Intergiciels pour la répartition
CORBA : Common Object Request Broker

Patrice Torguet
torguet@irit.fr
Université Paul Sabatier

Plan du cours

- Introduction à CORBA
- Architecture de l'ORB
- Implémentation de l'ORB
- Interopérabilité entre ORBs
- IDL
- Un Adapteur d'Objets : le POA
- Un service : le CosNaming
- Fonctionnement statique
- Exemple complet

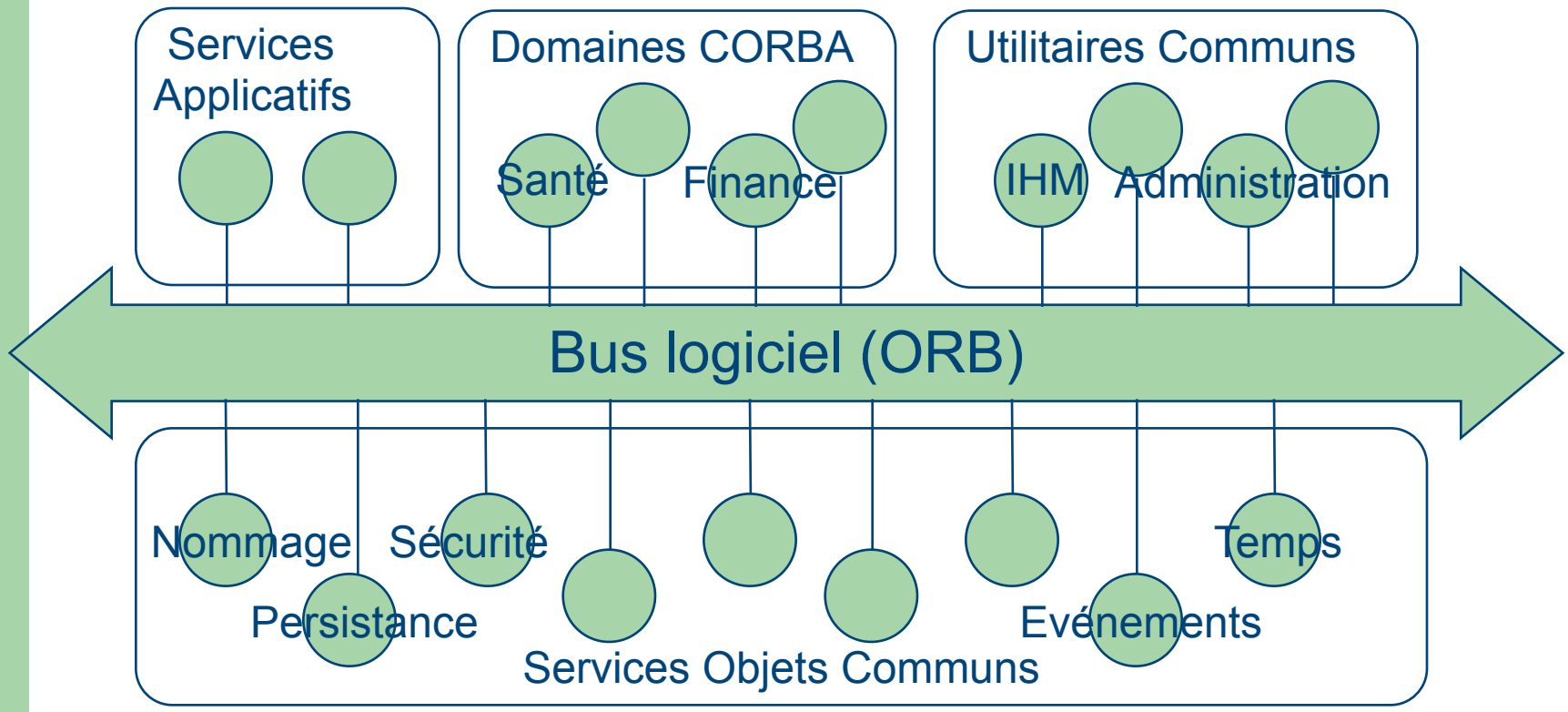
Introduction à CORBA

- RPC orientés objets + nombreux services
  - Invocation de méthodes sur des objets distribués
  - Indépendant du langage => utilisation d'un IDL (Interface Definition Language)
- Outils
  - Génération des souches/squelettes depuis l'IDL
  - Service de nommage complet, service de trading (annuaire page jaunes)
  - Gestion des événements, de la persistance, du temps-réel, du streaming...
- Multi-langage et multi-plateforme
  - langages gérés : C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python et IDLscript.
- Interopérabilité entre plusieurs ORBs : GIOP/IIOP

Introduction à CORBA

- L'OMA (Object Management Architecture) de l'OMG (Object Management Group)
  - L'OMG est un consortium international qui regroupe un très grand nombre d'organisations et d'entreprises.
  - Il définît un ensemble de standards qui tournent autour des technologies objet : UML, MOF, MDA, CORBA...
  - Il a définit une architecture distribuée orientée objet complète appelée l'OMA qui présente un ensemble de services communiquant entre eux via un bus logiciel l'ORB (Object Request Broker - Broker = Commissionnaire /Courtier).

Introduction à CORBA (l'OMA)



Introduction à CORBA

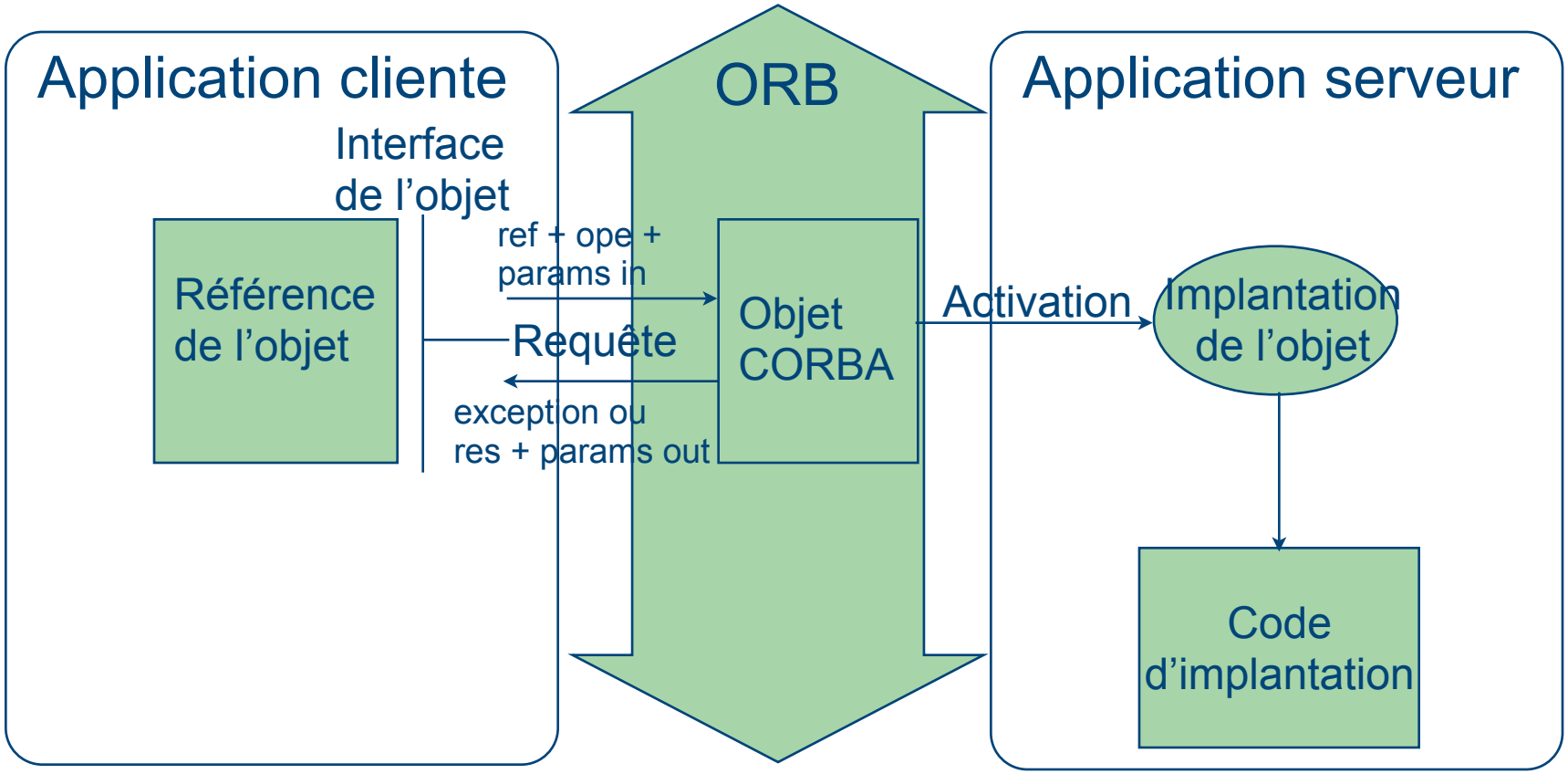
- Les objets applicatifs sont spécifiques à une application donnée (objets métiers spécifiques)
- Les domaines CORBA contiennent des services génériques pour un domaine d'application particulier (objets métiers génériques)
- Les services objets communs (Common Object Services) sont des services génériques orientés vers la distribution de l'application
- Les utilitaires communs (CORBA Facilities) sont des services communs à plusieurs applications

Introduction à CORBA

- Caractéristiques du bus logiciel
  - liaison avec les langages de programmation
  - transparence des invocations (même processus, même machine, autre machine)
  - invocations statiques (souches/squelettes) ou dynamiques (utilisation de référentiels d'interfaces)
  - système auto-descriptif : interfaces définies pour l'ORB, les référentiels, les adaptateurs d'objets...
  - activation automatique des objets
  - communication entre différents bus (protocoles GIOP et IIOP)

Introduction à CORBA

- Fontionnement du bus



Introduction à CORBA

- Fontionnement du bus
  - l'application cliente invoque des traitements sur des objets distants
  - la référence de l'objet est une structure de donnée désignant l'objet CORBA.
  - l'interface de l'objet (en IDL) définit un type abstrait de données (ensemble d'attributs et d'opérations). Elle contient la liste des signatures des opérations (nom, type de retour, type et noms des paramètres, types de passage de paramètre - in, out et inout - et exceptions)

Introduction à CORBA

- Fontionnement du bus
  - la requête est le mécanisme d'invocation d'une opération sur un objet. Elle contient, lors de l'émission, la référence de l'objet, le nom de l'opération et les valeurs des paramètres en entrée. Elle est transportée par le bus jusqu'à l'objet. En retour, elle contiendra soit une exception soit le résultat et les paramètres en sortie.
  - l'objet CORBA est un composant logiciel qui traite les requêtes
  - le processus d'activation permet d'associer une implantation à un objet CORBA (ce peut être dynamique)

Introduction à CORBA

- Fontionnement du bus
  - l'implantation de l'objet est soit une instance d'objet d'un langage à objets, une structure de donnée (pour C par exemple) ou encore un objet d'une BDOO. Elle stocke l'état de l'objet et fournit une implantation pour chaque opération.
  - le code d'implantation contient le traitement associé à chaque opération. Ce peut être un ensemble de fonctions C ou une classe Java, C++...
  - l'application serveur est une structure d'accueil pour des implantations d'objets et le code de ces objets.

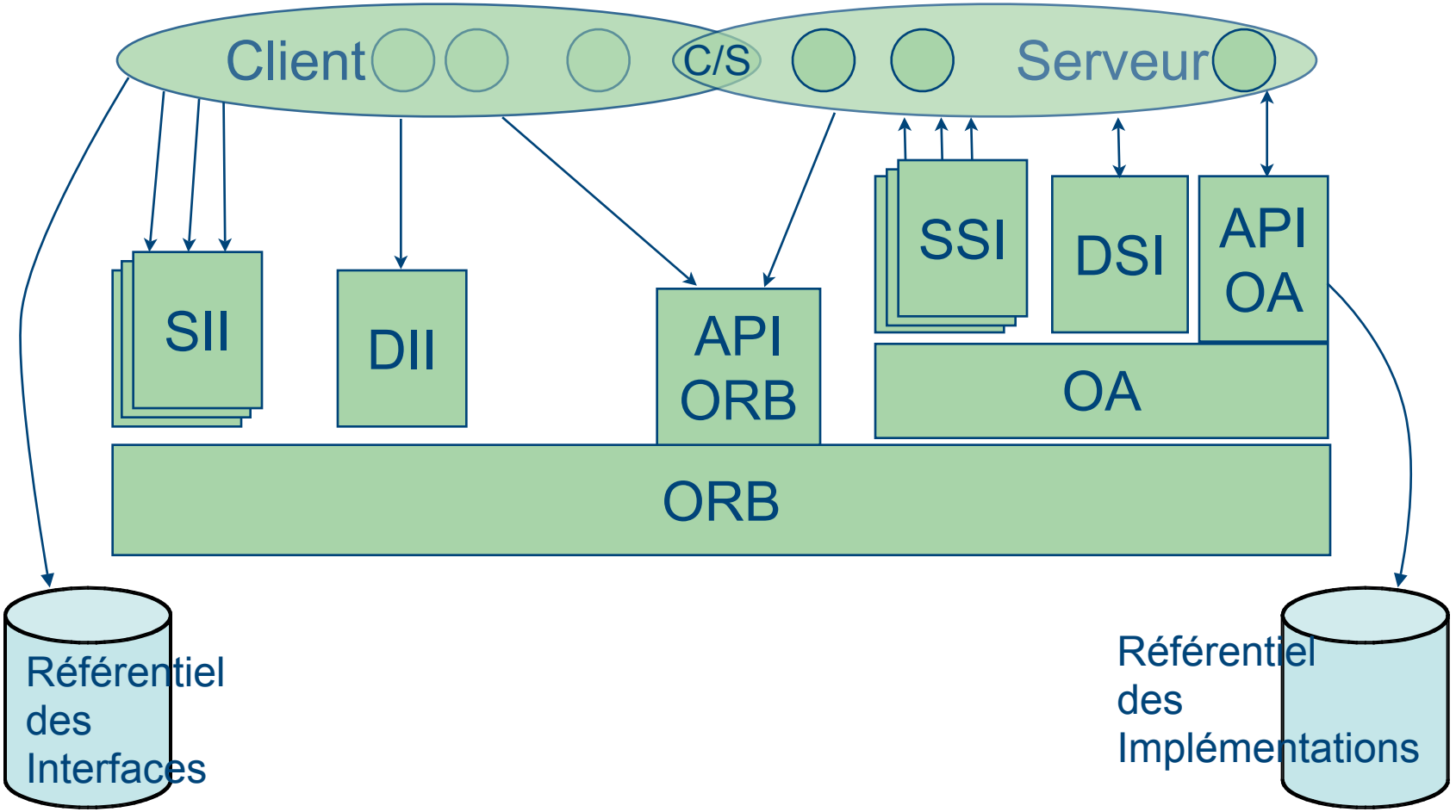
Introduction à CORBA

- L'IDL (Interface Definition Language)
  - C'est un langage qui permet de définir les interfaces des objets CORBA.
  - Il est indépendant de tout langage de programmation.
  - Il y a plusieurs modèles de traduction pour chaque langage de programmation.
  - Il offre des mécanismes de réutilisation par héritage typiques des langages à objets

Introduction à CORBA

- L'IDL (Interface Definition Language)
  - Il permet de définir :
    - les types des paramètres, résultats d'opérations et des attributs
    - les attributs et les opérations des interfaces
    - des exceptions
    - des constantes
    - des modules qui rassemblent des types et des interfaces

Les composants du Bus CORBA



Les composants du Bus CORBA

- Référentiel des interfaces
  - Permet de stocker les méta données (en IDL) décrivant les interfaces des objets
- SII : Interface d'Invocations Statiques
  - Résultat de la projection des interfaces IDL dans un langage donné. A chaque interface est associée une SII ou souche pour un langage donné. La souche permet d'invoquer les opérations à distance via des requêtes.
  - Une souche emballe les invocations (ref, id méthode et paramètres), utilisent l'ORB pour transmettre les requêtes d'invocation et déballent les résultats et paramètres out

Les composants du Bus CORBA

- DII : Interface d'Invocations Dynamiques
  - Permet de construire dynamiquement des requêtes vers des objets. Utilisé lorsqu'on n'a pas de souche.
- Interface de l'ORB
  - Permet d'accéder aux services généraux de l'ORB : initialisation, gestion des références...
- OA : Adaptateurs d'Objets
  - Bloc fonctionnel qui se charge de l'exécution des requêtes. Il y en a plusieurs de définis et le plus utilisé est maintenant le POA.

Les composants du Bus CORBA

- SSI : Interface de Squelettes Statiques
  - Les squelettes statiques sont le pendant des souches, ils déballent les requêtes d'invocation, exécutent les méthodes des objets d'implémentation et emballent les résultats à destination des clients
- DSI : Interface de Squelettes Dynamiques
  - Pendant du DII. Permet de gérer les requêtes d'invocations lorsqu'on n'a pas de squelette statique. Utilisé pour IDLscript et pour des passerelles.
- Référentiel des implémentations

Les composants du Bus CORBA

- Référentiel des implémentations
  - Contient l'ensemble des informations concernant l'implantation des objets (nom des exécutables contenant le code des objets, leurs politiques d'activation, les droits d'accès...)

Implémentation de l'ORB

- Diverses implémentations possibles :
  - ORB processus : tout se passe à l'intérieur d'un même processus. On n'utilise donc que les aspects orientés objets de CORBA. Par exemple en C.
  - ORB serveur ou système d'exploitation : permet de faire communiquer plusieurs processus sur la même machine. Peut être intégré au syst. d'expl. ou géré par un serveur. Exemple : Gnome Bonobo.
  - ORB distribué : les processus clients et serveurs peuvent se trouver sur des machines différentes. Ils utilisent une bibliothèque et un ou plusieurs serveurs pour l'invocation des requêtes distantes. Le plus utilisé.

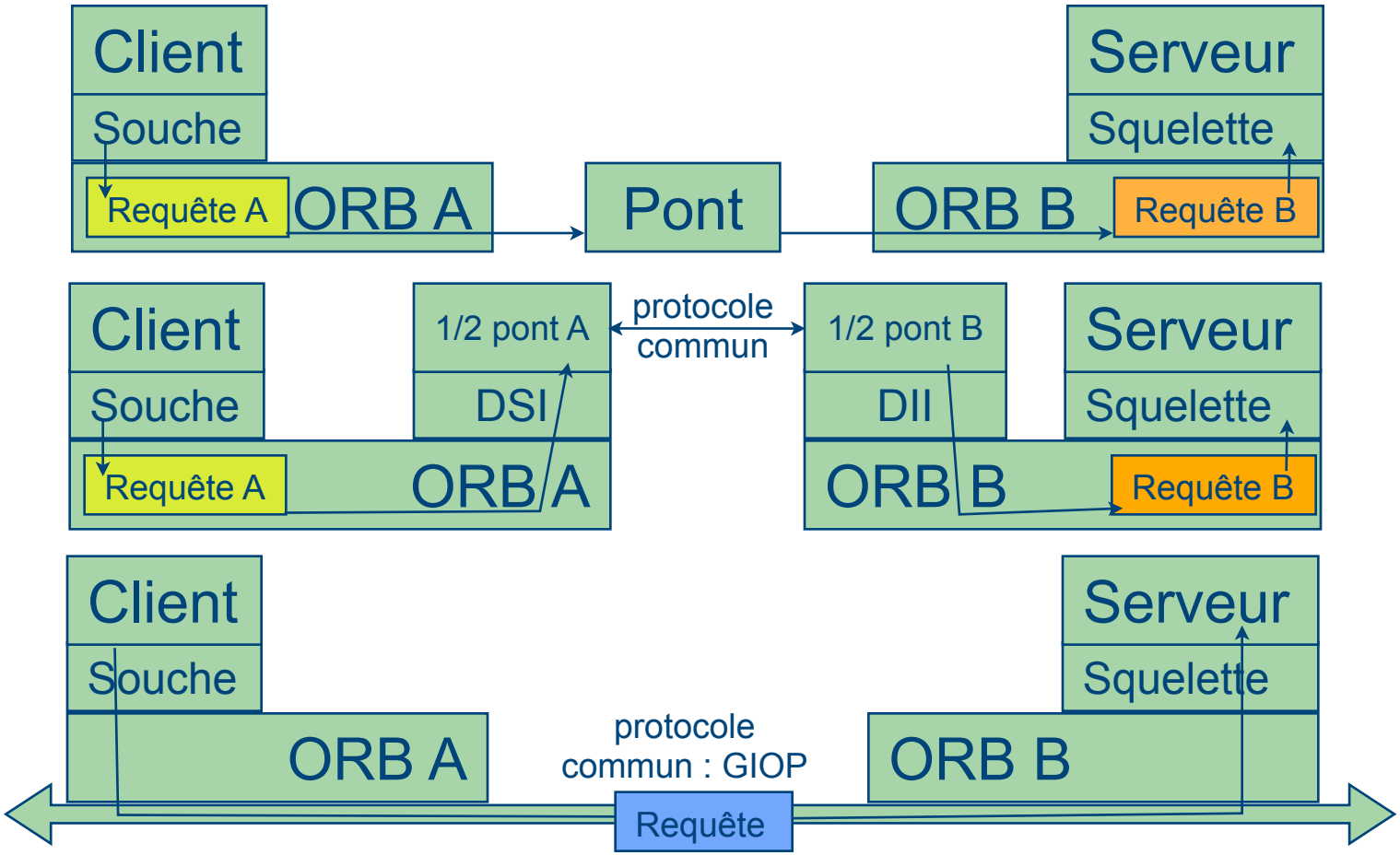
Implémentation de l'ORB

- Exemples d'ORB distribués :
  - OpenORB : ORB opensource en Java qu'on utilisera pour les TPs.
  - OmniORB : ORB opensource en C++ (permet aussi d'utiliser Python)
  - TAO (The Ace ORB) : ORB opensource en C/C++
  - Orbit : ORB opensource utilisé par Gnome
  - Orbix de la société Iona.
  - Visibroker de la société Borland.
  - ObjectBroker de la société BEA.

Interopérabilité entre ORBs

- Si client et serveur n'utilisent pas le même ORB on a 3 solutions :
  - Utilisation d'un pont de conversion de protocoles : il s'agit d'une application qui intercepte et traduit les requêtes d'un bus en requêtes d'un autre bus.
    Pb : il en faut un par couple de bus
  - Utilisation de deux demi-ponts génériques : ici on utilise les DSI et DII pour transformer les requêtes. On a un 1/2 pont par bus et un protocole commun.
  - Utilisation d'un protocole d'interopérabilité : GIOP (Generic Inter-ORB Protocol) et son implantation la plus utilisée : IIOP (Internet Inter-ORB Protocol)

Interopérabilité entre ORBs



Interopérabilité entre ORBs

- GIOP et IIOP définissent :
  - La représentation commune des données (CDR) qui offre une projection des types IDL en une représentation physique qui peut circuler sur le réseau. CDR n'utilise pas de codage à la source (comme XDR par exemple). Chaque machine envoie ces données dans son format binaire et la machine qui reçoit traduit si c'est nécessaire.
  - Les références d'objets interopérables (IOR) qui contiennent au minimum : un numéro de version du protocole utilisé, l'adresse de la machine destinatrice pour le protocole transport (ex. @IP + port pour TCP) et une clé identifiant l'objet dans le serveur

Interopérabilité entre ORBs

- GIOP et IIOP définissent :
  - Le format des messages (entêtes et données). Les protocoles définissent par exemple un message transportant une requête, un autre pour la réponse...
  - Des pré-requis sur la couche transport : spécifiant les contraintes sur le transport et l'ordonnancement des messages GIOP (on nécessite un protocole orienté connexion, fiable et ordonné sans limitation de taille - TCP pour IIOP) ainsi que la manière de gérer des connexions entre les applications (un multiplexage est possible).

IDL

- L'IDL permet de préciser :
  - Ce qui est échangé grâce à :
    - des définitions de types
    - des structures
    - des exceptions
  - Les interfaces des objets CORBA

IDL

- définitions de types
  - à partir de types de base (exemple : typedef string nom;)

Type IDL	Type Java correspondant
boolean	boolean
char	char
wchar	char
string	java.lang.String
wstring	java.lang.String
octet	byte
short	short
long	int
long long	long
float	float
double	double
unsigned short	short
unsigned long	int

IDL

- définitions de types
  - en définissant des tableaux fixes :
    typedef short tableau[3];
    typdef short matrice[3][3];
  - ou dynamiques :
    typedef sequence<float> flottants;
    typedef sequence<float, 100> flottantsMax100;
  - en définissant des énumérations :
    enum jours {lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche};

IDL

- définitions de structures
  - struct personne {
    string nom;
    string prenom;
    short age;
    };
- définition d'exceptions
  - exception Inconnu {
    string raison;
    };

IDL

- Interface d'un objet CORBA
  - On définit les méthodes à appeler à distance (et le sens de transfert des paramètres : in, out, in out)
  - On peut aussi utiliser les mots-clés attribute (resp. read-only attribute) pour définir 2 (resp. 1) méthode(s) d'accès
  - Des méthodes peuvent être asynchrones (déclarées oneway)
    - interface Horloge {
      attribute string timezone;
      readonly attribute long seconds;
      readonly attribute long microseconds;
      oneway void setTime(in long sec, in long usec) raises (Inconnu);
      };

IDL

- Interface d'un objet CORBA
  - L'héritage multiple entre interfaces est possible pour définir de nouveaux objets CORBA

```
interface Horloge2 : Horloge {
        void getTime(out long sec, out long usec);
};
```

  - Une méthode peut aussi travailler sur le type Object (objet CORBA quelconque) ou le type any (n'importe quel type, y compris structure, tableau, type simple...)

IDL

- On peut aussi :
  - Définir des constantes : const float PI = 3.14159;
  - Définir des modules (idem concept de package en Java)
  - Inclure d'autres fichiers IDL
    #include "autreIDL.idl"
  - Utiliser des directives de précompilation :
    - #define DEBUG
      #ifdef DEBUG
      attribute boolean debug;
      #endif

IDL

- Compilation IDL : génère les types, squelettes et souches dans un langage cible
- Exemple en Java
  - Soit le type IDL personne, le compilateur générera :
    - La classe personne correspondant : personne.java
    - Un fichier contenant des méthodes de classes utilitaires : personneHelper.java (exemple lecture/écriture du type sur un flot CORBA, comparaison de type, extraction depuis un any...)
    - Un fichier permettant de faire du passage par référence (out et inout) : personneHolder.java (classe qui a un attribut de type personne)

IDL

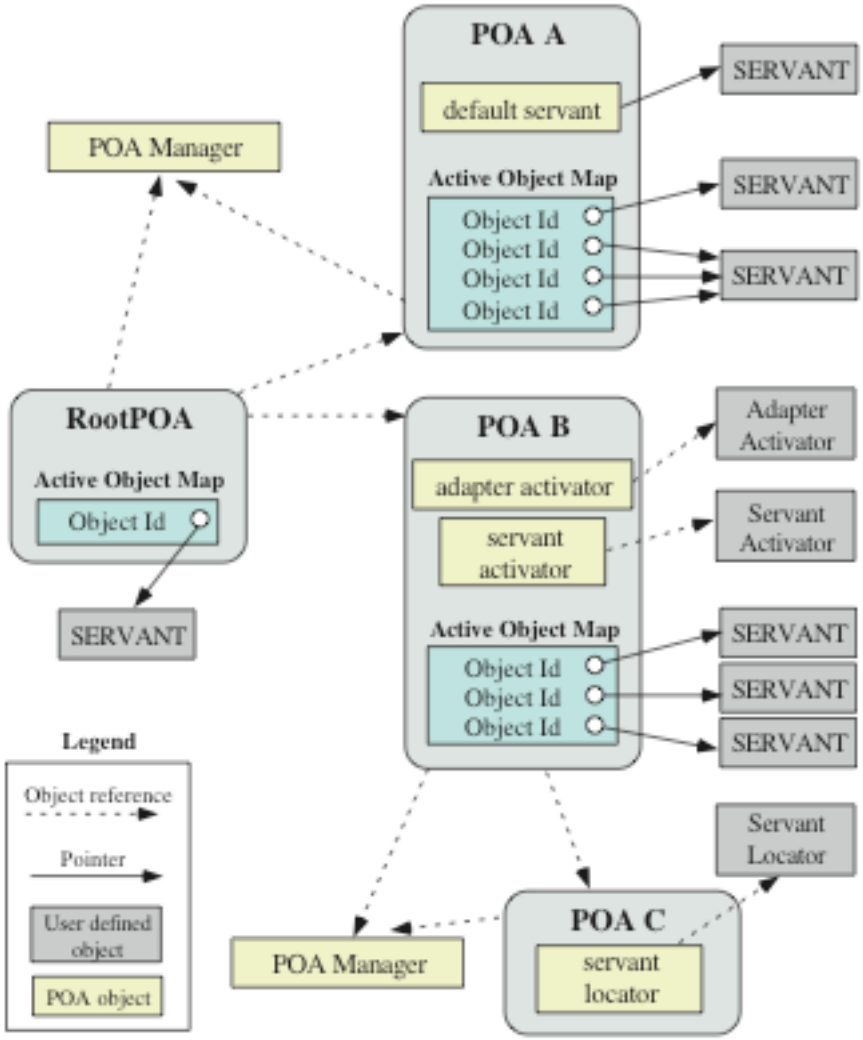
- Pour une interface IDL Horloge, le compilateur générera :
  - HorlogeOperations.java : interface java correspondant à Horloge (utilisée dans l'approche par délégation)
  - Horloge.java : hérite de la précédente (utilisée par le client)
  - HorlogePOA.java : squelette pour le POA par héritage
  - HorlogePOATie.java : squelette pour le POA par délégation
  - HorlogeHelper.java : méthodes utilitaires liées à Horloge (idem précédemment plus méthodes pour convertir un CORBA.Object en Horloge : narrow et unchecked_narrow)
  - HorlogeHolder.java : pour gérer Horloge en out ou inout
  - _HorlogeStub.java : stub client

Un Adapteur d'Objets : le POA

- Rôle : permettre l'écriture de servants (implémentation d'un objet CORBA) indépendamment d'un ORB particulier
- Les squelettes sont standardisés ainsi que les interactions OA/Servant et OA/Serveur (programme qui met en place le servant)
- Son comportement est paramétré par des politiques
- On peut utiliser des hiérarchies de POA (RootPOA est la racine et est prédéfini)

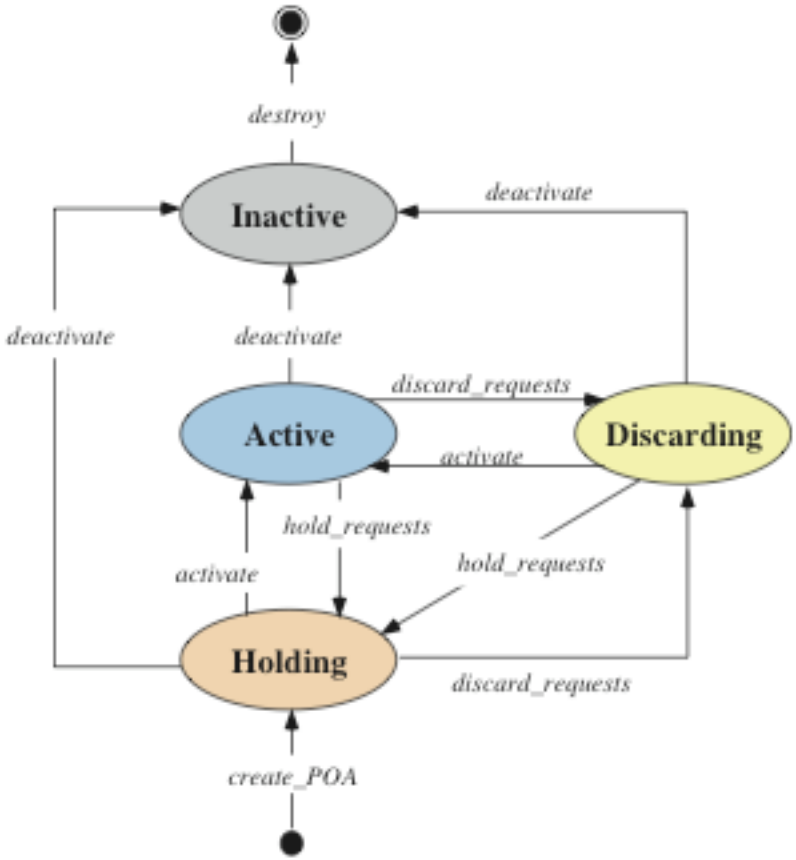
Un Adapteur d'Objets : le POA

- Architecture :



Un Adapteur d'Objets : le POA

- Cycle de vie du POA Manager



Un Adapteur d'Objets : le POA

- Le POA manager : gère la façon dont les requêtes sont traitées par un ensemble de POAs (traitées, mises en file d'attente, non traitées)
- Adapter Activator : gère les requêtes reçues pour un POA fils qui n'existe pas encore (il peut le créer à la demande)
- Servant Manager : permet de créer des servants à la demande
- Active Object Map : associe des servants aux objets CORBA

Un Adapteur d'Objets : le POA

- Politiques :
  - multi-thread ou mono-thread
  - persistance des servants
  - unicité des id d'objets à l'intérieur du POA
  - choix des id d'objets (appli/POA)
  - activation implicite des objets lors de leur création
  - stockage des associations objet CORBA/servant
  - comportement à tenir lors d'une requête sur un objet n'ayant pas de servant (erreur, servant par défaut ou manager de servants)

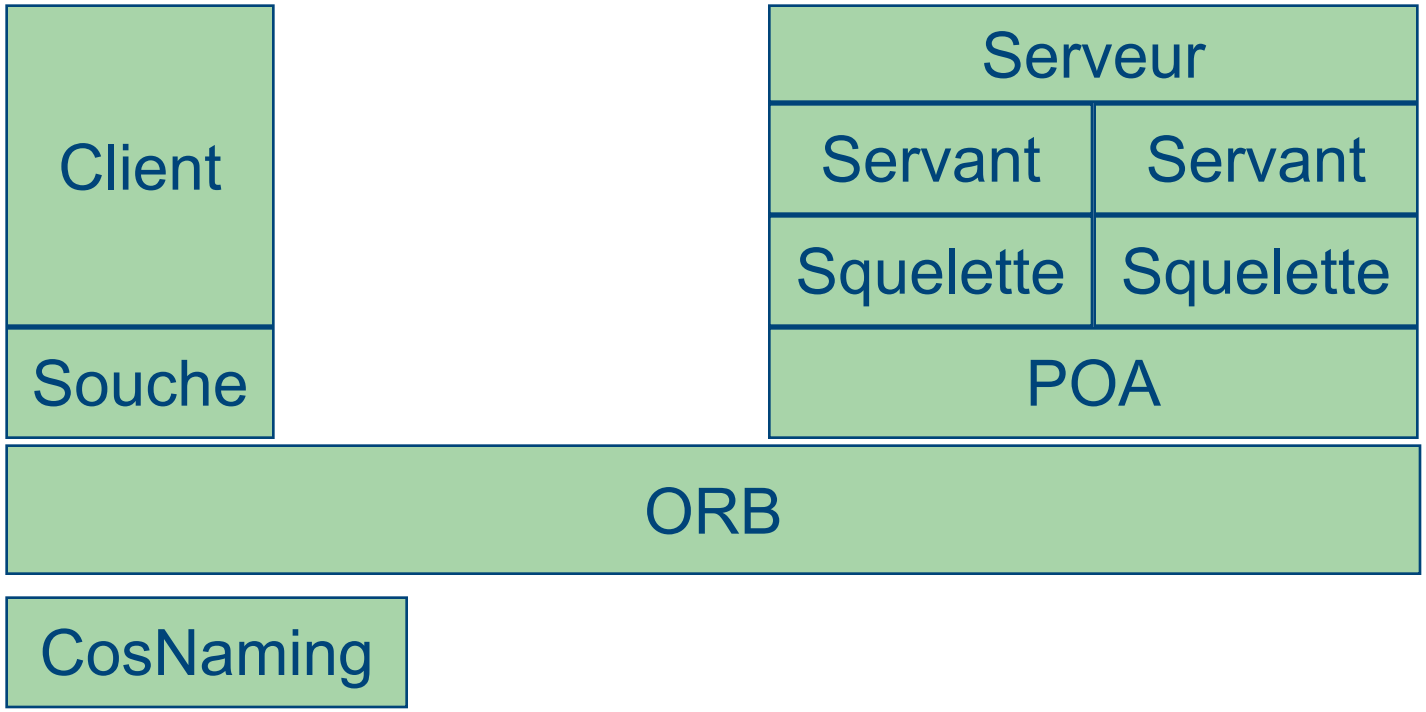
Un service : CosNaming

- Service de nommage réparti
- Définit des associations noms/IOR
- Permet de rechercher un IOR en précisant un nom
- Permet une hierarchie d'espaces de noms
- Permet une navigation des espaces de noms

Un service : CosNaming

- Namespace : un espace de nommage hiérarchique
- NamingContext : un objet CORBA qui matérialise un noeud de l'espace de nommage (il stocke les associations)
- Name : suite de NameComponents (nom simple : 1 seul composant, nom composé : plusieurs composants)
- NameComponent : deux parties id et kind (id est le nom, kind une description)

Fonctionnement statique



Fonctionnement statique

- Ecriture de la description de l'interface de l'objet CORBA en IDL
  - possibilité de décrire : types, structures, union, exception, tableaux, séquences
  - interfaces pouvant dériver les unes des autres et décrivant des attributs (en fait get/set) et méthodes implantés par l'objet CORBA
- Compilation IDL vers un (ou plusieurs) langages cibles
  - génération de la souche et du squelette ainsi que de fichiers pour gérer les types, structures... définis dans l'IDL

Fonctionnement statique

- Deux solutions pour implémenter un objet CORBA avec un langage à objets :
  - Approche par héritage : la classe du servant (implémentation de l'objet CORBA) dérive du squelette.
    - Problème dans un langage comme Java pour lequel il n'y a que de l'héritage unique...
  - Approche par délégation : on utilise deux objets : le délégué qui implante les méthodes et le squelette

Fonctionnement statique

- Fonctionnement côté serveur :
  - 1- Initialisation de l'ORB
  - 2- Initialisation du POA (ou autre OA)
  - 3- Création d'un (ou plusieurs) servant(s)
  - 4- Activation d'un servant au sein de l'OA (gestion du squelette)
  - 5- Activation du POA (passage état pending à état actif)
  - 6- Gestion des IOR (identifiants d'objets)
  - 7- Boucle de gestion des requêtes de l'ORB

Fonctionnement statique

- Fonctionnement côté client :
  - 1- Initialisation de l'ORB
  - 2- Gestion des IOR (exemple : recherche avec service de nommage)
  - 3- Utilisation de la souche pour appeler les méthodes distantes

Exemple complet

- Exemple complet :
  - module calc {
    exception DivisionParZero {};
    interface Calculatrice {
    float ajoute(in float x, in float y);
    float soustrait(in float x, in float y);
    float multiplie(in float x, in float y);
    float divise(in float x, in float y) raises
    (DivisionParZero);
    };
    };

Exemple complet

- Compilation IDL (exemple openORB) :
  - java org.openorb.compiler.IdlCompiler -verbose Calculatrice.idl -d .
  - Génère :
    - CalculatriceOperations.java : interface java correspondant à Calculatrice (utilisée dans l'approche par délégation)
    - Calculatrice.java : hérite de la précédente (utilisée par le client)
    - CalculatricePOA.java : squelette pour le POA par héritage
    - CalculatricePOATie.java : squelette pour POA par délégation
    - CalculatriceHelper.java : méthodes utilitaires liées à Calculatrice
    - CalculatriceHolder.java : pour gérer Calculatrice en out ou inout
    - _CalculatriceStub.java : stub client

Exemple complet

- DivisionParZero.java : exception java correspondante
- DivisionParZeroHelper.java : méthodes utilitaires liées
- DivisionParZeroHolder.java : pour utilisation en out ou inout

Exemple complet

- Implémentation du servant en Java par héritage :

```
public class CalculatriceImpl extends calc.CalculatricePOA
{
      public float ajouter(float x, float y)
      {
         return a1 + a2;
      }

     ...
      public float diviser(float x, float y) throws DivisionParZero {
           if(y==0) throw new DivisionParZero();
           return x / y;

      }
}
```

Exemple complet

- Implémentation du serveur en Java :

```
import org.omg.CosNaming.*;
public class Server {
        public static void main(String[] args) throws Exception {
//          Initialisation de l'ORB
            org.omg.CORBA.ORB orb = org.openorb.CORBA.ORB.init(args,null);
//          Récupération du POA racine
            org.omg.PortableServer.POA rootPOA =
org.omg.PortableServer.POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
//                Creation d'une instance de l'objet distant
            CalculatriceImpl objet = new CalculatriceImpl();
//                Activation de l'objet dans le POA
            byte[] monId = rootPOA.activate_object(objet);
//                Activation du POA
            rootPOA.the_POAManager().activate();
...
```

Exemple complet

- Implémentation du serveur en Java :

```
//          Recuperation de la reference du service de nommage
            org.omg.CORBA.Object ns = orb.resolve_initial_references("NameService");
//          Extraction d'un objet NamingContext correspondant
            NamingContext naming = NamingContextHelper.narrow(ns);
//          On associe l'objet au nom MaCalculatrice
            NameComponent[] name = new NameComponent[1];
            name[0] = new NameComponent();
            name[0].id="MaCalculatrice";
            name[0].kind="";
            naming.bind(name,rootPOA.servant_to_reference(objet));
            String IOR = orb.object_to_string(rootPOA.servant_to_reference(objet));
            System.out.println("Server running...\n"+IOR);
//          on attends la fin de l'exécution de l'ORB (attente infinie ici)
            orb.run();
        }
}
```

Exemple complet

- Implémentation du servant en Java par délégation :

```
public class CalculatriceImpl implements calc.CalculatriceOperations
{
      public float ajouter(float x, float y)
      {
         return a1 + a2;
      }

     ...
      public float diviser(float x, float y) throws DivisionParZero {
           if(y==0) throw new DivisionParZero();
           return x / y;

      }
}
```

Exemple complet

- Implémentation du serveur :

```
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContext;
import org.omg.CosNaming.NamingContextHelper;

public class Server {

        public static void main(String[] args) throws Exception {
//          Initialisation de l'ORB
           org.omg.CORBA.ORB orb = org.openorb.CORBA.ORB.init(args,null);
//          Récupération du POA racine
           org.omg.PortableServer.POA rootPOA =
org.omg.PortableServer.POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
//                Creation d'une instance de l'objet CORBA
           CalculatriceImpl délégué = new CalculatriceImpl();
           CalculatricePOATie squelette = new CalculatricePOATie(délégué);
//                Activation de l'objet dans le POA
           byte[] monId = rootPOA.activate_object(squelette);
//                Activation du POA
           rootPOA.the_POAManager().activate();

...
```

Exemple complet

- Implémentation du serveur :

```
//          Recuperation de la reference du service de nommage
            org.omg.CORBA.Object ns = orb.resolve_initial_references("NameService");
//          Extraction d'un objet NamingContext correspondant
            NamingContext naming = NamingContextHelper.narrow(ns);
//          On associe l'objet au nom MyWatch
            NameComponent[] name = new NameComponent[1];
            name[0] = new NameComponent();
            name[0].id="MaCalculatrice";
            name[0].kind="";
            naming.bind(name,rootPOA.servant_to_reference(squelette));
            String IOR = orb.object_to_string(rootPOA.servant_to_reference(squelette));
            System.out.println("Server running...\n"+IOR);
//          on attends la fin de l'exécution de l'ORB (attente infinie ici)
          orb.run();
        }
}
```

Exemple complet

- Implémentation du client en Java :

```
public class Client {

        public static void main(String[] args) throws Exception {
//          Initialisation de l'ORB
            org.omg.CORBA.ORB orb = org.openorb.CORBA.ORB.init(args,null);
//          Recuperation de la reference du service de nommage
            org.omg.CORBA.Object ns = orb.resolve_initial_references("NameService");
//          Extraction d'un objet NamingContext correspondant
            org.omg.CosNaming.NamingContext naming = org.omg.CosNaming.
            NamingContextHelper.narrow(ns);
//          On va rechercher la reference a l'objet distant a partir de son nom
            org.omg.CosNaming.NameComponent[] name = new org.omg.
            CosNaming.NameComponent[1];
            name[0] = new org.omg.CosNaming.NameComponent();
            name[0].id="MaCalculatrice";
            name[0].kind="";
            org.omg.CORBA.Object obj = naming.resolve(name);
...
```

Exemple complet

- Implémentation du client en Java :

```
                // On en "extrait" l'objet calculatrice correspondant
                calc.Calculatrice c = calc.CalculatriceHelper.narrow(obj);

                // On effectue l'operation
                System.out.println(c.ajoute(12,5));
        }
}
```

Remerciements et Questions ?

- Ce cours est basé sur :
  - Le cours sur CORBA de Sun : http://java.sun.com/developer/onlineTraining/corba/
  - Le cours sur CORBA de Thierry Desprats (IRIT)
  - Le livre “CORBA des concepts à la pratique” de JM Geib, C Gransart et P Merle
  - Le livre “Au coeur de CORBA avec Java” de J. Daniel
  - Les documents du site web de l’OMG : http://www.omg.org
  - L’article “An Overview of the CORBA Portable Object Adapter” d’I. Pyarali et D. C. Schmidt http://www.cse.wustl.edu/~schmidt/PDF/POA.pdf