Chapitre 7 CORBA

1. Introduction

L'hétérogénéité est actuellement un des grands problèmes des systèmes distribués. Elle se retrouve au niveau *hardware, système d'exploitation* et *langage*.

CORBA qui est à l'inverse d'autres architectures logicielles, n'est pas un outil ou un ensemble d'outils créés par un seul éditeur de logiciels. CORBA est une norme, un ensemble de spécifications et de recommandations rédigées par un groupe de travail nommé OMG (Object Management Group), auquel appartiennent la plupart des acteurs informatiques majeurs.

Depuis sa création, ce groupe travaille à la définition de "l'architecture distribuée idéale", désignée sous le sigle CORBA. Pour cela nous intéressons-nous aux concepts fondamentaux sur lesquels repose ce nouveau modèle : l'architecture distribuée et la programmation orientée objet.

1.1 Qu'est-ce que l'Architecture Distribuée ?

Avant de définir précisément la nature d'une architecture distribuée, nous allons procéder à un rappel historique des différents modèles, ce qui va nous permettre de retracer l'origine de l'architecture distribuée, et d'en comprendre tout le potentiel.

Partons de l'architecture client/serveur qui, au cours de ces dix dernières années, s'est progressivement imposée du fait de sa simplicité. Figure 1.1 illustre cette architecture. L'application (interface homme machine et traitements) est, dans sa quasi-totalité, située sur le poste de l'utilisateur. Un serveur de bases de données gère les données auxquelles l'application accède via des ordres SQL (requêtes de sélection, d'ajout, de modification ou encore de suppression de données, etc.). Les règles d'intégrité référentielles sont définies directement dans la base de données sous la forme de contraintes, de règles d'intégrité ou de déclencheurs. Quelques traitements existent dans la base sous la forme de procédures stockées, mais ils sont obligatoirement écrits en langage propriétaire (par exemple, le PL/SQL pour un serveur Oracle), un langage plutôt limitatif par rapport aux langages évolués du type C++, Java ou encore ObjectPascal.

Figure 1.1 Le client/serveur traditionnel.

Les limites de ce système sont vite apparues :

- Problèmes de performances lorsqu'un grand nombre d'utilisateurs se connecte en parallèle.
- Impossibilité d'écrire des traitements complexes au niveau du serveur.
- Mauvaise tolérance aux pannes et peu de mécanismes de répartition de charge : le lien entre client et serveur est direct, ce qui provoque un blocage complet du système lorsque l'un des éléments est défaillant….
-..Etc.

En réponse à tous ces problèmes est apparu un nouveau modèle d'architecture nommé client/serveur à *trois niveaux*, dit de seconde génération .Un intermédiaire est placé entre le client et le serveur (voir Figure 1.2). Ainsi, le client n'accède jamais directement au serveur. Il émet des requêtes et des demandes de traitement à ce poste, dit serveur de traitement ou serveur applicatif, qui exécute directement les traitements et transmet les requêtes au serveur de base de données.

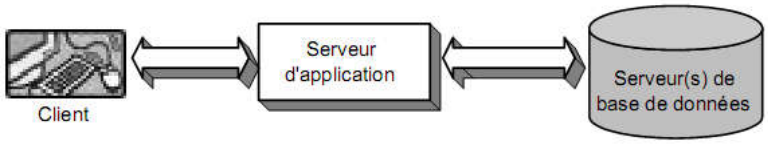

Figure 1.2 Le client/serveur à trois niveaux.

Dans cette architecture, on parle de niveaux ou de tiers pour identifier les différents éléments que constituent client, serveur d'application et serveur de base de données. A chaque niveau correspondent un rôle et des activités précises :

- **Client** : Caractérisé par l'interface homme machine de l'application, on cherche à placer le moins d'éléments possible sur ce poste. En fait, seule la couche de présentation est placée sur le client.

- **Serveur d'application** : La logique de l'application est déplacée ici. Ce poste est un relais entre le *client* et le *SGBD*, mais il n'est pas cantonné dans un simple rôle de transmetteur/récepteur puisque la logique applicative y est localisée. On y trouve de vrais traitements et règles de gestion qui peuvent n'avoir aucun lien direct avec les données physiques : ils sont écrits dans n'importe quel langage (Java, C++, Delphi, etc.) et, ainsi, ne souffrent d'aucune limitation quant à leur réalisation.

- **Serveur de base de données** : Comme en client/serveur de première génération, ce serveur sert au stockage de données et à l'exécution de traitements "proches des données" : procédures stockées écrites en langage propriétaire (par exemple, PL/SQL).

Les avantages de cette seconde génération de client/serveur sont multiples :
- Centralisation des traitements, ce qui facilite la maintenance et les évolutions.
- Partage possible entre des traitements communs à plusieurs applications.
- Dissociation entre le nombre d'utilisateurs et les connexions serveur : un serveur de traitement peut multiplexer les accès utilisateur et partager une même connexion au SGBD.

A la vue de tous ces avantages, on pourrait croire que cette architecture est la meilleure, celle qui va s'imposer dans les développements de grande envergure, mais... ce serait oublier quelques "détails" :

- La communication entre les différents éléments n'est pas standardisée, ou alors à un bas niveau (par exemple, TCP/IP), ce qui alourdit la charge de développement
- Il manque une souche commune, un ensemble d'outils à notre disposition pour simplifier les développements et, surtout, pour nous éviter une réécriture systématique de certains modules utilitaires.

Toutes ces raisons ont provoqué l'éclosion d'une nouvelle architecture, dite **distribuée,** dans laquelle les concepts d'objets, de protocole standard ou encore d'outils sont omniprésents (voir Figure 1.3). Dans cette illustration, vous constaterez la forte indépendance géographique

entre les différents éléments qui sont éclatés sur les différents ordinateurs. L'ensemble du dialogue entre ces machines est alors pris en charge par le *middleware* choisi.

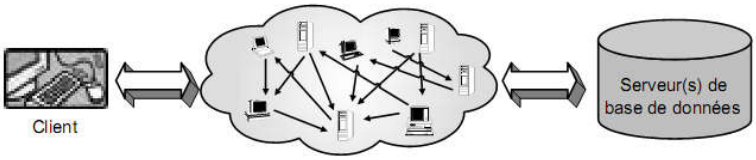

Figure 1.3 L'architecture distribuée.

1.2 CORBA et OMG (Object Management Group) :

L'Object Management Group (OMG) est un consortium international créé en 1989 et regroupant actuellement plus de 850 acteurs du monde informatique, comme *IBM, Sun, HP*. L'objectif de ce groupe est de faire émerger des standards pour la distribution d'applications hétérogènes à partir des technologies orientées objets et de permettre la réutilisabilité de composants logiciels, programmés dans différents langages. L'élément clé de la vision de l'OMG est CORBA (Common Object Request Architecture) : un bus d'objets répartis qui prend en charge les communications entre les différents objets.

CORBA a été conçu pour gérer l'hétérogénéité des machines et des langages de programmation. Pour résoudre ce problème, l'OMG a créé un langage commun : *OMG-IDL*, qui décrit l'interface des objets de façon abstraite. D'autre part, la clé de voûte de CORBA est le transport de requêtes entre objets : c'est le rôle du bus CORBA : ORB.

L'OMG a donc défini une architecture globale, visant à classifier les différents objets qui interviennent dans la construction d'applications réparties. Cette architecture comprend :
- un bus d'objets repartis : ORB
- des services de base (CORBAServices) qui fournissent les fonctions nécessaires à la plupart des applications réparties.
- des utilitaires communs qui répondent plus particulièrement aux besoins des utilisateurs

- des interfaces d'objets applicatifs qui définissent les objets créés par les utilisateurs de l'architecture

1.3 La notion de middleware :

Le middleware permet de résoudre les problèmes liés à l'intégration des différentes plates-formes utilisées dans vos développements. Il se charge de la communication entre applications ou parties d'applications différentes, éventuellement localisées sur des sites distants et, bien souvent, écrites dans des langages distincts. Il peut s'agir, par exemple, de faire dialoguer une applet Java avec une base de données DB2, de permettre à une feuille de calcul Excel d'obtenir des données à partir d'une base Oracle ou encore de réutiliser un vieux module COBOL sur un site central en affichant les données à partir d'une application graphique C++.

1.4 CORBA est-il un middleware ?

Bien sûr, CORBA est même un middleware "objet". Il permet à des systèmes hétérogènes de communiquer via un langage commun et un ensemble d'outils standardisés. Il repose sur la notion de bus logiciel, appelé ORB (Object Request Broker), dont c'est précisément le rôle.

Omniprésente dans CORBA, la *notion d'objet* est l'un des piliers de cette norme. L'objet est à CORBA ce que le microprocesseur est à l'ordinateur : ils sont indissociables ! La norme CORBA impose non seulement la définition d'objets, mais aussi l'utilisation d'objets existants, puisque tous les outils que vous allez devoir employer pour réaliser votre application sont aussi des objets. Vous pourrez ainsi bénéficier des apports de cette méthodologie en termes de réutilisabilité (héritage), de protection des informations (encapsulation) ou encore d'accès dynamique (polymorphisme).

1.5 La norme CORBA :

CORBA (Common Object Request Brocker Architecture) est une telle plate-forme. Il est issus des travaux de l'OMG (Object Management Group) et est basé sur la notion d'objets distribués : cette norme propose une architecture de gestion d'objets répartis (Object Management Architecture - OMA). Elle vise à prendre en compte toutes les fonctionnalités nécessaires à la construction d'applications réparties.

Corba a été conçu pour permettre à des composants intelligents (des objets) de se découvrir les uns les autres et d'interopérer sur un bus. Mais en plus, cette norme définit un vaste ensemble de services associés au bus qui permettent de créer/supprimer des objets, de leur assigner un nom, de les rendre persistant, d'externaliser leur état, de définir des relations entre eux, de traiter des transactions, ... Ainsi, grâce à ces services, on peut créer un objet « ordinaire » puis le rendre persistant et/ou fonctionnel par la simple utilisation de ces services.

Ainsi, les objets Corba peuvent être distribués sur plusieurs sites →nécessités de mécanisme d'appel de méthodes à distance. Le langage objet doit coopérer avec le système d'exploitation et les couches de communication réseau pour réaliser les invocations entre les objets localisés sur des sites distants.

CORBA intègre donc un bus permettant la connexion d'applications. Les objets communiquent alors de manière transparente par l'intermédiaire d'objets souche appelés aussi mandataire, proxies ou surrogates :

-Une souche-cliente (intégrée dans une interface) est un représentant local d'un objet distant : il connaît la localisation de l'objet et implante les mécanismes de communication pour invoquer l'objet (extension des RPC).

-Une souche-serveur (intégré dans le serveur) est une « extension » à un objet, cette extension étant chargée de transférer une demande distante (d'une souche-client) en un appel local de méthode.

CORBA est construit sur ce mécanisme qui donne aux objets la capacité via des interfaces décrites initialement dans un langage propre à CORBA, le langage IDL (Interface Description Language) d'utiliser des services distants.

2.1 L'architecture globale de CORBA

Pour faciliter la construction des applications reparties efficaces et pour assurer l'intéropérabilite, et l'intégration des applications hétérogènes, l'OMG a propose une architecture globale d'écrite dans l'Object Management Architecture Guide (l'OMA Guide). Cette architecture constitue en fait un modèle de construction et d'organisation des infrastructures de la plate-forme répartie CORBA. Elle est globale car elle prend en compte toutes les fonctionnalités nécessaires à la construction d'une application répartie aussi bien au niveau technique comme la recherche des objets dans le système distribué, qu'au niveau applicatif (voir la figure suivante).

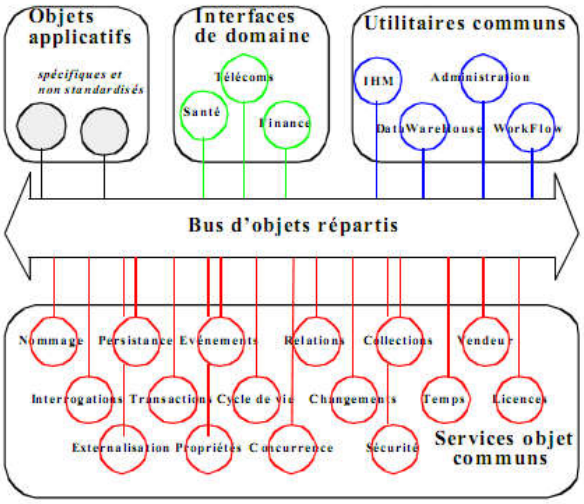

Figure 2.3 :L'architecture globale de OMG-CORBA

2.1.1 Le bus d'objet répartis :

Le bus Corba (Bus d objets répartis) se charge de masquer les différences entre les langages d'implantation des objets (aujourd 'hui C++, CORBA, Smallatlak, COBOL, JAVA, Ada, C), les systèmes d'exploitation et les architectures matérielles.

Il rend transparente la localisation des objets. EN capturant les appels il est responsable de retrouver un objet qui implémente une requête...Le bus d'objets répartis CORBA est appelé négociateur ou transitaire de requêtes objet. Son rôle est de fournir une infrastructure ou MiddleWare pour réaliser des applications distribuées à l'aide d'objets hétérogènes.

2.1.2 Les services objet communs (CORBAservices) :

Ils fournissent sous forme d'objets CORBA, spécifiés grâce au langage OMG-IDL, les fonctions systèmes nécessaires à la plupart des applications réparties. Actuellement, l'OMG a défini des services pour les annuaires (Nommage et Vendeur), le cycle de vie des objets, les relations entre objets, les événements, les transactions, la sécurité, la persistance, (il y en a 15)etc. Au fur et à mesure des besoins, l'OMG ajoute de nouveaux services communs.

On peut citer comme exemple

- ***la recherche d'un objet*** **:** Cette catégorie de services offre les mécanismes pour rechercher/retrouver dynamiquement sur le bus les objets nécessaires aux applications. Ce sont les équivalents des annuaires téléphoniques :

· Le service Nommage (Naming Service) est l'équivalent des « pages blanches » : les objets sont désignés par des noms symboliques. Cet annuaire est matérialisé par un graphe de répertoires de désignation.

· Le service Vendeur (Trader Service) est l'équivalent des « pages jaunes » : les objets peuvent être recherchés en fonction de leurs caractéristiques.

- ***La vie d'un objet*** **:** Cette catégorie regroupe les services prenant en charge les différentes étapes de la vie des objets CORBA.

-Le service Cycle de Vie (Life Cycle Service) décrit des interfaces pour la création, la copie, le déplacement et la destruction des objets sur le bus. Il définit pour cela la notion de fabriques d'objets («Object Factory»).

-Le service Propriétés (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 Relations (Relationship Service) sert à gérer des associations dynamiques (appartenance, inclusion, référence, auteur, emploi,...) reliant des objets sur le bus. Il permet aussi de manipuler des graphes d'objets.

-Le service Externalisation (Externalization Service) apporte un mécanisme standard pour fixer ou extraire des objets du bus. La migration, le passage par valeur, et la sauvegarde des objets doivent reposer sur ce service.

-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 Interrogations (Query Service) permet d'interroger les attributs des objets. Il repose sur les langages standard d'interrogation comme SQL3 ou OQL. L'interface Query permet de manipuler les requêtes comme des objets CORBA. Les objets résultats sont mis dans une collection. Le service peut fédérer des espaces d'objets hétérogènes.

-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.

- ***La surtée de fonctionnement*** Cette catégorie de services fournit les fonctions système assurant la sûreté de fonctionnement nécessaire à des applications réparties.

-Le service Sécurité (Security Service) permet d'identifier et d'authentifier les clients, de chiffrer et de certifier les communications et de contrôler les autorisations d'accès. Ce service utilise les notions de serveurs d'authentification, de clients/rôles/droits (et délégation de droits), d'IIOP sécurisé (utilisant Kerberos ou SSL).

-Le service Transactions (Object Transaction Service) assure l'exécution de traitements transactionnels impliquant des objets distribués et des bases de données en fournissant les propriétés ACID.

-Le service Concurrence (Concurrency Service) fournit les mécanismes pour contrôler et ordonnancer les invocations concurrentes sur les objets. Le mécanisme proposé est le verrou. Ce service est conçu pour être utilisé conjointement avec le service Transactions.

- ***Les communications asynchrones*** Par défaut, la coopération des objets CORBA est réalisée selon un mode de communication client/serveur asynchrone. Toutefois, il existe un ensemble de services assurant des communications asynchrones.

-Le service Evénements (Event Service) permet aux objets de produire des événements asynchrones à destination d'objets consommateurs à travers des canaux d'événements. Les canaux fournissent deux modes de fonctionnement. Dans le mode « push », le producteur a l'initiative de la production, le consommateur est notifié des événements. Dans le mode « pull », le consommateur demande explicitement les événements, le producteur est alors sollicité.

- Le service Notification (Notification Service) est une extension du service précédent. Les consommateurs sont uniquement notifiés des événements les intéressant. Pour cela, ils posent des filtres sur le canal réduisant ainsi le trafic réseau engendré par la propagation des événements inintéressants.

- Le service Messagerie (CORBA Messaging [OMG-CM]) définit un nouveau modèle de communication asynchrone permettant de gérer des requêtes persistantes lorsque l'objet appelant et l'objet appelé ne sont pas présents simultanément sur le bus. Cela permet d'avoir des fonctionnalités proches des MOMs

2.1.3 Les utilitaire communs :

Elles sont des canevas d'objets construits au dessus des services pour offrir des fonctionnalités de plus haut niveau: interface utilisateur, gestion de l'information, administration du système et la gestion de la tâche.

Interface utilisateur : les scripts CORBA

Gestion de l'information : modélisation,

Administration du système: instrumentation (interface standard pour collecter les infos sur la charge de travail des composants, leur consommation de ressources, leur réactivité, les débits de communication, les erreurs et les pannes diverses);

2.1.4 Les interface de domaine :

Elles sont dédiées à des segments de marché verticaux ou économiques. Elles permettent, par exemple l'interopérabilité et la collaboration dans les domaines de la santé (objet Patient), des finances (monnaie, convertisseur de monnaie) des Télécom ou du commerce électronique.

Leur objectif est de pouvoir assurer l'interopérabilité sémantique entre les systèmes d'informations d'entreprises d'un même métier : les « Business Object Frameworks » (BOFs).

2.1.5 Les objet applicatif (Application Objects) :

Ils sont ceux qui sont spécifiques à une application répartie et ne sont donc pas standardisés. Toutefois, dès que le rôle de ces objets apparaît dans plus d'une application ils peuvent alors rentrer dans une des catégories précédentes et donc être standardisés par l'OMG.

2.2 Langage de description d'interfaces (OMG-IDL)

2.2.1 La notion d'IDL :

Le langage OMG-IDL est la base de la force de CORBA. Il a été créé pour rendre compatible l'utilisation de plusieurs applications écrites dans différents langages de programmation, ceci s'appelle l'interopérabilité. Il cherche à masquer l'hétérogénéité des applications

Pour ceci, il a été défini pour décrire uniquement les interfaces indépendamment du langage d'implantation.

Pourquoi décrire uniquement les interfaces ?

Au départ, l'OMG voulait créer un langage unique qui utiliserait les avantages de chaque langage déjà existant. En effet, actuellement quand on programme une application, on utilise souvent un unique langage de programmation. Idéalement, on pourrait choisir le langage le mieux approprié à chaque partie de l'application et réutiliser du code existant. La solution idéale consisterait donc à mélanger tous les langages pour créer une application très performante.

Ceci est malheureusement utopique car il existe de nombreuses incompatibilités entre les langages. Chacun définit sa propre vision : de la structuration des données : type abstrait, langage objet ou non, ... de la structuration de code : fonction, package, classe ... Ceci rend les langages rarement compatibles.

D'où l'idée de l'OMG de créer un langage objet : OMG-IDL qui ne décrit que les interfaces des objets, de façon indépendante du langage d'implantation. L'interface d'un objet comporte :

- les opérations
- le nom des méthodes, ses paramètres, son type de retour
- les attributs, les types et les exceptions.

Ainsi, une application qui comporte plusieurs objets écrits dans différents langages peut les lier grâce à ce nouveau langage, qui isole l'interface de l'implantation.

Du côté client, on a uniquement l'interface de l'objet ;

Du côté serveur, en plus de l'interface des objets, on a aussi l'implantation dans des langages de programmation tels que C, C++, Java, Smalltalk, Cobol ... Le bus CORBA permet de lier l'interface de l'implantation.

2.2.2 La projection vers un langage de programmation :

Une projection est la traduction d'une spécification OMG-IDL dans un langage d'implantation. Pour permettre la portabilité des applications d'un bus vers un autre, les règles de projection sont normalisées et fixent précisément la traduction de chaque construction IDL en une ou des constructions du langage cible et les règles d'utilisation correcte de ces traductions. Actuellement, ces règles existent pour les langages C, C++, SmallTalk, Ada, Java et Cobol orienté objet. Nous ne détaillons pas ici ces règles, La projection est réalisée par un pré-compilateur IDL dépendant du langage cible et de l'implantation du bus CORBA cible. Ainsi, chaque produit CORBA fournit un pré-compilateur IDL pour chacun des langages supportés. Le code des applications est alors portable d'un bus à un autre car les souches/squelettes générés s'utilisent toujours de la même manière quel que soit le produit CORBA. Par contre, le code des souches et des squelettes IDL n'est pas forcément portable car il dépend de l'implantation du bus pour lequel ils ont été générés.

La figure suivante illustre la pré-compilation d'un contrat IDL vers les langages cibles C++ et Java. Comme les deux applications vont dialoguer à travers IIOP, on peut très bien d'un côté utiliser un pré-compilateur IDL/C++ fourni par un bus A et de l'autre côté utiliser un pré-compilateur IDL/Java fourni par un bus B.

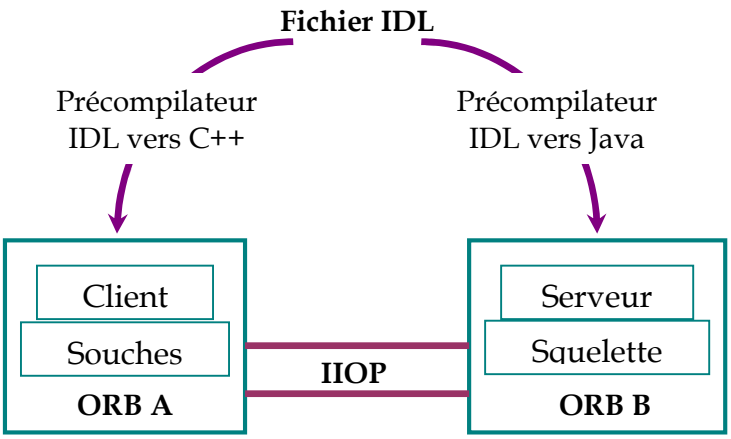

Figure 2.2 : Pré-compilation du contrat Idl vers deux langages différents

La souche est la partie du code générée automatiquement par un compilateur IDL vers un langage de programmation cible. Ce code est utilisé par le client lors des invocations statiques. Il est le lien entre le client et l'ORB. Il traduit :

- les invocations du client en messages transmissibles sur le réseau : opération "marshalling",
- les messages qui proviennent de l'ORB dans le langage choisi : opération "unmarshalling".

Le squelette est la partie du code générée automatiquement par un compilateur IDL vers un langage de programmation cible. Ce code est utilisé par l'adaptateur d'objet lors des invocations statiques. Il est le lien entre l'ORB et l'implémentation de l'objet.

IIOP : (Internet Inter-ORB Protocol) : GIOP (General Inter-ORB Protocol) spécifie un ensemble de formats pour les messages ainsi qu'une représentation commune des données échangées entre les ORBs. La représentation des données communes est basée sur la spécification CDR (Common Data Representation). IIOP est l'implémentation du protocole GIOP basé sur TCP/IP.

2.2.3 La mise en place d'une application CORBA :

L'OMG n'impose pas de processus de conception et de développement d'applications distribuées : la norme CORBA laisse une entière liberté sur le choix des outils à mettre en œuvre.

Néanmoins, la mise en place d'une application CORBA suit toujours à peu près le même scénario :

1. **La définition du contrat IDL** : à partir du cahier des charges, il faut définir les objets composant l'application à l'aide d'une méthodologie orientée objet (e.g. OMT). Cette modélisation est ensuite traduite sous la forme de contrats IDL composés des interfaces des objets et des types de données utiles aux échanges d'informations entre les objets.
2. **La pré-compilation du contrat IDL** : les interfaces des objets sont décrites dans des fichiers texte. Le pré-compilateur prend en entrée un tel fichier et opère un contrôle syntaxique et sémantique des définitions OMG-IDL contenues dans ce fichier. Le pré-compilateur peut aussi charger ces définitions dans le référentiel des interfaces. C'est la partie frontale commune à tous les pré-compilateurs IDL.
3. **La projection vers les langages de programmation** : le pré-compilateur IDL génère le code des souches qui sera utilisé par les applications clientes des interfaces décrites dans le fichier IDL, ainsi que le code des squelettes pour les programmes serveurs implantant ces types. Cette projection est spécifique à chaque langage de programmation : un environnement CORBA fournit un pré-compilateur IDL pour chacun des langages supportés.
4. **L'implantation des interfaces IDL** : en complétant et/ou en réutilisant le code généré pour es squelettes, le développeur implante les objets dans le langage de son choix ou dans le langage le mieux adapté à la réalisation de ses objets. Il doit tout de même se plier aux règles de projection vers ce langage.

5 **L'implantation des serveurs d'objets** : le développeur doit écrire les programmes serveurs qui incluent l'implantation des objets et les squelettes pré-générés. Ces programmes contiennent le code pour se connecter au bus, instancier les objets racines du serveur, rendre publiques les références sur ces objets à l'aide par exemple du service Nommage et se mettre en attente de requêtes pour ces objets. Le nombre de programmes serveurs est tout de même souvent limité.

6 **L'implantation des applications clientes des objets** : le développeur écrit un ensemble de programmes clients qui agissent sur les objets en les parcourant et en invoquant des opérations sur ceux-ci. Ces programmes incluent le code des souches, le code pour l'interface Homme-Machine et le code spécifique à l'application. Les clients obtiennent les références des objets serveurs en consultant le service Nommage. Il faut tout de même souvent développer une multitude de programmes clients des objets en fonction des rôles et des activités de chacun des utilisateurs de l'application répartie.

7 **L'installation et la configuration des serveurs** : cette phase consiste à installer dans le référentiel des implantations les serveurs pour automatiser leur activation lorsque des requêtes arrivent pour leurs objets.

8. **La diffusion et la configuration des clients** : une fois les programmes clients mis au point, il est nécessaire de diffuser les exécutables sur les sites de leur future utilisation et de configurer les sites clients pour qu'ils sachent où se trouvent les serveurs utilisés.

9 **L'exécution répartie de l'application** : enfin, l'exploitation de l'application peut commencer. Le bus d'objets répartis CORBA assure alors les communications entre les programmes clients et les objets via le protocole IIOP.

2.3 Le bus CORBA

Le bus CORBA est l'intermédiaire/négociateur à travers lequel les objets vont pouvoir dialoguer. Il permet de réaliser des applications composées d'objets répartis.

2.3.1 Les caractéristiques :

Il fournit les caractéristiques suivantes :

· La liaison avec « tous » les langages de programmation : cependant, actuellement l'OMG a seulement défini officiellement cette liaison pour les langages C, C++, SmallTalk, Ada, COBOL et Java.

· La transparence des invocations : les requêtes aux objets semblent toujours être locales, le bus CORBA se chargeant de les acheminer en utilisant le canal de communication le plus approprié.

· 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 et transparente des objets : les objets sont en mémoire uniquement s'ils sont utilisés par des applications clientes.

· L'interopérabilité entre bus : à partir de la norme CORBA 2.0, un protocole générique de transport des requêtes (GIOP pour General Inter-ORB Protocol) a été défini permettant l'interconnexion de bus CORBA provenant de fournisseurs distincts, une de ses instanciations est l'Internet Inter-ORB Protocol (IIOP) fonctionnant au dessus de TCP/IP.

2.3.2 Les composants :

Le bus CORBA fournit les composantes suivantes :

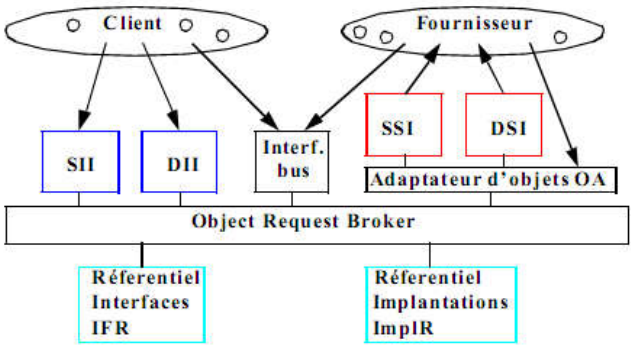

Figure 2.3 : les composant de bus CORBA

1- ORB (Object Request Broker) est le noyau de transport des requêtes aux objets. Il intègre au minimum les protocoles GIOP et IIOP. L'interface au bus fournit les primitives de base comme l'initialisation de l'ORB.

Donc l'ORB est un Middleware qui gère les relations client/serveur entre les objets, il gère :

- La localisation d'objet
- La désignation des objets
- L'empaquetage des paramètres (marshalling)
- Le dépaquetage des paramètres (unmarshalling)
- L'invocation des méthodes

- La gestion des exceptions
- L'activation automatique et transparente des objets

2-SII (Static Invocation Interface) : est l'interface d'invocations statiques permettant de soumettre des requêtes contrôlées à la compilation des programmes. Cette interface est générée à partir de définitions OMG-IDL.

3-DII (Dynamic Invocation Interface) : est l'interface d'invocations dynamiques permettant de construire dynamiquement des requêtes vers n'importe quel objet CORBA sans générer/utiliser une interface SII.

4-IFR (Interface Repository) : est le référentiel des interfaces contenant une représentation des interfaces OMG-IDL accessible par les applications durant l'exécution.

5-SSI (Skeleton Static Interface) : est l'interface de squelettes statiques qui permet à l'implantation des objets de recevoir les requêtes leur étant destinées. Cette interface est générée comme l'interface SII.

6-DSI (Dynamic Skeleton Interface) : est l'interface de squelettes dynamiques qui permet d'intercepter dynamiquement toute requête sans générer une interface SSI. C'est le pendant de DII pour un serveur.

7-OA (Object Adapter) : est l'adaptateur d'objets qui s'occupe de créer les objets CORBA, de maintenir les associations entre objets CORBA et implantations et de réaliser l'activation automatique si nécessaire.

8-ImplR (Implementation Repository) : est le référentiel des implantations qui contient l'information nécessaire à l'activation. Ce référentiel est spécifique à chaque produit CORBA

Ces différentes composantes sont toutes décrites dans le langage OMG-IDL, ce qui les rend accessibles au travers du bus (e.g. le référentiel des interfaces).

2.3.3 Les protocoles réseaux :

Tout bus à la norme CORBA 2.0 doit fournir les protocoles GIOP et IIOP. Le protocole GIOP définit une représentation commune des données (CDR ou Common Data Representation), un format de références d'objet interopérable (IOR ou Interoperable Object Reference) et un ensemble de messages de transport des requêtes aux objets (Request, Reply, …). Cependant, GIOP est seulement un protocole générique, IIOP fournit alors une

implantation de GIOP au dessus de TCP/IP et donc d'Internet. Les IORs dans le contexte d'IIOP doivent contenir :

- Le nom complet de l'interface OMG-IDL de l'objet ;
- L'adresse IP de la machine Internet où est localisé l'objet ;
- Un port IP pour se connecter au serveur de l'objet ;
- Une clé pour désigner l'objet dans le serveur. Son format est libre et il est donc différent pour chaque implantation du bus CORBA.

Toutefois, un bus CORBA peut contenir d'autres protocoles de transport des requêtes aux objets. Par exemple, nous travaillons actuellement sur un bus multi-protocoles permettant d'avoir simultanément des communications en IIOP, UDP-IOP et Multicast-IOP.

2.3 Les avantages et les inconvénients de modèle d'objet CORBA :

2.3.1 Les avantages :

CORBA supporte plusieurs langages. CORBA supporte aussi un mélange de ces langages dans une même application distribuée.

- Modélisation plus naturelle car plus proche du monde réel (en théorie grâce au modèle objet !)
- modèle qui assure la réutilisabilité des composants
- indépendance de localisation des objets clients et serveurs

2.3.2 Les inconvénients :

- absence d'un état global observable (pas de mémoire commune)
- Performance
- Déboguage complexe

Conclusion

Dans cet exposé on a présenté CORBA comme un standard défini par l'Object Management Group (OMG) tel que ce groupe a proposé une architecture globale d'écrite dans l'Object Management Architecture Guide (l'OMA) L'objectif de ce groupe est de faire émerger des standards pour l'intégration d'applications distribuées hétérogènes à partir de technologies ou bien le modèle objet.

CORBA a donc essentiellement été conçu pour résoudre le problème d'hétérogénéité des langages. Il sait exploiter les principales qualités du langage Java : portabilité, sécurité et facilité d'emploi, tout en révolutionnant son système d'invocation à distance. D'autre part, CORBA comporte des services objet communs (CORBAservices) qui fournissent sous forme d'objets CORBA, spécifiés grâce au langage IDL, les fonctions systèmes nécessaires à la plupart des applications réparties. Ils contribuent à assurer l'interopérabilité entre les diverses implémentations et une compatibilité avec beaucoup de technologies existantes.

Donc CORBA est une excellente réponse aux exigences de qualité du génie logiciel :

- réutilisabilité de code existant, intégrité et compatibilité grâce au langage IDL ;
- extensibilité grâce aux services proposés par Corba ;
- portabilité grâce au bus.

Glossaire

CORBA : « Common Object Request Brocker Architecture » est une norme rédigée par l'OMG. Elle définit une architecture distribuée fondée sur le concept d'objet.

OMG : Est un consortium international créé son objectif est de faire émerger des standards pour la distribution d'applications hétérogènes à partir des technologies orientées objets

ODP-RM TINA : Le Consortium **TINA** (**TINA**-C) et le modèle de référence pour le traitement ouvert réparti (**RM-ODP**)
« Object Distributed Processsing - Reference Model » Le but de **RM-ODP** est de proposer un standard permettant de décrire et de réaliser des systèmes distribués pouvant interagir sans pour cela imposer une technologie ou des composants particuliers

OMA : « Object Management Architecture » Cette architecture donne des orientations sur la manière dont la normalisation des interfaces du composant tel que ellel spécifiée des services au sien d'objets CORBA

ORB : Middleware qui gère les relations client/serveur entre les objets.

COBOL : (COmmon Business Oriented Language) est un langage de programmation de troisième génération défini par des normes internationales qui suivent, à chaque révision, l'évolution des techniques de programmation

IDL « Interface Description Language »Est une abstraction d'un type d'objets CORBA

Encapsulation : A travers la projection de IDL en un langage de programmation (objet) spécifique, tout code existant et qui peut etre encapsulé en un objet CORBA.

Marshalling : Un processus qui fait appel permette à un programme de faire appel une fonction sur une autre machine distance tel que il convertis les argument passés en paramètres en un flux d'octets

Unmarshalling : Il convertis le flux d'octets en arguments passés en paramètres.

Intéropérabilite : Les objets CRBA se communiquent par l'intermédiaire du protocole à travers le bus qui support la distinction des langages de programmation