TABLE DES MATIERES

INTRODUCTION GENERALE **8**

PARTIE I : ÉTAT DE L'ART **9**

CHAPITRE I : PRESENTATION DE CORBA **11**

I.Introduction 11
II. Modèle client/serveur de CORBA 11
1.Architecture client/serveur traditionnelle 11
2.Architecture client/serveur à trois tiers 11
3.Architecture distribuée 11
III.L'architecture globale de CORBA 13
1.L'ORB (Object Request Broker) 13
2.L'IDL (*Interface Definition Language*) 14
a.Les fonctionnalités 14
b.Les caractéristiques de l'IDL 17
c.Les limites actuelles 17
d.La projection vers un langage de programmation 18
IV. Les composantes de CORBA 19
V.Les protocoles réseaux 20
VI. Les services objet communs 21
1.La recherche d'objets 21
2.La vie des objets 21
3.La sûreté de fonctionnement 22
4.Les communications asynchrones 22
VII. Les interfaces de domaine 23
VIII. Les limites de CORBA 23

CHAPITRE II : SERVICE D'EVENEMENT **25**

I.Introduction 25
II. La communication par événement 25
III. Principes de conception d'un service d'événement (Design Principles)**....** 26
IV. Qualité de service 27
1.Modules et interfaces 27
a.CosEventComm 27
b.CosEventChannelAdminmodule 27
2.Le canal d'événement 28
3.Le modéle de communication push avec un chanel d'événement. 28
4.Le modèle de communication pull avec un chanel d'événement**...** 29
5.Communication mixte avec un canal d'événement 29
V.Modèle de la plateforme indépendante (PIM) 30
1.Le Package CosLightweightEventComm 30
2.Le Package CosLightweightEventChannel 30

PARTIE II : CONCEPTION **31**

CHAPITRE III : CONCEPTION D'UN SERVICE D'EVENEMENT FIABLE ET TEMPORISE.....33

I.Introduction 33
II. Les limitations des services d'évènement CORBA.....34
1.Contrainte de temps de livraison.....34
2.Contrainte de Fiabilité 35
III. Protocoles de diffusion fiable (*multicast*) 37
1.Définition et Principes de base 37
2.Problème de fiabilité 37
IV. Conclusion.....39

PARTIE III : REALISATION.....40

CHAPITRE IV : PARTIE PRATIQUE.....42

I.Introduction 42
II. Architecture TAO.....42
III. Prise en main de l'environnement de développement 43
1.Installation de TAO pour l'application Serveur 44
a.Modification de la variable PATH 44
b.Création du fichier config.h 45
c.Compilation de TAO 45
2.Installation de JacORB pour l'application Cliente.....45
a.Installation de Ant.....45
b.Modification de la variable PATH 45
c.Installation de JacORB 45
d.Génération du script de lancement des applications Java avec JacORB.....46
3.Génération du compilateur IDL.....46
IV. Les différentes étapes pour la mise en place de notre prototype 47
V.Conclusion.....48

CONCLUSION & PERSPECTIVES.....50

BIBLIOGRAPHIE & NETOGRAPHIE 52

TABLE DES FIGURES

FIGURE 1 : LE MODELE CLIENT/SERVEUR. 12
FIGURE 2 : ARCHITECTURE GLOBALE DE CORBA. 13
FIGURE 3 : LA NOTION DE CONTRAT IDL 14
FIGURE 4 : LES TYPES DE META-DONNEES 15
FIGURE 5 : PRE-COMPILATION DU CONTRAT IDL VERS DEUX LANGAGES DIFFERENTS 19
FIGURE 6 : LES DIFFERENTES COMPOSANTES DE CORBA 20
FIGURE 7 : MODELE PUSH 25
FIGURE 8 : MODELE PULL 26
FIGURE 9 : MODELE DE COMMUNICATION *EVENT CHANNEL (PUSH)* 28
FIGURE 10 : MODELE DE COMMUNICATION *EVENT CHANNEL (PULL)* 29
FIGURE 11 : MODELE DE COMMUNICATION *EVENT CHANNEL* (COMMUNICATION MIXTE) 29
FIGURE 12 : COMPOSANTES D'UN SERVICE D'EVENEMENT CORBA 33
FIGURE 13 : MODELE DE COMMUNICATION *PUSH* 34
FIGURE 14 : PRINCIPE DE LA COMMUNICATION *MULTICAST* 37
FIGURE 15 : LES DIFFERENTS FACTEURS D'ECHELLE 38

INTRODUCTION GENERALE

INTRODUCTION GENERALE

En général l'informatique répartie présente un défi majeur de l'informatique actuelle et à venir. Pour faire face au défi on a besoin de développer les applications reparties de manière simple tout en limitant les coûts.
Dans le monde CORBA, l'invocation standard d'une méthode consiste en une exécution synchrone d'une opération fournie par un objet. Cependant la plupart des applications reparties nécessitent un modèle de communication fortement découplé entre les objets communiquant. Ce besoin de découplage entre les objets a conduit L'OMG (l'Object Management Group) à définir un nouveau type d'interfaces qui assure la communication asynchrone entre objets, ces interfaces sont intitulées les services d'évènements qui constituent un des principaux dispositifs d'interaction asynchrone dans les systèmes répartis.
Un service d'évènement comprend des producteurs d'évènements, des consommateurs d'évènements, et des canaux d'évènement.
Le principe de fonctionnement du service d'évènements est fondé sur un canal d'évènements dans lequel des producteurs d'évènement émettent des données tandis que des consommateurs reçoivent ces données. Le modèle de communication se base sur l'architecture Client /Serveur, Le serveur va jouer le rôle du producteur tandis que le client va jouer le rôle de consommateur. Les producteurs peuvent générer des évènements sans avoir connaître les identités des consommateurs. De même, les consommateurs peuvent recevoir des évènements sans avoir connaître les identités des producteurs.
Pour initier la communication d'évènements il y a deux approches ***push*** et ***pull***
Dans le modèle ***push*** le producteur prend l'initiative de transférer des données évènements vers le consommateur, dans l'autre modèle le consommateur prend l'initiative de solliciter des données évènements du producteur.
Cependant, les spécifications actuelles de ces services ne permettent pas d'exprimer des contraintes temporelles et des contraintes de fiabilité dans l'envoi des évènements, Considérant ces lacunes nous proposons dans ce mémoire une approche pour la prise en compte de tels contraintes dans l'envoie des évènements.

Plan du mémoire : Le mémoire est organisé en trois grandes parties. La première
Constitue un état d'art sur la norme CORBA et les services d'évènements, puis une étude un peu plus approfondie sur la conception des services d'évènement.
Dans la deuxième partie, on présente le problème lié aux spécifications actuelles des services d'évènement et nous présentation une solution qui tient en compte l'intégration des contraintes temporelles et des contraintes de fiabilité dans l'envoi des évènements, et on termine cette partie par une présentation sur les protocoles de diffusion fiable (multicast),qui sont utiliser pour la diffusion d'information, les mises à jour logiciels, l'intégration des applications réparties et les applications interactives en réseau.
La troisième partie est dédiée aux expérimentation, qui prend en charge une implémentation de la norme CORBA (TAO) gratuit et open source et supporte les contrainte temps réel.
Puis nous avons préparé l'environnement de programmation afin de montrer une démonstration de TAO et de réaliser notre prototype « un service d'évènement du type CORBA fonctionnant sur un réseau local IP, et qui prend en compte des contraintes de temps et de fiabilité ».

Partie I

État de l’art

Présentation de CORBA

CHAPITRE 1

Chapitre I : Présentation de CORBA

I. Introduction

L'*Object Management Group* (OMG) est un consortium international créé en 1989 et regroupant actuellement plus de 850 acteurs du monde informatique : des constructeurs, des producteurs de logiciel des utilisateurs et des institutionnels et universités. L'objectif de ce groupe est de faire émerger des standards pour l'intégration d'applications distribuées hétérogènes à partir des technologies orientées objet. Ainsi les concepts-clés visé sont la l'interopérabilité, réutilisabilité, et la portabilité de composants logiciels, programmés dans différents langages.
Le cœur de la vision de l'OMG est CORBA (*Common Object Request Broker Architecture*) : qui est un *middleware* orienté objet et repose sur le protocole TCP / IP.
L'idée consiste à créer des objets qui seront accessibles par le client et le serveur.
CORBA propose un support d'exécution masquant les couches techniques d'un système réparti et il prend en charge les communications entre les composants logiciels formant les applications réparties et hétérogènes.

II. Modèle client/serveur de CORBA

1. Architecture client/serveur traditionnelle

L'application cliente est située sur le poste client. Le lien entre le serveur et le client est direct. L'application cliente accède aux données de la base via des requêtes SQL. Peu de traitements sont localisés au niveau du serveur.

2. Architecture client/serveur à trois tiers

Le client n'accède pas directement au serveur. Il émet des requêtes au serveur d'application qui exécute les traitements et à son tour faire transmettre les requêtes au serveur de données.
Cette architecture ne repose pas sur le concept d'objet.

3. Architecture distribuée

Dans cette architecture l'ensemble des dialogues entre machines est pris en charge par le middleware (dans notre cas c'est CORBA). Il prend en charge tous les problèmes de communication liés à l'intégration des différentes plates-formes utilisées (langages distincts, machines distantes, environnements hétérogènes)
CORBA propose donc un modèle client/serveur de coopération entre des objets répartis, chaque application peut exporter certains services sous forme d'objet *CORBA*, cette notion de client serveur intervient uniquement lors de l'utilisation de l'objet.
Les interactions entre applications sont faites à travers les invocations de procédures distantes.
L'application utilisant un objet est un client, et celle en attente des requêtes est le Serveur tandis que la partie du serveur implantant l'objet est appelée le servant.

La coopération client-serveur se déroule de la manière suivante :

- Le client détient une référence sur un objet CORBA qui permet de le localiser sur le bus CORBA.
- Le client dispose de l'interface de l'objet CORBA qui définit ses opérations et ses attributs exprimés dans le langage IDL (voir ci-dessous).
- Le client invoque une requête sur l'objet CORBA.
- Le bus CORBA achemine cette requête vers l'objet CORBA tout en masquant les problèmes d'hétérogénéité liés aux systèmes d'exploitation, langages, machines, réseaux.
- L'objet CORBA est associé à un objet d'implantation
- Le serveur détient l'objet d'implantation qui code l'objet CORBA et gère son état temporaire.

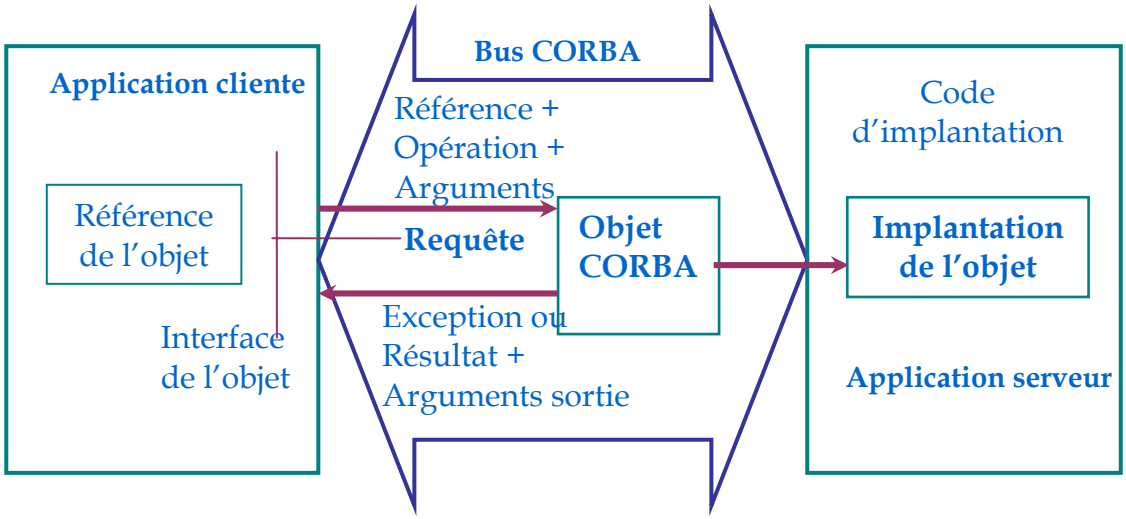


Figure 1 : Le modèle client/serveur.

Les composantes de la figure1 sont :

- La référence d'objet est une structure désignant l'objet et contenant l'information nécessaire pour le localiser sur le bus CORBA.
- L'interface de l'objet est le type abstrait de l'objet, elle se définit par l'intermédiaire du langage OMG-IDL,
- Le bus CORBA gère les transferts entre les clients et les serveurs en masquant tous les problèmes d'hétérogénéité,
- La requête est le mécanisme d'invocation d'une opération ou d'accès à un attribut de l'objet,
- L'objet CORBA est le composant logiciel cible. C'est une entité virtuelle gérée par le bus CORBA,
- L'activation est le processus d'association d'un objet d'implantation à un objet CORBA,
- Le code d'implantation rassemble les traitements associés à l'implantation des opérations de l'objet.

III. L'architecture globale de CORBA

L'OMG a défini une architecture globale pour classifier les différents objets qui participent dans la construction d'applications réparties. Cette architecture comprend un bus d'objets répartis : l'*ORB* (voir ci-dessous) qui est l'intermédiaire à travers lequel les objets vont pouvoir dialoguer. Des services de base (*CorbaServices*) qui fournissent les fonctions nécessaires à la plupart des applications réparties. Ces services sont spécifiés grâce au langage OMG-IDL (voir ci-dessous). Des utilitaires communs (*Common Facilities*) qui répondent plus particulièrement aux besoins des utilisateurs. Des interfaces d'objets applicatifs (*Application Interfaces*) qui définissent les objets créés par les utilisateurs. Des interfaces de domaines (*Domain Interfaces*) qui définissent des interfaces spécialisées répondant aux besoins spécifiques de tel ou tel marché comme la finance ou la santé.
La figure 2 illustre cette architecture.

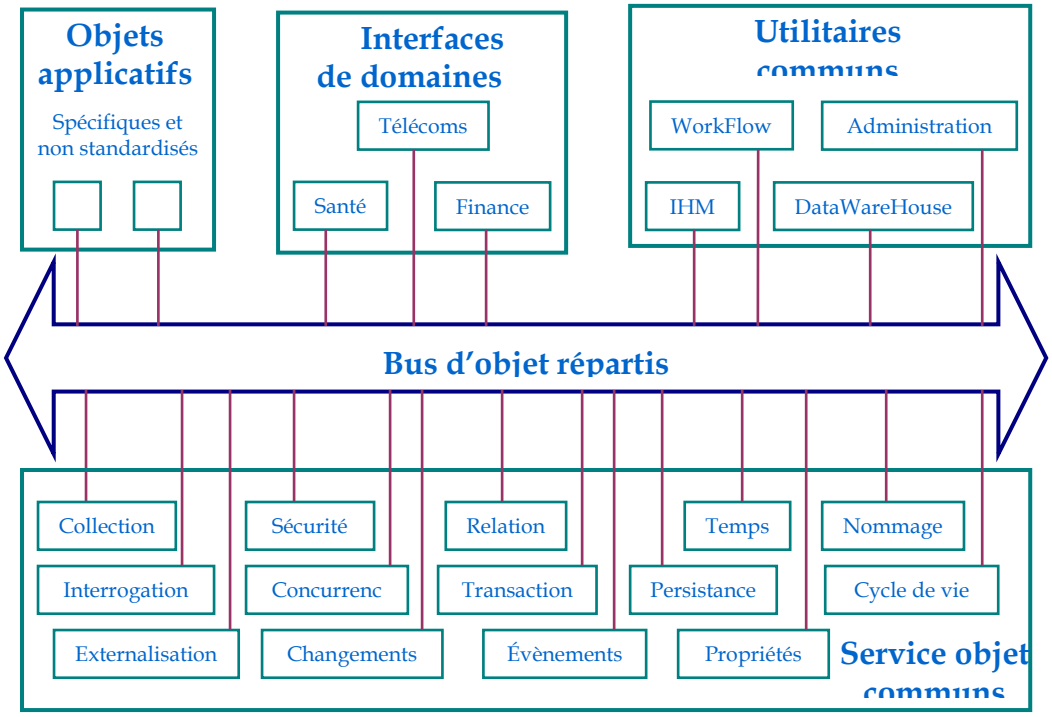


Figure 2 : Architecture globale de CORBA.

1. L'ORB (Object Request Broker)

L'ORB (*Object Request Broker*) est une entité (en réalité en ensemble des interfaces) qui fournit des mécanismes d'interrogations permettant de récupérer des objets, des procédures qui constituent une application répartie. Donc c'est l'intermédiaire/négociateur à travers lequel les objets vont pouvoir dialoguer et ce de manière transparent.
Le noyau de communication (*ORB Core*) transporte les requêtes du client vers l'implantation de l'objet cible, selon diverses techniques dépendantes de la localisation. Si les objets sont situés sur des machines différentes, L'*ORB Core* utilise le protocole IIOP (*Internet Inter-ORB Protocol*) et si les objets sont sur le même site il utilise des outils de communication interprocessus. Ainsi l'*ORB* est responsable de trouver l'implantation de l'objet, de l'activer,

de délivrer la requête à l'objet et de retourner la réponse à l'appelant. Ce noyau n'est pas directement accessible par les programmeurs.

2. L'IDL (*Interface Definition Language*)

IDL est un langage de définition d'interface orienté objet. Il a été créé pour fournir l'interopérabilité entre plusieurs applications répartir, flexibles écrites dans différents langage de programmation, Il cherche à masquer l'hétérogénéité de ces applications et il définit les types des objets en spécifiant leurs interfaces. Cette spécification des applications répartie flexibles sur des plateformes hétérogènes nécessite une séparation stricte entre l'interface de l'objet et de son implémentation.
D'où l'idée de créer un langage objet : OMG-IDL qui ne décrit que les interfaces des objets, et ceci de manière indépendante du langage d'implantation. L'interface d'un objet contient les éléments suivent :

- Les opérations,
- Les paramètres,
- Les noms des méthodes,
- Le type de retour,
- Les attributs,
- Les types et les exceptions.

Du côté client, on a uniquement l'interface de l'objet.
Du côté serveur, on a l'interface des objets et on a aussi l'implantation dans des langages de programmation tels que (C, C++, Smalltalk, Java, Cobol)
L'interface et l'implantation sont liées par l'intermédiaire du bus CORBA.
Un compilateur IDL génère des *stubs* (les souches) client et des *skeletons* (squelettes) serveurs. Le client invoque localement les *stubs* pour accéder aux objets. Les *stubs* IDL créent des requêtes, qui vont être transportées par le bus CORBA, puis délivrées par celui-ci aux *skeletons* IDL qui les délégueront aux objets.

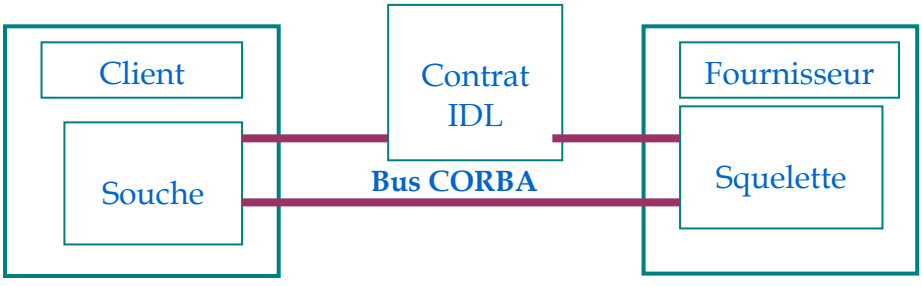


Figure 3 : La notion de contrat IDL

a. Les fonctionnalités

La liste suivante présente les constructions offertes par le langage OMG-IDL et leurs utilités :

- Un **pragma** permet de fixer les identifiants désignant les définitions OMG-IDL à l'intérieur du référentiel des interfaces. Ces identifiants assurent une désignation unique et cohérente des définitions OMG-IDL quel que soit le référentiel des interfaces consulté, i.e. ils sont utilisés pour fédérer des IFRs.

La forme la plus utilisée est le pragma prefix qui fixe l'organisation définissant un contrat IDL en utilisant son nom de domaine Internet. Le pragma version permet de définir le numéro de version d'une spécification OMG-IDL.

- Un **module** sert à regrouper des définitions de types qui ont un intérêt commun. Cette structuration permet aussi de limiter les conflits de noms pouvant intervenir entre plusieurs spécifications, il est facilement imaginable que le nom Convertisseur soit utilisé dans une autre spécification. Les conflits de nom de module sont facilement réglés par le pragma prefix.
- Les **types de données de base** sont ceux couramment rencontrés en informatique :
  - void, short, unsigned short, long, unsigned long, long long (64 bits),
  - unsigned long long, float, double, long double (128 bits),
  - boolean, octet, char, string, wchar et wstring (format de caractères international) et fixed pour les nombres à précision fixe.
- Les **types de méta-données** (TypeCode et any) sont une composante spécifique à l'OMG-IDL et nouvelle par rapport à d'autres langages IDL. Le type TypeCode permet de stocker la description de n'importe quel type IDL. Le type any permet de stocker une valeur IDL de n'importe quel type en conservant son TypeCode. Ces méta-types permettent de spécifier des contrats IDL génériques indépendants des types de données manipulés, e.g. une pile d'any stocke n'importe quelle valeur.

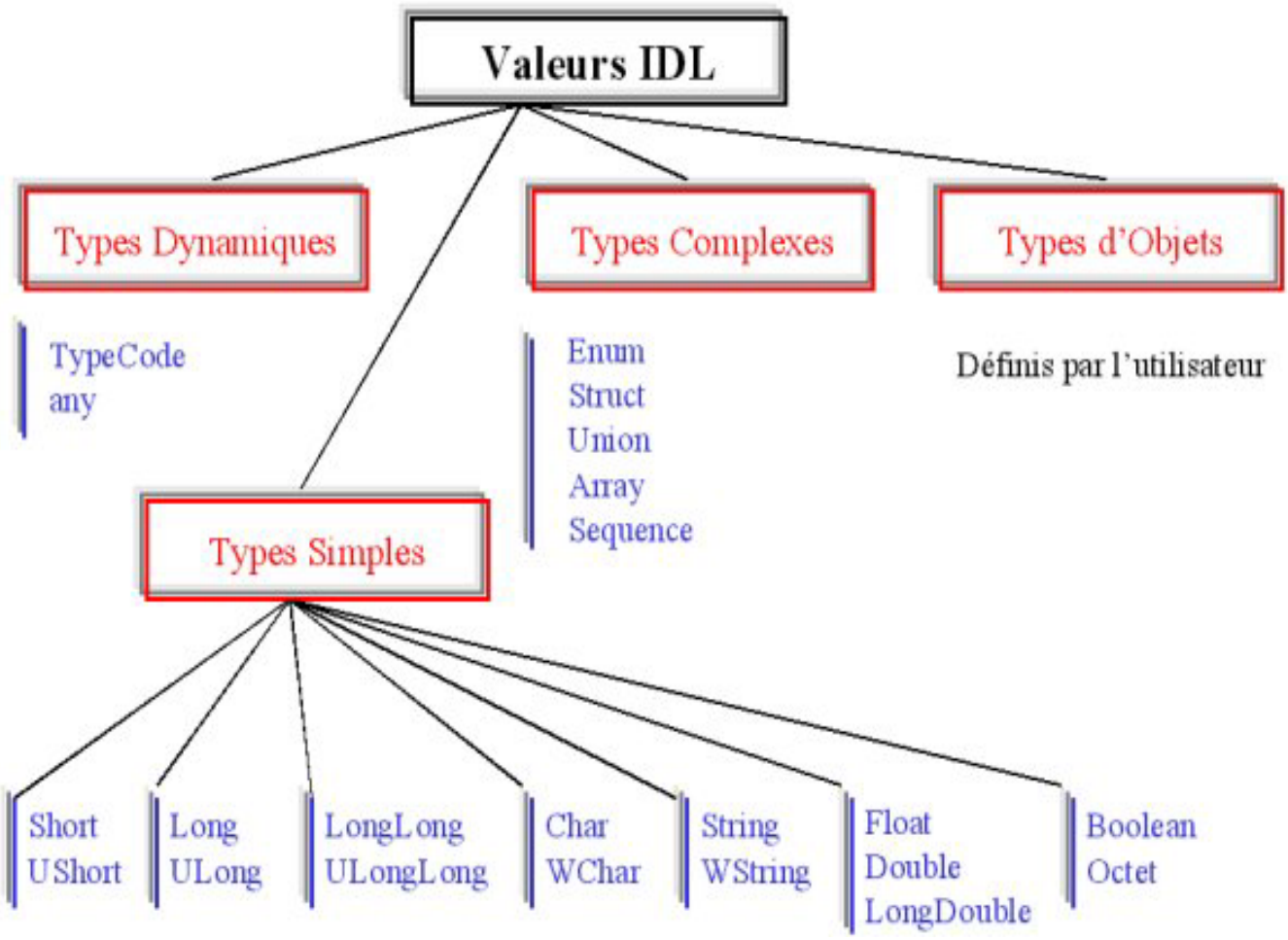


Figure 4 : Les types de méta-données

- Une **constante** se définit par un type simple, un nom et une valeur évaluable à la compilation (exemple : const double PI = 3.14),
- Un **alias** 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 plutôt qu'un entier 16 bits signer,
- Une **énumération** définit un type discret via un ensemble d'identificateurs. Par exemple, il est plus parlant d'énumérer les jours de la semaine que d'utiliser un entier pour coder un Jour_Dans_La_Semaine,
- Une **structure** définit une fiche regroupant des champs. Cette construction est fortement employée car elle permet de transférer des structures de données composées entre objets CORBA,
- Une **union** juxtapose un ensemble de champs, le choix étant arbitré par un discriminant de type simple. Toutefois, cette construction est très rarement utilisée et on préférera plutôt utiliser le type any,
- Un **tableau** sert à transmettre un ensemble de taille fixe de données homogènes. Mais cette construction est rarement utilisée car on lui préfère la suivante,
- Une **séquence** 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,
- 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 ou de plusieurs champs,
- Une **interface** décrit les opérations fournies par un type d'objets CORBA. Une interface IDL peut hériter de plusieurs autres interfaces. L'héritage multiple et répété est autorisé car ici on parle seulement d'héritage de spécifications. La seule contrainte imposée est qu'une interface ne peut pas hériter de deux interfaces qui définissent un même nom d'opérations. Ce problème doit être résolu manuellement en évitant/éliminant les conflits de noms. La surcharge est donc aussi interdite en OMG-IDL. Actuellement, un objet CORBA ne peut implanter qu'une seule interface IDL. CORBA 3.0 intégrera la notion d'interfaces multiples comme cela existe déjà en Java ou avec COM,
- 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 paramétre se caractérise par un mode de passage, un type et un nom formel. Les modes de passages autorisés sont in, out et inout. Le résultat et les paramètres peuvent être de n'importe quel type exprimable en IDL. Par défaut, l'invocation d'une opération est synchrone. Cependant, il est possible de spécifier qu'une opération est asynchrone (oneway), c'est-à-dire que le résultat est de type void, que tous les paramètres sont en mode in et qu'aucune exception ne peut être déclenchée. Malheureusement, CORBA ne spécifie pas la sémantique d'exé ion d'une opération oneway : l'invocation peut échouer, être exécutée plusieurs fois sans que l'appelant ou l'appelé puissent en être avertis. Toutefois, dans la majorité des implantations CORBA, l'invocation d'une telle opération

équivaut à un envoi fiabilisé de messages. De plus, l'OMG travaille sur les spécifications d'un service pour les communications asynchrones (CORBA Messaging),

- 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). Il faut tout de même noter que le terme IDL attribut est trompeur, l'implantation d'un attribut IDL peut être faite par un traitement quelconque, par exemple : l'implantation de l'attribut temperature d'un Thermometre consultera un capteur physique tandis que celle de l'attribut annee_courante d'une Date sera plutôt réalisée par une variable d'état de l'objet.

b. Les caractéristiques de l'IDL

Le Language IDL est caractérisé par :

- La gestion de l'hétérogénéité des différentes applications (en masquant les systèmes d'exploitation, réseaux, langages…),
- La réutilisabilité du code existant,
- La portabilité des applications d'un bus CORBA vers l'autre (interopérabilité).

c. Les limites actuelles

Bien que le langage OMG-IDL soit simple et assez complet, il est actuellement limité sur certains points on cite à titre d'exemple:

- Les types de données de base sont limités en nombre et l'utilisateur de CORBA ne peut pas en ajouter facilement. Cependant, l'OMG ne s'interdit pas d'introduire de nouveaux types au fur et à mesure des besoins,
- L'impossibilité de spécifier des types intervalles qui seraient tout de même pratique pour exprimer l'heure (Secondes, Minutes, Heures),
- L'impossibilité de d'étendre la définition d'une union ou d'une structure, cela peut être gênant pour la réutilisation et la spécialisation de spécification IDL. Par exemple, il est impossible d'enrichir la structure *Date* dans le contexte d'un nouveau service de dates,
- L'interdiction de conflits de noms à l'intérieur d'un module ou d'une interface impliquant l'interdiction de surcharger des opérations, c'est-à-dire définir deux opérations ayant le même nom mais des signatures différentes.
- Le passage par valeur de structures de données et non d'objets. Par exemple, si l'on veut transférer un graphe d'objets, il faut l'aplatir dans une séquence de structures. Il serait préférable comme avec Java/RMI de pouvoir passer l'objet graphe par valeur. L'OMG travaille actuellement sur une extension de l'OMG-IDL pour exprimer le passage d'objets par valeur, cela sera disponible dans CORBA 3.0,
- Toutefois, l'OMG est un lieu de standardisation très actif et donc si les limites soulevées ci-dessus deviennent à l'avenir des inconvénients majeurs au développement de CORBA alors on peut être sûr que de nouvelles spécifications viendront améliorer le langage OMG-IDL. D'autres limites peuvent encore être exprimées mais celles-ci nécessiteront de nouveaux langages **:**
    - Le langage OMG-IDL ne sert pas à exprimer la sémantique des objets comme des contraintes, des pré et post conditions sur les

opérations. L'expression de la sémantique permettrait de tester et valider automatiquement les objets.

- La description de la qualité de service, c'est-à-dire les caractéristiques d'une implantation, Pour exprimer les contraintes de temps et de fiabilité et les prendre en compte dans les canaux d'évènement ne sont pas exprimables avec le langage OMG-IDL.
- La répartition et les liaisons entre les objets, c'est-à-dire l'architecture logicielle d'une application, ne s'exprime pas non plus *via* le langage OMG-IDL. l'OMG travaille aussi sur un modèle de composants [OMG-MC] qui devrait palier ce problème.
- Pour conclure, on peut considérer que le langage OMG-IDL est un langage pivot entre les applications. Et pour chaque partie de l'application il permet de choisir le Language le plus approprié.

d. 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. Toutefois, voici quelques exemples de règles de projection :

Construction OMG-IDL	**Projection en C++**	**Projection en Java**
module C { … };	Au choix du produit CORBA : namespace C { … } ou class C { … } ou préfixe C	`package C`
`interface J:I {…};`	class J : public virtual I {…};	`interface J extends I {…}`
`String`	`char *`	`java.lang.String`

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.

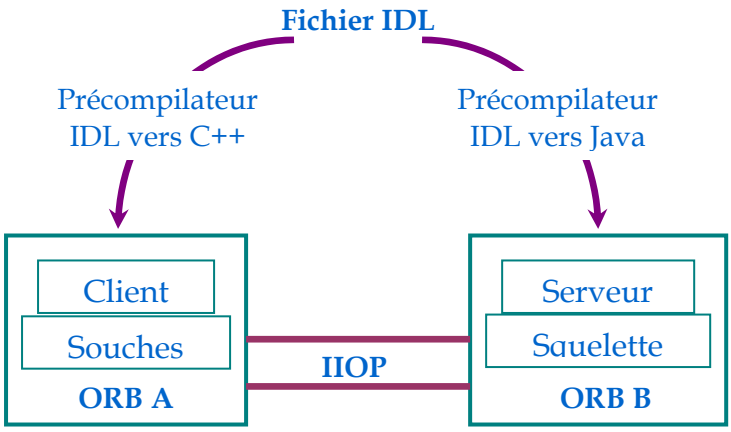


Figure 5 : Pré-compilation du contrat IDL vers deux langages différents

IV. Les composantes de CORBA

Nous pouvons distinguer les composantes du bus CORBA suivantes (voir figure 6) :

- **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,
- **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,
- **IFR (*Interface Repository*) :** est le référentiel des interfaces contenant une représentation des interfaces OMGIDL accessible par les applications durant l'exécution,
- **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,
- **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,
- **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,
- **ImplR (*Implementation Repository*) :** est le référentiel des implantations qui contient l'information nécessaire à l'activation. Ce référentiel est spécifique à chaque produit CORBA.

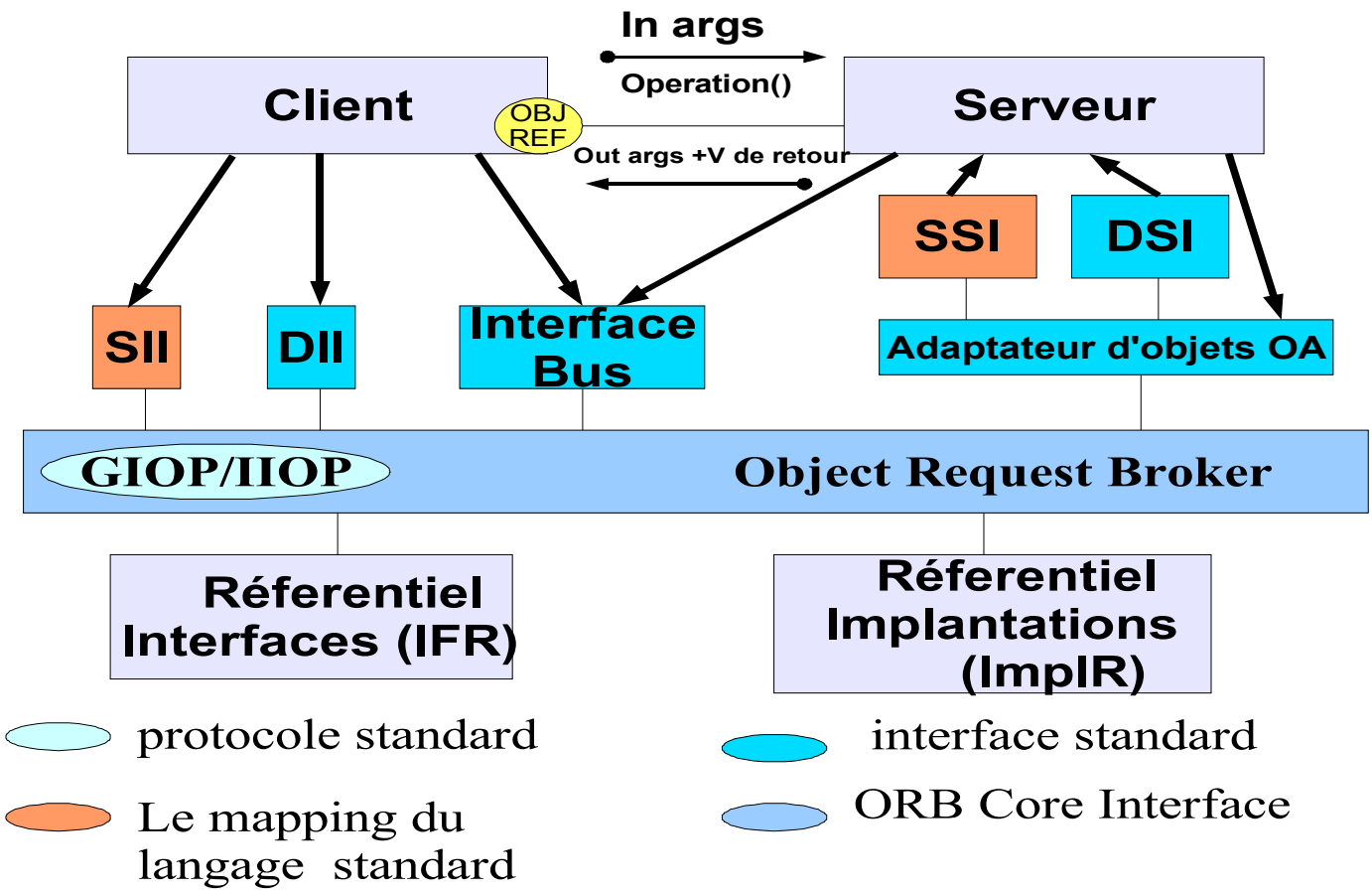


Figure 6 : Les différentes composantes de CORBA

V. Les protocoles réseaux

A partir de la norme CORBA 2.0 le bus CORBA doit fournir les protocoles GIOP et IIOP.
Le protocole GIOP pour General Inter-ORB Protocol est un protocole générique de transport des requête définit une représentation commune des données (CDR ou Common Data Representation), un format de références d'objet interopérable (IOR ou Interoperable Object Reference) et qui permet l'interconnexion de bus CORBA produit par de fournisseur distincts.
L'**IIOP** est basé sur TCP/IP et correspond à un protocole d'échange, c'est une instanciation du protocole **GIOP** et spécifie la manière dont les messages **GIOP** sont échangés au-dessus de TCP/IP.
Dans le contexte d'IIOP, les ***IORs*** 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 dépend de l'implantation du bus CORBA.

VI. Les services objet communs

Les services objet communs sont ensemble de services de niveau système formaté comme des objets avec une interface spécifiée en IDL et ont pour objectif de standardiser les interfaces des fonctions système nécessaires à la construction et l'exécution de la plupart des applications réparties. Un service est caractérisé par les interfaces qu'il fournit et par les objets qui fournissent ces interfaces.

1. La recherche d'objets

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 de Nommage pour *Naming Service* : les objets sont désignés par des noms symboliques.ce service est l'équivalent des "*pages blanches*",
- Le service Vendeur pour Trader Service : en utilisant ce service les objets peuvent être recherchés en fonction de leurs caractéristiques.
  Ce servie est l'équivalent des "*pages jaunes*".

2. La vie des objets

Cette catégorie regroupe les services prenant en charge les différentes étapes de la vie des objets CORBA.

- Le service Cycle de Vie pour Life Cycle Service décrit des interfaces pour la création, la copie, le déplacement et la destruction des objets sur le bus. Il définit et utilise la notion de fabriques d'objets Object Factory pour faire cala,
- Le service Propriétés pour Property Service permet aux utilisateurs d'associer dynamiquement des valeurs nommées à des objets. Ces propriétés ne modifient pas l'interface de l'IDL, mais représentent des besoins caractéristiques du client,
- Le service Relations pour Relationship Service sert à gérer des associations dynamiques (appartenance, inclusion, référence, emploi,...) reliant des objets sur le bus CORBA. Il permet aussi de manipuler des graphes d'objets,
- Le service Externalisation pour 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 pour 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 PersistentObject et d'un mécanisme d'externalisation,
- Le Service Interrogations pour *Query Service* permet d'interroger les attributs des objets. Il repose sur les langages standards 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 Collections pour *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,..) 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 pour Versionning Service permet de suivre et de gérer 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 en cours d'être spécifié officiellement.

3. La sûreté de fonctionnement

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

Le service **Sécurité** pour Security Service permet d'authentifier et d'identifier les clients, de crypter et de certifier les communications et de contrôler les autorisations d'accès. Ce service utilise les notions de serveurs d'authentification, et délégation de droits, d'IIOP sécurisé qui implémente Kerberos ou SSL.

- Le service Transactions pour Object Transaction Service assure l'exécution de traitements transactionnels impliquant des objets distribués et des bases de données,
- Le service Concurrence pour Concurrency Service fournit les mécanismes pour ordonnancer et contrôler les invocations concurrentes sur les objets. Le mécanisme proposé pour gérer le problème de concurrence est le verrou. Ce service est conçu pour être utilisé parallèlement avec le service Transactions.

4. Les communications asynchrones

Par défaut, la communication entre les objets CORBA est réalisée selon un mode de communication client serveur synchrone. Cependant, pour assurer des communications asynchrones entre des objets CORBA, il existe un ensemble de services spécifique pour assurer ce type de communication.

- Le service Evénements ou 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 Pull et Push. Dans le mode push, le producteur à l'initiative de la production et le consommateur est notifié des événements produits par le producteur. Dans le mode pull, le consommateur demande explicitement les événements et dans ce cas le producteur est alors sollicité.
- Le service Notification ou Notification Service est une extension du service d'événement. Les consommateurs sont uniquement notifiés des événements les intéressant. Pour cela, ils posent des filtres sur le canal d'événement réduisant ainsi le trafic réseau produit par la propagation des événements inintéressants.
- Le service Messagerie (CORBA Messaging) définit un nouveau modèle de communication asynchrone permettant de gérer des requêtes persistantes lorsque l'objet appelant et l'objet appelé n'est pas présent simultanément sur le bus CORBA.
- Le service Temps pour Time Service fournit des interfaces permettant d'obtenir une horloge globale sur le bus CORBA, de mesurer le temps et de synchroniser les objets CORBA.
- Le service Licences pour Licensing Service permet de mesurer et de contrôler l'utilisation des objets.

VII. Les interfaces de domaine

Les interfaces de domaines définissent des interfaces spécialisées répondant aux besoins spécifiques de tel ou tel marché comme la santé ou la finance par exemple.
Leurs caractéristiques sont :

- Spécification d'interfaces IDL,
- Standardisées par l'OMG,
- Leurs fonctionnalités peuvent être étendues ou spécialisées par héritage.

VIII. Les limites de CORBA

CORBA facilite énormément le développement d'applications reparties. Il constitue une Solution pour les applications dont le seul critère est le Best effort. Cependant il ne convient pas aux applications temps réel et de haute performance pour les raisons suivantes :

- Manque d'interfaces pour spécifier la QoS : il n'y a pas d'interface pour la spécification des contraintes de la QoS de bout en bout. Par exemple, CORBA ne dit pas comment les clients indiquent à l'ORB la périodicité de certaines opérations,
- Pas de spécification de la politique du contrôle d'admission : un serveur vidéo peut préférer utiliser la bande passante disponible pour servir un nombre limite de clients et refuser tout nouveau client, plutôt que d'accepter tous les nouveaux clients et dégrader la qualité de la vidéo,
- Le non respect de la QoS : CORBA ne garantit pas la QoS de bout en bout. Un ORB qui travaille selon la technique FIFO peut bloquer l'exécution des requêtes Prioritaires au profit des requêtes non prioritaires. Il ne permet même pas de spécifier La priorité d'une requête de plus, les spécifications CORBA n'imposent pas `a l'ORB de notifier les clients quand les ressources du réseau et l'ORB distant ne sont pas disponibles momentanément. Vu l'incapacité des implémentation actuels à satisfaire les applications très exigeantes en terme de contrainte de temps et de fiabilités,
- Comme l'architecture CORBA manque de mécanismes lui permettant de prendre en compte les contraintes temporelles liées au temps réel. Cependant, la modification seule du fonctionnement interne de l'architecture ne suffit pas pour remédier ce problème.
- Notre choix est porté sur une implémentation de CORBA qui tient en compte la notion de temps réel et facilite la programmation réseaux (aspect qualité de service).cette implémentation sera détaillé dans la partie pratique (Chapitre).

CHAPITRE 2

Service d'Évènement

Chapitre II : Service d'évènement

I. Introduction

Dans CORBA, l'invocation standard d'une méthode consiste en une exécution synchrone d'une opération fournie par un objet. Si l'opération définit des paramètres ou des valeurs de retour, des données sont communiquées entre le client et le serveur
Pour que la requête soit réussie, le client et le serveur doivent être disponibles. Si une requête échoue parce que le serveur est indisponible, le client reçoit une exception et doit prendre de mesure appropriée. Pour la plupart des applications, un modèle de communication plus découplé entre objets est exigé.
Pour cette raison l'OMG a défini une interface *event service* qui permet la communication asynchrone entre objets.
Le modèle de l'OMG est basé sur le modèle producteur/consommateur.

II. La communication par événement

Le service d'événement découple la communication entre les objets.
Il définit deux rôles pour les objets: producteur et consommateur. Les producteurs fournissent les données évènement, les consommateurs consomment ces données. La communication de ces données entre producteurs et consommateurs se fait par les requêtes CORBA standards
Il y a deux approches pour initier la communication d'événement entre les producteurs et les consommateurs,
Les deux approches pour initier la communication d'événement s'appellent le modèle Push et le modèle Pull.

- **Le modèle push:** il permet à un producteur d'initier le transfert des données évènements vers le consommateur. Pour établir une communication du modèle push, les consommateurs et les producteurs échangent les références objet de PushConsumer et de PushSupplier. La communication d'événement peut être terminée en appelant une opération de disconnect_push_consumer sur l'interface de PushConsumer ou en appelant une opération disconnect_push_supplier sur l'interface de PushSupplier. Si la référence d'objet de PushSupplier est zéro, le raccordement ne peut pas être cassé par l'intermédiaire du producteur. La figure 7 illustre la communication de *push-modèle* entre un producteur et un consommateur.

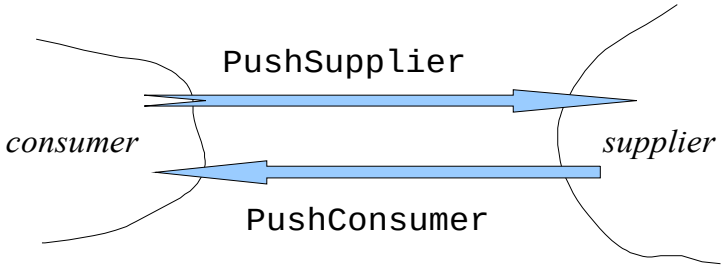


Figure 7 : Modèle push

- **Le modèle pull:** il permet à un consommateur de solliciter des données évènements d'un producteur. Pour établir une communication du *modèle push*, les consommateurs et les producteurs échangent les références objet de *PullConsumer* et de *PullSupplier*.

La communication d'événement peut être terminée en appelant une opération de *disconnect_pull_consumer* sur l'interface de *PullConsumer* ou en appelant une opération *disconnect_pull_supplier* sur l'interface de *PullSupplier*. Si la référence d'objet de *PullConsumerr* est zéro, le raccordement ne peut pas être cassé par l'intermédiaire du producteur.

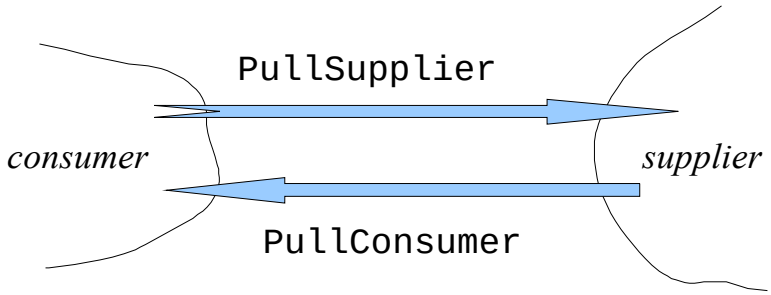


Figure 8 : Modèle pull

La figure 8 illustre la communication de *pull-modèle* entre un producteur et un consommateur.

Dans le *modèle push* le producteur a donc l'initiative et dans le modèle *pull* le consommateur a l'initiative.

- **La communication** peut être générique ou typée:
  - **Dans le cas d'une communication générique:** toute communication se fait au moyen d'opérations *push* et *pull* génériques qui prennent un seul paramètre englobant toutes les données évènements.
  - **Dans le cas d'une communication typée:** toute communication se fait via des opérations définies en IDL.

Ce service définit également des objets *event channel*. Un *event channel* est un objet qui permet à de multiples producteurs de communiquer avec de multiples consommateurs d'une manière asynchrone. Un *event channel* est aussi à la fois un producteur et un consommateur d'évènements; c'est un objet CORBA standard et donc la communication avec un *event channel* se fait par les requêtes CORBA standards.

Le service d'événement peut être utilisé pour fournir l'état de changement d'un objet Quand un objet est changé (son état est modifié), on peut produire un événement qui est propagé à tous les objets relié avec ce dernier.

III. Principes de conception d'un service d'événement (Design Principles)

La conception de service d'événement satisfait les principes suivants

- Les événements fonctionnent dans un environnement distribué. La conception ne dépend pas du service global, critique, ou centralisé,
- Les services d'événement permettent le couplage de multiples consommateurs d'un événement et de multiples fournisseurs d'événement,
- Des consommateurs peuvent demander des événements ou être avisés des événements, ce qui est plus approprié pour la conception et l'exécution d'application
- Les consommateurs et les producteurs des événements soutiennent les interfaces standards d'OMG IDL ; aucun prolongement à CORBA n'est nécessaire pour définir ces interfaces,
- Un fournisseur peut publier une seul simple requête standard pour communiquer des données d'événement à tous les consommateurs,

- ☞ Les fournisseurs peuvent produire des événements sans savoir les identités des consommateurs. Réciproquement, les consommateurs peuvent recevoir des événements sans savoir les identités des fournisseurs,
- ☞ Les interfaces de service d'événement autorisent différentes qualités de service, pour différents niveaux de fiabilité et elle tient compte également des extensions futures pour de fonctionnalité additionnelle.

IV. Qualité de service

Les domaines d'application exigeant la communication de type événement nécessite des conditions diverses sur la fiabilité et dans la plupart de cas elle utilise la sémantique (*best effort*), formellement aucune implémentation du service d'événement ne peut optimiser une gamme diverse des impératifs techniques. Par conséquent, des réalisations multiples des services d'événement doivent être prévues, avec différents services visés pour différents environnements.

1. Modules et interfaces

Les interfaces de services communs d'objet d'OMG (COS) sont définies en utilisant le langage OMG-IDL. Les deux composants primaires dans l'architecture de service d'événements sont le module de *CosEventComm* et le *CosEventChannelAdminmodule*.

a. CosEventComm

Les modèles de communication montrés ci-dessus (*pull* et *push*) sont soutenus par quatre interfaces simples : *PushConsumer*, *PushSupplier*, et *PullSupplier* et *PullConsumer*. Ces interfaces sont définies dans un module, OMG IDL appelé **CosEventComm**

```
module CosEventComm {
        exception Disconnected{};
        interface PushConsumer {
        void push (in any data) raises(Disconnected);
        void disconnect_push_consumer();
        };
        interface PushSupplier {
        void disconnect_push_supplier();
        };
        interface PullSupplier {
        any pull () raises(Disconnected);
        any try_pull (out boolean has_event)
        raises(Disconnected);
        void disconnect_pull_supplier();
        };
        interface PullConsumer {
        void disconnect_pull_consumer(); };
        };
```

b. CosEventChannelAdminmodule

Ce module définit les interfaces pour établir la connexion entre les producteurs et les consommateurs. L'établissement de la connexion est un processus multi-step. L'OMG IDL

montré ci-dessous illustre les opérations d'Event Channel que les consommateurs et les producteurs doivent appeler d'abord.

```
module CosEventChannelAdmin {
                    //Création des proxies qui permet à des
                    //consommateurs de se relier à un canal d'événement.
                    interface ConsumerAdmin {/*… */};
                    // Création des proxies qui permet à des producteurs
                    //de se relier à un canal d'événement.
                    interface SupplierAdmin {/*… */};
                    interface EventChannel
                    {
                    // Renvoie une référence d'objet pour créer des
                    //proxies pour le producteur.
                    ConsumerAdmin for_consumers ();
                    // Renvoie d'une référence d'objet pour créer des
                    //proxies pour le consommateur
                    SupplierAdmin for_suppliers ();
                    // arrêt du canal d'événement
                    void destroy ();
                    };
};
```

2. Le canal d'événement

On a déjà définit ce qu'est un canal d'événement on signale juste que le canal d'événement n'a pas besoin de fournir les données à son consommateur en même temps qu'il consomme les données de son producteur.

3. Le modèle de communication *push* avec un chanel d'événement

Le producteur initie l'envoie des données d'événement au canal d'événement, le canal d'événement, à son tour envoie les données au consommateur.
Le schéma ci-dessous (figure 9) illustre une communication de type *push* entre un producteur et le canal d'événement, et un consommateur et le canal d'événement.

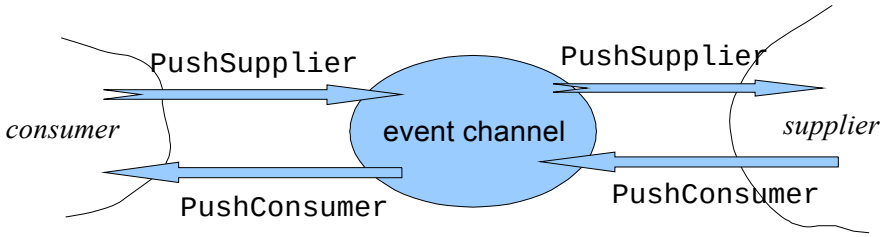


Figure 9 : Modèle de communication *event channel (push)*

4. Le modèle de communication *pull* avec un chanel d'événement

Le consommateur tire les données d'événement du canal d'événement ; le canal d'événement, à son tour tire les données d'événement du producteur.
Le schéma ci-dessous (figure 10) illustre une communication de type pull entre un consommateur et le canal d'événement, et un producteur et le canal d'événement.

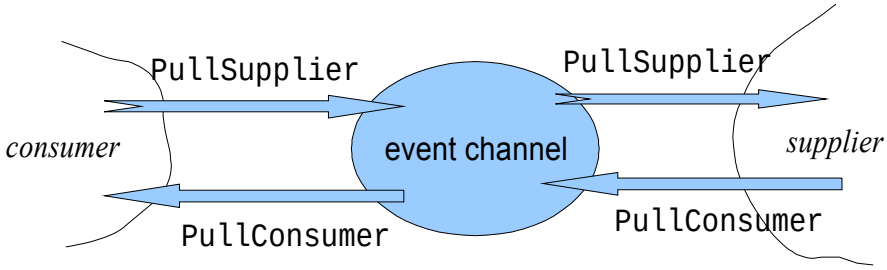


Figure 10 : Modèle de communication *event channel (pull)*

5. Communication mixte avec un canal d'événement

Un canal d'événement peut communiquer avec un producteur en utilisant un modèle de communication, et communique avec un consommateur en utilisant un modèle différent de celui utilisé avec le producteur.

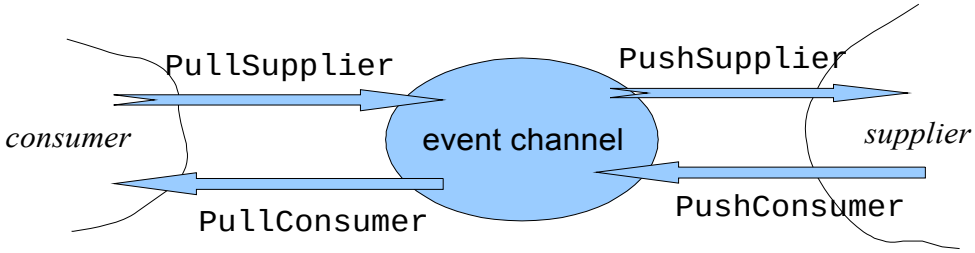


Figure 11 : Modèle de communication *event channel* (communication mixte)

Le schéma ci-dessus (figure 11) illustre une communication de *push-modèle* entre un producteur et un canal d'événement, et une communication de *pull-modèle* entre un consommateur et le canal d'événement. Le consommateur tire les données d'événement que le producteur a envoyé au canal d'événement. D'une façon générale le canal d'événement peut consommer les événements d'un ou plusieurs producteurs, et fournit des événements à un ou plusieurs consommateurs.

V. Modèle de la plateforme indépendante (PIM)

Cette section définit le modèle de la plateforme indépendante (PIM) pour le service d'événement léger. Le service d'événement léger est prévu pour être un sous-ensemble du service d'événement CORBA. Cette sous ensemble est modélisé par des package, des interfaces, et des classes en cas des spécifications de service Les descriptions des interfaces, des opérations et de leur sémantique sont également prévues pour être identiques à ceux définies par l'excédent de cet sous-ensemble (Version 1.1, March 2001)

1. Le Package CosLightweightEventComm

Le package *CosLightweightEventComm* define les interfaces enter push consumers et *push suppliers*. Seulement le *push model* est supporté par le service d'événement léger.

2. Le Package CosLightweightEventChannel

Le *CosLightweightEventChannelAdmin* établissant les connexions entre les producteurs et les consommateurs, seulement le *push model* est soutenus par le service léger d'événement.

Partie II

Conception

CHAPITRE 3

Conception d'un Service d'Évènement Fiable et Temporisé

Chapitre III : Conception d'un service d'évènement Fiable et Temporisé

I. Introduction

Les services d'évènements sont des interfaces définies par l'OMG qui permettent la communication asynchrone entre des objets CORBA. Et constituent un des principaux dispositifs d'interaction asynchrone dans les systèmes répartis.
Le fonctionnement principal d'un service d'évènements (voir Figure) est basé sur un canal d'évènements dans lequel des fournisseurs émettent des données et les consommateurs reçoivent ces données. Le serveur va émettre les données vers le canal d'évènements et le client qui est prévenu par le canal fait la réception des ces données.
Ce modèle est appelé le modèle **push** (écriture d'évènements). Cependant Il existe un autre modèle nommé **pull** (lecture d'évènements).
Ce modèle **pull,** permet à un consommateur de solliciter des données évènements d'un producteur. Dans ce cas le consommateur a donc l'initiative de faire le transfert des données évènements.

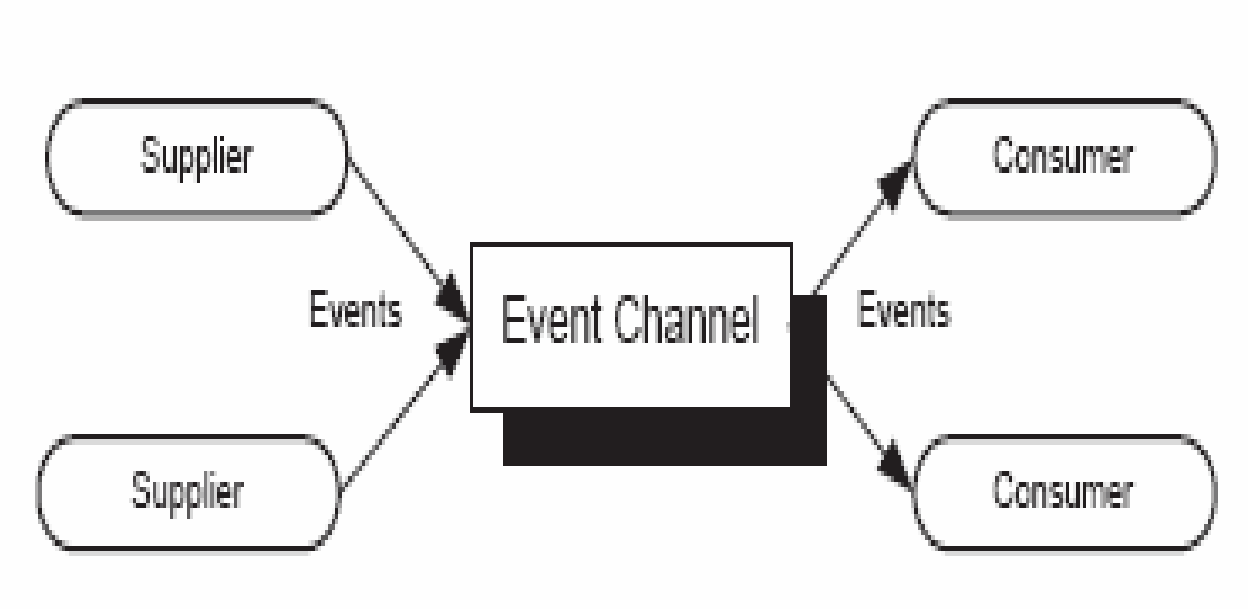


Figure 12 : Composantes d'un service d'évènement CORBA

Les serveurs et clients sont isolés les uns des autres par des proxies internes du service d'évènements. Le serveur ne dialogue qu'avec son *proxy* consumer qui va envoyer l'évènement vers le *proxy* supplier vers son client.
L'objectif de cette architecture est bien que serveurs et clients soient le plus découplés possible. Cependant, les spécifications actuelles de ces services d'évènement ne permettent pas d'exprimer des contraintes de temporelles et les contraintes de fiabilité dans l'envoi des évènements.
Apres avoir fait un état de l'art sur les services d'évènement (Chapitre précédent) et en tenant compte de ces limitation des contrainte on va proposer dans ce Chapitre une méthode pour

exprimer les contraintes de temps et de fiabilité et les prendre en compte dans les canaux d'évènement.

II. Les limitations des services d'évènement CORBA

Le service d'événement CORBA adresse beaucoup de limitations, les contraintes des applications réparties qui exigent un niveau de fiabilité n'était pas explicitement respecté lors de la spécification de ces services on ne sont pas définies on peut cité à titre d'exemple :

- **Support de QoS :** Dans les systèmes qui exigent des contraintes temporelles, les événements doivent être traités de sorte que les consommateurs puissent satisfaire leurs besoins en terme de qualité de service, tels que la fiabilité, la priorité, et l'ordre de délivrance de évènement, et le *timeliness*. Cependant, le service d'événement de CORBA ne fournit aucune interface ou politique pour soutenir ces propriétés de qualité de service. Par exemple, il n'y a aucune interface qui assure que l'ordre de délivrance des évènements a été respecté, ou que le flux produit est bien égal au flux consommé.
- **Mécanisme de filtrage des évènements :** pour supporter divers modèles de filtrage des évènements on peut chaîner plusieurs canal d'évènement, il en résulte que les événements non exigés par le consommateur seront rejetés automatiquement par un canal d'événement.

Tenant compte de ces lacunes on va proposer une conception pour spécifier les contraintes temporelles et de fiabilité lors des interactions producteur/consommateur via un canal d'évènement.

1. Contrainte de temps de livraison

Nous commençons par la contrainte de temps (temps de livraison), en supposant que les horloges de différent site sont synchronisées et qu'il y a plusieurs consommateurs et plusieurs producteurs qui se communiquent via un canal d'évènement, comme dans la figure ci-dessous.

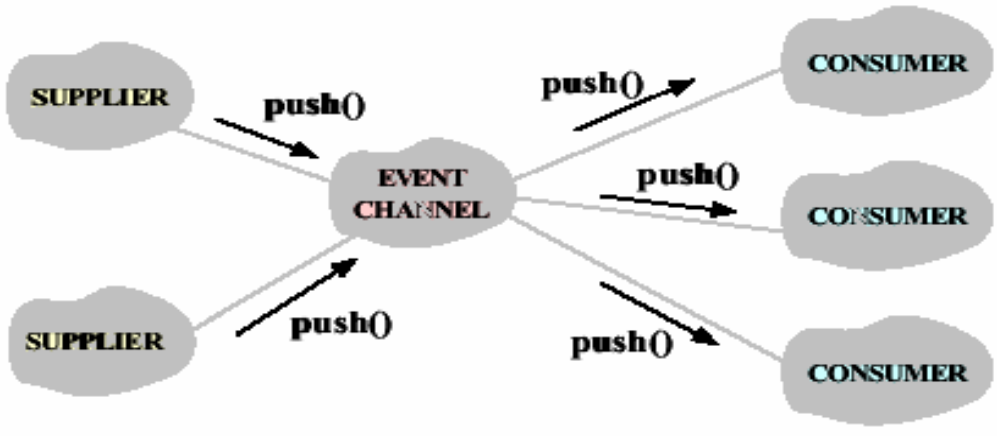


Figure 13 : Modèle de Communication *push*

Légende :

Supplier : Producteur (fournisseur)
Consumer : Consommateur (client)

Nous allons attribuer a chaque producteur un id qui sera unique, puis pour chaque évènement produit par le producteur *producteur-id* on précise le temps d'envoie, et la plage de temps maximum pour que cet évènement soit consommer par le consommateur. La structure suivante représente cette conception :

<Producteur-id, event, dateenvoi, plagedetemps>

Les deux derniers paramètres expriment la qualité de service (les contraintes de temps)
L'évènement envoyé par le producteur sera consommé en premier temps par le canal d'évènement et après un temps Δ le canal d'évènement livrera l'évènement au consommateur
Lorsque l'évènement arrive au consommateur, ce dernier doit enregistrer le temps d'arriver d'évènement dans un variable ***time_livre,*** il y a deux scénarios selon le calcul fait par le consommateur :

Scénario 1 : Soit le consommateur rejette le message d'évènement
Scénario 2 Soit le consommateur accepte de consommer l'évènement

Le consommateur doit calculer la quantité ***time_livre – (dateenvoi+ Δ)***

Si (***time_livre – (dateenvoi+ Δ)) > plagedetemps***

{
Le choix du consommateur est le ***scénario 1***
}

Sinon {
L*e choix du consommateur est* le ***scénario 2***
}

L'idée consiste de comparer la différence entre le temps de livraison et la somme du temps d'envoie et le temps Δ que passe le canal d'évènement avant la livraison du message et nous comparons par la suite le résultat avec la plage permis (contrainte de livraison) Ceci vérifie la contrainte qui tient en compte le délai de livraison d'un évènement.

2. Contrainte de Fiabilité

Comme contrainte de fiabilité nous allons étudier la propriété d'ordre de délivrance de messages, c'est-à-dire s'assurer que le consommateur consomme dans le même ordre de productions de événements. Pour ce là on va utiliser la relation de causalité et on définie un horloge logique qui a chaque message m_i produit par le producteur délivre un entier C (m_i) de tel sorte que si le producteur a produit deux message m_i et m_j tel que m_i est produit avant m_j on aura l'inégalité.

$$C(mi) < C(mj)$$

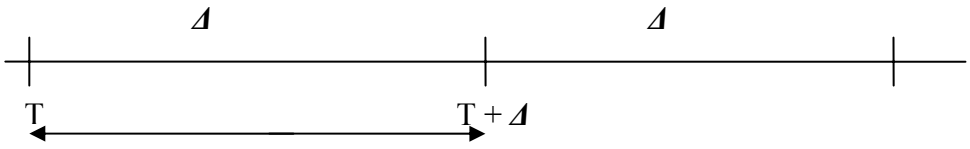
Donc a chaque fois que le canal d'évènement livre un ensemble des évènements au consommateur, ce dernier doit commencer à consommer l'évènement m_i dont le ***$C(m_i)$*** vérifie l'égalité

$$C(m_i) = \inf_{I \in [1,k]} [C(m_1), C(m_2), C(m_3), \ldots, C(m_k)]$$

Avec k est le nombre des évènements délivrer par le canal d'évènement au consommateur, mais il faut faire attention lors de l'utilisation de cette formule car on a trois variables temps différents :

- ☞ Date d'envoie di message,
- ☞ Date de réception du message (par le canal d'évènement et non par le consommateur),
- ☞ Date de livraison du message par le consommateur.

Donc on applique la formule ci-dessus entre le canal d'événements et le consommateur et à ce moment là on est sur que le consommateur consomme le premier évènement produit par le producteur (par rapport aux messages qui ont été délivrer par le canal d'évènement)



Période de réception
d'évènement par le canal

T : La date d'envoi du message
***Δ* :** La période dont le canal d'événement garde le message avant de le délivré au consommateur.
***Δ*+*T* :** La date de livraison du message vers le consommateur.

Cette conception nous permet d'exprimer les contraintes de temps et de fiabilité au sein d'un service d'évènement (lors des interactions entre un producteur et un consommateur via un canal d'événement) Dans le reste du chapitre on fait un aperçu sur les protocoles de diffusion *multicast.*

III. Protocoles de diffusion fiable (*multicast*)

Pendant quelques décennies, une recherche importante a eu lieu et dont le but est de développer de nouveaux protocoles de transport multicast. Cependant, plusieurs protocoles ont été implémentés et sont en phase de test. Les recherches dans ce domaine confirme qu'à l'heure actuelle, aucun protocole multicast ne suffise a lui tout seul pour tous les types d'applications multicast qui peuvent exister.

1. Définition et Principes de base

Un service de communication *multicast* offre un moyen efficace de diffuser des unités de données à un groupe de récepteurs. Le *multicast IP* fournit au niveau réseau un support efficace pour la diffusion non fiable des paquets pour un grand nombre d'applications tel que la visioconférence, ftp,mise à jour d'informations réparties, applications coopératives et, simulations distribuées,…Certaines de ces applications prévoient la mise en rapport d'un nombre de participants de l'ordre de plusieurs milliers et peuvent aussi, en plus de l'efficacité du routage, nécessiter une grande fiabilité dans la délivrance des données. La figure 13 résume grosso modo le principe de *multicast.*

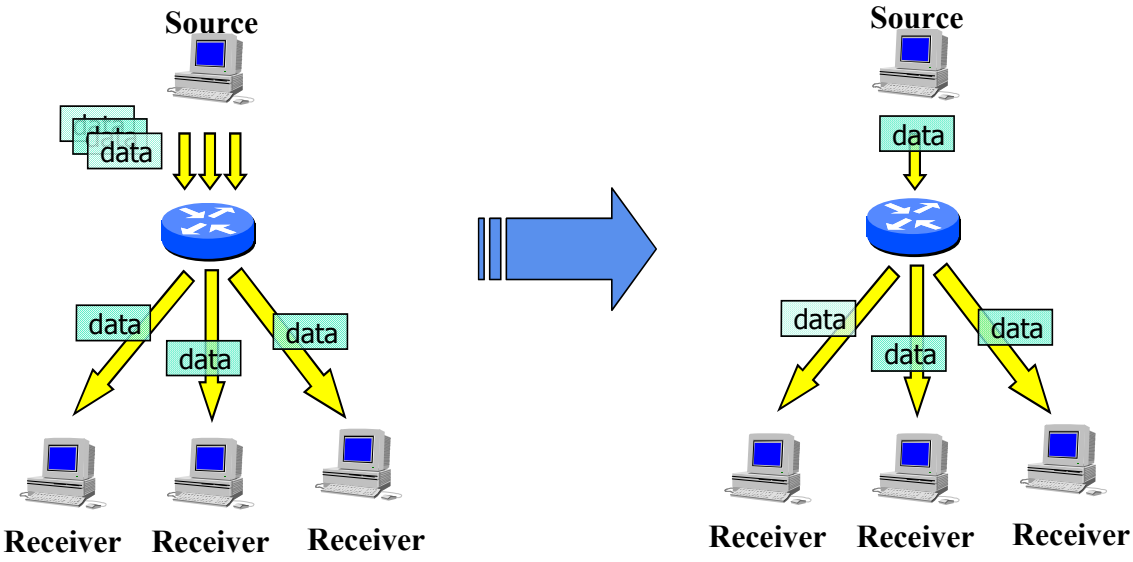


Figure 14 : Principe de la communication *multicast*

Une livraison *multicast* fiable doit satisfaire les propriétés suivantes : Validité, Unanimité, et en fin l'intégrité.

2. Problème de fiabilité

Le problème de la fiabilité pour les communications point à point est déjà bien maîtrisé et des solutions moins satisfaisantes ont été déployées. Par contre l'exigence de la fiabilité dans le contexte du multicast est un problème plus difficile à et les solutions sont moins évidentes, surtout sur des réseaux hétérogènes et étendus c'est le cas du réseau Internet. La conception de protocoles de diffusion fiable et efficaces est plus difficile et doit tenir compte des contraintes imposées par la résistance au facteur d'échelle. L'implosion et la surcharge de la source par les messages de contrôle ACK/NAK, l'exposition des récepteurs à des

transmissions dupliquées ou le contrôle de congestion en sont des exemples (voir figure 15). Malgré plusieurs années d'efforts consacrées à la conception de protocoles de diffusion fiable au dessus d'*IP multicast*, il est toujours aussi difficile de fournir une solution de *bout en bout* capable de satisfaire à l'ensemble des contraintes liées au facteur d'échelle.

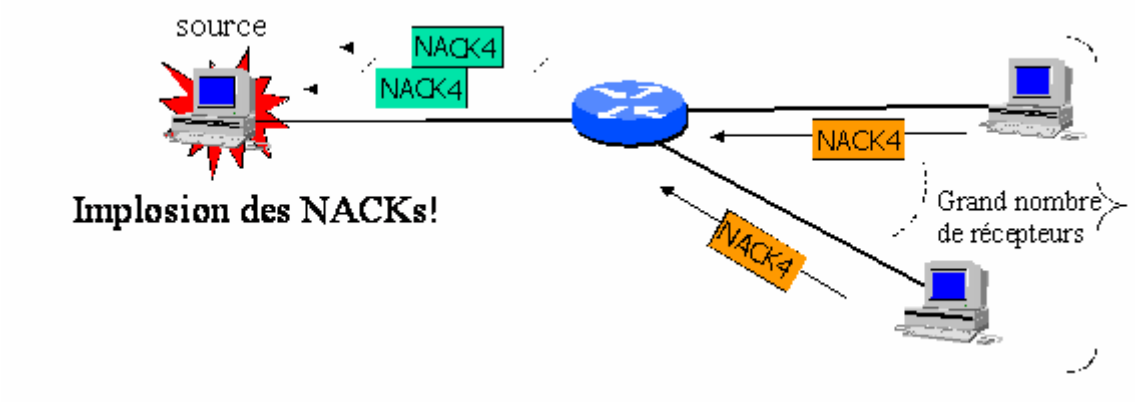


Figure 15 : Les différents facteurs d'échelle

Lorsque il y a plusieurs destinateurs de nouvelles techniques sont à implémenter, tel que ne pas délivrer des accusés de réception à chaque paquet reçu mais, à la place, envoyer des accusés de non réception (negative acknowledges, pour NACKs) pour les données qui n'ont pas été reçues. Pour ce qui est du nombre de protocoles Multicast, il en existe énormément. On présente les plus connus :

- IP/Multicast : ce protocole est une extension du protocole IP lui-même qui permet d'envoyer un paquet à plusieurs personnes simultanément,
- IGMP : C'est un protocole d'interaction entre le ou les routeurs multicast du LAN et les hotes multicast de ce même LAN. Son rôle est de distribuer les datagrammes multicast sur le LAN.

Ce protocole fait partie du protocole IP et comprend deux types de messages. un message d'interrogation du routeur et un autre message de réponse

- RTP (*Real-Time Transport Protocole*) : le protocole de transport temps réel (concerne les conférences multimédia multi-partite (transferts de vidéo et de sons),
- SRM (*Scalable Reliable Multicast*) est conçu pour les applications temps réel comme les mise à jours ou la diffusion de médias,
- URGC (*Uniform Reliable Group Communication Protocol*) permet des transactions fiables et ordonnées, c'est un protocole basé sur un contrôle centralisé,
- MFTP (*Protocole Multicast de Transport de Fichiers*) se décrit de lui-même,
- LBRM (*Log-Based Receiver-reliable Multicast*) est un protocole qui permet de garder une trace de tous les paquets envoyés par un serveur, ces derniers indique à l'émetteur s'il doit retransmettre les données ou s'il peut les supprimer correctement dans le cas où tous les destinataires ont bien reçu les données.

IV. Conclusion

La conception d'une approche pour exprimer les contraintes de temps et de fiabilités lors de la communication producteur / consommateur via un canal d'événement a nécessité beaucoup d'effort et de travail personnel. En effet, il a fallut s'imprégner rapidement aux concepts liés au middleware CORBA pour comprendre le fonctionnement d'un tel système. La majorité des articles fournie par l'OMG sont tous en Anglais. Ceci a rendu notre tâche encore plus difficile. Malgré une difficulté énorme au départ j'ai tout de même pris le temps de me documenter et d'étudier les concepts clés de manière à fournir un résultat correct.

Partie III

Réalisation

CHAPITRE 4

Partie Pratique

Chapitre IV : Partie Pratique

I. Introduction

Dans cette partie nous allons préparer l'environnement de programmation afin de réaliser un prototype de service d'événement du type CORBA sur un réseau local IP pouvant prendre en compte des contraintes de temps et de fiabilité ou à défaut l'un des type de contraintes.
Vu la difficulté de la programmation en CORBA d'ailleurs c'est l'un des principaux inconvénients de cette norme nous ne pouvons pas s'attarder sur la pratique de CORBA dans le cadre de DEA, car le moindre petit programme peut prendre des jours, voire des mois de mise au point.
On va donc faire le choix de notre plate forme c'est-à-dire l'implémentation CORBA qui nous aide à mieux pour réaliser notre prototype et en suite après avoir choisi le système d'exploitation on va compiler et installer notre implémentation CORBA. Puis nous faisons une évaluation de TAO à travers un exemple Client/Serveur.
Après avoir mené une étude bibliographique et une comparaison entres les différentes implémentations proposées de la norme CORBA et en tenant compte de notre problème qui touche les contraintes temporelle qui doivent implémenté au niveau des services d'événements on a abouti de choisir l'implémentation ***TAO*** de l'Université de Washington qui est évidement basé sur l'architecture CORBA, et qui offre une qualité de service de bout en bout (au niveau ***ORB-Core***) et se place dans un contexte de temps-reel dur.

II. Architecture TAO

TAO est une architecture de l'Université de Washington, fondée sur l'architecture CORBA, et se focalisant sur les informations concernant la performance d'exécution en du temps et l'ordonnancement des évènements. De ce fait, le besoin de transmettre les requêtes et leurs réponses dans un temps minimal a conduit à un grand travail d'optimisation aux niveaux des mécanismes de CORBA et du réseau d'interconnexion.
TAO repose sur ACE (Adaptive Communication Environment), un composant conçu par la même les même chercheurs.
TAO et ACE représentent une implémentation de CORBA qui permet aux clients d'invoquer des requêtes via des objets distribués avec une transparence par rapport à la localisation de l'objet, de la plate forme, du langage de programmation, des protocoles de communications et d'interconnexion. Le langage IDL a été entendu en IDL temps-reel (Real-time IDL) pour inclure une sémantique permettant la specication de la qualite de service, et des besoins en terme de CPU. La traduction des différents paramètres vers les couches inférieures a également été optimise l'adaptateur d'objet et de l'ORB ont également fait l'objet de modification. Le premier a été modifié afin de permettre une distribution des requêtes aux méthodes plus rapides par la mise en place d'un mécanisme de démultiplexage comprenant moins d'étapes.
L'adaptateur d'objet bénéficie de plus de son propre Ordonnanceur c'est un Ordonnanceur temps-reel pour déterminer l'ordonnancement d'une requête et affecter une priorité aux processus implique dans cette requête. Le second inclut un protocole de *communication inter-ORB temps-reel*, le protocole générale *inter-ORB* (GIOP).
En fin au niveau du réseau de communication, TAO propose un sous-système, appelé le sous-système Gigabit I/O, destiné a optimiser les entrées/sorties des systèmes d'exploitation.
De plus, TAO utilise un ensemble de services temps-reel destines a renforcer l'amélioration des performances, parmi lesquels :

- **Un service d'ordonnancement:** TAO inclut en réalité deux politiques d'ordonnancement. En effet, l'hypothèse de TAO est que le temps d'exécution et que les besoins en ressources sont Connus pour les requêtes. Ces conditions permettent de considérer que les ressources autres que le processeur peuvent être ordonnancées selon un algorithme dynamique. Pour le Processeur, TAO inclut un service d'ordonnancement statique, inspire de l'algorithme d'ordonnancement ***rate*** *monotonic*, évaluant d'une part la faisabilité d'une requête en terme de CPU, l'assignation d'une priorité cette requête cela si le requête est considérée comme faisable et prenant en charge les changements de paramètres. Dans le cas d'exécution périodique des processus, TAO utilise un mécanisme d'ordonnancement temps-reel;
- **un service d'événement :** Ce service permet a deux extrémités de pouvoir fonctionner de manière asynchrone. TAO constitue une approche intéressante de la gestion de la qualite de service dans CORBA par les très nombreuses solutions qu'elle propose. Cependant, l'adoption de la plupart de ces solutions doivent s'accompagner de modifications importantes dans CORBA mais ceci n'influe pas sur notre mémoire.

Les améliorations de ***TAO*** par rapport au ***CORBA*** standard sont :

- La spécification et la validation de la qualité de service (inter ORB),
- Les caractéristiques temps réel,
- L'optimisation de la performance,
- Les informations de la qualité de service sont spécifiées par des interfaces IDLs prédéfinies,
- Le support d'ordonnancement temps réel,
- Service d'éventement supporte l'exécution temps réel,
- Une extension temps réel du protocole GIOP nommé RTIOP,
- Adaptateur d'objet temps réel,
- Des optimisations du multiplexage, démultiplexage des requêtes,
- la gestion d'événements temps réel,
- TAO réduit aussi de manière significative la taille des souches qui est souvent conséquente dans les ORBs.

Pour conclure TAO est défini par ses fondateurs comme étant conçu avec une extrême habilité et un savoir faire qui permettrait l'automatisation de la garantie d'une certain qualité de service, d'un certain degré de performance, et du temps réel des applications répartie.
La version déjà testée de TAO est livrée sous forme de code source donc avant de l'utiliser il faut faire une compilation et une génération de ses libraires associées.

III. Prise en main de l'environnement de développement

Le but de notre prototype consiste à faire communiquer un producteur et un consommateur d'événement en utilisant les services d'événements fournis par CORBA cette communication se déroule via en canal d'événement et sur un réseau IP**.** La problématique consiste à faire ajouter des contraintes temporelles et des contraintes de fiabilités au service d'événement l'or des interactions producteur/consommateur, plus précisément l'objectif est de faire implémenter la conception qu'on a déjà fait au chapitrer précèdent.
Le principe de communication proposé par CORBA consiste à crée des objets qui seront accessibles par le client (consommateur d'événement) et par le serveur (producteur d'événement), ce mécanisme existe déjà pour des application java distantes, le bon exemple est *RMI* pour *Remote Method Invication* et pour la platforme *.NET* c'est *.NET Remontig*

Donc l'utilisation de CORBA n'est pas nécessaire si l'on veut communiquer deux applications (client/serveur) C++ ou deux applications (client/serveur) *java*, cependant entre une application *C* et une application java, la technologie CORBA devient inévitable.
Comme notre client et notre serveur ne sont pas sur la même site on va choisir l'*ORB TAO* pour implémenter notre serveur (*TAO* est en ***C*++** donc il n'est pas souhaitable de choisir un *ORB* en *Java* car ce dernier est très lourd).
Et un *ORB* en java pour faire l'interface de notre Client pour cela notre choix porte sur *JacORB* qui est entièrement gratuit et open source

1. Installation de TAO pour l'application Serveur

Pour implémenter notre serveur on va utiliser le langage *C++*, pour cela nous allons utiliser *TAO* (*The Ace Orb*), *ACE* signifiant *Adaptive Communication Environment*. *TAO* s'installe sur une multitude de systèmes d'exploitation, nous allons décrire son installation sous la plateforme Windows XP.
Pour bien installer TAO c'est mieux de respecter l'ordre des étapes suivantes :
Apres avoir télécharger TAO (on l'enregistre dans un emplacement précis) on crée un dossier prototype qui contient aussi un dossier CORBA puis on extraire l'archive .zip (de TAO) dans un répertoire au choix comme dans l'exemple D:\Prototype\CORBA.

Un répertoire ACE_wrappers sera automatiquement créé sous l'arborescence

D:\Prototype\CORBA.

C:\prototype\CORBAdoit bien sûr correspondre au répertoire dans lequel l'archive .zip a été extraite.
On crée les deux variables d'environnement suivantes avec leur valeur respective :

ACE_ROOT = D:\Prototype\CORBA\ACE_wrappers
TAO_ROOT = D:\Prototype\CORBA\ACE_wrappers\TAO

Désormais, dans la suite de cette installation, *%ACE_ROOT% et %TAO_ROOT%* se réfèreront respectivement aux répertoires suivants :

D:\Prototype\CORBA\ACE_wrappers
D:\Prototype\CORBA\ACE_wrappers\TAO

a. Modification de la variable PATH

Puis on modifier la variable PATH a la fin du champ valeur**,** rajouter en séparant bien chaque valeur par un point virgule :

;%ACE_ROOT%\bin;%TAO_ROOT%\orbsvcs\Naming_Service;%TAO_ROOT%\orbsvcs\Cos Event_Service

TAO peut être compilé sur plusieurs plateformes différentes : Windows, Linux, UNIX, etc ... Mais auparavant, il faut créé un fichier d'entête lui indiquant sur quelle plateforme il va être compilé.

b. Création du fichier config.h

Pour indiquer au compilateur qu'a choisi la plateforme Windows, il suffit de créer un fichier ***config.h*** dans le répertoire ***%ACE_ROOT%\ace*** dans lequel on écrit la ligne suivante :

#include "ace/config-win32.h"

c. Compilation de TAO

Lorsque nous avons téléchargé et décompressé l'archive Zip de TAO, il n'y a aucun binaire livré avec cet archive, seules les sources sont présentes. C'est pourquoi il est maintenant nécessaire de compiler TAO afin de générer les binaires nécessaires à son exécution.
Sous Windows, il y a plusieurs façons de compiler TAO. La plus simple consiste à utiliser Visual Studio. Puisque qu'il est compatible avec la version 6 on a choisi Visual Studio 98
Pour compiler notre CORBA.
Pour cela, il faut ouvrir le fichier %TAO_ROOT%\TAOACE.dsw. Et on lance la compilation en utilisant l'interface graphique du VS 6, et à ce moment là il faut prendre son mal en patience, car la compilation peut durer plus d'une heure suivant la config utilisée.
Voila, on vient de terminer l'installation de notre CORBA, TAO est donc prêt à être utilisé, nous allons maintenant nous occuper de la partie qui concerne notre Client donc c'est l'installation de l'orb *JacORB*.

2. Installation de JacORB pour l'application Cliente

Au coté Client, nous allons utiliser *JacORB*. *JacORB* est entièrement écrit en Java d'où sa portabilité sur tous les systèmes d'exploitation supportant ce langage.
Pour bien installer *JacORB* c'est mieux de respecter l'ordre des étapes suivantes :
On enregistre puis on extraire l'archive .zip dans un répertoire au choix comme dans l'exemple D:\Prototype\CORBA\JacORB.

a. Installation de Ant

Il est absolument nécessaire d'installer un outil nommé Ant avant de commencer l'installation de JacORB. Ant est une sorte de Makefile pour les programmes Java, c'est un outil gratuit, Ant va nous servir durant les étapes d'installation de JacORB.
On enregistre puis on extraire l'archive .zip dans un répertoire au choix comme D:\Prototype\Java.un répertoire apache-ant-1.6.5 sera automatiquement créé sous l'arborescence D:\Prototype\Java.
Puis on crée les deux variables d'environnement suivantes avec leur valeur respective :

ANT_HOME = D:\Prototype\Java\apache-ant-1.6.5
JAVA_HOME = D:\Prototype\Java\j2sdk1.4.2_06

Ceci suppose que la version du *JDK de Java est la 1.4.2_06*

b. Modification de la variable PATH

A la fin de la valeur de la variable PATH, on rajoutera :

;%ANT_HOME%\bin

c. Installation de JacORB

On crée les deux variables d'environnement suivantes avec leur valeur respective :

JRE_HOME = D:\Prototype\Java\j2sdk1.4.2_06\jre
JACORB_HOME = D:\Prototype\CORBA\JacORB

Puis on modifie la variable PATH, à la fin de la valeur de la variable PATH, on rajoutera :

;%JACORB_HOME%\bin

On copie le fichier *orb.properties* qui se trouve dans *%JACORB_HOME%\etc* dans *%JRE_HOME%\lib* afin que l'ORB Java utilisé soit bien JacORB et non pas l'ORB intégré au JDK de Sun.
Puis on edite le fichier *orb.properties* et on modifie la ligne suivante comme suit :

jacorb.config.dir=D:/Prototype/CORBA/JacORB

Cette ligne va signifier à *JacORB* d'utiliser un autre fichier de propriétés qui se trouve en fait dans *%JACORB_HOME%\etc*. Cet autre fichier s'appelle *jacorb.properties* et il peut être créé à partir du fichier *jacorb_properties.template* qui se trouve également dans *%JACORB_HOME%\etc*. Pour cela, on fait une copie du fichier *jacorb_properties.template* en jacorb.properties. On reviendra ultérieurement à ce fichier *jacorb.properties*.

d. Génération du script de lancement des applications Java avec JacORB

Afin d'exécuter une application Java avec JacORB, nous aurons besoin d'un script nommé jaco.bat. Pourtant, ce script n'existe pas par défaut à l'installation de JacORB. Nous allons le générer grâce à Ant. Pour cela, il suffit de se placer dans le répertoire %JACORB_HOME% et d'exécuter la commande :

ant jaco

Après l'exécution de la commande précédente deux fichiers vont être créés : *jaco* et *jaco.bat*. Le premier fichier correspond au script exécutable pour Linux/UNIX tandis que le second correspond au script exécutable pour Windows. C'est la deuxième qui nous intéresse

3. Génération du compilateur IDL

Le langage *IDL* (Intermediate Development Language) est, déjà défini dans la premier partie on va juste mentionné que c'est un langage intermédiaire qui permet de décrire des objets qui vont pouvoir être implémentés en Java ou en C++ et grâce auxquels un client et un serveur vont communiquer. Un compilateur *IDL* permet ainsi de traduire un objet décrit en langage *IDL* en Java en utilisant le compilateur *IDL* de *JacORB* ou en C++ en utilisant avec le compilateur *IDL* de *TAO*.
Le compilateur *IDL* de *TAO* a été généré automatiquement lorsque nous avons compilé *TAO*. En revanche, pour *JacORB* nous allons le générer nous-mêmes grâce à *Ant* :

Donc pour cela on se place dans le répertoire %JACORB_HOME% et on exécute la commande :

Ant idlcmd

Après l'exécution de la commande précédente deux fichiers vont être créés : *idl et idl.bat.* Le premier correspond au script exécutable pour Linux/UNIX tandis que le second correspond au script exécutable pour Windows. C'est la deuxième qui nous intéresse.

IV. Les différentes étapes pour la mise en place de notre prototype

La norme corba n'impose pas de processus de conception et de développement d'applications distribuées ceci veut dire que la mise en place d'une application CORBA suit toujours à peu près le même scénario :

- **La définition du contrat IDL :** il faut modéliser les objets composant notre prototype à l'aide d'une méthodologie orientée objet .Cette modélisation est ensuite traduite sous forme de contrats IDL composés des interfaces des objets et des types de données utiles pour assurer la communication entre les objets.
- **Le Pré-compilation du contrat IDL :** les interfaces des objets sont décrites dans des fichiers texte simple. 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 prés compilateurs IDL.
- **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 l'IDL, ainsi que le code des squelettes pour les programmes serveurs implantant ces types. Cette projection est spécifique à chaque langage de programmation, ceci dépend de L'environnement CORBA utilisé.
- **L'implantation des interfaces IDL :** en complétant le code généré pour les squelettes, on doit implanter les objets dans le langage le mieux adapté à la réalisation de ses objets. On doit tout de même respecter les règles de projection vers ce langage.
- **L'implantation des serveurs d'objets :** on 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 en utilisant par exemple le service Nommage et se mettre en attente de requêtes pour ces objets.
- **L'implantation des applications clientes des objets :** on doit 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 par exemple le service Nommage.
- **L'installation et la configuration des serveurs :** cette phase consiste à installer dans le référentiel des implantations les serveurs nécessaire pour automatiser leur activation lorsque des requêtes arrivent pour leurs objets.
- **La diffusion et la configuration des clients :** une fois les programmes clients mis au point, il est fortement nécessaire de diffuser les exécutables sur les sites concerné et de configurer les sites clients pour qu'ils sachent où se trouvent les serveurs utilisés.
- **L'exécution répartie de l'application :** et en fin, Grâce au protocole IIOP, l'ORB CORBA permet d'assurer les communications entre le client et le serveur donc notre application est prêt d'être exploiter .

V. Conclusion

Dans le chapitre précèdent nous avons fait une conception d'une approche qui permettre d'ajouter les contraintes de temps et de fiabilités lors de la communication par les services d'évènements ce qui a nécessité beaucoup d'effort et du travail.
Dans ce chapitre après avoir installé et compiler notre environnement CORBA On a évalué l'environnement TAO (qui est une implémentation gratuit et open source de la norme CORBA) et nous avons mis un exemple d'application client/serveur à objets.
Faute de temps, notre prototype est un cours de développement mais le difficile a été franchi.

CONCLUSION & PERSPECTIVES

CONCLUSION

CONCLUSION & PERSPECTIVES

L'objectif de ce mémoire était de concevoir une solution pour la prise en compte des contraintes de temps, et de fiabilité dans l'expression et l'acheminement d'un évènement, et les moyens de réaliser ce type de service.

Dans une première partie, nous avons présenté le contexte et ses problèmes. En suite nous avons introduit une présentation complète de la norme CORBA. La partie axée sur l'étude de la littérature de services d'évènement nous a permis d'éclaircir nos idées, et de souligner le besoin en terme de Qualité de service pour les services d'évènement. Dans la deuxième partie, nous avons fournit une conception pour exprimer les contraintes de temps et de fiabilités lors de la communication producteur / consommateur via le canal d'événement .puis dans la dernier partie dédiée aux expérimentations on a évalué l'environnement CORBA via son implémentation TAO qui est accessible librement et dotée de contraintes temps réel et on a mis en œuvre une application client / serveur à objet.

Tout au cours de ce mémoire, pour exprimer les contraintes temporelles, nous avons supposé que les horloges de différents sites de notre système sont synchronisées. Par conséquent, il serait intéressant, pour un travail futur, d'étudier ce problème.

Vu aussi la complexité de CORBA, d'ailleurs c'est l'un des principaux inconvénients de ce standard, nous n'avons pas pu achever notre prototype mais le difficile a été franchi, le travail restant sera une passerelle pour une future thèse dans le domaine de qualité de service de système d'objets reparti.

BIBIOLGRAPHIE & NETOGRAPHIE

BIBLIOGRAPHIE

BIBLIOGRAPHIE & NETOGRAPHIE

[1] D. Houatra, OMG: "Event Service Specification, version 1.2", The Object Management Group, http://www.omg.org/docs/formal/04-10-02.pdf, October 2004.

[2] D. Houatra, "QoS-constrained Event Communications in Distributed Virtual Environments", DOA'2000 – 2nd, Sept. 21-23, Antwerp, Belgium, pp.71-80.

[3] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service", ACM Transactions on Computer Systems, 19(3):332-383, Aug 2001.

[4] http://java.sun.com/products/jini/2.1/doc/specs/html/event-spec.html., Evènements JiniTM.

[5] Don Box, Luis Felipe Cabrera, Craig Critchley &a ll, "Web Services Eventing", Joint IBM, BEA Systems, Microsoft, Computer Associates, Sun Microsystems, and TIBCO Software spec. : http://www-128.ibm.com/developerworks/webservices/library/specification/ws-eventing, 2004.

[6] http://www.cs.wustl.edu/~schmidt/TAO.html, TAO.

[7] Didier Lc Tien, Christian Bac Support de média continus dans CORBA : Etat De l'art et proposition, 2004.

[8] Jean-Marc Geib - Christophe Gransart - Philippe Merle, CORBA : des concepts à la pratique, 2004.

[9] D. Houatra, "Internet-Based Reliable Multicast Services : Protocol Specifications for Distributed Simulation, General Survey and Comparison", November 1999.

[10] Marc-Olivier Killijian, Présentation de CORBA :http://corba.developpez.com, 19 mars 2006.

[11] Pradeep Gore, Ron Cytron, Douglas Schmidt, Carlos O'Ryan "Designing and Optimizing a Scalable CORBA Notification Service", 2001

[12] Laurence Duchien, "Le modèle Objet de OMG-CORBA", 95

[13] S. Wilson, H. Sayers, M. D. J. McNeill, " Using CORBA Middleware to Support the Development of Distributed Virtual Environment Applications"

[14] Chris Greenhalgh, Steve Benford, Gail Reynard, "A QoS Architecture for Collaborative Virtual Environments"

[15] Gérard Florin, "Cours d'introduction Modèles d'interactions pour le client serveur et exemples d'architectures les implantant"

[16] Douglas C. Schmidt, "Patterns and Performance of Real-time Object Request Brokers High-performance, Real- time ORBs Douglas C. Schmidt"

[17] Sacha Krakowiak, "Protocoles de groupes et diffusion",http://sardes.inrialpes.fr/people/krakowia

[18] Douglas C. Schmidt, " An Overview of OMG CORBA Event services",http://www.cs.wustl.edu/_schmidt/