquement le 06 juillet 2006

Devant le jury d'examen composé de :

Mr	CHIKH Mohamed Amine	Président
Mr	BENZIAN Yaghmoracen	Examinateur
Mr	B. BENADDA	Examinateur
Mr	A. BENAMAR	Encadreur

Promotion 2005/2006

Dédicaces

Je dédie ce modeste travail :

- à mes chers parents pour leurs encouragements, leur soutien moral, spirituel et leur tolérance durant toutes mes années d'études ; tous les mots restent faibles pour exprimer mes sentiments ; qu'ils trouvent à travers ce travail les fruits et la récompense de leurs efforts.

- à mon grand frère Mahamat Saleh H. et à Mahamat Elhadj Abdoulaye pour leur soutien sans faille durant toutes ces années d'études en Algérie ;

- à tous mes frères et sœurs en particulier à Marco, et à tous les cousins ;

- à la mémoire de mon cher frère et ami Dr Saleh Haroun S ;

- à la mémoire de mon petit frère Oumar Abdoulaye Chérif ;

- à toutes celles et à tous ceux qui me tiennent à cœur et qui ont quitté cette vie, paix à leur âme ;

- à tous mes ami(e) s tchadiens et algériens ;

- à toute la communauté makanda d'Algérie particulièrement à celle de Tlemcen ;

- aux familles Abderrahim Hadjaro, Abdel Salam Bouari au Tchad ;

- aux familles TAÎBI, AZZOUZ, BENADLA, particulièrement au maître de karaté Djillali TAIBI, AZZOUZ M, Amine le mignon en Algérie ;

- à la jeunesse algérienne et tchadienne ;

- à toutes celles et à tous ceux qui m'ont aidé dans ce travail et dans mes études ;

- à tous les adeptes de la philosophie rasta et les défenseurs de la justice ;

- à toutes et à tous ceux qui sont victimes d'injustice sur cette terre ;

AHMAT.Hisseine O.

Remerciements

A l'issue de ce modeste travail, effectué au laboratoire de Télécommunications de l'université Abou Bekr BELKAÏD-Tlemcen, sous l'assistance de mon encadreur Monsieur **BENAMAR Abdelkerim** à qui il m'est agréable d'exprimer mes remerciements les plus sincères et ma profonde reconnaissance pour son aide. Je suis sensible à la confiance et à l'honneur qu'il m'a accordés en acceptant de diriger ce travail. Son attention, sa bienveillance et son appui sans faille ont été des encouragements décisifs pour mener à terme ce projet. Ses suggestions, ses encouragements, ses enseignements et son soutien m'ont été très précieux. Je lui exprime ma vive et respectueuse gratitude.

Je voudrais exprimer ma gratitude à la coopération Algéro-Tchadienne dont je suis bénéficiaire. Je remercie particulièrement l'Algérie à qui je dois ma formation d'Ingénieur d'Etat en Informatique.

Je remercie profondément **Mr CHIKH Mohamed Amine**, de l'honneur qu'il me fait de présider ce jury de soutenance ; je suis particulièrement reconnaissant à son accueil, ses conseils toujours chaleureux et attentionnés.

Ma profonde et vive gratitude s'adressent à Messieurs **BENADA Belkacem** et **BENZIAN Yaghmoracen**, pour l'honneur et le privilège qu'ils me font d'examiner ce travail.

Je témoigne vivement de ma profonde reconnaissance à toutes les enseignantes et à tous les enseignants du département d'Informatique particulièrement à celles et à ceux qui ont contribué à ma formation.

Je suis infiniment reconnaissant au générale Dadoum D Jaures et TRAORE BRAMA, pour leurs conseils indélébiles ; les mots me manquent pour leur exprimer ma gratitude.

Je remercie vivement toute la communauté des étudiants étrangers de l'Algérie, particulièrement ceux de Tlemcen. Ces remerciements vont

spécialement à Oumar A,Chahallah Y, le SG Touré B, Bechir H,Moustapha A, Mht Nour N, Mht Baba S, Barry K, Moussa A Cheny, Dallah, Dr Sakine M, mon guerrier Moussa,Acheik Abdoulaye Chérif, Mht T Hadjaro, MOULAY Mezian, KOITA B qui ont utilisé toutes leurs énergies pour m'aider durant ce travail. Les mots me manquent pour leur exprimer mon amitié.

Je remercie tous les étudiants tchadiens d'Algérie particulièrement ceux de Tlemcen pour leur bonne collaboration.

A toute personne ayant contribué d'une manière ou d'une autre à ce travail, je dis ma sincère gratitude.

J'exprime toute ma sympathie à mes promotionnels; je garde un bon souvenir de chacune et de chacun.

Enfin, c'est par un grand hommage à ma famille que je me permets de tourner cette page. Mes parents, mes sœurs, mes frères et mes amis. Je les remercie de leurs efforts.

Développement d'une application par la norme CORBA OUSMAN A Hissien 2006

Résumé

L'évolution des réseaux n'est pas sans effet dans les modes d'échanges des données entre les entreprises. Très vite, suite aux limites rencontrées dans les architectures centralisées, d'autres types d'architectures dites distribuées ont vu le jour. Ces dernières sont caractérisées par une évolution du client serveur jusqu'aux technologies par objets distribués.
La technologie CORBA (Common Object Request Broker Architecture) de l'OMG (Object Management Group) s'inscrit naturellement dans ce sens. En effet, elle s'articule autour d'un bus logiciel appelé ORB (Object Request Broker) qui fournit une plate-forme d'exécution interopérable tant au niveau matériel que logiciel.

Mots clés : architectures centralisées, architectures distribuées, client /serveur, objets, CORBA, ORB, bus, interopérable.

Abstract

The evolution of networks is not without effect in the way to change data between enterprises. Early, by the limit of centralized architectures, other kinds of architectures called distributed are born. These one are characterized by the evolution of client/server until distributed objects.
By it side, the CORBA (Common Object Request Broker Architecture) technology of OMG (Object Management Group) enrols naturally in this way. Indeed, it articulate around the bus called ORB (Object Request Broker) that provides a platform of interoperable execution either in hardware or in software.

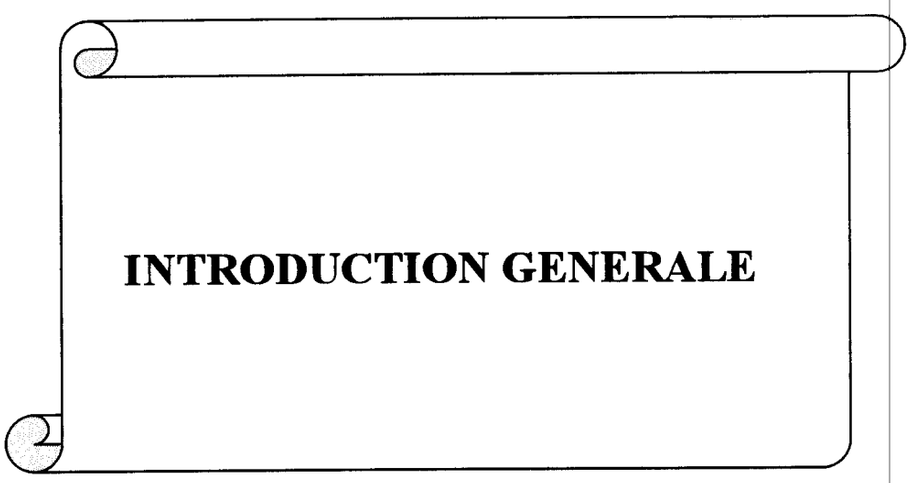
Key words: centralized architectures, distributed architectures, client/server, objects, CORBA, OMG, bus, ORB, interoperable.

Liste des figures

Page

« Les grandes œuvres se distinguent par leur accessibilité car elles n'appartiennent pas au patrimoine de quelques élus mais à celui de tous les hommes doués de bon sens. »

Manuel Gonzalez Prada



INTRODUCTION GENERALE

Les besoins croissants de s'informer de manière fiable et efficace, l'évolution des réseaux et technologies de communication au sein des entreprises, mais aussi l'évolution des modes d'échanges des données entre les entreprises, s'inscrivent parfaitement dans le phénomène de mondialisation et de suppression des frontières au sein des entreprises et de la société et, par ailleurs induisent des bouleversements dans l'ingénierie informatique. Des programmes monolithiques sur une machine centrale, on est passé en quelques années seulement à la mise en place des véritables applications tournant sur plusieurs machines géographiquement éloignées et parfois très hétérogènes.

D'abord, la question qui se pose est de savoir comment garantir la facilité d'évolution de telles applications en permettant de modifier et d'en ajouter des composants logiciels et, ainsi donc, assurer un développement incrémental et progressif ?

Ensuite, c'est aussi à se demander comment intégrer et faire coopérer des logiciels existants en garantissant un fonctionnement cohérent des nouvelles comme des anciennes applications informatiques ?

Enfin, comment minimiser les coûts de développement inhérents aux facteurs de répartition en cachant au mieux les problèmes qui en découlent ?

Les réponses à ces questions sont donc aussi bien un enjeu stratégique qu'un défi technique pour les concepteurs/développeurs de ces applications et les ont conduit à mettre en place de nouveaux outils et concepts de programmation.

Parmi ces concepts, le modèle de programmation orienté objets fournit un support à la modélisation du monde réel pour la mise en oeuvre de nouveaux services.

Le concept de la programmation repartie, de son côté devient incontournable, non seulement pour des raisons d'éloignement géographique, mais aussi pour permettre aux utilisateurs de partager des informations et des ressources matérielles. De ces modèles est né un certain nombre de standards tel que **ODP (Open Distributed Processing) de l'ISO-ITU-T, java RMI de Sun, COM (Component Object Model) /DCOM (Distributed Component Object Model) de Microsoft, les spécifications CORBA (Common Object Request Broker Architecture) de l'OMG (Object Management Group)**, etc. Ces standards doivent offrir un modèle global pour supporter les

besoins d'uniformité de l'informatique repartie : interopérabilité, transparence, efficacité, distribution et réutilisabilité des objets issus de ces applications.

Le dernier standard cité ci-dessus servira de support pour notre travail de mémoire de fin d'études qui consiste à développer une application à travers le bus CORBA dans un environnement reparti.

La mise en œuvre de ce travail s'articule autour de quatre chapitres et sera structurée comme suit :

Le premier chapitre détaille l'architecture centralisée et ses limites ainsi que les motivations ayant poussé l'avènement des architectures distribuées.

Le deuxième chapitre présente le standard de développement reparti de l'OMG qu'est CORBA

Le troisième chapitre illustre le langage IDL (interface Definition Language) utilisé pour la spécification d'interface et les différents services de l'architecture OMA (Object Management Architecture).

Le quatrième et dernier chapitre met en exergue notre mise en œuvre pratique des concepts théoriques sur un fragment d'application de commerce électronique.

Enfin, le tout sera suivi d'une conclusion générale qui fera la synthèse de notre travail en ouvrant de nouvelles perspectives.

I Introduction

La technologie des réseaux fait aujourd'hui irruption au coeur du système de communication des entreprises. Ces dernières voient leurs dimensions grandir spacialement et doivent donc réaliser la coopération de leurs activités, évoluer leurs applications et naturellement les faire interopérer.

C'est dans cette optique que l'adoption des standards s'avère tant nécessaire qu'incontournable.

Dans ce chapitre, nous esquissons les évolutions des différents modèles d'architectures ainsi que les technologies de standardisations sous jacentes.

II Les architectures centralisées

II .1 Définition

On appelle par architecture centralisée, une architecture dont la logique de traitement est concentrée au niveau de la machine hôte. L'utilisateur n'interagit avec celle-ci qu'au travers d'un terminal passif généralement en mode caractère.

C'était surtout l'architecture des legacy systems, c'est-à-dire les environnements existants avant l'introduction des nouvelles visions des systèmes d'informations, notamment avant l'utilisation du modèle client/serveur et du concept orienté objet. Le mainframe ou machine hôte centralise toute l'intelligence du système, les traitements et le stockage des données. L'interface utilisateur (passive) ne sert qu'à la présentation à distance des données et au contrôle [1].

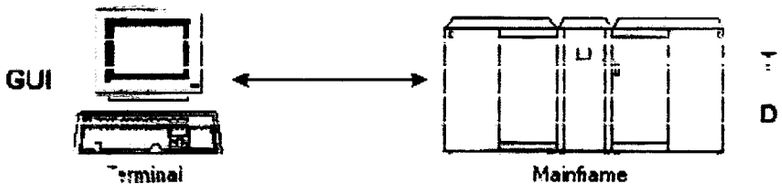


Figure 1.1 Architecture centralisée de type Terminal/Mainframe

Caractéristiques:

- Systèmes propriétaires, provenant d'un seul OS (Operating System);

- Applications monolithiques;
- Traitements au niveau du Serveur Central;
- Gros systèmes mêlant interfaces, règles métiers (logique applicative) et stockages de données;
- Terminaux passifs.

II .2 Avantages et inconvénients

Avantages ;

- Facilité d'administration;
- Performance : la centralisation de la puissance sur une seule et même machine permet une utilisation optimale des ressources et se prête très bien aux traitements par lots (en *batch*).
- Sécurité et fiabilité;

Inconvénients :

- Interface utilisateur en mode caractère peu convivial;
- Systèmes non ouverts vers d'autres, dépendance d'un fabricant particulier.
- Maintenance logicielle peu aisée et délicate, et donc un coût très élevé pour la maintenance logicielle.
- Problèmes de compatibilité : les « Hosts » (ou «Minis»), n'étaient parfois pas compatibles dans la gamme d'un même constructeur. Le simple passage d'un modèle à un autre au sein du même constructeur impliquait souvent une réécriture des applications.

III Les Architectures distribués

III .1 Définitions

Un système réparti (SR) « Distributed System (DS) » se compose d'un ensemble d'ordinateurs reliés par un réseau informatique et équipés d'un logiciel réparti.

Le logiciel d'un SR permet à des ordinateurs de coordonner leurs activités et de partager les ressources du système : équipements, logiciels et données [1][2].

Cependant, on introduit la notion de transparence selon laquelle un système réparti est un ensemble d'ordinateurs indépendants apparaissant à l'utilisateur comme un système unique et cohérent, dans lequel :

- les machines sont autonomes ;
- les utilisateurs ont l'impression d'utiliser un seul système.

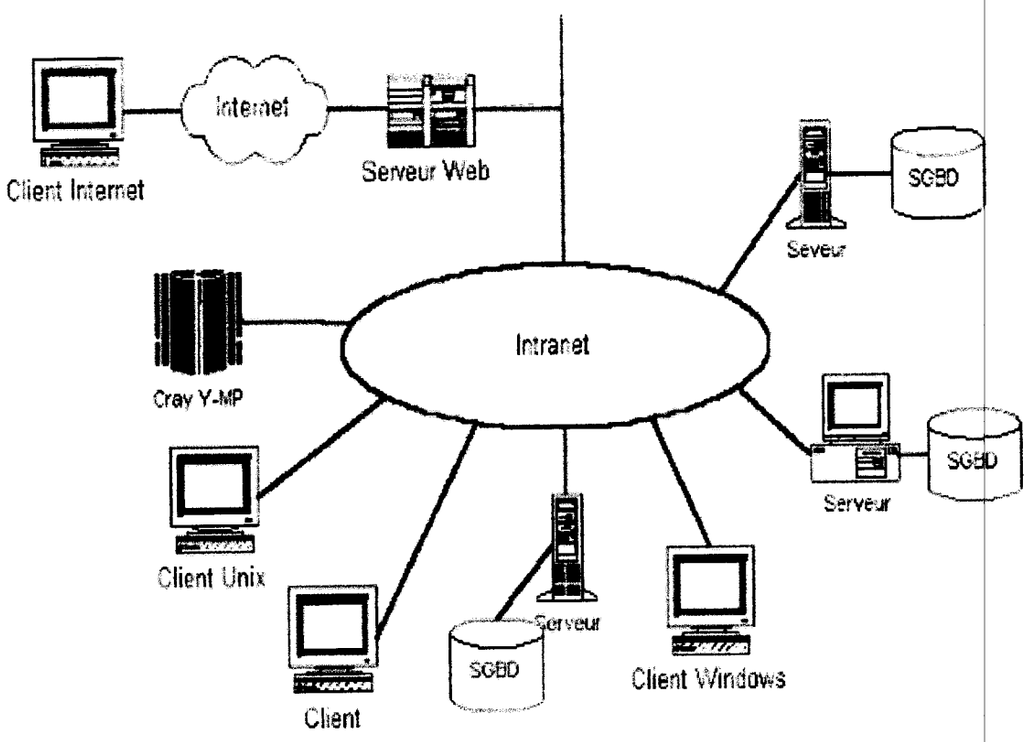


Figure I.2 Un exemple de système distribué

Caractéristiques [1][2] :

- Partage de données entre les utilisateurs

  Par exemple : système de réservation aérienne

- Partage de périphériques coûteux

  Par exemple : imprimante laser couleur, périphériques d'archivage

• Scalabilité : fonctionner dans différentes échelles (de quelques postes et serveur(s) à plusieurs centaines de postes de travail et de serveurs, à plusieurs réseaux locaux reliés par Internet).

• Concurrence : Plusieurs processus s'exécutent en parallèle (ou pouvant s'exécuter en parallèle).

• Transparence : de données, d'emplacement, dans le simple but de masquer la distribution.

La notion de transparence correspond à la possibilité d'accéder à un service ou à une donnée indépendamment de certaines propriétés liées à la distribution. Cette notion de transparence présente différents aspects [3]:

- Uniformité des accès : correspond à la possibilité d'accéder à une ressource par une interface unique, que cette ressource soit locale ou distante (la séparation physique entre machines et les différences matériels/logiciels pour les accès sont invisibles par l'utilisateur).

- Localisation : exprime la possibilité d'accéder à une ressource sans en savoir la localisation (localisation des ressources non perceptible).

- Réplication : permet d'accéder à des exemplaires multiples de ressources sans en connaître l'existence à des fins de performances et/ou de fiabilité;

- Mobilité / Migration: autorise la mobilité (ou migration) d'objets (ressources ou processus) sans affecter le déroulement des opérations en cours (sans modification de leurs noms, de la manière d'y accéder et de l'environnement d'un utilisateur).

- Défaillance: correspond au masquage des pannes de sites ou d'inaccessibilité des objets jusqu'au recouvrement de l'erreur correspondante.

• Ouverture :

- Permettre l'ajout d'autres composantes logicielles ou matérielles sans perturber le fonctionnement du système.

- Flexibilité (facilité d'utilisation et de configuration).

- Interopérabilité des matériels (de fournisseurs différents).

• Tolérance aux pannes :

- Cas de pannes détectés et récupérés.

- Redondances (réplication) matérielles et logicielles.

- Permettre à un utilisateur/programme de ne pas s'interrompre (ou même se rendre compte) à cause de la panne d'une ressource.

● Performances : un système réparti peut offrir la possibilité de distribuer les traitements sur plusieurs machines ou de permettre l'exécution d'une application sur un serveur adapté à ses besoins.

III .2 Avantages et Inconvénients

Avantages [2][3] :

● L'hétérogénéité : plusieurs machines de nature différente peuvent être reliées via un même médium de communication et ainsi pouvoir échanger de l'information.

● La croissance modulaire : la configuration d'une architecture distribuée peut croître en fonction des besoins réels des utilisateurs.

● Une plus grande disponibilité : un élément défaillant d'une architecture peut être mis hors service sans entraîner un arrêt complet du système.

Inconvénients :

● Logiciel : manque d'expérience dans la conception, la mise en œuvre et l'utilisation des logiciels distribués.

● Réseau de communication

- saturation
- perte de messages

● Difficulté d'administration : l'administration du système est gérée par tous les sites

● Sécurité : difficulté d'assurer la protection des données confidentielles réparties dans des sites géographiquement éloignés.

IV Quelques définitions

IV .1 Définition d'un serveur

On appelle serveur un programme qui **offre un service** sur le réseau. Le serveur accepte des requêtes, les traite et renvoie le résultat au demandeur. Le terme serveur s'applique aussi bien à la machine sur laquelle s'exécute le logiciel serveur qu'au logiciel lui-même [2]. Pour pouvoir offrir ces services en permanence, le serveur doit être sur un site avec **accès permanent** et s'exécuter en

permanence. Un programme serveur peut gérer, en général, plusieurs requêtes à la fois provenant de clients différents ou identiques.

IV .2 Définition d'un client

On appelle client un programme qui **utilise le service** offert par un serveur. Le client envoie une requête et reçoit la réponse. Un programme client peut, en général, utiliser simultanément plusieurs connexions vers des serveurs différents ou identiques [2].

IV .3 Identification d'un service

Un service est un comportement défini par contrat, qui peut être implémenté et fourni par un serveur pour être utilisé par un client, sur la base exclusive d'un contrat .Un site peut offrir plusieurs services. Chacun de ces services est fourni sur **un port de communication** identifié par un **numéro**. Ce numéro identifie le service quel que soit le site (le service HTTP est offert sur le port numéro 80, FTP le numéro 21, TELNET le numéro 23, ...). Pour accéder donc à un service, il faut l'adresse du site et le numéro du port [2].

IV .4 Architecture client/serveur

Selon Alain Lefèbvre « est conforme au client-serveur une application qui fait appel à des services distants au travers d'échanges de message (les requêtes) plutôt que par un échange de fichiers» [2].

A notre avis, cette définition de Lefèbvre nous paraît assez restrictive, c'est pourquoi nous choisissons la définition suivante qui nous semble relativement exhaustive : tout couple de processus, séparés logiquement, résidant éventuellement sur la même machine physique et dialoguant entre eux afin, pour l'un, de demander un service et pour l'autre de répondre à ce service, peuvent prétendre participer à une relation de type « Client - Serveur » [1].

En effet, c'est la description du **fonctionnement coopératif** entre le serveur et le client (les services Internet sont conçus selon cette architecture). Ainsi, chaque application est composée de logiciel serveur et logiciel client. A un logiciel serveur, peut correspondre plusieurs logiciels clients développés dans différents environnements: Unix, Mac, PC...; la seule obligation est le

respect du protocole entre les deux processus communicants. Ce protocole étant décrit dans un RFC (Request For Comment).

Un programme s'exécutant sur une machine logique ayant un support de connexions (ou ports) vers d'autres ordinateurs au travers d'un réseau offre ses services. Sur une autre machine logique, proposant elle aussi un support de connexions (ou ports) vers d'autres ordinateurs au travers d'un réseau, s'exécute un programme qui émet des requêtes vers des machines faisant office de serveur, afin d'utiliser les services proposés. C'est l'OS (Operating System) qui se charge d'attribuer librement au client le port qu'il veut. Ainsi donc, le serveur reçoit la requête et répond à l'aide de l'adresse IP de la machine client et son port [1].

Ces deux machines logiques (car le programme client peut fonctionner sur la même machine physique que le programme serveur) utilisent un ensemble de protocoles pour communiquer entre elles. Ces protocoles sont un ensemble de règles à respecter pour pouvoir communiquer correctement selon le but recherché (transmission d'informations confidentielles, transmission rapide d'informations peu importantes, etc.).

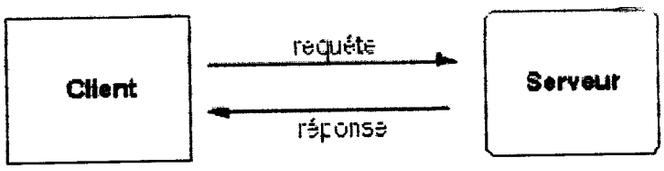


Figure 1.3 Client/Serveur

V. Les évolutions des différents modèles d'architectures client-serveur

V.1 Le modèle client-serveur à deux niveaux

Au cours des années 1980, les évolutions du paysage informatique ont remis en cause les conditions même d'organisation du travail et de production des logiciels. Elles ont été caractérisées par[2] :

- une augmentation des performances matérielles de la micro-informatique (vitesse des processeurs, capacité de stockage accrue) ;

- une fiabilité accrue du matériel ;
- une diminution significative des coûts ;
- une banalisation de l'utilisation de logiciels de bureautique qui ne sont plus seulement réservés à des informaticiens ;
- l'arrivée des interfaces en mode graphique.

Ceci a créé les conditions d'une nouvelle approche des systèmes d'information en rupture avec une architecture basée sur un ordinateur central (*mainframe*) et des terminaux (clavier-écran) qui lui sont reliés. Les années 1990 voient émerger une nouvelle architecture d'application nommée de manière générique « client-serveur » qui repose sur le mécanisme suivant :

1. une application sur le poste client formule une requête vers une autre application présente sur un ordinateur serveur ;
2. le serveur exécute le traitement demandé par le client (par exemple l'interrogation d'une base de données) et retourne les résultats ;
3. le client récupère le résultat et prend en charge au moins sa présentation.

C'est ce qu'on rencontre souvent avec les systèmes de gestion des bases de données (SGBD). Dans ce cas de figure on fait la séparation en deux couches physiques. D'un côté il y a la machine du client et de l'autre la machine du serveur. Le serveur fournit directement les ressources demandées par le client sans qu'il ne fasse recours à une autre application [1].

Dans la figure ci-dessous, les traitements sont effectués sur le poste de travail. Les interfaces utilisateurs servent aux contrôles et à la présentation locale au poste de travail et la gestion des données se déroule sur le serveur.

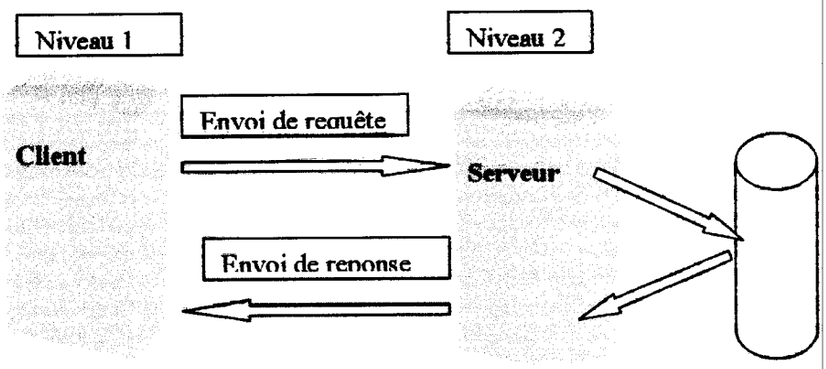


Figure 1.4 :l'architecture Client-serveur à 2 niveaux.

V .2 Le modèle client-serveur à trois niveaux

Appelé également architecture à 3 niveaux. On parle dans ce cas d'architecture partagée car il existe un serveur intermédiaire (généralement un serveur de base de données) hormis le client et le serveur d'application. Ce dernier requiert les services du serveur intermédiaire pour satisfaire la requête du client [1].

Dans ce cas on dispose de trois supports physiques et on distribue les différentes couches logiques sur les trois. La couche d'application peut être séparée des serveurs de données SGBD.

Les traitements sont effectués sur le poste de travail. Les interfaces utilisateurs servent aux contrôles et à la présentation locale au poste de travail dans l'exemple illustré par la figure 1.5.

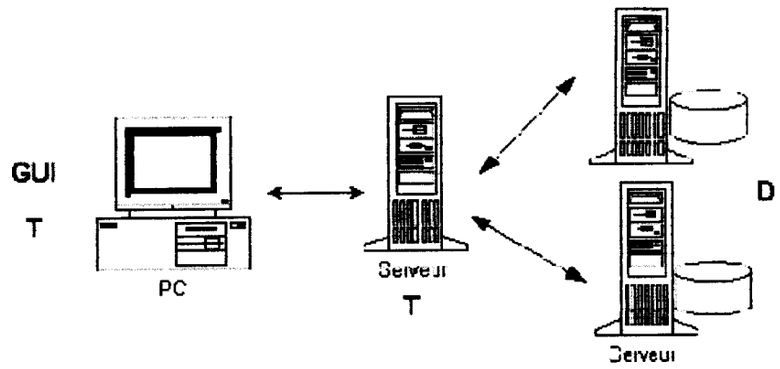


Figure 1.5 Architecture Client/Serveur 3 couches

V. 3 Architecture multi niveaux

Comme nous l'avons vu précédemment dans l'architecture à 3 niveaux, chaque serveur fournit une tâche bien définie. Cette architecture peut donc se résumer en une imbrication de niveaux. Dans une vision beaucoup plus étendue, l'architecture à 3 niveaux s'assimile à une architecture à n niveaux comme illustré par la figure suivante :

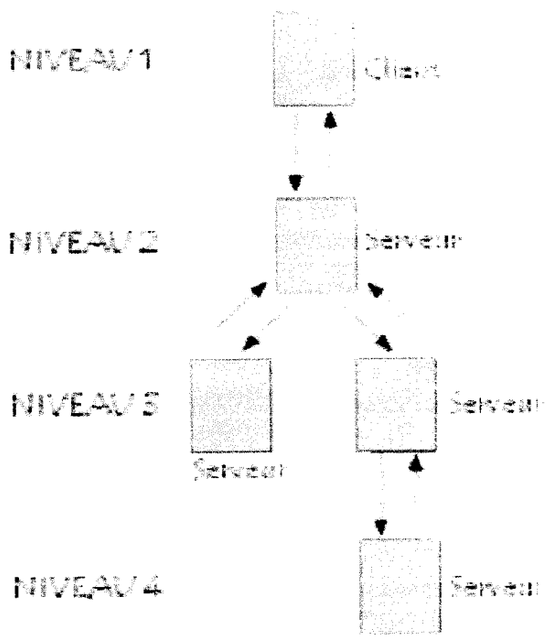


Figure 1.6 :l'architecture Client-serveur multi niveaux [1]

V.4 Avantages du modèle Client-serveur

Le modèle Client-serveur est particulièrement recommandé pour des réseaux nécessitant un grand niveau de fiabilité ; ses principaux atouts sont [2]:

- **une meilleure sécurité**: Lors de la connexion un client ne voit que le serveur, et non les autres clients. De même, le nombre de points d'entrée permettant l'accès aux données est limité.
- **une administration au niveau serveur**: les clients sont administrés par le serveur.
- **facilité d'évolution**: Une architecture client/serveur est évolutive car il est très facile de rajouter ou d'enlever des clients, et même des serveurs sans perturber le fonctionnement de l'ensemble et sans modifications majeures.
- **meilleure fiabilité :** En cas de panne, seul le serveur fait l'objet d'une réparation, et non le client.
- **unicité de l'information :** toutes les informations sont stockées dans le serveur. De cette manière, les informations restent identiques, ce qui permettrait à chaque client d'accéder toujours aux mêmes informations.

V. 5 Inconvénients du modèle Client-serveur

Malgré ces nombreux avantages, cette architecture a cependant quelques inconvénients qui sont [2] :

- **un coût élevé** dû à la technicité du serveur.
- **un maillon faible**: arrêt du système en cas de panne du serveur,
- **la faible interopérabilité :** le modèle dépend d'un fabricant particulier (solution propriétaire).
- **Problèmes de congestion en entrée du serveur :** lors de l'arrivée massive de requêtes (congestion de communication).
- **Problèmes de performances** : lors de la connexion d'un très grand nombre de clients en même temps.

A la vue de ces nombreux avantages par rapport à ces inconvénients, on pourrait croire que cette architecture est relativement idéale, celle qui va s'imposer dans le développement d'application de grande envergure, mais ce serait oublier cependant quelques détails :

- la communication entre le client et le serveur n'est pas standardisée alourdissant ainsi de surcroît la charge de développement [3].
- le concept d'objet n'apparaît pas ici : on appelle à distance des traitements sous forme de RPC (Remote Procedure Call) comme si ces traitements sont locaux. En agissant ainsi, on passe à côté des beaux principes de réutilisabilité, de maintenance, de fiabilité prônés et mis en évidence par la programmation orientée objets.
- en outre, il manque une souche commune (appelons là pour l'instant "logiciel du milieu") à notre disposition pour simplifier les développements et, surtout évitant ainsi de réécrire systématiquement certains services de l'application [3].

De là, surgissent des dérivatifs ayant causé l'éclosion d'une nouvelle architecture dite distribuée dans laquelle les concepts d'objets, de protocole standard ou encore d'outils sont omniprésents déclassant en effet le mode de programmation procédurale et plaçant à son créneau la programmation répartie.

VI la programmation orientée objet

VI. 1 Définition d'un objet

Le monde dans lequel nous vivons est constitué d'objets. Ils sont caractérisés par leur taille, leur poids, leur couleur, etc. Certains de ces objets sont très petits tel un grain de sable, d'autres sont très grands comme un immeuble. A la différence du grain de sable et de l'immeuble qui ont une existence très concrète, grâce à leur aspect matériel, nous utilisons également des objets de nature "virtuelle" n'ayant pas d'existence matérielle comme un compte en banque, ... Nous sommes donc confrontés en permanence à des objets. Une des grandes caractéristiques de l'intelligence humaine est sa capacité à manipuler des concepts complexes à différents niveaux d'abstraction et ce, de façon quasiment innée. Ainsi, les méthodes actuelles de développement informatique permettent de manipuler le concept d'objets.

L'informaticien et le non informaticien peuvent avoir un langage commun basé sur ce concept. Un objet au sens informatique du terme permet de désigner une représentation "abstraite" d'une chose concrète du monde réel ou virtuel. Un objet (au sens informatique, que nous appellerons maintenant objet) présente les 3 caractéristiques suivantes :

Objet = État + Comportement + Identité

L 'état d'un objet définit la valeur des données (ou attributs) ; par exemple dans le cas d'un objet Moto, celui-ci pourrait être caractérisé par les attributs suivants : la marque, la couleur, le poids, la puissance, la quantité d'essence... Ce que l'on représente graphiquement par :

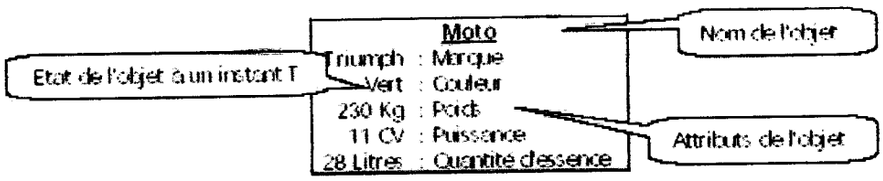


Figure 1.7 : Représentation de l'état d'un objet

L'état de l'objet peut être amené à changer durant son cycle de vie. Par exemple, la quantité d'essence et le poids de la moto varient en permanence lorsque celle-ci roule.

Le comportement d'un objet indique toutes les compétences de celui-ci et décrit les actions et les réactions qu'il peut avoir. Chaque élément de base du comportement est appelé opération. Les opérations d'un objet sont déclenchées suite à une stimulation externe de l'utilisateur qui appuie sur un bouton ou du programmeur qui appelle une opération. Il existe un lien très étroit entre le comportement d'un objet et son état.

Effectivement, l'état d'un objet peut être modifié par l'appel d'une opération et les actions effectuées par une opération dépendent de l'état de l'objet. Reprenons notre exemple de l'objet Moto. Nous pourrions définir les opérations Démarrer, Rouler, Stopper. L'opération Démarrer n'a de sens que si l'information "moteur éteint" est vérifiée. De même, l'opération Stopper n'aura d'effet que si les opérations Démarrer et Rouler ont déjà été exécutées modifiant ainsi l'état de l'objet moto de "moto stoppée" à "moto en déplacement".

L'identité : En plus de son état et de son comportement, un objet possède une identité qui caractérise son existence propre. L'identité permet de distinguer tout objet de façon non ambiguë, et cela indépendamment de son état. Cela permet, entre autres, de distinguer deux objets dont toutes les valeurs d'attributs sont identiques. L'identité est souvent construite à partir d'un identifiant issu naturellement du domaine du problème.

Ainsi une moto est identifiée par son numéro de série, un compte en banque par son numéro de compte.

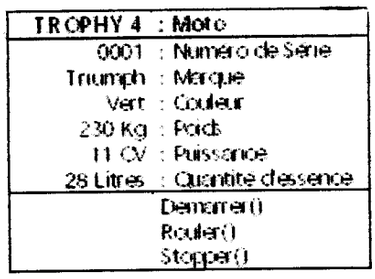


Figure 1.8 : Représentation complète d'un objet Moto selon la méthode UML

La partie gauche illustre un objet particulier (ou instance), la "Trophy 4" de classe "Moto" illustrée dans la partie droite.

Comme son nom l'indique une classe permet de définir une classification. Ainsi tous les objets "moto" partagent une même classe à savoir la "classe moto". Les notions de classe et d'objet sont relativement difficiles à définir indépendamment l'une de l'autre. La classe d'un objet définit par ses attributs et ses opérations, l'état, le comportement et l'identité d'un objet de façon abstraite ; alors qu'un objet est une représentation concrète d'une classe. On parle d'instance d'une classe.

Dans notre exemple la classe moto définit que tout objet de classe moto comporte un numéro de série, et l'instance TROPHY 4 de la classe moto défini un objet unique via son numéro de série.

VI.2 Caractéristiques de la programmation orientée objet

Dans la programmation orientée objet, les objets ont essentiellement des attributs et sont doués de comportements. Les attributs décrivent l'objet et permettent de le distinguer des autres. Le comportement est la description des actions que peut effectuer l'objet.

Les principales caractéristiques de l'orienté objet sont l'encapsulation, l'héritage et le polymorphisme.

L'**encapsulation** est le principe qui permet de regrouper les données et fonctions au sein d'une classe. Ceci permet de protéger les données encapsulées des accès intempestifs. Elle garantit aussi aux utilisateurs des objets de ne pas se préoccuper de la réalisation des abstractions fournies par les objets.

On entend par **héritage** la relation entre classes qui permet le partage de propriétés définies dans une classe. L'héritage favorise la réutilisation des classes.

Le **polymorphisme** est cette technique de programmation qui s'applique aux méthodes et permet à ces dernières de réagir différemment en fonction du type d'objet traité.

VI. 3 les objets distribués ou répartis

Un objet distribué est un composant logiciel capable d'interagir et de collaborer avec d'autres composants objets à travers un réseau, sur différents systèmes d'exploitations [4]. Selon ce même auteur, ces objets peuvent être implémenter avec différents langages de programmation, dans des applications indépendantes et sur des machines de constructeurs hétérogènes.
Le concept des objets combiné avec l'architecture client serveur permet de construire des applications informatiques très performantes.

VI. 4 Application Middleware

Le middleware (ou élément intermédiaire) désigne l'ensemble des composants logiciels nécessaires à la communication entre des programmes indépendants souvent distants. Il permet de faire l'intégration et l'interopérabilité entre programmes conçus avec différents langages ; entre un programme Java et un programme C++ par exemple.

Le middleware sert de schéma de base aux applications réparties, il facilite le changement de répartition des couches sur les différents supports physiques du système.

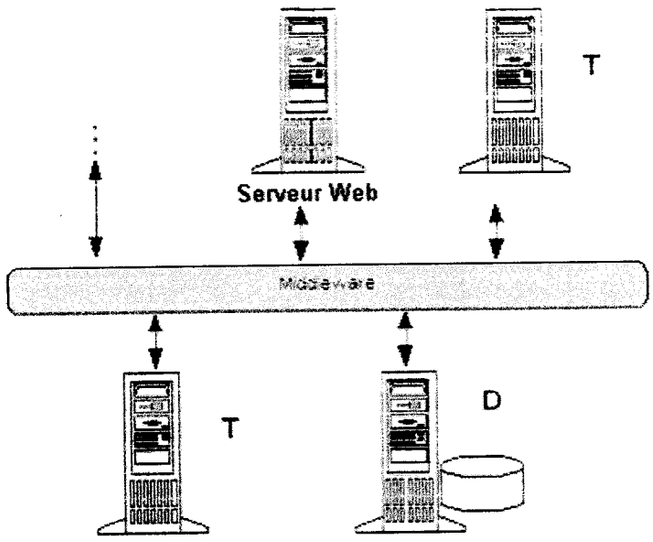


Figure 1.9 : Bus middleware [4]

VII. Les technologies de la programmation distribuée

VII .1 Objectifs de la programmation distribuée

L'objectif de la programmation distribuée est de produire une transparence dans la localisation des objets distants qui se trouvent sur une autre machine, et faciliter ainsi l'accès et l'utilisation de ces objets distants comme des objets locaux.

Les fonctionnalités apportées par cette technique de programmation sont alors [4] :

- localiser et charger de manière dynamique les classes distantes,
- localiser et produire des références sur ces objets distants,
- permettre des appels distants sur ces objets, en les passant en paramètre comme arguments et les retourner comme valeurs de traitements,
- enfin notifier aux applications utilisatrices sur le réseau des succès de contact, des échecs survenus et autres problèmes.

VII.2 Le Remote Procedure Call (RPC)

L'appel de procédures distantes (RPC : Remote Procedure Call) est un moyen pour exécuter des programmes distants à travers le réseau. Le but du RPC est de cacher au processus

demandeur le fait que l'appel se déroule à distance et de permettre la programmation aisée de programmes client/serveur sous une forme transparente aux utilisateurs.

Les RPC étendent donc la notion d'appel procédural à un ensemble des machines [6].

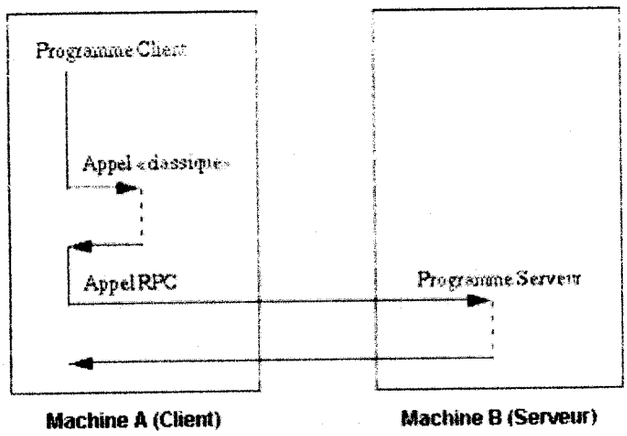


Figure 1.10 : Remote Procedure Call [4]

Le RPC va permettre de distribuer le programme principal et les procédures sur des systèmes distants :

- le programme principal devient un programme client,
- les procédures appelées constituent un programme serveur.

Ainsi, au lieu de manipuler directement des sockets, avec le RPC, le programmeur va avoir l'illusion de faire des appels à des procédures locales alors même que ces appels impliquent une communication avec un hôte distant pour envoyer et recevoir des données.

Comme il simule un appel de procédure, c'est un mécanisme synchrone, puisque l'appelant reste bloqué jusqu'à la terminaison de la procédure [4]. La difficulté essentielle à résoudre pour offrir un service d'appel de procédures à distance est le maintien des propriétés implicitement associées à un appel de procédure locale :

- Codage et décodage des arguments des procédures et de la valeur retournée par les fonctions,
- effets de bord permanents,
- fiabilité (la procédure retourne toujours un résultat).

VII.3 Le Message Passing (MP)

Le Message Passing est le moyen par lequel deux processus communiquent entre eux en s'envoyant des messages (données, instructions, synchronisation, signaux d'interruptions, etc.). On peut distinguer deux modèles de programmation [4] :

Synchrone : Un passage de message synchronise les processus émetteur et récepteur dans le temps et l'espace. Ceci se fait en implémentant le concept de "rendez-vous".

Asynchrone : Une communication asynchrone n'exige pas que les processus émetteur ou récepteur soient synchrones. Ce paradigme requiert l'utilisation de buffers de canaux de communication ou d'une boite à lettres globale.

Les outils les plus célèbres pour implémenter le message passing sont PVM (Parallele Virtual Machine) et MPI (Message Passing Interface).

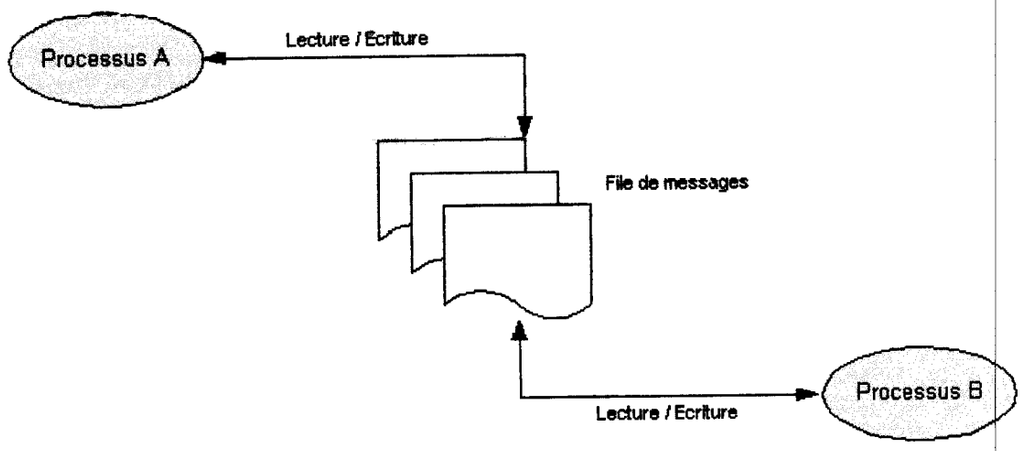


Figure 1.11 : Message Passing asynchrone

VII.4 La technique Remote Method Invocation (RMI) de Java

VII.4.1 Présentation

Le Remote Method Invocation RMI est la technique des objets distribués en java. Le RMI est une API (Application Programming Interface) java intégrée au JDK (Java Developpement Tool Kit) depuis le JDK1.1 et qui permet de distribuer des objets entre applications développées avec du code java [7].

La RPC a été conçu pour la programmation procédurale et ne prévoit rien en ce qui concerne la programmation objet. C'est l'adaptation de la RPC à la programmation objet qui a donné naissance au RMI (Remote Method Invocation) en Java.

A ses débuts, il était uniquement réservé aux objets java, mais avec le JNI (Java's Native Interface) on peut l'utiliser avec d'autres langages.

Le RMI est un mécanisme qui permet l'appel de méthodes entre objets java s'exécutant sur des machines virtuelles différentes (espaces d'adressage distincts), sur la même machine ou sur des machines distantes reliées par un réseau.

Le RMI utilise les sockets pour la communication et code ses échanges avec un protocole propriétaire : le RMP (Remote Method Protocol).

Les objectifs de RMI sont :

- Permettre d'invoquer plus facilement des méthodes Java entre applets ou applications sur des machines virtuelles différentes
- S'intégrer le plus facilement et le plus naturellement dans Java de manière à promouvoir l'écriture d'applications réparties dans ce langage
- rendre transparent l'accès aux objets distribués sur un réseau.
- faciliter la mise en œuvre et l'utilisation d'objets distants java.
- Permettre à une applet d'appeler directement une application Java sur son serveur.
- préserver la sécurité inhérente à l'environnement java :
  - RMISecurityManager et le Distributed Garbage Collector (DGC)

VII.4.2 L'architecture du protocole RMI

Le système RMI contient 3 couches qui sont : la couche des amorces (stub/skeleton), la couche des références et la couche de transport. Chacune est indépendante de l'autre et utilise un protocole spécifique ; par exemple, le transport utilise actuellement TCP .

La transmission des objets utilise deux techniques :

- la sérialisation ;
- le chargement dynamique du stub permet au client de charger dynamiquement le stub quand il a seulement l'interface.

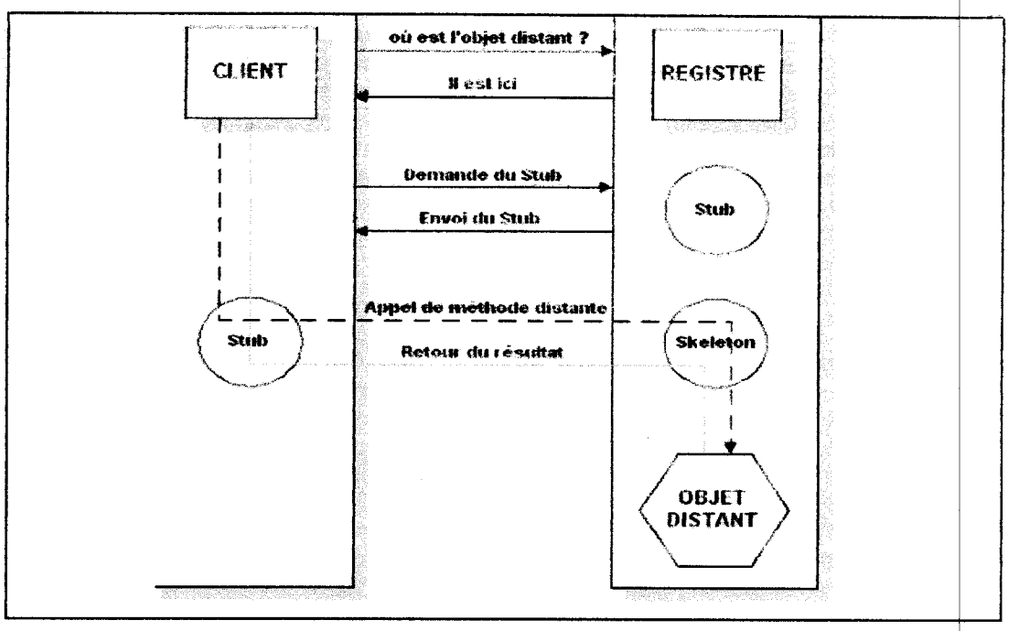


Figure 1.12 : Exemple d'appel distant par un client RMI [4].

La liaison entre les trois couches est représentée dans la Figure 1.13.

Le **stub** (utilisé par le client) est une implémentation des interfaces distantes et a pour but de :

- appeler l'objet distant (par un appel à la couche référence distante) ;
- empaqueter les arguments de la méthode sur un flux de données (stream) ;
- informer la couche référence distante que le RMI doit être fait ; - (attendre la valeur de retour) ;
- dépaqueter la valeur de retour ou l'exception s'il y en a ;
- informer la couche référence distante que l'appel est fini.

Le **skeleton** (côté serveur) contient une méthode qui appelle les méthodes de l'objet distant. Il a pour but de :

- dépaqueter les arguments de l'appel ;
- appeler la méthode concernée ;
- empaqueter la valeur de retour ou l'exception, s'il y en a.

La couche référence distante gère le type de RMI, qui peut être par exemple :

- appel point à point ;
- appel multi-cast (par diffusion) ;

- **support pour objet persistant ;**
- **stratégies de reconnexion (si l'objet devient inaccessible).**

La couche transport s'occupe de la communication proprement dite. Elle prend en charge :

- **les connexions réseau ;**
- **l'écoute des appels ;**
- **la gestion de la table des objets enregistrés ;**
- **la localisation du serveur lors d'un appel.**

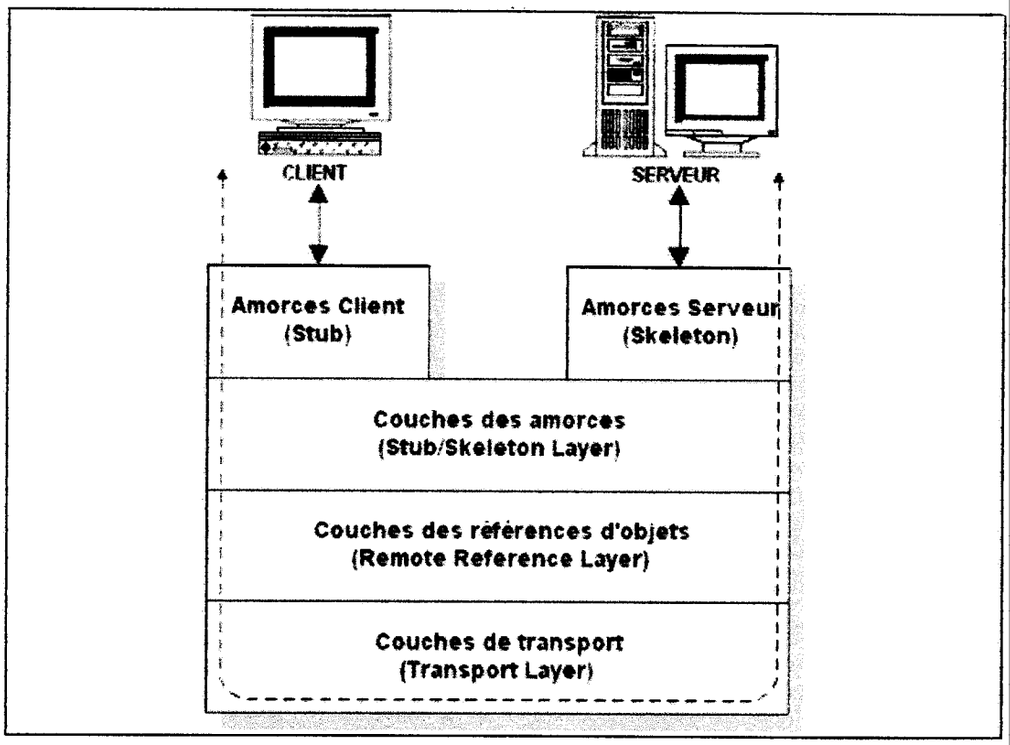


Figure 1.13 Les trois couches RMI [4]

VII.5 La technique CORBA de l'OMG

VII.5 .1 Présentation

CORBA (Common Object Request Broker Architecture) est une norme de spécification, de recommandation et de définition de l'architecture distribuée "idéale"[8], proposée par l'OMG (Object Management Group), qui offre une solution au problème d'interopérabilité des applications hétérogènes. Comme toutes les autres solutions distribuées, elle exploite les objets distribués et le principe d'appel à des méthodes distantes. La particularité de CORBA est l'intégration de services implantés par des fournisseurs de logiciels en exploitant le langage IDL (Interface Definition Language) et le protocole IIOP pour la communication sur le réseau.

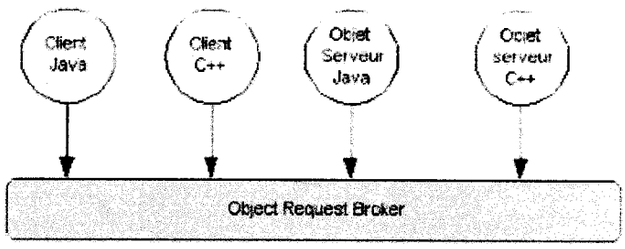


Figure 1.14 Le bus CORBA.

VII .5 . 2 L'architecture OMA

L'OMA (Object Management Architecture) est une architecture globale prenant en compte toutes les technologies nécessaires à la construction d'applications dans un environnement distribué.

En effet celle-ci vise à classifier les différents objets qui interviennent dans une application en fonction de leurs rôles.

VII.6 La technique DCOM de Microsoft

DCOM (appelé initialement Network OLE) est une extension du modèle de composants objets COM (Component Object Model) initié par Microsoft pour permettre à plusieurs composants situés sur une même machine (tournant sous l'environnement Windows uniquement) de communiquer entre eux.

Selon la formule de Microsoft, DCOM vient combler les lacunes de COM pour la distribution et la communication d'objets issus des langages Microsoft (Visual C, C++) sur des plates-formes clientes hétérogènes. Mais le serveur doit être de type Windows 9x ou NT. En d'autres termes DCOM est simplement « COM avec un plus long câble ».

Un objet défini est accessible seulement par ses interfaces.

Un GUID (Globally Unique IDentifier) permet de designer chaque interface.

DCOM est basé sur l'environnement DCE (Distributed Computing Environment), dont il exploite en particulier les appels de procédures distantes RPC (Remote Procedure Call), comparables dans leur principe au système RMI. Par conséquent, à chaque fois qu'on envoie un message DCOM à un objet distant, ce message est en réalité transporté via RPC.

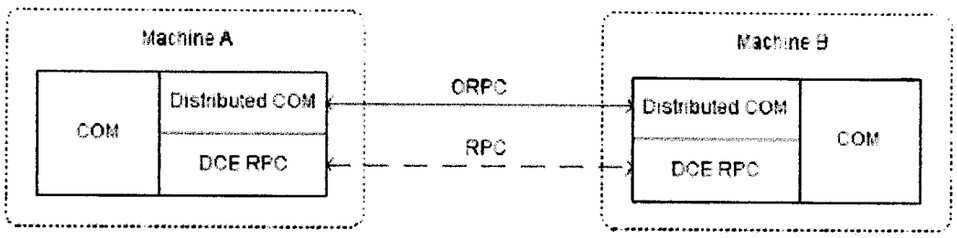
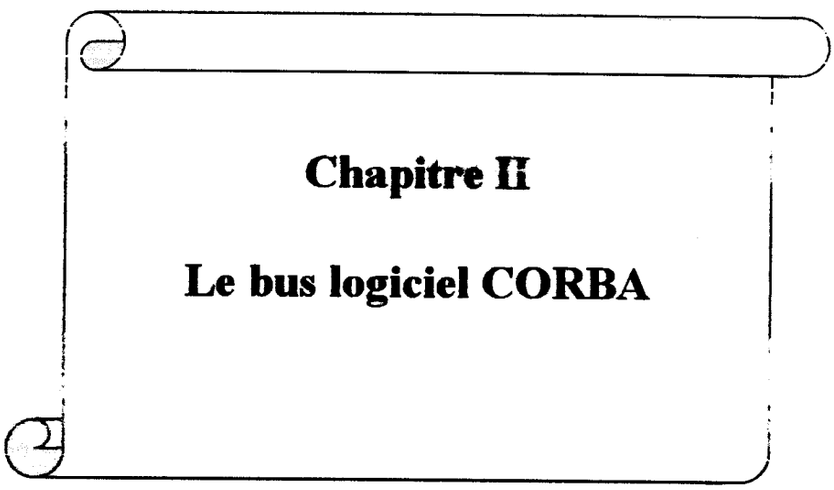


Figure 1.15 Architecture DCOM

DCOM supporte les objets distants à travers le protocole Object Remote Procedure Call (ORPC, qui est au-dessus de RPC et qui interagit avec les services d'exécutions de COM). Les objets serveurs ont des Interfaces décrivant chacune un des comportements de l'objet.

VIII CONCLUSION

Au terme de ce chapitre, force est de constater avec évidence la nécessité combien grandissante de vouloir faire coopérer des plates formes logicielles hétérogènes malgré les charges inhérentes de développements. Cependant, un certain nombre de standards a émergé ; le choix des technologies sous jacentes ne dépend que des contextes d'utilisation. Si RMI convient bien aux Intranets de moyenne envergure, DCOM constitue une alternative dans le cas de systèmes "tout Windows". En revanche, CORBA est très indiqué pour les environnements hétérogènes et, c'est ce dernier standard qui sera l'objet du chapitre suivant.



Chapitre II

Le bus logiciel CORBA

I. INTRODUCTION

Dans le chapitre précèdent, nous avons passé en revue les différents problèmes et limites des architectures centralisées faisant ainsi émergé des motivations ayant fait naître les architectures distribuées. Ces dernières ont connu plusieurs évolutions à savoir du client serveur simple en passant par les RPC pour aboutir aux technologies de la programmation par objets distribuées. Parmi celles-ci, le standard de l'OMG CORBA a particulièrement suscité notre attention du fait de la simple raison qu'il offre une plate forme d'exécution interopérable, et ce indépendamment de l'hétérogénéité matérielle que logicielle.
Ainsi donc, le présent chapitre met en exergue le bus logiciel CORBA. l'architecture autour de laquelle dont il s'articule ainsi que son fonctionnement et son anatomie.

II.1 L'OMG, son Histoire et ses Objectifs

L'OMG (Object Management Architecture) a été fondé en 1989. Il s'agit d'un consortium international regroupant actuellement près de 1000 acteurs du monde informatique dans lequel on trouve des : constructeurs (ex : IBM, Sun, HP, INTEL), éditeurs de logiciels (ex : Netscape, Inprise, Microsoft, Iona Tech.), utilisateurs (ex : Boeing, Alcatel, NASA). universités (ex : INRIA, CERN, LIFL).

L'OMG poursuit les objectifs suivants :

- Promouvoir la conception d'applications informatiques distribuées interopérables et ouvertes.
- Développer des spécifications (spécifier les interfaces) mais pas de produits (implémentations ouvertes à la concurrence.
- Définir une architecture globale appelée OMA (Object Management Architecture), fondée sur les technologies orientées objet permettant la réutilisabilité et la portabilité des composants logiciels, l'hétérogénéité et l'interopérabilité des environnements informatiques.

II.2 Présentation, motivations et objectifs de CORBA

CORBA (Common Object Request Broker Architecture) est une architecture logicielle conçue pour permettre à des applications distribuées sur un environnement hétérogène de communiquer, quel que soit leur langage de programmation et leur localisation. Cette

technologie utilise une approche orientée objet pour créer des composants logiciels réutilisables et partageables [10]. Chacun des objets exposés encapsule les détails des opérations internes qu'il implémente, et présente une interface bien définie lui servant à cet effet de point d'entrée. Son intérêt réside dans la définition et la promotion d'une architecture et d'un ensemble de spécifications, basés sur la technologie objet, réalisant l'interopérabilité entre applications distribuées sur des réseaux de systèmes hétérogènes. Les motivations ayant poussées l'avènement de CORBA sont [11] :

- construire un environnement dont les spécifications sont standardisées.
- s'affranchir des solutions purement propriétaires.
- offrir une plate-forme multi-systèmes et multi-langages

Depuis sa mise en œuvre, CORBA n'a cessé d'évoluer et de s'améliorer pour parvenir à s'imposer à l'heure actuelle dans de très nombreux secteurs tels que la finance, les télécommunications ou encore le milieu médical pour lesquels il a d'ailleurs développé des services sur-mesure. Par ailleurs, et vu que les objectifs de CORBA sont :

- Fournir un environnement ouvert
  - les membres participent aux spécifications
- Fournir un environnement portable
  - les API sont définis pour rendre les applications portables
    (quelque soit le produit CORBA utilisé)
- Fournir un environnement interopérable
  - Permettre aux applications CORBA de collaborer entre elles.

Il est indéniable que CORBA fasse partie aujourd'hui des nouvelles technologies les plus en vogue. De nombreux centres de recherche et développement l'étudient et le placent au cœur de leurs nouveaux projets.

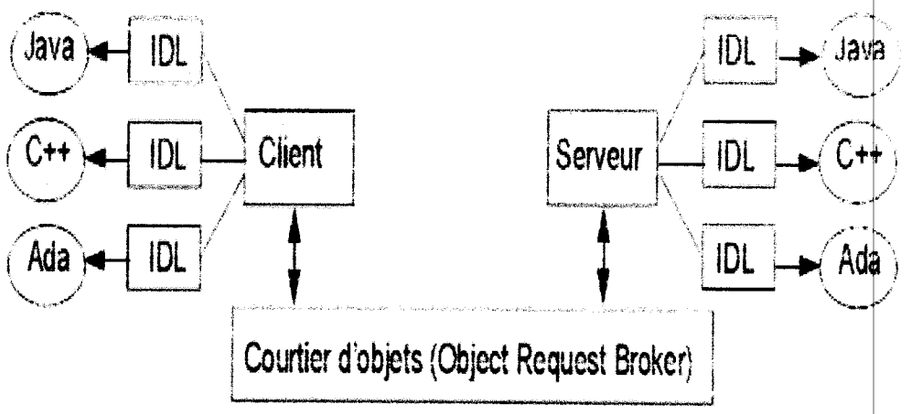


Figure II.1 Vue générale de CORBA [12]

Dans ce schéma, l'ORB (Object Request Broker) apparaît comme l'organe central réalisant l'interaction entre une application cliente et un objet CORBA. Le client n'a aucun besoin de savoir où l'objet en question se trouve exactement. Tout ce que le client doit savoir c'est le nom de l'objet et comment utiliser son interface. L'ORB se charge de trouver l'objet CORBA où qu'il soit, et de router, de manière appropriée, les requêtes et les résultats.

II.3 L'objet CORBA

II.3 .1 Définition du concept d'objet dans CORBA

Les objets CORBA d'un point de vue concret sont des objets tout à fait traditionnels implémentés dans le langage choisi par le développeur (C++, Java ou encore Smalltalk). Leur seule particularité consiste à exporter leur interface sous une forme bien définie, ce qui permet à des applications extérieures d'accéder à leurs services.

La définition des interfaces des objets CORBA est faite en utilisant le langage IDL (Interface Definition Language). Ce langage normalisé par l'OMG permet de spécifier un certain nombre d'informations concernant les objets du système et notamment les méthodes que les clients pourront invoquer sur ces objets.

Ceci permet de respecter un des principes majeurs de CORBA, à savoir, séparer de façon stricte les interfaces des objets de leur implémentation. De cette façon, le bus CORBA parvient à faire communiquer des clients Java avec des serveurs C++ par exemple, sans que ni l'un ni l'autre n'ait besoin de savoir dans quel langage les objets sont véritablement implémentés. En effet, seules les interfaces IDL sont nécessaires aux clients pour savoir quelles méthodes ils peuvent invoquer sur les serveurs.

II.3.2 Le modèle objet client/serveur dans CORBA

Le bus CORBA propose un modèle orienté objet client/serveur abstrait et de coopération entre les applications réparties. Chaque application peut exporter certains de ses services sous forme d'objets CORBA : c'est l'abstraction de ce modèle. Les interactions entre les applications sont alors matérialisées par des invocations à distance des méthodes des objets : c'est la coopération. La notion client/serveur intervient au moment de l'utilisation de l'objet : l'application implémentant l'objet fait office de serveur, l'application utilisant l'objet représente le client. Concrètement, nous pouvons décrire le fonctionnement de ce modèle par :

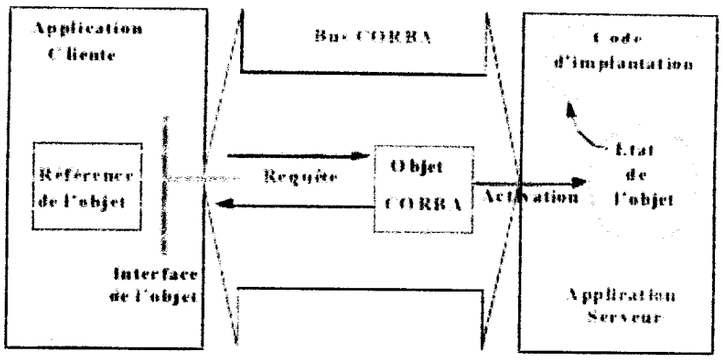


Figure II.2 Client/Serveur objet [8]

II.3.3 Vocabulaire CORBA

En effet, de maniere intuitive, nous utilisons les termes comme client, serveur, requêtes, objet CORBA et ce, en faisant appel à la signification qu'on leur accorde généralement. Cependant, comme toute technologie qui se respecte, CORBA possède sa propre terminologie [10] :

- Un **objet CORBA** est une entité virtuelle identifiable et encapsulée qui peut être localisée par l'ORB et qui peut être la cible des requêtes client. Cette entité est virtuelle dans le sens qu'elle n'a pas d'existence concrète avant qu'elle soit implémentée par un servant. Chaque objet CORBA est associé à une référence d'objet qui forme son identité.
- Un **client** est une entité qui formule des requêtes à un objet CORBA. Il peut exister ans le même espace d'adresses que l'objet CORBA invoqué ou dans un espace d'adresses complètement différent. Le terme client a du sens seulement dans le contexte d'une requête particulière. Le client accède à l'objet CORBA par l'intermédiaire d'une référence d'objet.
- Une **requête** est l'invocation d'une méthode ou d'une opération d'un objet CORBA par un client. La requête part du client vers un objet CORBA exécuté par un serveur et le résultat est renvoyé au client.
- Un **serveur** est une application contenant, dans son espace d'adresses (peut héberger), un ou plusieurs objets CORBA dont chacun est accessible indépendamment des autres objets du serveur. Comme le client, ce terme n'a de sens seulement dans le contexte d'une requête particulière, car une application serveur d'une requête peut être client pour une autre.
- Une **référence d'objet** est une structure de données (assimilable à un pointeur) contenue dans l'espace mémoire de l'application cliente capable d'identifier, de localiser et d'accéder à un objet CORBA sur un serveur. Chaque objet CORBA est associé à une référence d'objet qui forme son identité et que deux objets CORBA du même type (appartenant à un même serveur) ont deux identités différentes.

Les clients utilisent des références d'objet mais ne peuvent pas créer ou en modifier.

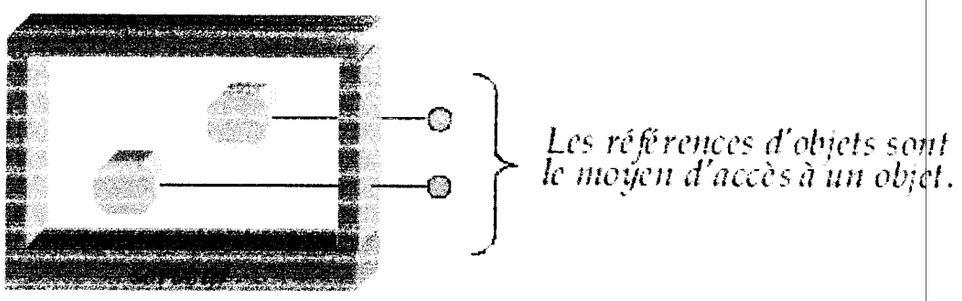


Figure II.3 Illustration des références d'objets

- **L'activation** est le processus d'association d'un objet d'implantation à un objet CORBA.
- **L'implantation de l'objet** est l'entité codant l'objet CORBA à un instant donné et gérant un **état de l'objet** temporaire. Au cours du temps, un même objet CORBA peut se voir associer des implantations différentes.
- **Le code d'implantation de l'objet** regroupe les traitements associés à chacune des opérations de l'interface de l'objet CORBA à l'implantation des opérations de l'objet CORBA. Cela peut être par exemple une classe Java aussi bien qu'un ensemble de fonctions C.

III. L'architecture OMA

L'OMA (Object Management Architecture) est une architecture globale prenant en compte toutes les technologies nécessaires à la construction d'applications dans un environnement distribué.

En effet celle-ci vise à classifier les différents objets qui interviennent dans une application en fonction de leurs rôles :

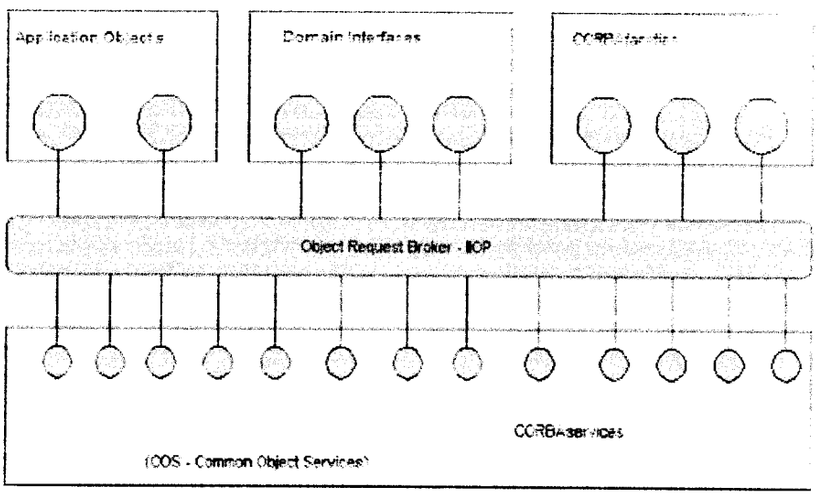


Figure II.4 Architecture de l'OMA

L'architecture générique définit une gamme d'outils complète pour permettre d'écrire des applications réparties. Il s'agit notamment de :

•**L'ORB** (Object Request Broker) : bus logiciel assurant le transport des requêtes et jouant le rôle de courtier de requêtes entre les objets clients et serveurs. Quand un client sollicite un ORB par l'intermédiaire d'une invocation de service, l'ORB localise le serveur d'objets et prépare la requête. Ensuite, il communique les données et transfère le contrôle au site serveur [9].

Ainsi l'ORB permet aux objets d'émettre des requêtes destinées à d'autres objets, locaux ou à distance, et d'en recevoir les réponses. En utilisant un ORB, un objet client peut, de manière transparente, appeler une méthode sur un objet serveur, lequel peut se trouver sur la même machine ou sur une autre machine du réseau. Le client n'a pas à connaître l'endroit où se trouve l'objet, ni son langage de programmation, ni son système d'exploitation, ni aucun autre aspect du système ne faisant pas partie de l'interface de l'objet.

•Les **CORBAservices** fournissent les fonctions de base pour l'implantation et l'utilisation des objets permettant de ce fait de simplifier le développement et d'en diminuer le temps. Il s'agit par exemple des mécanismes de nommage des objets ou encore de sécurité des communications,

•Les **CORBAfacilities** (utilitaires communs) fournissent un ensemble de fonctions pour les applications distribuées. Il s'agit par exemple de la gestion de l'interface utilisateur ou de la gestion des informations,

•Les **Domain Interfaces** fournissent des objets utilitaires adaptés à un secteur d'activité spécifique.

•Les **Application Objects** (objets applicatifs) sont les objets qui répondent spécifiquement aux besoins d'une application utilisateur donnée. Ils ne peuvent donc pas être standardisés. Toutefois, dès qu'ils commencent à devenir génériques pour un domaine, il est recommandé de demander leur inclusion dans les interfaces standard des domaines concernés.

IV. L'ORB ou bus logiciel CORBA

Dans une application conforme au modèle CORBA, chaque objet interagit avec son environnement en invoquant et en recevant des requêtes par l'intermédiaire de l'ORB. En effet, celui-ci est un courtier d'objet prenant en charge le dialogue entre les objets serveurs et les différents clients qui s'y connectent. Son rôle est de mettre une infrastructure de communication (middleware) à travers laquelle les objets distants du système vont dialoguer.

On peut résumer le rôle de l'ORB par :

- Identification des objets par des références d'objet
- Localisation d'objets
- Activation du serveur h l'objet
- Activation de l'objet
- Acheminement des requêtes

Pour faire partie du système, chaque objet doit exporter son interface à tous les autres sites et possède une référence globale, afin d'étendre son accessibilité, c'est-à-dire afin de devenir visible de l'extérieur.

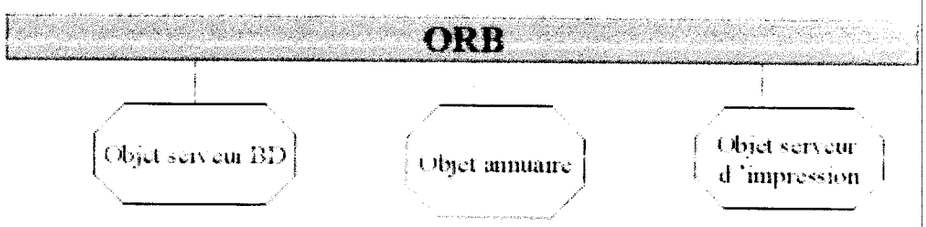


Figure II.5 l'ORB

Ainsi donc, quand un client invoque le serveur via la requête, l'ORB :

- localise l'implantation objet du serveur (service),
- achemine la requête vers l'objet "serveur" (après avoir activé ce dernier si nécessaire), et
- renvoie la réponse au client.

Le client n'a aucune connaissance sur la localisation du serveur dans le réseau.

IV.I Mécanisme de localisation d'un objet serveur

- Le client et serveur ne partagent pas le même espace mémoire,
- Lorsque le serveur construit l'objet d'implémentation, une **référence locale** à l'espace mémoire du serveur lui est attribué,
- Pour que l'objet d'implémentation soit invocable à distance, une **référence globale** lui est attribuée par le courtier,
- Cette référence globale doit ensuite être transmise au client,
- lors d'une invocation, le courtier utilise la référence globale pour localiser le serveur.

IV.2 Caractéristiques de l'ORB

Il fournit les caractéristiques suivantes [8] :

• **La liaison avec de nombreux langages de programmation** : En effet l'un des atouts majeurs du bus CORBA est la liberté laissée au programmeur de choisir le langage de programmation le mieux adapté pour implanter chacun des objets serveurs et chacune des applications. Cette liberté est due au fait de la séparation entre l'interface et l'implantation des objets offertes par le langage de description d'interfaces appelé IDL (pour Interface Description Language).

• **La transparence des invocations** : les requêtes aux objets semblent toujours locales, le bus CORBA se chargeant de les acheminer en utilisant le canal de communication le plus approprié. Ce fait donne concrètement aux clients l'impression qu'il invoque une méthode locale. Mais qu'en réalité le bus CORBA cache tous les détails et les technicités inhérentes à la répartition. De ce fait, le serveur affirme mettre la disponibilité de ses objets afin que les clients s'y connectent.

• **L'invocation statique et dynamique** : ces deux mécanismes complémentaires permettent de soumettre les requêtes aux objets. En statique, les invocations sont contrôlées à la compilation. En dynamique, les invocations doivent être contrôlées à l'exécution.

• **L'activation automatique d'objets** : c'est un mécanisme permettant au bus CORBA de ne chargé en mémoire que les objets utilisés par les clients, et ceci dans le soucis d'éviter d'encombrer la mémoire. Au fait, lorsqu'un client invoque un objet pour lequel aucune implantation n'est active, alors le bus CORBA se charge d'activer automatiquement l'implantation adaptée à cet objet c'est-à-dire associer un espace mémoire pour stocker l'état de l'objet et un contexte d'exécution pour les opérations.

• **L'interopérabilité entre différents bus** : Un ORB peut fonctionner en mode « standalone » ou être interconnecté à un autre ORB (provenant d'un autre fournisseur) grâce aux services des protocoles :

– GIOP (General Inter-Orb Protocol) : ensemble de messages et représentations communes des données (CDR : Common Data Representation) pour assurer la communication entre ORB.

– IIOP (Internet Inter-Orb Protocol) : définit la façon dont les messages GIOP s'échangent sur un réseau TCP/IP. Donc, il permet le dialogue entre les objets distribués à travers le monde entier.

– ESIOP (Environment Specific Inter-ORB Protocol) : pour les interactions particulières entre réseaux spécifiques.

Brièvement, il s'agit là d'une définition d'un ensemble des règles et des protocoles permettant la communication entre les différentes implantations du bus et ce, grâce à des passerelles convertissant les messages entre les différents bus.

IV.3 Anatomie et fonctionnement de l'ORB

L'Objeot Request Broker (ORB) constitue la colonne vertébrale de toute application bâtie sur l'architecture CORBA et est responsable de toutes les interactions entre Clients et Objets serveurs. Il doit être vu comme un ensemble logique constitué de plusieurs composants dont chacun est doté d'une tâche (service) spécifique plutôt qu'une bibliothèque ou un processus particulier.

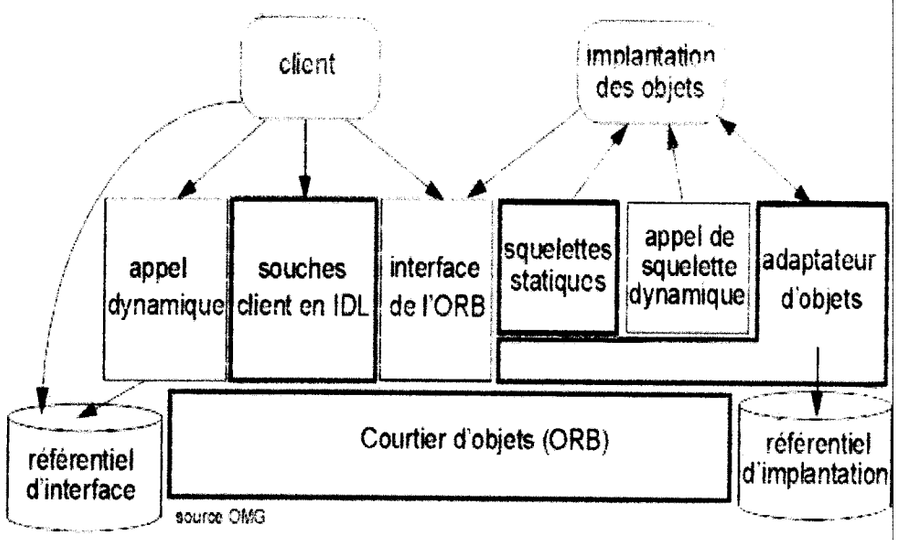


Figure II.6 Anatomie de l'ORB

IV.4 Les composantes du bus CORBA

- **Le noyau de communication** de l'ORB (ORB core) assure le transport des requêtes entre les objets selon diverses techniques dépendantes de la localisation. En effet il s'occupe de :

  — gère la localisation des objets dans l'environnement
  — implante les protocoles de communication entre objets
  — accessible au travers d'un ensemble de primitives

• **L'interface d'invocation statique (ou Static Invocation Interface : SII)** est une **souche** ou **stub** résultant de la projection des descriptions IDL dans un langage de programmation donné et permettant au client de soumettre des requêtes contrôlées à la compilation des programmes. A chaque interface IDL correspond une SII. Le stub joue le rôle de proxy (mandataire) local pour un objet serveur à distance, c'est qui permet d'ailleurs au client de voir à travers le stub, l'interface de l'objet et d'y invoquer les opérations de manière transparente en ignorant la localisation de l'objet et les détails techniques de son implantation. Lors d'une invocation, le stub permet de :

— préparer les paramètres d'entrée, ce qu'on appelle généralement l'assemblage ou le « marshalling » de l'opération.

— décoder les paramètres de sortie et les résultats, ce qu'on appelle généralement le désassemblage ou le « unmarshalling » de l'opération.

— déléguer le transport de l'invocation au noyau.

• **L'interface d'invocation dynamique (ou Dynamic Invocation Interface : DII)** est comme les proxys, utilisée pour invoquer des requêtes sur l'objet à la seule différence que ces requêtes sont créées dynamiquement (contrôlées à l'exécution i.e. dont on ne connaît pas l'interface à la compilation). En recourant à la DII, un client devra spécifier les éléments de la requête (préciser l'objet cible, l'opération à exécuter et un ensemble de paramètres pour cette opération) qui ne sont accessibles qu'à travers le référentiel des interfaces [13].

• **Le référentiel des interfaces (ou interface repository : IR)** C'est une base de données contenant toutes les définitions IDL d'objets, d'interfaces, de modules accessible par les applications clientes durant l'exécution. Les définitions d'objets sont fournies par les compilateurs d'IDL ou par des fonctions d'écriture du référentiel. Les référentiels d'interfaces peuvent se fédérer et coopérer à travers les ORBs.

• **L'interface du bus** fournit les primitives de base (notamment quelques API) pour l'initialisation, le paramétrage et l'instanciation des références d'objets. En outre elle constitue un point de départ aussi bien pour les applications clientes que serveurs pour accéder à toutes les composantes techniques du bus [8].

• **L'interface de squelettes statiques (ou Static Skeleton Interface : SSI)** est un **squelette** ou **skeleton** généré à la compilation à partir de l'interface IDL. En effet, il y'a un

squelette par type d'objet invocable c'est-à-dire qu'à chaque service exporté par le serveur lui correspond un skeleton. En d'autres termes, on peut dire tout simplement qu'il contient la signature de l'implantation d'objet générée à partir des interfaces définies en IDL. Le skeloton est le symétrique du stub côté serveur et permet entre autres de :

– déballer les paramètres des requêtes pour les transmettre aux objets d'implantation d'entrée, ce qu'on appelle généralement le désassemblage ou le « unmarshalling » de l'opération.

– Emballer les résultats à destination des clients, ce qu'on appelle généralement l'assemblage ou le « marshalling » de l'opération.

– déléguer le transport de l'invocation au noyau.

• **L'interface de squelettes dynamiques (ou Dynamic Skeleton Interface : DSI)** est l'équivalent pour les serveurs d'objets de la DII pour les clients. Elle permet de recevoir des requêtes sans disposer à priori des squelettes (SSI) de déballage c'est à dire sans que le squelette de l'objet soit statiquement compilé dans le programme. L'implémentation de l'objet est atteinte à partir d'une interface qui offre l'accès au nom de l'opération et à ses paramètres. L'implémentation de l'objet donne à l'ORB la description des paramètres de l'opération et l'ORB lui transmet les valeurs de chacun des paramètres d'entrée. Ensuite l'implémentation de l'objet donne à l'ORB la valeur de chacun des paramètres de sortie ou une exception que l'ORB transmet au client [13].

• **Le référentiel des implantations (ou Implantations Repository : IR)** appelé encore Dictionnaire d'implémentation est une sorte de base de données des objets serveurs et qui contient les informations nécessaires qui permettent à l'ORB de localiser et d'activer les implantations des objets à la demande. Ce référentiel n'est utilisé qu'à l'exécution.

• **L'adaptateur d'objets (ou Object Adaptator : OA)** est un mécanisme permettant à l'implantation d'un objet d'accéder aux services de l'ORB. En outre, il fournit un environnement d'exécution complet pour une application serveur. Parmi les fonctions offertes par l'adaptateur d'objets, on note entre autres :

– Interface entre les objets CORBA et l'ORB (moyen utilisé par les objets CORBA pour accéder aux services offerts par l'ORB).

– Enregistrement et recherche des implantations d'objets dans le référentiel des implantations.

– Génération de références pour les objets serveurs (ID ou références d'objets).

– Gestion de l'instanciation des objets serveurs (correspondance entre les références propres de l'ORB et celles de l'implantation)

– Activation des processus dans le serveur

– Aiguillage des invocations de méthodes vers les objets serveurs

– Traitement des requêtes client entrantes

Donc, nous pouvons dire qu'un adaptateur d'objet ressemble beaucoup plus à un composant logiciel dont le rôle principal est de faire en sorte que des objets CORBA soient accessibles à leurs clients potentiels. De ce point de vue, un adaptateur d'objet peut être vue comme une prise de courant dans laquelle on branche des objets CORBA. Il en va ainsi que la responsabilité primordiale de l'adaptateur d'objet est d'assurer que l'appel d'une méthode qu'elle soit locale ou distante, atteigne l'objet CORBA auquel elle est destinée [10].

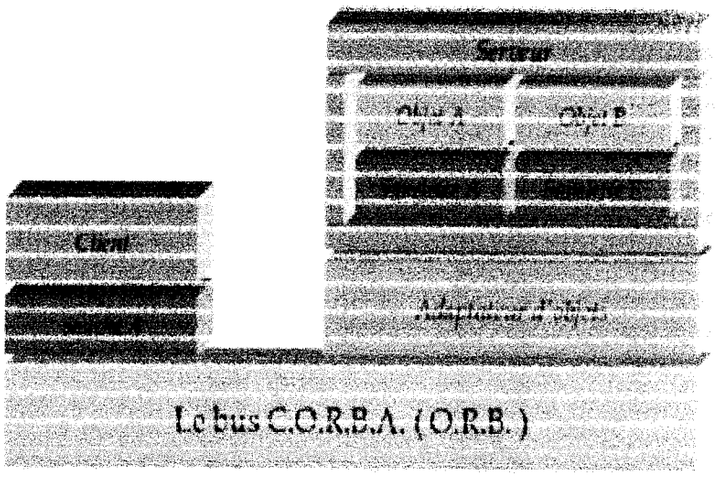


Figure II.7 L'adaptateur d'objets

V. Décomposition des tâches du courtier ou ORB

Les tâches du courtier ont été partagées entre :

- L'ORB présent sur le client et le serveur et qui permet :
  - D'assurer la localisation du serveur qui héberge l'objet,
  - D'assurer l'activation du serveur (si besoin),
  - D'attribuer la partie serveur de la référence d'objet,
  - D'assurer l'acheminement de la requête vers le serveur, puis vers l'adaptateur d'objets.

- L'adaptateur d'objet présent uniquement sur le serveur et permet :
  - D'assurer la localisation de l'objet serveur (ou serviteur),
  - D'assurer l'activation du serviteur (si besoin),
  - D'attribuer la partie locale de la référence d'objet,
  - De transmettre la requête au mandataire du serviteur.

VI. Déroulement d'une requête

Pour formuler une requête, le client peut utiliser l'interface d'Invocation Dynamique ou les Proxy IDL. Du côté de l'objet, il peut recevoir la requête à travers le Squelette Statique ou le Squelette Dynamique. Pendant le traitement de la requête l'implantation de l'objet fait appel à l'Adaptateur d'Objets de l'ORB [13] :

1. Le client accède à la référence de l'objet pour connaître le type d'objet et les opérations offertes. Cet accès fait appel à l'un des Interfaces d'Invocation (Statique ou Dynamique). Les requêtes statiques et dynamiques satisfont la même sémantique, donc le récepteur ne peut faire aucune différence entre ces deux types de requête.
2. L'ORB localise l'implémentation appropriée, transmet les paramètres et transfert le contrôle à l'implémentation d'objet par le squelette dynamique ou statique. Pour traiter la requête, l'implantation d'objet sollicite des services de l'ORB par l'Adaptateur d'objet. Les résultats sont transmis au client dès que la requête est terminée.

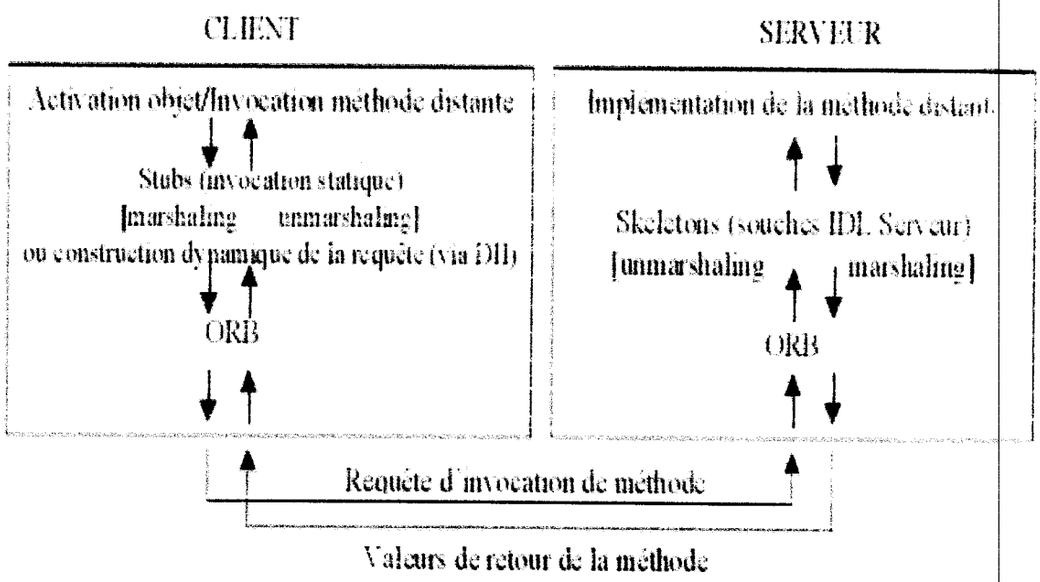


Figure II.6 Illustration d'une requête

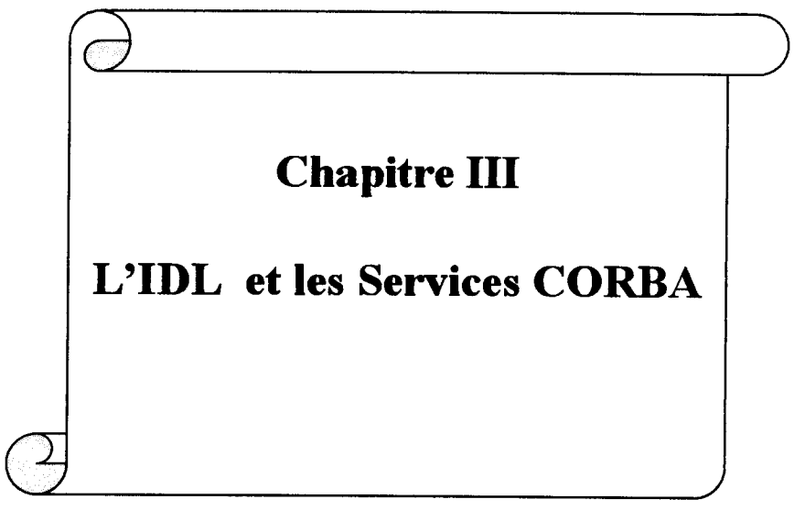
VII CONCLUSION

En conclusion, CORBA est la composante clé de l'OMA (object Management Architecture) de l'OMG spécifiée pour faire face aux défis du développement des systèmes distribués qui sont :

Rendre le développement des applications distribuées aussi facile que le développement des programmes centralisés.

Offrir une infrastructure pour intégrer les composantes d'une application dans un système distribué.

Cette section a décrit de façon sommaire les principales composantes de l'ORB qui constitue d'ailleurs la colonne vertébrale de cette architecture, cependant, il n'en demeure pas moins vrai que le langage OMG-IDL fournit les articulations c'est-à-dire tous les points de liaison entre le bus à objets et les éléments qui s'y attachent. C'est ce qui fera d'ailleurs le point central du chapitre prochain avec les différents services de l'architecture OMA.



Chapitre III

L'IDL et les Services CORBA

I. INTRODUCTION

Le précédent chapitre a fait un tour d'horizon sur l'anatomie du noyau de l'architecture CORBA, les blocs fonctionnels qui le composent et son fonctionnement. Cependant pour arriver à masquer les divers problèmes liés à l'interopérabilité, l'OMG a défini le langage IDL (Interface Definition Language). L'IDL permet d'exprimer sous la forme de contrats, la coopération entre les fournisseurs et les utilisateurs de services, en séparant l'interface de l'implantation des objets. L'IDL a une syntaxe proche du C++ et peut être projeté dans différents langages comme le Java, le C, le C++, le COBOL et d'autres encore.
Ainsi donc, le présent chapitre met d'abord l'accent sur l'importance, les différentes constructions syntaxiques et sémantiques, les principaux types de ce langage ainsi que les mécanismes de projection de ce langage vers les langages de programmation. Ensuite nous finirons par mettre en exergue les différents services offerts par l'architecture OMA.

II. Présentation de l'IDL

Le langage IDL pour Interface Description Language est un langage purement descriptif conçu pour la description des services orientés objet pour l'environnement CORBA. Il permet d'exprimer, sous la forme de contrats IDL, la coopération entre les fournisseurs et les utilisateurs de services, en séparant l'interface de l'implantation des objets et en masquant les divers problèmes liés à l'interopérabilité, l'hétérogénéité et la localisation de ceux-ci [8].
L'IDL est un langage utilisé pour décrire au sein d'une interface (vue cliente de l'objet) la liste des services offerts (ensemble de fonctions) auxquels ont accès les clients de l'objet.
C'est un langage de spécification qui offre très peu de détails sur l'implantation mais des correspondances vers différents langages de programmation peuvent être définies. Par conséquent, le client ne peut être écrit en IDL mais dans l'un des langages pour lequel la correspondance des concepts de l'IDL est définie. L'IDL permet l'interopérabilité entre clients et serveur d'objets écrits dans différents langages de programmation. L'IDL est constitué d'un ensemble de conventions lexicales comprenant des mots clés, des commentaires, des identificateurs et des littéraux. Sa grammaire est un sous- ensemble de celle de C++ augmenté des instructions nécessaires aux mécanismes d'invocation des

opérations. Elle supporte la syntaxe des constantes, des types et de la déclaration des opérations, mais elle n'inclut pas la définition des structures algorithmiques.

Toutefois, l'IDL ne permet qu'une interopérabilité uniquement syntaxique se limitant seulement au niveau des définitions d'interfaces, offrant donc un vocabulaire commun aux différents langages en laissant à chacun d'eux de définir une représentation (i.e.une projection) des concepts OMA en son sein. De ce fait, deux langages hétérogènes peuvent alors « interopérer » [14].

II.1 Le contrat IDL

Un contrat IDL spécifie les types manipulés par un ensemble d'applications réparties, c'est-à-dire les types d'objets (ou interfaces IDL) et les types de données échangées entre les objets. Ainsi, le contrat IDL isole donc les clients et fournisseurs de l'infrastructure logicielle et matérielle les mettant en relation à travers le bus CORBA.

En effet le contrat, défini par l'IDL lie les fournisseurs des services d'objets distribués à leurs clients. Les fournisseurs décrivent grâce au langage IDL l'ensemble des interfaces des objets qu'ils veulent fournir à leurs clients. Ensuite un client qui invoque une méthode distante va communiquer avec l'interface précisée par l'IDL. De son côté, l'objet serveur implémente l'interface indiquée par l'IDL.

Les contrats IDL sont projetés en souches IDL (stubs) dans l'environnement de programmation du client et en squelettes IDL (skeletons) dans l'environnement de programmation du serveur. Le client invoque localement les souches pour accéder aux objets. Les souches IDL construisent des requêtes, qui vont être transportées par le bus, puis délivrées par celui-ci aux squelettes IDL qui les délégueront aux objets. Ainsi le langage OMG-IDL est la clé de voûte du bus d'objets répartis CORBA.

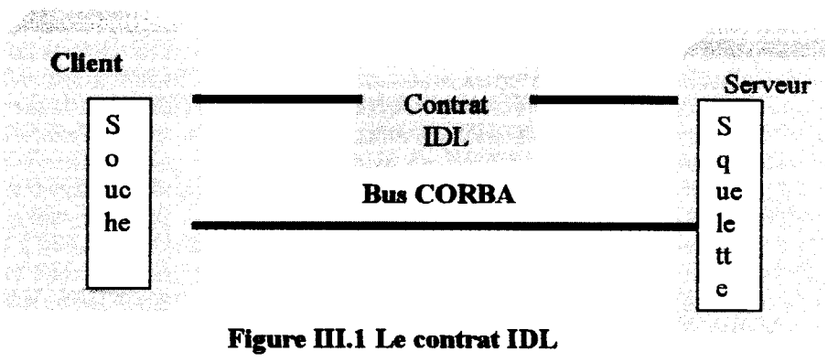


Figure III.1 Le contrat IDL

II.2 Notion d'interface IDL

Comme nous l'avons évoqué ci haut, l'IDL est un langage de description des interfaces contractuelles pour les services proposés par les objets CORBA.

Une interface IDL décrit donc, les opérations et les attributs accessibles à distance fournis par un type d'objets CORBA. Cette description ne présente que la partie publique des objets et ne précise aucun détail sur l'implantation de ceux-ci. Nous pouvons donc les comparer aux interfaces dans le langage java ou encore aux classes abstraites du langage c++.

II.3 Le modèle objet des interfaces

Apparaissant comme à la base de la modélisation d'une application reposant sur l'architecture CORBA, les interfaces fournissent donc un modèle objet primordial qu'il faut appréhender et qui repose bien en effet sur le principe de base du paradigme objet.

- **L'héritage :** il est possible qu'une interface hérite d'une autre interface, cependant celui-ci n'est qu'un héritage de déclarations d'attributs ou d'opérations et non d'implémentation de méthodes. En outre, le langage IDL autorise l'héritage multiple d'interfaces à condition qu'il n'y ait pas d'ambiguïté, c'est-à-dire qu'il est interdit d'hériter d'un même nom d'opération ou d'attribut au travers de plusieurs interfaces.

Ainsi, nous sommes obligés de conclure que l'héritage multiple est supporté en IDL de manière restreinte [14] (i.e. tant qu'il n'y a pas de conflit).

En ce qui concerne la surcharge et la redéfinition, voici les principales règles :

D'abord, nous mettons un petit détail en évidence, à savoir ne pas confondre la surcharge (overload) de la redéfinition (override). Dans le premier cas, le même nom de méthode est utilisé avec des arguments différents (par leur nombre ou leur type, i.e.signature différente) et une implémentation différente dans une même classe ou une sous-classe. Dans le second cas, la même méthode peut être redéfinie dans une sous-classe et sert à spécifier ou à modifier le comportement de base (i.e. même signature).

Dans tous les cas, ni l'une, ni l'autre n'est autorisée dans le modèle de l'OMG. Si l'absence de l'overriding peut se justifier par le fait que l'on peut référencer à partir d'une interface dérivée une opération ou un attribut de même nom situé dans une interface de base grâce à l'opérateur '::'. Pour l'OMG, le langage IDL traite uniquement de la description d'interfaces, il serait donc anormal de répéter la déclaration des opérations ou d'attributs dans l'arbre

d'héritage. Cependant, il n'existe pas de mécanisme de protections au sein même d'une hiérarchie comme en C++, aussi l'héritage peut nuire à l'encapsulation.

En revanche, l'absence de l'overloading est uniquement due à une concession faite au langage C qui ne supportait pas ce concept [14].

- **L'encapsulation :** L'atout majeur du modèle de l'OMG est sans contesté l'encapsulation. Avec le langage IDL, la séparation interface/implémentation est obligatoire. Cette séparation simple et naturelle présente l'interface comme une membrane protectrice qui isole l'objet et lui garantit le contrôle de ses données et de l'exécution des opérations qu'il définit.

En outre, la substitution logique d'une implémentation par une autre du côté serveur ne modifie pas l'interface et donc ne concerne pas le client, de même que la substitution physique (i.e. déplacement d'un objet dans un autre processus, vers une autre machine) aussi.

- **Le polymorphisme :** Dans le cas des interfaces ce concept est assuré par l'indépendance des interfaces vis-à-vis de leur implémentation. En effet, il est possible d'implémenter une interface de multiples façons dans de multiples objets serveurs différents, ce qui donne au fait à un client de se connecter à plusieurs instances d'objets serveurs différents via la même interface. Cela permet donc de mettre le stub une fois pour toutes au niveau des clients qui peuvent se connecter à de nombreux serveurs implémentant la même interface dans le cas d'un dialogue en mode client/serveur [3].

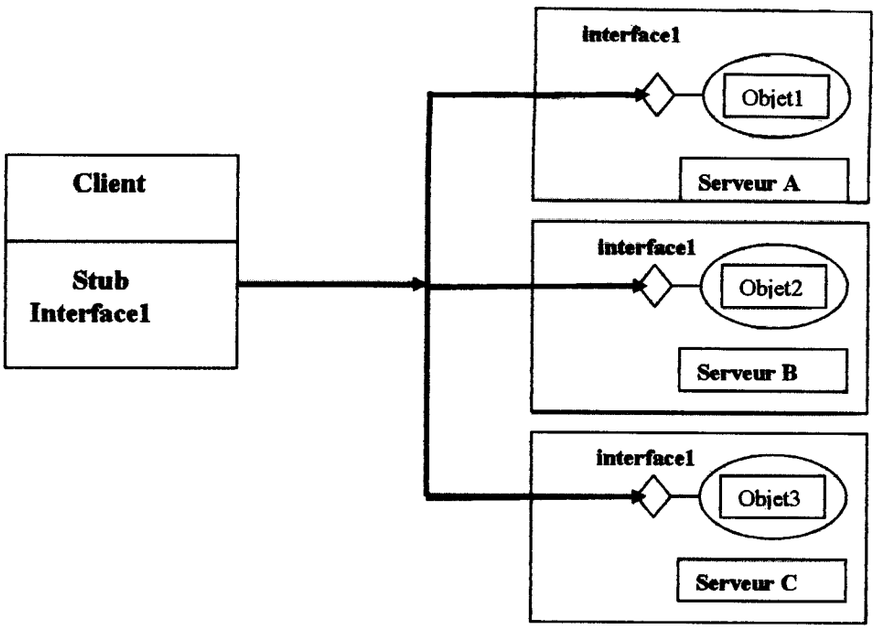


Figure III.2. Intérêt du polymorphisme

III. Les éléments du langage IDL

Dans cette partie nous nous attachons à présenter de façon sommaire les principaux éléments et les différentes constructions formant la syntaxe du langage IDL. Cependant cette description

III.1 Les types de données de base

Les types de données de base sont ceux couramment rencontrés en informatique [8][3] :

- **void** : En effet ce type ne représente pas réellement un type de donnée et sert principalement à définir des fonctions sans retour de valeur.
- **short, unsigned short:** entiers signés ou non sur 16 bits;
- **long, unsigned long**: entiers signés ou non sur 32 bits;
- **long long, unsigned long long** : entiers signés ou non sur 64 bits;
- **float** : nombres réels flottants 32 bits au format standard IEEE;
- **double** : nombres réels flottants 64 bits au format standard IEEE;
- **long double** : nombres réels flottants 128 bits;
- **boolean** : valeurs booléennes FALSE ou TRUE;
- **octet** : permet de stocker des valeurs sur 8 bits ne subissant aucune transformation lors du transport par l'ORB;
- **char** : caractères 8 bits ISO Latin-1;

- **wchar** : caractères au format international;
- **string** : chaînes de caractères;
- **wstring** : chaînes de caractères au format international;
- **fixed** : nombres réels à précision fixe.
- **any** :c'est le type « joker » qui peut contenir n'importe quel type de paramètre défini dans l'IDL. On l'emploi souvent lorsque l'on ignore à l'avance le type de paramètre à envoyer ou à recevoir.

III.2. Les constantes

Une constante (const) se définit par un type simple, un nom et une valeur évaluable à la compilation. A l'exception des types **any** et **octet**, tous les autres types élémentaires de l'IDL permettent la déclaration des constantes.

const double PI = 3.1415;

III.3. Les alias de type

Un alias de type (typedef) permet de créer de nouveaux types en renommant des types déjà définis. Par exemple, il est plus clair de spécifier qu'une opération retourne un Jour ou une Annee plutôt qu'un entier 16 bits signé.

typedef unsigned short Jour;

typedef unsigned short Annee;

III.4. Les énumérations

Une énumération (enum) définit un type discret via un ensemble d'identificateurs. Par exemple, il est plus parlant d'énumérer les mois dans l'année que d'utiliser un entier pour coder un Mois.

enum Mois {

Janvier, Fevrier, Mars, Avril, Mai, Juin, Juillet, Aout,

Septembre, Octobre, Novembre, Decembre

};

III.5. Les structures

Une structure (struct) définit un enregistrement correspondant à un agrégat des champs de types différents (e.g. Date). Cette construction est fortement employée car elle permet de

réduire le nombre de paramètres nécessaires aux opérations lors des transferts des structures de données composées entre objets CORBA.

```
struct Date {
        Jour j;
        Mois m;
        Annee a;
};
```

III.6. Les tableaux

Un tableau (array) sert à transmettre un ensemble de taille fixe de données homogènes. Mais cette construction est rarement utilisée dans des spécifications IDL du fait de la simple raison imposée par l'obligation de dimensionner les tableaux dans la phase de conception, et donc on fait recours aux tableaux dynamiques.

```
typedef float [10] Tableau ;
```

III.7. Les séquences

Une séquence (sequence) permet de transférer un ensemble de données homogènes dont la taille sera fixée à l'exécution et non à la définition comme pour un tableau (e.g. DesDates). Ce qui permet de transporter des ensembles non bornés des valeurs lors du dialogue entre clients et fournisseurs d'objets CORBA.

```
typedef sequence<Date> DesDates;
```

III.8. Les exceptions

Une exception spécifie une structure de données permettant à une opération de signaler les cas d'erreurs ou de problèmes exceptionnels pouvant survenir lors de son invocation. Une exception se compose de zéro, un ou plusieurs champs (e.g. MauvaiseDate).

```
exception MauvaiseDate {
        string raison;
};
```

III.9. Les interfaces

Une interface décrit les opérations fournies par un type d'objets CORBA (e.g. Calendrier). Il est important aussi de noter que tous les constituants d'une interface sont publics.

```
interface Calendrier {
// ...
};
```

III.10. Les attributs

Un attribut est une manière raccourcie d'exprimer une paire d'opérations pour consulter et modifier une propriété d'un objet CORBA. Il se caractérise par un type et un nom. De plus, on peut spécifier si l'attribut est en lecture seule (readonly) ou consultation/modification (mode par défaut).

```
interface CarteIdentité {
        attribute string nom;
        attribute string prénom;
        attribute short jour;
        attribute short mois;
        attribute long annee;
        readonly attribute short age;
    };
```

III.11. Les opérations

Les opérations représentent avec les attributs l'essence même des interfaces, puisque c'est à travers ceux deux types d'éléments que les clients pourront accéder à l'objet CORBA [3].
En effet, une opération se définit par une signature qui comprend le type du résultat, le nom de l'opération, la liste des paramètres et la liste des exceptions éventuellement déclenchées lors de l'invocation. Un paramètre se caractérise par un mode de passage, un type et un nom formel. Les modes de passages autorisés sont :

- **in** : mode de passage du client vers l'objet.
- **out** : mode de passage en retour de l'objet vers le client
- **inout** : mode de passage dans les deux sens.

Le résultat et les paramètres peuvent être de n'importe quel type exprimable en IDL.

```
Par exemple : boolean verifier_date (in Date d);
              void jour_suivant (inout Date d);
```

Par défaut, l'invocation d'une opération est synchrone, c'est-à-dire que le client reste bloqué tant que le serveur n'a pas traité la requête. Cependant, il est possible de spécifier qu'une opération est asynchrone (**oneway**), dans ce cas l'appelant n'attend pas la fin de l'exécution de l'opération et ignore donc si elle a été effectivement exécutée. Cela induit à ce que le résultat soit de type void, que tous les paramètres soient en mode **in** et qu'aucune exception ne puisse être déclenchée. Malheureusement, CORBA ne spécifie pas la sémantique d'exécution

d'une opération oneway : l'invocation peut échouer, être exécutée plusieurs fois sans que l'appelant ou l'appelée puissent en être avertis.

Toutefois, dans la majorité des implantations CORBA, l'invocation d'une telle opération équivaut à un envoi fiabilisé de messages [8].

IV. Les Mappings IDL vers langage de programmation

N'étant que purement descriptif, le langage OMG-IDL n'est qu'un point de départ. Il faut que les interfaces « prennent corps » [14] au sein de l'ORB.

Pour cela, un mécanisme permettant de faire correspondre les définitions IDL en des objets implémentés dans des langages de programmation particuliers. Ce mécanisme est tout simplement appelé mapping ou projection. Par ailleurs, chaque produit CORBA fournit donc un pré-compilateur IDL dédié à un langage d'implémentation supporté [8], permettant ainsi, à partir de l'interface IDL, de générer des fichiers (talons et squelettes IDL) qui serviront lors d'une invocation statique.

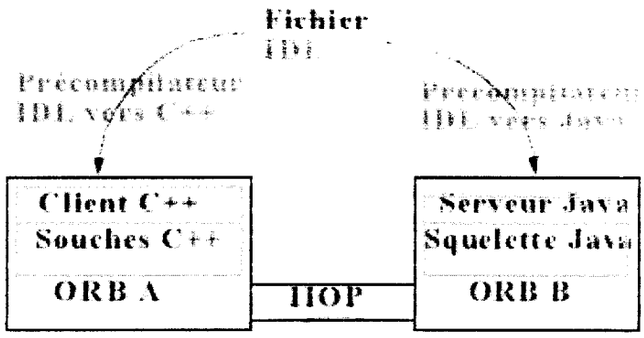


Figure III.3. Illustration d'une pré-compilation IDL [8].

Nous voyons bien sur la figure ci-dessus que chaque langage étant spécifique, la projection vers ce langage l'est aussi. Malheureusement, contraint par les nombreuses incompatibilités entre les langages jusqu'à l'heure actuelle, l'OMG n'a pu normaliser que les projections de quelques langages tels que le C, C++, Smalltalk, Ada95, Cobol, et Java pour ne citer que ceux là. Toutefois, des évolutions sont en amélioration constante au sein de l'OMG, et qui pourront toucher les champs d'autres langages pour renforcer « l'universalité » de CORBA dans le « monde » des langages.

Le moindre détail de ces projections ne peut incontestablement être négligé du fait de la simple raison qu'une spécification incomplète d'une projection pour un langage donné induira l'inutilisabilité de la technologie CORBA pour ce langage.

IV.1 Règles de projections

Dans cette partie, nous allons brièvement présenter quelques exemples des projections CORBA en les présentant sous forme de tableau. Il s'agit particulièrement du mécanisme de projection en Java. Cependant, il est à remarquer au niveau du tableau III.5, les constructions IDL qui n'ont pas d'équivalent en Java sont traduites par des classes.

IDL	Java
[unsigned] short	short
[unsigned] long	int
[unsigned] long long	long
float	float
double	double
char, wchar	char
string, wstring	Java.lang.String
boolean	boolean
octet	byte
any	org.omg.CORBA..Any
void	void
long double	non supporté

Figue III.4. Types IDL et leur projection en java

IDL	Java
module	package
interface	interface
attribute	Méthodes *setter/getter*
opération	Méthode Java
struct, enum, union	Classes Java
sequence	Tableau Java
const	static final
exception	Sous-classe de java.lang.Exception

Figure III.5. Projection des types construits

V. Les services CORBA

Comme nous l'avons évoqué sommaire au premier chapitre, l'OMG a développé sous forme d'objets CORBA un ensemble de tâches qualifiées sous le vocable de « services » souvent récurrentes lors de mise en place d'application CORBA, et ce, dans un souci de réduire au mieux les charges des développeurs. Ces services sont regroupés sous l'appellation générique de **COS** (Common Object Services) [3], et COSS (Common Object Services Specification) pour leur spécification.

Cela va sans dire que ces services fournissent sans doute des APIs d'objets CORBA dotés des interfaces qui rendent leur manipulation facile et leur intégration au sein d'une nouvelle application.

Dans les lignes qui suivent, nous présentons les différentes fonctionnalités des ces services et la façon de l'incorporer dans une application.

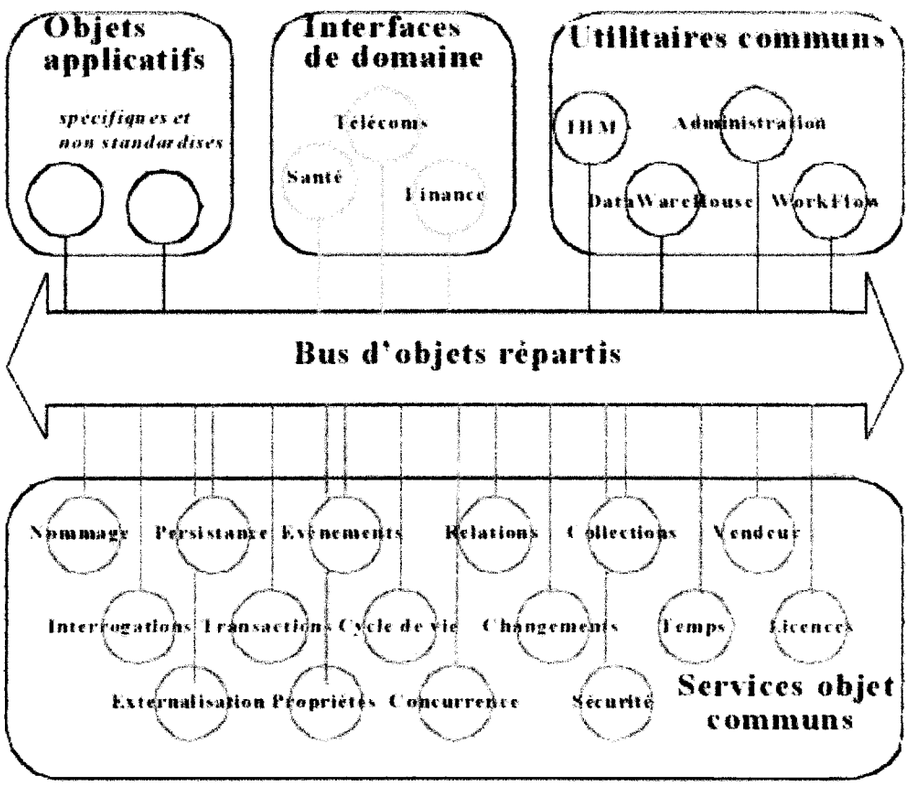


Figure III.6 Mise en évidence des services de l'OMA [3]

V.1. Description des services

Les services de recherches d'objets offrent les mécanismes pour rechercher/retrouver dynamiquement sur le bus les objets nécessaires aux applications : le service de nommage (Naming Service) et le service vendeur (Trader Service).

D'autres services prennent en charge les différentes étapes de la vie des objets CORBA : le service cycle de vie (Life Cycle Service), le service propriétés (Property Service), le service relations (Relationship Service), le service externalisation (Externalization Service), Le service persistance (Persistent Object Service), le service interrogations (Query Service), le service collections (Collection Service)...

D'autres services encore fournissent les fonctions système assurant la sûreté de fonctionnement nécessaire à des applications réparties : le service sécurité (Security Service), le service transactions (Object Transaction Service), le service concurrence (Concurrency Service).

Par défaut, la coopération des objets CORBA est réalisée selon un mode de communication client/-serveur synchrone. Toutefois, il existe un ensemble de services assurant des communications asynchrones : le service événements (Event Service), le service notification (Notification Service), le service messagerie (CORBA Messaging Service).

Il existe encore d'autres services divers et variés: le service temps (Time Service), le service licences (Licensing Service) [15]...

V.2. Le service de Nommage

Appelé encore Naming service ou CosNaming dans la terminologie CORBA [3], le service de Nommage est l'un des premier à avoir été implémenté et est disponible dans de nombreuses implémentations CORBA car il répond aux besoin les plus élémentaires d'applications distribuées.

Son objectif est donc de permettre aux utilisateurs et à leur programmes de trouver dynamiquement à l'exécution les objets qui leur sont nécessaires [8]. Il définit un espace de désignation symbolique des objets. Cet espace est structuré par un graphe de contextes de nommage (interface NamingContext) [15] faisant ainsi office d'espace de nommage réparti.

Chaque contexte maintient une liste d'associations entre des noms symboliques et des références d'objet. A l'intérieur d'un contexte, un nom (struct NameComponent) doit être

(bind), mettre à jour une telle association (rebind), le connecter ou reconnecter à un contexte (bind_context et rebind_context), rechercher la référence désignée par un chemin (resolve), détruire une association (unbind), créer un nouveau contexte indépendant (new context), créer un contexte et le connecter (bind new context), détruire définitivement le contexte (destroy), et finalement lister son contenu (list). Ces opérations déclenchent selon les différents cas de mauvaises utilisations les exceptions NotFound, CannotProceed, InvalidName, AlreadyBound et NotEmpty.

La figure III.7 met en évidence le processus de nommage qui se résume à :

- Du côté client

1. Initialise l'ORB
2. Obtient la référence du service de nommage
3. Obtient la référence d'un objet à partir de son nom
4. Conversion explicite vers le type d'objet
5. Invocation de la méthode

- Du côté serveur

1. Initialisation de l'ORB et du POA
2. Création de l'objet
3. Obtient la référence du service de nommage
4. Enregistre l'objet avec un nom
5. Active l'objet et attend les requêtes

V.3. Le service Vendeur

Le service Vendeur ou Trader service est un service qui permet aussi la mise en place de référencement d'objets et qui offre la possibilité de découvrir des références d'objets à l'exécution. Cependant dans ce cas de figure, les fournisseurs enregistrent leurs objets serveurs auprès de ce service en les caractérisant par un ensemble de propriétés [8]. Ainsi donc, pour retrouver un objet on fait la recherche par ses caractéristiques.

Dans la figure ci-dessous, l'application serveur va exporter vers le service vendeur une référence sur un de ses objets avec la description associée [8]. Les applications clientes qui désirent utiliser cet objet vont interroger le service vendeur en fournissant des critères de sélection, ce dernier retourne une liste d'objets qui satisfont aux critères de la demande. L'application cliente importe ainsi la référence de l'objet choisi et l'utilise à travers le bus CORBA.

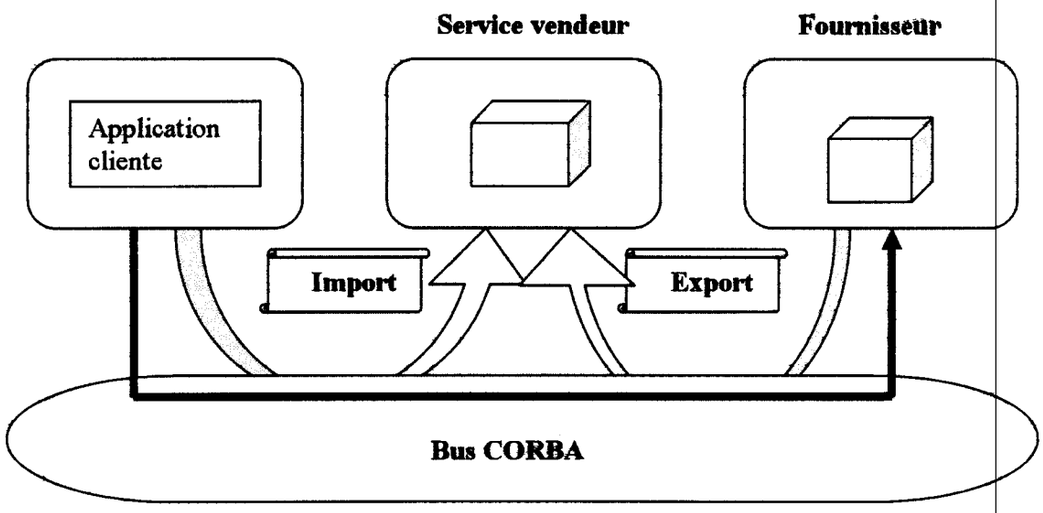


Figure III.8 Le fonctionnement du service vendeur

V.4. Le survol des autres services

- **Le Service Evénement (ou Event Service)** définit un objet appelé « canal d'événements » qui collecte des événements asynchrones et les distribue à des objets consommateurs.

En effet les clients ont besoin d'être notifiés par un événement ou un callback (rappel du client) lors des cas de figure où un changement survient au niveau du serveur.

- **Le Service cycle de vie (ou Life Cycle Service)** définit les opérations de création, copie déplacement et la suppression des objets sur le bus.

- **Le Service Transaction (ou Object Transaction Service)** assure l'exécution de traitements transactionnels impliquant des objets distribués et des bases de données.

- **Le Service Relation (ou Relationship Service)** permet de gérer des associations dynamiques (appartenance, inclusion…) entre les objets.

• **Le Service Propriété (ou Property Service)** permet aux utilisateurs d'associer dynamiquement des valeurs nommées à des objets. Ces propriétés ne modifient pas l'interface IDL, mais représentent des besoins spécifiques du client comme par exemple des annotations.

• **Le Service Sécurité (ou Security Service)** permet d'identifier et d'authentifier les clients, de chiffrer et de certifier les communications et de contrôler les autorisations d'accès.

• **Le Service Concurrence (ou Concurrency Service)** fournit un mécanisme de gestion des accès concurrents à une ressource (mécanisme de verrou).

• **Le Service contrôle de licences** fournit des mécanismes de base pour contrôler, mesurer et limiter l'utilisation des objets.

• **Le service Persistance (Persistent Object Service)** offre des interfaces communes à un mécanisme permettant de stocker des objets sur un support persistant. Quel que soit le support utilisé, ce service s'utilise de la même manière via un «Persistent Object Manager». Un objet persistant doit hériter de l'interface «Persistent Object» et d'un mécanisme d'externalisation.

• **Le service Externalisation (Externalization Service)** fournit un mécanisme standard pour fixer et extraire des objets du bus. Le déplacement et la sauvegarde reposent sur ce service.

• **Le service Persistance (Persistent Object Service)** fournit une interface unique afin de stocker les objets de manière persistante sur un support de stockage.

•**Le service Collections (Collection Service)** permet de manipuler d'une manière uniforme des objets sous la forme de collections et d'itérateurs. Les structures de données classiques (listes, piles, tas,..) sont construites par sous-classement. Ce service est aussi conçu pour être utilisé avec le service d'interrogations pour stocker les résultats de requêtes.

• **Le service Changements (Versionning Service)** permet de gérer et de suivre l'évolution des différentes versions des objets. Ce service maintient des informations sur les évolutions des interfaces et des implantations. Cependant, il n'est pas encore spécifié officiellement [8].

Brièvement, on peut citer le **service Temps (Time Service), le service Messagerie** (CORBA Messaging)

VI. CONCLUSION

Ce chapitre a présenté quelques concepts fondamentaux concernant L'IDL de CORBA, et un certain nombre de services qui ne sont ni plus ni moins que des objets CORBA accessibles à travers leurs interfaces et qui peuvent s'avérer très utiles aux applications développées par l'utilisateur. Parmi ces services, nous pouvons citer le service de nommage "Naming Service" permettant d'échanger des références d'objets en les nommant ou encore le service événement « l'Event Service » qui permet de notifier aux différents objets du système que des évènements particuliers se sont produits mettent en évidence les ambitions grandissantes de l'OMG de diminuer les surcoûts de développement des applications réparties. Toutefois, rien ne vaut que la pratique et que pour y arriver, des schémas directeurs (étapes de développement) sont à suivre.

C'est justement d'ailleurs le but du chapitre suivant qui sera consacré à l'application et présentera les étapes saillantes de l'implémentation.

Chapitre IV

Présentation de l'application

I. INTRODUCTION

Comme cela a été mentionné à la fin du chapitre précédent, le moyen le plus efficace pour concrétiser et acquérir un concept théorique demeure toujours dans sa mise en oeuvre pratique.

C'est pour cette raison que nous avons choisi de mettre en place un fragment d'application répartie de commerce électronique (e-commerce) tournant sur le bus CORBA et dans laquelle clients et serveur sont mis en jeu via des invocations distantes. Cette application permet d'abord d'une part à des clients situés sur des sites éloignés de se connecter au serveur de notre boutique virtuelle et d'y lancer des requêtes. D'autre part, le serveur localise l'objet cible et exécute la méthode appropriée pour répondre aux sollicitations des applications clientes.

Ainsi donc, le présent chapitre met l'accent sur l'application, les outils utilisés et son déploiement dans un environnement réparti.

II. Les outils utilisés

II.1 Environnement de développement utilisé

Si l'algorithmique est une discipline universelle qui consiste à construire un traitement informatique en analysant les différentes tâches élémentaires et ce, dans un langage très proche du langage naturel, la programmation, en d'autres termes l'écriture effective des ordres que la machine doit exécuter nécessite un langage particulier. Il existe de nos jours d'innombrables langages de programmation, plus ou moins répandus, ou plus ou moins dédiés à tel ou tel type d'application particulier. Parmi eux, notre choix s'est orienté vers le langage java avec l'EDI (Environnement de Développement Intégré) JBuilder.

II.2 Raisons du choix de l'environnement JBuilder

Nous ne saurions commencer sans toutefois évoquer le langage java mis en place par Sun en janvier 1995[16]. En effet, celui-ci est un langage purement orienté objet qui permet d'écrire de façon simple et claire des programmes portables sur la majorité des plates-formes.

JBuilder est un environnement de développement mis sur pied par Borland Software Corporation et qui offre une librairie riche, des possibilités complètes d'intégration avec les meilleurs serveurs d'application sur le marché comme IAS (Inprise Application Server), le BES (Borland Entreprise Server), JBoss, Tomcat ou encore le Borland AppServer (Borland Application Server). C'est ce qui fait de lui, un outil de prédilection pour le développement dans le monde java.

Il est important de noter que certains de ces serveurs cités ci haut constituent une solution payante.

II.4 Le Serveur d'application utilisé

Pour mettre en place une application CORBA, le recours à un serveur d'application devient incontournable. Dans ce sens, suite aux contraintes logiciels rencontrées au départ de notre projet, nous avons opté pour le développement de notre application à la version 4 de JBuilder.

En effet celle-ci inclut gratuitement l'IAS (terme découlant de l'ancienne appellation de Borland : Inprise) comme serveur d'application.

II.5 Implémentation de CORBA utilisée : Visibroker

Comme nous l'avons souligné ci haut, notre serveur d'application IAS contient le produit **visibroker** pour java qui constitue l'implémentation CORBA actuelle de Borland.

En effet, visibroker constitue le principal ORB utilisé dans le développement des applications reparties avec la technologie CORBA et dont une des principales caractéristiques est [10] :

- l'architecture **Smart Agent :** le SmartAgent ou OSagent de visibroker est un service de répertoire dynamique utilisé tant par les applications clientes que par les objets CORBA. Il enregistre tous les objets CORBA actifs et disponibles sur le réseau et les localise pour servir les invocations des applications clientes. Lorsqu'un objet se charge, il se référence au niveau l'OSagent, donc dans la liste des objets accessibles [3]. Par la suite, le Smart Agent (ou Agent tout court) effectue des interrogations fréquentes de type broadcast périodiques notamment lors de son lancement afin de détecter les objets actifs pour mettre à jour sa liste des références internes d'objets. En outre l'Agent permet aussi d'assurer la tolérance aux pannes et la répartition des charges (mécanisme qui repartit l'étalement des connexions clientes sur les objets disponibles) lorsque plusieurs instances d'un même objet sont actives sur le réseau.

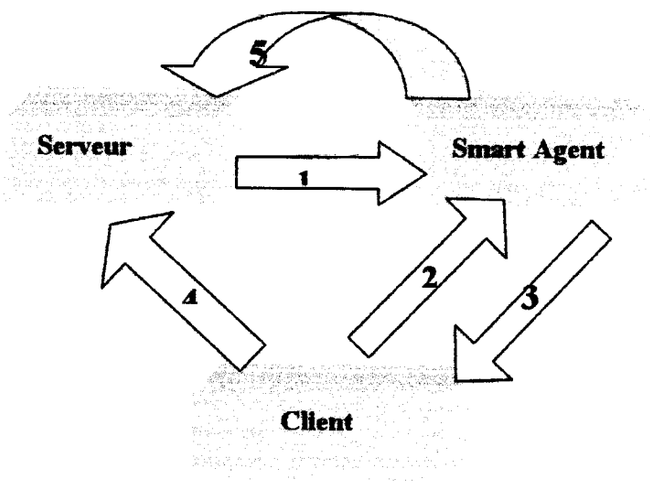


Figure IV.1 Relation entre client, serveur et Smart Agent [3]

Dans la figure ci-dessus, on y trouve lors du lancement de l'application :

1. enregistrement de l'objet serveur sur l'agent visibroker
2. demande de connexion d'un client à un objet distant
3. en réponse retour de la référence à l'objet distant
4. dialogue direct entre le client et l'objet
5. accès périodiques (tolérance aux pannes)

III. Les étapes de développement d'une application CORBA

Dans cette partie nous présentons les étapes à suivre pour développer une application distribuée avec CORBA [17].

Etape 1 : la première chose à faire est de créer l'interface IDL des objets en définissant les opérations fournies et qui seront accessibles aux applications distantes.

Etape 2 : après la spécification des interfaces IDL, l'étape suivante est de compiler l'IDL pour générer les fichiers stub et skeleton.

Etape 3 : Implémenter l'objet c'est-à-dire créer la classe de l'objet dont on souhaite mettre à dispositions des différents clients afin de le concrétiser et de lui donner un corps à travers un servant. Bien qu'il n'y ait aucune règle formelle de ce point de vue [10], les conventions veulent que les servants soient des instances d'une classe dont le nom est <nom interface>Impl.

Etape 4 : à cette étape de développement, on passe à la réalisation du serveur qui contiendra l'objet. En effet celui-ci se charge de l'initialisation de l'ORB, du routage des requêtes et des résultats. Pour rendre accessibles les objets CORBA aux clients, le serveur doit les enregistrer par l'utilisation de :

- l'IOR (Interoperable Object Reference) est une information permettant d'identifier de manière unique et non ambiguë tout objet CORBA dans un ORB [12].En d'autres termes, c'est une référence unique d'un objet CORBA qu'un client peut obtenir et utiliser afin d'accéder aux objets CORBA. Elle contient toutes les informations de localisation nécessaires

Type de l'objet	Adresse réseau	Clé de l'objet

Figure IV.2 Structure d'une IOR [12]

Type de l'objet : permet de différencier des types d'objets différents

Adresse réseau : adresse IP et numéro de port acceptant des invocations de méthodes pour cet objet

Clé de l'objet : identité de l'adaptateur sur ce site et de l'objet sur cet adaptateur.

Par exemple :

IDL:monObjet:1.0	milo:1805	OA7. obj_979

- le service de nom peut être aussi un moyen pour le serveur d'enregistrer les noms des objets CORBA crées et activés. Au fait, au lieu d'avoir à gérer l'IOR de chaque objet CORBA, le serveur l'enregistre sous un nom quelconque et c'est ce nom qui est ensuite utilisé par les clients afin d'obtenir les références d'objets.

En somme, la logique de fonctionnement d'un serveur est de :

- initialiser l'ORB et le BOA
- créer l'objet et l'enregistrer au niveau du BOA
- et enfin la mise en attente

Etape 5 : et puis enfin, il faut passer à la réalisation de l'application cliente et dont la logique de fonctionnement se résume à [17] :

- l'initialisation de l'ORB
- la connexion à l'objet distant
- et puis l'utilisation (appels des méthodes)

IV. Présentation de l'application de commerce électronique (e-commerce)

IV. 1 Définitions du commerce électronique

Dans cette partie, nous explicitons la notion du commerce électronique et qui fait soulever deux grandes questions.

-quelles activités relatives au commerce électronique relèvent du commerce ?

-faut-il prendre en compte le commerce électronique sur l'Internet ou sur tous les réseaux supports ?

Pour ce faire, les définitions ci-dessous donnent des éléments de réponse à ces questions.

D'abord « le commerce électronique désigne l'ensemble de transactions marchandes effectuées sur un réseau électronique ouvert par l'intermédiaire d'ordinateurs ou d'autres terminaux interactifs. »[18].

« Le commerce électronique est une forme d'achat et vente en ligne au cours duquel des services sont échangés entre consommateurs et fournisseurs se basant sur la commande, la fourniture et le règlement »[18].

Dans le cas de notre projet, et dans le simple souci de faciliter la compréhension du e-commerce nous nous limitons juste sur une définition plus restrictive et qui tient seulement sur les achats et ventes de biens et articles effectués sur une infrastructure réseau, qu'ils soient payés ou non, livrés ou non en ligne

IV.2 Description de notre application

Nous proposons dans cette partie de décrire notre application nommée **BOBY.INFO** mise en place. Elle est constituée des objets modélisant des articles d'un stock au niveau d'un magasin virtuel accessible à distance, d'un serveur d'objets et de quelques programmes clients qui utilisent ce serveur. Ces deux parties de l'application (serveur d'objets et clients) s'exécutent au dessus du bus d'objets répartis visibroker, qui est une implémentation CORBA commercialisée par Borland. Nous tenons aussi à rappeler qu'elle ne prend pas en charge les transactions par carte de crédit qui représentent les transferts électroniques de fonds faisant intervenir l'activité bancaire [18].

Le serveur stocke la référence, le libellé, le prix, la quantité disponible d'un article d'une part et d'autre part le pseudo, le solde, l'e-mail, le téléphone, et enfin l'adresse de localisation physique d'un client.

Pour cela notre serveur est capable de répondre à des invocations telles que :

- Du côté des articles du stock, le menu de l'interface suivante le mentionne

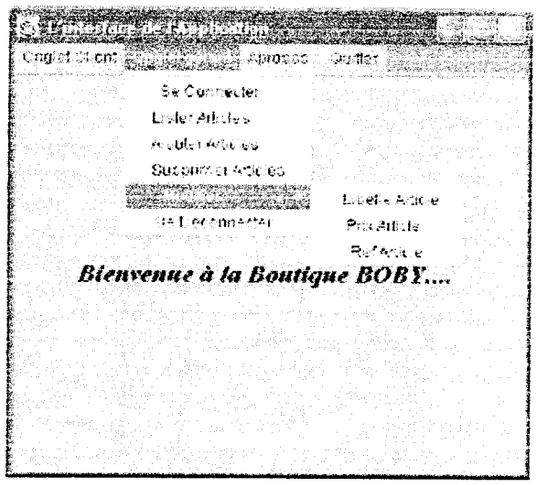


Figure IV.3 Onglet gérant

Les éléments de ce menu ne sont accessibles que par le gérant du magasin qui se connecte par un mot de passe.

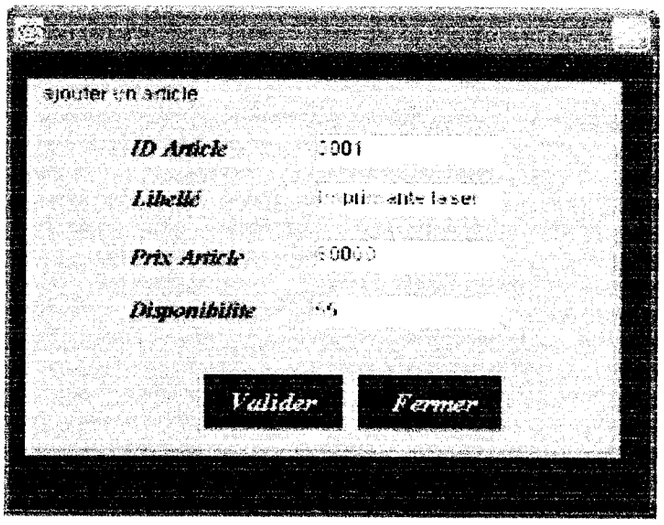


Figure IV.4 L'ajout d'un article

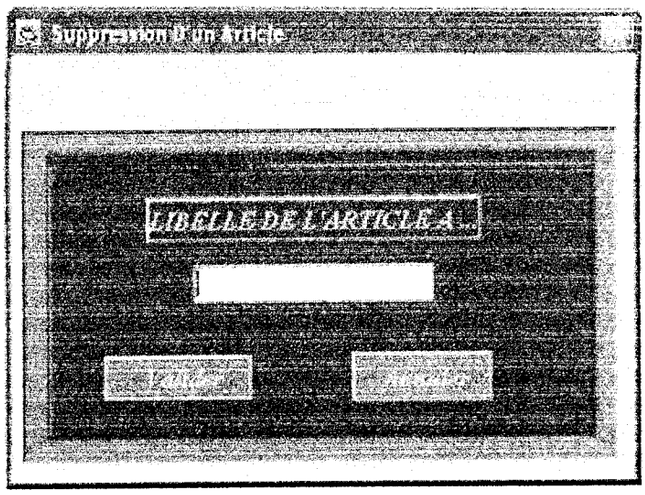


Figure IV.5 La suppression d'un article

- Du côté des clients, l'onglet Client du menu ci-dessus l'illustre :

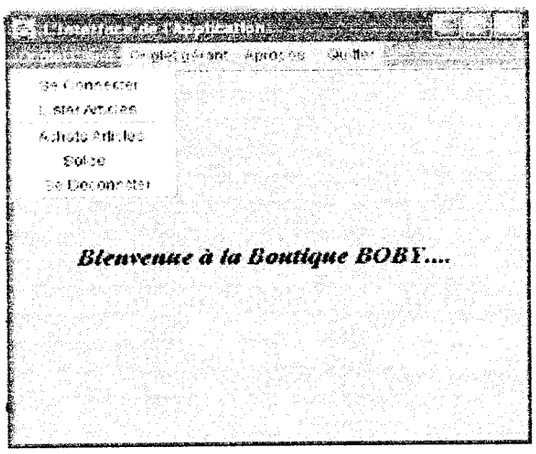


Figure IV.6 Onglet Client

Un client n'ayant pas fait de connexion ne peut utiliser l'application. Ceci étant, tout client doit d'abord passer la procédure de connexion qui lui donne le droit de faire les invocations distantes. L'interface ci – dessous demande au client d'entrer les informations comme nous l'avons évoqué ; il s'agit du Pseudo, du Solde, de l'e-Mail, du Numéro de téléphone et enfin de l'adresse sur laquelle est localisé le client afin de lui acheminer les articles achetés.

L'interface ci-dessous permet à un client de faire ses achats. Elle contient deux listes dont la première présente tous les articles disponibles et la seconde permet au client de voir les articles et les quantités commandées avant de passer à la validation des ses achats.

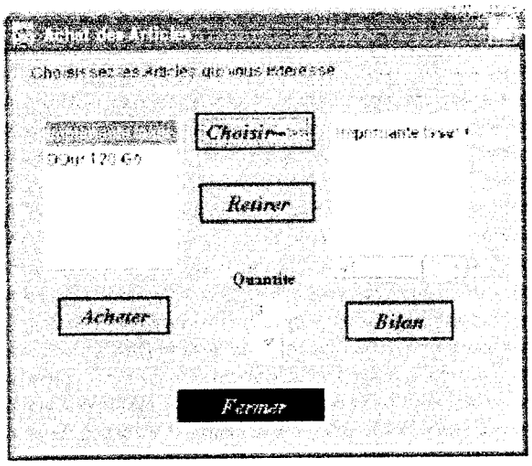


Figure IV.9 Interface des achats

Après avoir fait les achats des articles qui l'intéressent, le client peut à n'importe quel moment consulter son solde afin éventuellement d'entrevoir d'autres achats.

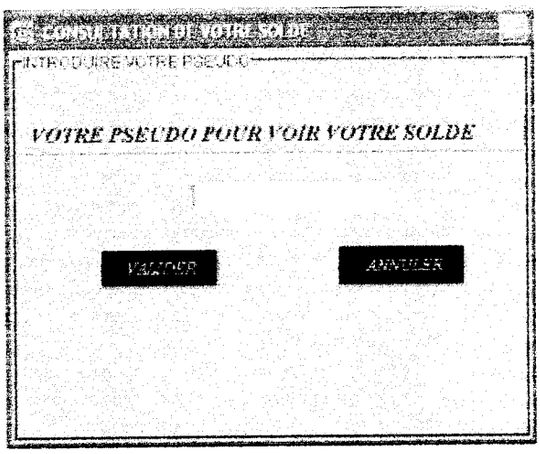


Figure IV.10 Consultation du solde

Dans tout ça, le plus important est de voir la trace laissée par les invocations au niveau du serveur distant et cela est illustré par :

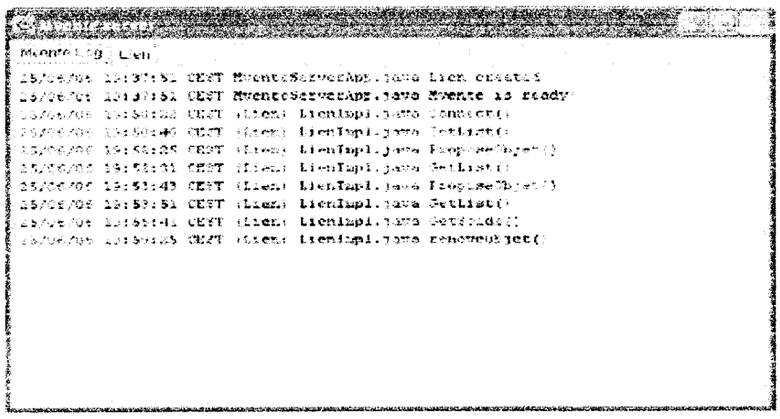


Figure IV.11 Trace des invocations

IV.3 Exécution de l'application

Nous avons utilisé le réseau du laboratoire de télécommunication pour pouvoir exécuter l'application sur deux postes reliés en réseau dont sur l'un, se trouve uniquement le programme client (l'invocateur) et l'autre le programme serveur (l'implémentation). Ainsi donc, pour exécuter l'application, il faut d'abord :

1) Exécuter le Smart Agent
2) Exécuter l'application serveur.
3) Exécuter l'application cliente

La fenêtre suivante qui représente la page d'accueil apparaît :

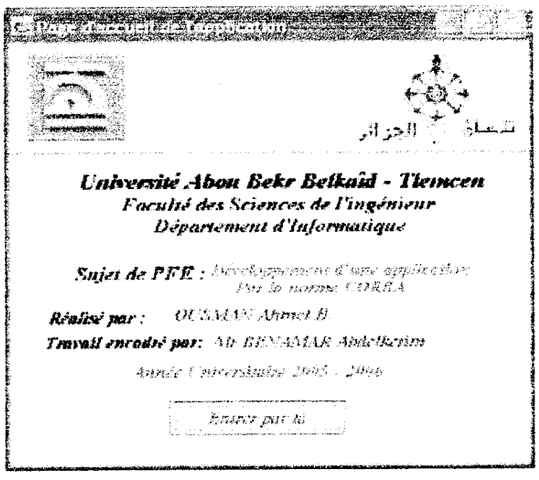


Figure IV.12 Page d'accueil

V. Conclusion

En somme, nous venons de mettre en place un fragment d'application répartie de commerce électronique en utilisant le bus CORBA. Cela aurait sans doute été un peu complexe si nous n'avions pas utilisé JBuilder et bénéficié heureusement de la richesse de sa bibliothèque et de la multitude d'experts dont il dispose et dont nous n'avons pas hésité à faire usage dans toutes les étapes et tout au long du processus de réalisation des opérations de notre application.

CONCLUSION GENERALE

Au terme de notre travail, ainsi qu'à la modeste contribution que nous avons apportée et illustrée en donnant les étapes techniques de construction d'un fragment d'application repartie de commerce électronique au dessus du bus CORBA, il convient de récapituler les résultats et les conclusions qu'ils impliquent. L'informatique est planétaire à tel point que cela engendre de besoins constants de présenter les données à un endroit, aller les chercher à un autre et parfois effectuer des traitements à un troisième. Pour cela, le recours à des standards de développement d'applications reparties devient donc incontournable. Aussi faut-il constater que c'est dans ce sens que de nombreuses solutions telles que java RMI,DCOM et CORBA ont été mises en place pour le développement d'applications reparties. Cependant à travers le thème étudié, la solution CORBA de l'OMG se distingue des autres car elle offre :

– Une solution ouverte et évolutive : choisir CORBA revient à ne pas s'enfermer dans une solution propriétaire évoluant difficilement.

– Une architecture modulaire : grâce à l'OMA, une application répartie n'est pas construite à partir de zéro mais réutilise des composants logiciels offrant des fonctions orientées système, utilisateur et/ou métier.

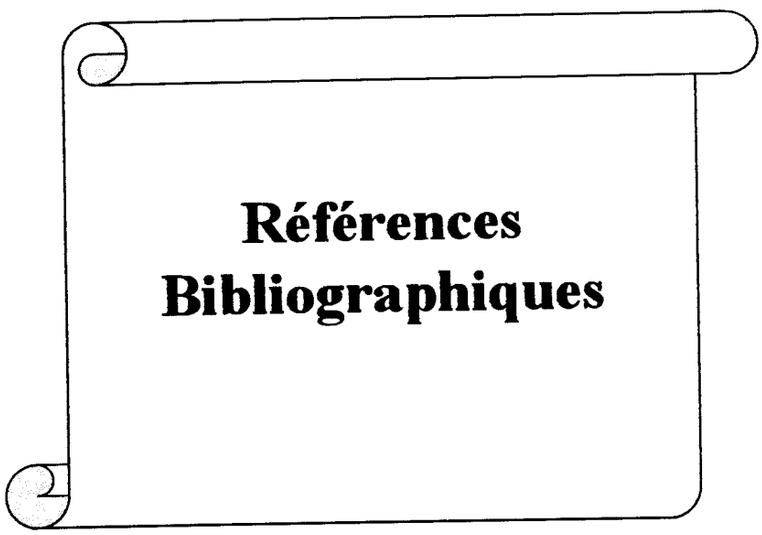
– L'interopérabilité entre composants hétérogènes : CORBA fournit les abstractions et mécanismes offrant un bus orienté objet d'intégration de composants logiciels conçus avec des technologies hétérogènes (langages, systèmes d'exploitation, machines et réseaux). Ainsi, diverses technologies peuvent être mises en œuvre et coopérer dans la même application.

– Elle se base sur un bus qui constitue incontestablement la colonne vertébrale et un langage purement descriptif qui fournit les articulations.

D'une part, toutes ces raisons ont fait que l'intérêt pour CORBA va grandissant et sa mise en œuvre est devenue l'un des atouts majeurs du développement d'applications réparties.

D'autre part la nouvelle approche MDA (Model Driven Architecture) de l'OMG qui définit une méthode de conception basée sur les modèles permettant d'obtenir à partir d'un schéma de base unique sa réalisation sur n'importe quelle plate-forme (supportée par le MDA) fait de CORBA une solution d'avenir portée par les plus grands éditeurs des logiciels.

Toutefois, il n'en demeure pas moins vrai que le choix d'une architecture ne dépend que des contextes d'utilisation lors du développement d'une application repartie.



Références Bibliographiques

Références bibliographiques des chapitres

[1] : Pillou, J-F. ; 2003-Internet fonctionnement

[En ligne : http://www.commentçamarche.net].

[2] : Alain Lefèbvre : l'architecture client-serveur (1994).

[3] : David, A ; Gilles, M & Laurent, R ; 1999 – Développer avec CORBA en JAVA et en C++

[4] : Kondo Messiba A ; 2000 – Data Provider : Un serveur de données sous RMI de Java. Mem . Postgrade en Informatique et organisation. Univ. de Lausanne

[En ligne : http://www.hec.unil.ch/cms_inforge/KondoMessibaAdjallah].

[5] : Sasha Krakowiak ; 2005-2006- Construction d'applications réparties et parallèles. Cours Master 2 Recherche "Systèmes et logiciel". Univ. Joseph Fourier.

[En ligne : http://www.sardes.inrialpes.fr/~krakowia].

[6] : Benamar, K & Abdelhamid, D ; 2003-2004 – Parallélisme avec passage de messages. Mem .Ing en Informatique. Univ Abou Bekr Belkaid de Tlemcen.

[7] : Pierre Yves, C ; David, D & Aurélien, G ; 2002 – Technologies et Architectures Internet:CORBA, COM, XML, J2EE, .NET, WEB Services.

[8] : Jean, M-G ; Christophe, G & Philippe, M; 1999 – CORBA : Des concepts à la pratique

[9] : Marie-Claude P ; S.D. Reconfiguration d'applications réparties : application au bus logiciel CORBA
Thèse Doct. es Sc. Informatique, Communication et système. Inst. Nat. Polytech. De Grenoble

[En ligne : http://www.sci-serv.inrialpes.fr/papers/files/99-Pelligrini-phd].

[10] : Nicolas Duminil ; 2002 – J2EE avec JBuilder 7 Enterprise.

[11] : Samia Bouzefrane ; SD. Les systèmes et applications reparties et leur programmation.

[En ligne : http://cedric.cnam.fr/~bouzefra]

[12] : Laurence Duchien ; SD. Le bus logiciel CORBA.

[En ligne : www.cedric.cnam.fr/~duchien]

[13] : Georges E, Kouamou ; 1998. Les Standards de SIMES : Installation et Utilisation de MICO, une réalisation de CORBA.

[En ligne : http://www.inria.fr].

[14] : Thomas Ledoux ; 1998. Réflexion dans les systèmes répartis : application à CORBA et Smalltalk

Thèse Doct. es Sc. Informatique. École doctorale « Sciences Pour l'Ingénieur » de Nantes

[En ligne : www.emn.fr/x-info/obasco/bibliography/papers/Ledoux_phd1998.pdf]

[15] : Aurélien Esnard ; 2004. Introduction à CORBA.

[En ligne : http:/www.labri.fr/perso/Esnard/Enseignement/corba/intro.pdf]

[16] : Irène Charon ; 2003 – Le langage Java : Concepts et pratique

[17]: Gerald, B; Andreas, V; Keith, D; 2001 – Java Programming With CORBA: Advanced Techniques for Building Distributed Applications. Third Edition

[18]: Benghozi, P-J; Licoppe, C; Rallet, A; 2001- Internet et commerce electronique