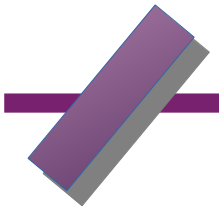
Construction d'application avec CORBA

Introduction



CORBA : bus pour l'interopérabilité

CORBA (Common Object Request Broker Architecture) est une architecture proposée et maintenue parl'OMG 1 (Object Management Group) pour permettre l'intégration d'une grande variété de systèmes hétérogènes distribués orientés objet. CORBA est la spécification d'architecture la plus répandue pour développer des systèmes distribués.

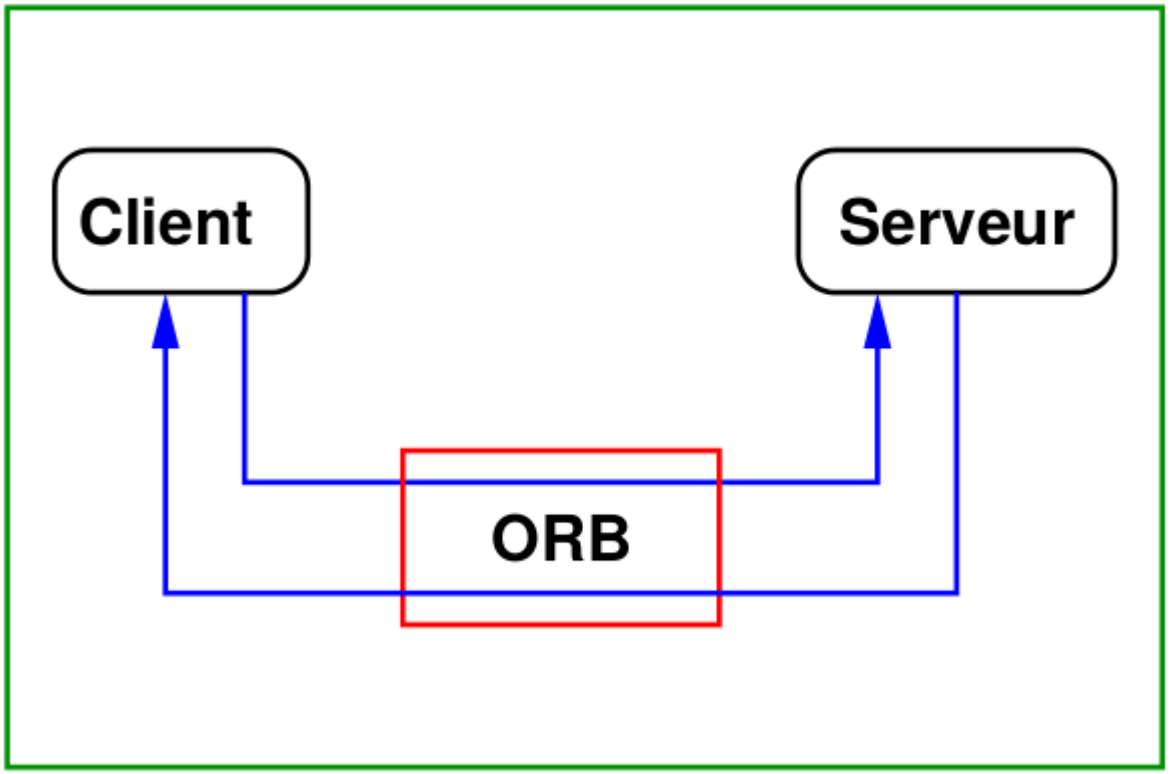
CORBA est conçue pour supporter des applications distribuées orientées objet et développées selon le modèle client-serveur. On dit aussi que CORBA est un bus réparti

Principe du bus logiciel

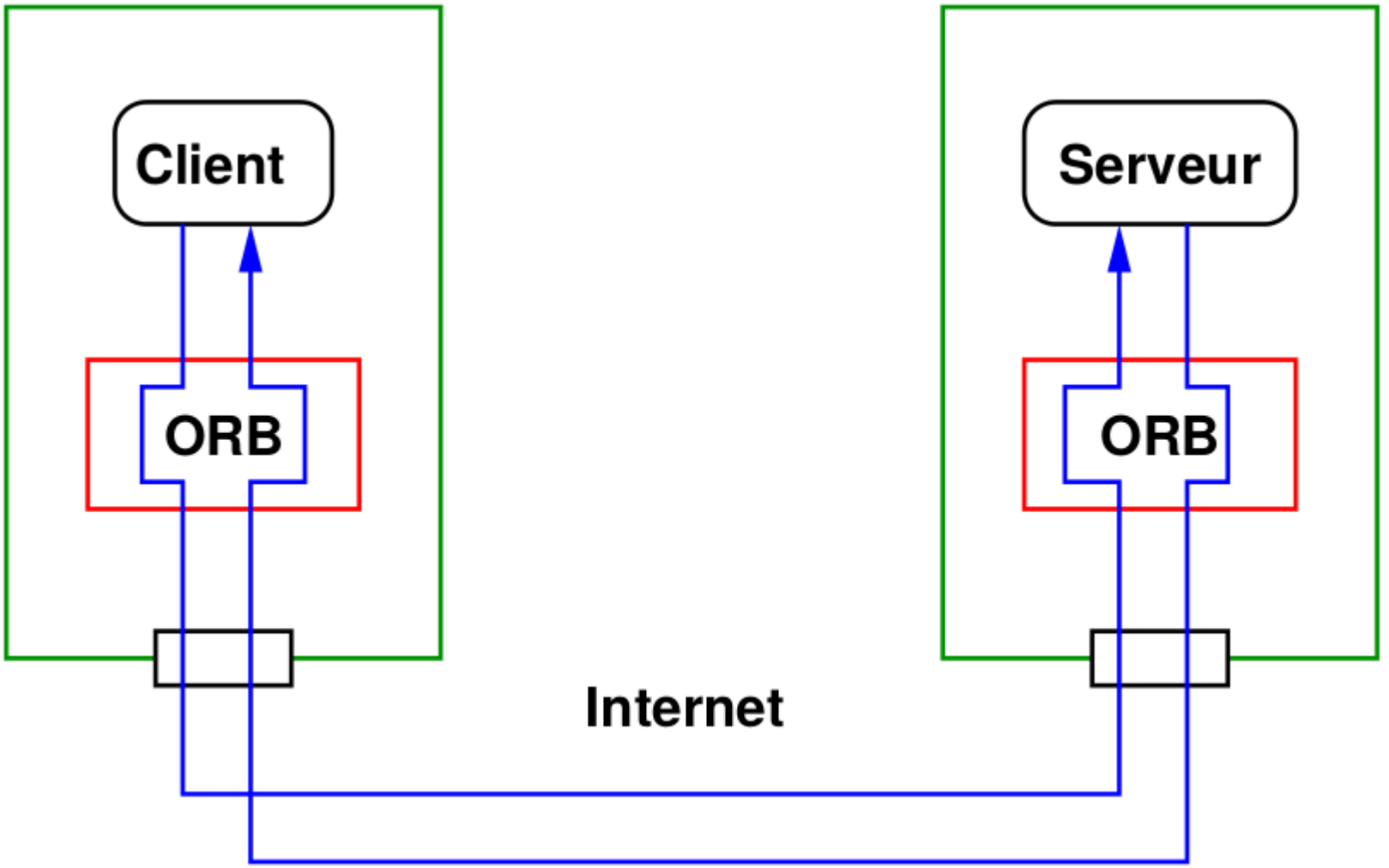
L'ORB est un bus logiciel :

- tout appel depuis un client vers un serveur passe obligatoirement par au moins un ORB qui se charge de tout :
  - localisation et activation du serveur
  - indépendance par rapport au langage, au système et au hardware
  - transparence réseau
- les ORBs communiquent entre eux par le General Inter-Orb Protocol, très souvent la version TCP/IP : Internet Inter-Orb Protocol (IIOP)
- communications client et ORB : stubs et API directe (appel à certains services)
- communications serveur et ORB : skeleton, Object Adapter et API

Une seule machine



Deux machines



Organisation des spécifications CORBA

- **un modèle objet**
- **un langage pour décrire les objets (IDL)**
- **une traduction de l'IDL vers les langages classiques (C, Java, etc.)**
- **des APIs :**
  - **décrites en IDL et associées à une sémantique précisepour l'ORB (fonctions du noyau CORBA**
  - **pour les Object Adapters (pour l'implémentation concrète des objets CORBA)**
  - **pour les services et les facilities**
- **des spécifications de plus bas niveau pour assurer l'interopérabilité entre les ORBs (GIOP et IIOP par exemple)**
- **un modèle de composants (CORBA Component Model)**

de type entreprise Java Bean, mais indépendants du langage (depuis la version 3.0)

Déroulement de requête sous CORBA

Le client n'a pas besoin de connaître :

- **la localisation de l'objet invoqué,**
- **le langage avec lequel a été programmé l'objet,**
- **le système d'exploitation sur lequel est implanté l'objet,**
- **la façon dont l'objet est implanté par le serveur.**

Pour pouvoir faire une requête, le client doit connaître la référence de l'objet, le type et l'opération à exécuter.

Une fois cette information obtenue, le client peut initialiser la requête. La requête se fait en appelant les routines représentées par les stubs ou en les construisant de manière dynamique. Un stub est

spécifique à un objet

Déroulement de requête sous CORBA

Quand la requête a quitté le client, l'ORB est le responsable de :

-la localisation de l'implémentation de l'objet,

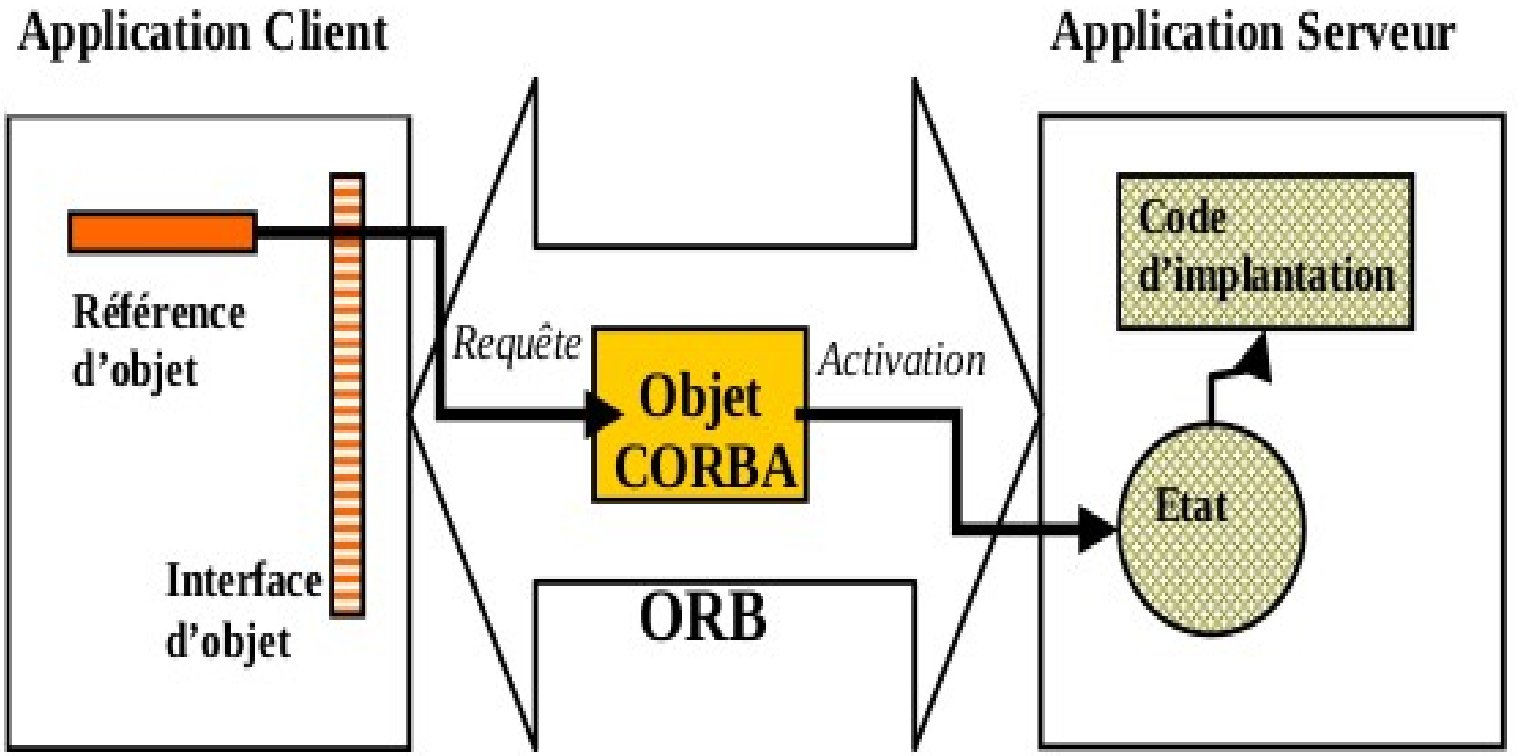
- la transmission des paramètres

- le contrôle de l'implémentation d'objet.

Le transfert d'information se réalise au travers des squelettes IDL ou des squelettes dynamiques. Les squelettes sont spécifiques à l'interface et à l'adaptateur d'objets. Si au moment d'exécuter la requête, l'implémentation de l'objet a besoin des services de l'ORB, elle peut communiquer avec l'ORB à travers l'Adaptateur d'Objets.

Une fois la requête satisfaite le contrôle et les résultats sont retournés au client.

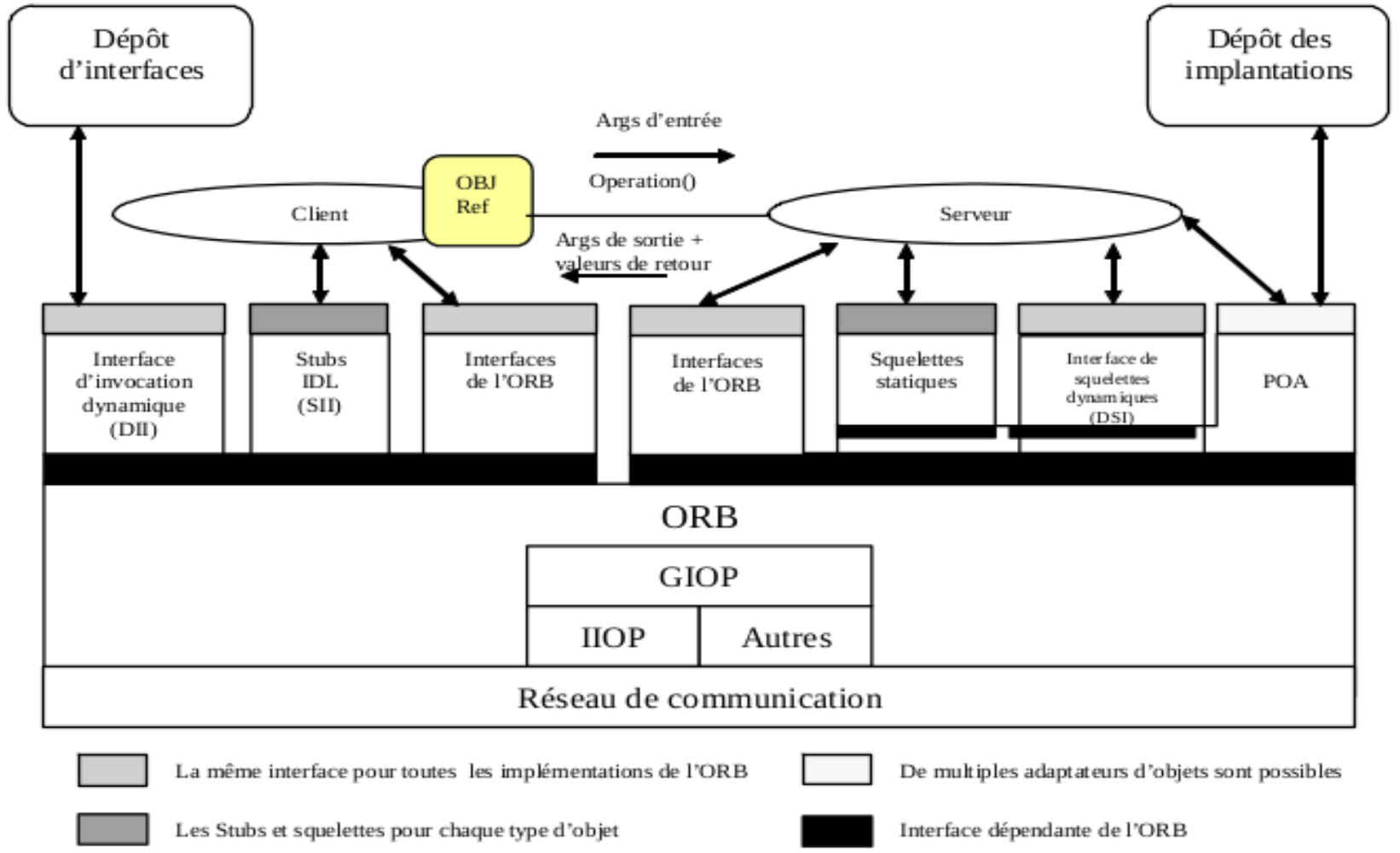
Déroulement de requête sous CORBA



Principe simplifié requête via CORBA

Composants de l'architecture CORBA

Les principaux composants (obligatoires ou optionnels) de CORBA sont :



Composants de l'architecture CORBA

• **Souches (stubs) :** permettent au client de faire des requêtes (mais ces requêtes doivent être connues avant la compilation). Lors d'un appel de méthode, le stub associé permet de préparer les paramètres d'entrée et de décoder les paramètres de sortie et résultat.

• **Interface d'invocation statique (SII : static invocation interface) :** interface pour les invocations statiques (c'est-à-dire des requêtes connues à la compilation).

• **Interface dynamique d'invocation (DII : dynamic invocation interface) :** interface pour les invocations dynamiques (c'est-à-dire des requêtes construites seulement à l'exécution).

• **Interface de l'ORB :** permet l'accès aux services de l'ORB par les clients et serveurs.

• **Squelettes statiques :** permettent aux objets de recevoir les requêtes définies avant la compilation. Lors d'un appel de méthode, le squelette associé décode les paramètres d'entrée et prépare les paramètres de sortie et résultat.

• **Interface de squelette dynamique (DSI : dynamic Skeleton Interface) : p**ermet de gérer dynamiquement les requêtes pour lesquelles des interfaces statiques n'ont pas été spécifiées.

Composants de l'architecture CORBA

• **POA (Portable Object Adaptor) :** c'est le composant qui s'occupe de créer les objets, de maintenir les associations entre objets et implantations et de réaliser l'activation des objets.

• **ORB :** c'est le noyau de CORBA pour le transport des requêtes et des réponses. Il Intègre, au minimum, les protocoles d'interopérabilité GIOP (General Inter-ORB Protocol) et IIOP (Internet Inter-ORB Protocol).

• **Dépôt d'interfaces ("Interface repository") :** contient la description des interfaces IDL accessibles aux applications à l'exécution.

• **Dépôt d'implantations ("implementation repository") :** contient la description les implantations d'objet.

• **Référence d'objet :** à chaque objet est associé une référence (chaîne de bits) permettant de faire le lien entre l'objet et son nom. Un même objet peut être associé à plusieurs références dans le temps. A tout instant, une référence ne correspond qu'à un seul objet.

Implémentation d'un objet

Partie très subtile de CORBA :

- **les langages cibles ne sont pas tous orientés objets : un objet CORBA n'est pas obligatoirement représenté par un objet**
- **un domestique (servant) est un "truc" (un objet dans un langage OO) qui implémente un ou plusieurs objets CORBA :**

  ☞ **l'idée de base est qu'une requête à un objet CORBA peut être traitée par un domestique alors que la requête suivante sera traitée par un autre domestique**

  ☞ **de plus, un même domestique peut répondre à des requêtes correspondant à des objets différents**

- **l'association objet/domestique est gérée grâce à un Object Adapter, en particulier le Portable Object Adapter le POA propose en plus l'activation et la persistence des objets**

Interface Definition Language

IDL :

langage permettant de décrire les interfaces des objets CORBA

purement déclaratif associé à une traduction officielle (mapping) vers la plupart des langages classiques (C, C++, Java, Smalltalk, etc.)

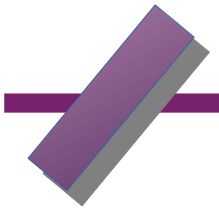
syntaxe très inspirée du C++

Exemple :

Hello.idl

```
module HelloApp {
interface Hello
{
string sayHelloTo(in string firstname, in string lastname);
};
};
```

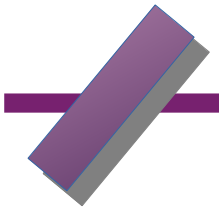
Syntaxe IDL



Eléments de syntaxe :

- commentaires du C++ (et de Java)
- preprocessing strictement identique à celui du C++(#include, #define, etc.)
- un seul espace de noms par portée
- case sensitive, mais un peu étrange :
  - si on utilise le nom TotO, on doit toujours l'écrire de cette façon
  - deux noms ne peuvent pas avoir pour seule différence la case des lettres qui les composent (i.e., si toTo est utilisé dans une portée, TOTO n'est plus permis)
- un fichier IDL est constitué de définitions, chaque définition se terminant par un point virgule

Définitions IDL



Un fichier IDL définit (ou déclare) :

- des types (principe des typedef du C++)
- des constantes (principe des const du C++) des exceptions
- des interfaces (principe des interfaces de Java)
- des modules (principe des namespaces du C++)
- des “valeurs” (types objets passés par valeur)

Types de base

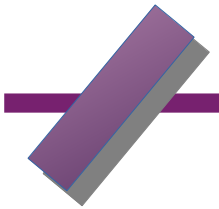
Les définitions s'appuient sur des types de base classiques :

- **entiers signés ou non (short (2 octets), long (4 octets) et long long (8 octets))**
- **réels (IEEE float, double, long double ainsi qu'un type fixed correspondant à des réels en virgule fixe)**
- **caractères (char ISO-Latin 1, un octet, et wchar pour les autres formats type UNICODE)**
- **booléen (boolean)**

Définition de types

exactement le même principe qu'en C/C++

Module



module nom {... } ;

- les modules CORBA ont pour rôle d'organiser les autres définitions
- version CORBA des
  - package Java
  - namespace C++
- structure hiérarchique
- noms complets : principe du C++, séparateur ::. Par

exemple : HelloApp::Hello

- un module crée un espace de noms

Déclaration des opérations

- **Une opération(similaires aux méthodes de Java et C++) se définit par une signature qui comprend le nom de l'opération, le type du résultat, la liste des paramètres et la liste des exceptions éventuellement déclenchées lors de l'invocation. Un paramètre se caractérise par un mode de passage, un type et un nom formel. Les modes de passages autorisés sont in , out et inout . Le résultat et les paramètres peuvent être de n'importe quel type exprimable en IDL. Par défaut,**
- **une opération peut lever une ou plusieurs exceptions, ce qui s'indique par une clause raises (similaire à throws)**
- **l'invocation d'une opération est synchrone (c'est-à-dire bloquant). Cependant, il est possible de spécifier qu'une opération est asynchrone (oneway),**

Déclaration des opérations

Exemple

```
interface I1 {
unsigned short obtenir_nombre_articles ();
float obtenir_prix_article (in long num_article) ;
void operX (in T1 a, out T2 b, in/out T3 c) ;
}
interface AnnuairePersonnel {
exception existeDeja {T_Nom nom);} ;
exceptiion inconnu {T_Nom nom);} ;
AjouterPersonne (in T_Personne personne) raises (existeDeja) ;
T_Personne ObtenirInfoPersonne (in T_Nom nom) raises (inconnu) ;
Void ModifierPersonne (in T_Nom nom, in T_Personne personne)
raises (inconnu) ;
}
```

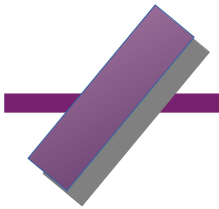
Déclaration d'attributs

En plus des opérations (que l'on peut invoquer), une interface peut définir des attributs (des variables) accessibles (c'est-à-dire qui peuvent être lus ou écrits) par un appelant. Le mot clé readonly signifie que l'attribut est accessible en lecture seulement. Il faut noter que l'on peut se passer des attributs en définissant des variables internes auxquelles on associe des fonctions de lecture et écriture.

Exemple

```
interface Interf1 {
enum T_materiau { ruban, glace };
struct T_position {float x, y; };
attribute float radius;
attribute T_materiau materiau ;
readonly attribute T_position position ;
};
```

Accès à l'ORB



Pour pouvoir écrire un programme CORBA (client ou serveur), il faut accèder au "noyau" CORBA, c'est-à-dire à l'ORB :

- **-l'ORB est une interface CORBA (décrite en PIDL) qui correspond à un pseudo objet CORBA**
- **pseudo objet : "truc" spécifié en pseudo IDL qui ne correspond pas toujours à un vrai objet CORBA (dépend du mapping) mais qui s'utilise en général comme un objet CORBA**
- **propose une opération de bootstrapping, init, qui permet d'obtenir une référence vers l'ORB (après l'avoir initialisé)**

Services de l'ORB

L'ORB propose :

- une résolution simple (resolve_initial_references) : associe des noms à des références importantes. Par exemple, les deux objets les plus importants :
  - **NameService** : le service de publication/découverte de CORBA
  - **RootPOA** : l'Object Adapter principal
- une conversion (bidirectionnelle) entre références et chaînes de caractères portables en ORB (string_to_object_to_string)

une opération de connexion **(connect)** d'un domestique (servant) à l'ORB, ce qui à pour effet de rendre disponible

une implémentation d'un objet

- beaucoup d'autres opérations, essentiellement destinées

à l'aspect dynamique de CORBA (découverte d'interfaces,

etc.)

Object

Object est la représentation d'une reférence vers un objet CORBA

l'interface définit diverses opérations utiles :

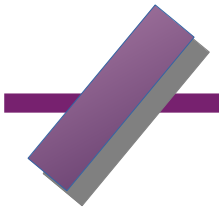
- gestion de l'aspect dynamique :
  - obtention d'une description de l'interface de l'objet référencé
  - création d'un objet Request
- manipulation de la référence (copie, association à un objet, implémentation d'une interface, etc.)

Processus de développement

1. **Ecriture de l'interface du serveur en IDL**
2. **Traduction de l'interface en :**
   - **l'équivalent dans le langage visé (par exemple une interface en Java)**
   - **des stubs pour les clients**
   - **des skeletons pour le serveur**
3. **Développement indépendant du serveur et des clients**

Il faut connaître la traduction pour pouvoir programmer !

Cas de Java



Nombreuses implémentations de CORBA pour Java :

- **Le jdk 1.4 (spécifications 2.3)**
- **Openorb**
- **JacORB (spécifications 2.4)**

pleins de choses commerciales...

Exemple traité ici : le jdk 1.4. Outil de base : idlj pour la traduction d'une interface idl en Java. Quelques éléments du mapping :

- **module CORBA → package Java**
- **interface <interface> → interface Java**

<interface>Operations mais aussi <interface>

Exemple

Hello.idl

```
module HelloApp {
interface Hello
{
string sayHelloTo(in string firstname,
in string lastname);
};
};
```

HelloApp/HelloOperations

```
package HelloApp;
public interface HelloOperations
{
String sayHelloTo (String firstname, String lastname);
}
```

Programmation du domestique

en utilisant le POA

idlj -fall Hello.idl engendre (entre autre) :

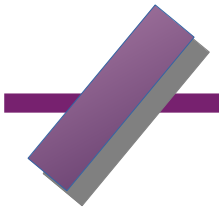
- HelloOperations.java
- HelloPOA.java : objet de base dont on hérite pour développer le serveur

Exemple :

HelloImpl

```
import HelloApp.*;
public class HelloImpl extends HelloPOA {
public String sayHelloTo (String firstname,
String lastname) {
return "Bonjour "+firstname+" "+lastname;
}
}
```

Démarrage du serveur



1. ORB.init : initialisation de l'ORB

2. obtention du RootPOA (avec resolve_initial_references) et activation de celui-ci grâce à son POAManager

3. mise en place du domestique :

(a) création de l'objet

(b) enregistrement auprès du POA

(c) obtention du NamingContextExt

(d) enregistrement de l'objet dans Naming Service

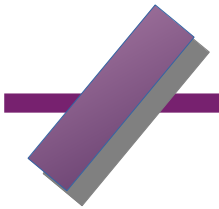
Exemple: HelloStart

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
public class HelloStart {
public static void main(String args[]) {
try{
// initialisation de l'ORB
ORB orb = ORB.init(args, null);
// obtention du POA racine
POA rootpoa =POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
// activation de celui-ci
rootpoa.the_POAManager().activate();
HelloImpl helloImpl = new HelloImpl();
// enregistrement aupr`es du POA
org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
```

Exemple(2): HelloStart

```
// obtention du Naming Service
NamingContextExt ncRef =NamingContextExtHelper.narrow(
orb.resolve_initial_references("NameService"));
// enregistrement du serveur
NameComponent path[] = ncRef.to_name("Hello");
ncRef.rebind(path, ref);
System.out.println("D´emarrage r´eussit");
orb.run();
} catch (Exception e) {
System.err.println("Erreur : " + e);
e.printStackTrace(System.out);
}
System.out.println("Termin´e !");
}
}
```

Démarrage effectif



- **il faut démarrer un serveur de nom !**
- **avec le jdk 1.4, on utilise orbd**
- **option importante : -ORBInitialPort. Précise le port du serveur de nom**
- **orbd est un serveur persistant : si on le redémarre, les associations ne sont pas perdues**
- **démarrage du serveur : java HelloStart**

  -ORBInitialPort 1050 (même option !)

Le Client

- **mêmes grands principes que le démarrage du serveur**
- **synopsis :**
  1. **initialisation de l'ORB**
  2. **obtention du NamingContextExt**
  3. **récupération de la référence vers l'objet cherché**
  4. **cast (narrow) et appel des méthodes voulues**
- **démarrage effectif : java HelloClient -ORBInitialPort**
- **1050**

Exemple: HelloClient

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class HelloClient {
public static void main(String args[]) {
try{
ORB orb = ORB.init(args, null);
NamingContextExt ncRef =NamingContextExtHelper.narrow(
orb.resolve_initial_references("NameService"));
Hello helloImpl = HelloHelper.narrow(ncRef.resolve_str("Hello"));
System.out.println(helloImpl.sayHelloTo("Joe","Bob"));
} catch (Exception e) {
System.out.println("Erreur : " + e) ;
e.printStackTrace(System.out);
}
```