Objets distribués - Corba - Code Mobile

Damien Olivier

Damien.Olivier@univ-lehavre.fr

Laboratoire d'informatique du Havre

Introduction

On évoque les technologies actuelles permettant la distribution d'objets dans des environnements de réseaux informatiques : les objets distribués avec Java/RMI, les brokers d'objets à la norme CORBA, les agents mobiles, les applications aux au e-commerce.

Organisation du cours

1. Concepts sur les systèmes distribués à objets et l'IDL/CORBA *(Lundi 20/01/2003 - D. Olivier)*
2. Les bases de CORBA : ORB, projections, invocation statique *(Mardi 21/01/2003 - D. Olivier)*
3. Service de nommage, adaptateur d'objet (POA) et référentiel d'interfaces en CORBA *(Vendredi 24/01/2003 - D. Olivier)*
4. Les mécanismes dynamiques en CORBA *(Lundi 27/01/2003 - D. Olivier)*
5. Codes mobiles et Aglets *(Mardi 28/01/2003 - D. Olivier)*
6. Applications professionnelles : le e-commerce (1) *(Lundi 18/02/2003 - B. Adouobo)*
7. Applications professionnelles : le e-commerce (2) *(Lundi 19/02/2003 - B. Adouobo)*
8. Applications professionnelles : le e-commerce (3) *(Mardi 20/02/2003 - B. Adouobo)*

Bilio / Webographie de base

- OMG : `http://www.omg.org`
- Livres :

“Objets répartis, guide de survie”

“Client server programming with Java & Corba”

Orfali, Edwards, Harley - ITP France

“CORBA : des concepts à la pratique”

Geib, Gransart, Merle - Masson 97

“The CORBA reference guide”

A. Pope - Addison Wesley 97

“Au cœur de Corba avec Java”

J. Daniel - Vuibert 2000

- Web :

ORBacus `http://www.ooc.com`

CorbaScript `http://corbaweb.lifl.fr`

D. Schmidt Page perso `http://www.cs.wustl.edu/~schmidt/`

U. Vienne `http://www.infosys.tuwien.ac.at/Research/Corba/`

Portail `http://www.cetus-links.org/oo_corba.html`

5.1 Serveur de noms

5.1.1 A quoi sert un serveur de noms ?

5. Serveurs de noms, POA et IFR

5.1 Serveur de noms

5.1.1 A quoi sert un serveur de noms ?

5.1.2 Organisation arborescente du serveur de noms

5. Serveurs de noms, POA et IFR

5.1 Serveur de noms

5.1.1 A quoi sert un serveur de noms ?

5.1.2 Organisation arborescente du serveur de noms

5.1.3 Mise en œuvre en C++

5. Serveurs de noms, POA et IFR

5.1 Serveur de noms

5.1.1 A quoi sert un serveur de noms ?

5.1.2 Organisation arborescente du serveur de noms

5.1.3 Mise en œuvre en C++

5.1.4 Implémentation en Java

5.1.1 A quoi sert un serveur de noms ?

- Jusqu'à présent, utilisation nécessaire d'un fichier `compteur.ref` commun au serveur et au client : il permet l'échange des références des objets ... Peu adapté à des applications distribuées sur plusieurs machines ! !

5.1.1 A quoi sert un serveur de noms ?

- Jusqu'à présent, utilisation nécessaire d'un fichier `compteur.ref` commun au serveur et au client : il permet l'échange des références des objets ... Peu adapté à des applications distribuées sur plusieurs machines ! !
- Plusieurs solutions

5.1.1 A quoi sert un serveur de noms ?

- Jusqu'à présent, utilisation nécessaire d'un fichier `compteur.ref` commun au serveur et au client : il permet l'échange des références des objets ... Peu adapté à des applications distribuées sur plusieurs machines ! !
- Plusieurs solutions
  - Tranférer le fichier "manuellement" par ftp par exemple ;

5.1.1 A quoi sert un serveur de noms ?

- Jusqu'à présent, utilisation nécessaire d'un fichier `compteur.ref` commun au serveur et au client : il permet l'échange des références des objets ... Peu adapté à des applications distribuées sur plusieurs machines ! !
- Plusieurs solutions
  - Tranférer le fichier "manuellement" par ftp par exemple ;
  - Utiliser un filesystem réseau (NFS) ;

5.1.1 A quoi sert un serveur de noms ?

- Jusqu'à présent, utilisation nécessaire d'un fichier `compteur.ref` commun au serveur et au client : il permet l'échange des références des objets ... Peu adapté à des applications distribuées sur plusieurs machines ! !
- Plusieurs solutions
  - Tranférer le fichier "manuellement" par ftp par exemple ;
  - Utiliser un filesystem réseau (NFS) ;
  - Récupérer le fichier en utilisant HTTP et les URL (cf. TP) ;

5.1.1 A quoi sert un serveur de noms ?

- Jusqu'à présent, utilisation nécessaire d'un fichier `compteur.ref` commun au serveur et au client : il permet l'échange des références des objets ... Peu adapté à des applications distribuées sur plusieurs machines ! !
- Plusieurs solutions
  - Tranférer le fichier "manuellement" par ftp par exemple ;
  - Utiliser un filesystem réseau (NFS) ;
  - Récupérer le fichier en utilisant HTTP et les URL (cf. TP) ;
  - Utiliser un objet notoire CORBA qui joue le rôle d'annuaire.

5.1.1 A quoi sert un serveur de noms ?

- Jusqu'à présent, utilisation nécessaire d'un fichier `compteur.ref` commun au serveur et au client : il permet l'échange des références des objets ... Peu adapté à des applications distribuées sur plusieurs machines ! !
- Plusieurs solutions
  - Tranférer le fichier "manuellement" par ftp par exemple ;
  - Utiliser un filesystem réseau (NFS) ;
  - Récupérer le fichier en utilisant HTTP et les URL (cf. TP) ;
  - Utiliser un objet notoire CORBA qui joue le rôle d'annuaire.
- Un serveur de nom ou annuaire va permettre d'associer un nom identifiant l'objet réparti (analogue à un DNS).

5.1.1 A quoi sert un serveur de noms ?

Serveur de noms local ou distribué ?

- Il peut être local sur la machine faisant tourner serveur et client

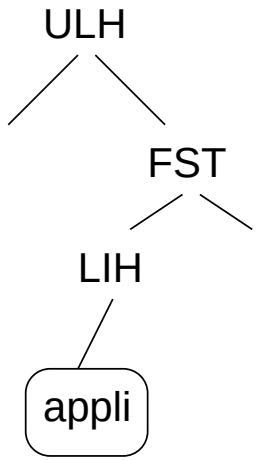
5.1.1 A quoi sert un serveur de noms ?

Serveur de noms local ou distribué ?

- Il peut être local sur la machine faisant tourner serveur et client
- ou il peut être distribué et invoqué à distance : on peut alors réellement mettre en œuvre des applications réparties sur un réseau.

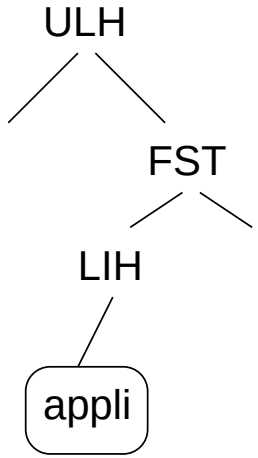
5.1.2 Organisation arborescente du serveur de noms

- Désignation des objets CORBA à l'aide de noms ;



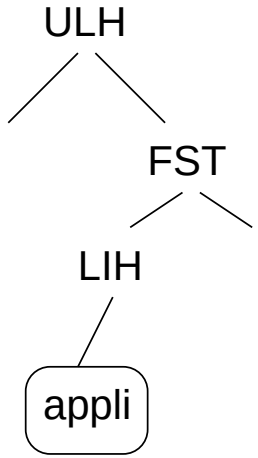
5.1.2 Organisation arborescente du serveur de noms

- Désignation des objets CORBA à l'aide de noms ;
- Un nom est une séquence ordonnées de composants correspondant à une suite de contextes et se terminant par le nom de l'objet dans ces contextes.



5.1.2 Organisation arborescente du serveur de noms

- Désignation des objets CORBA à l'aide de noms ;
- Un nom est une séquence ordonnées de composants correspondant à une suite de contextes et se terminant par le nom de l'objet dans ces contextes.
- Il s'agit d'une organisation arborescente similaire à celle des répertoires et des fi chiers dans un système d'exploitation.



5.1.3 Mise en œuvre en C++

- On conserve dans l'état les fi chiers précédents décrivant le service proposé, à savoir :
  - `"compteur.idl"`
  - `"compteur_impl.h"`
  - `"compteur_impl.cpp"`
- L'utilisation du serveur de nom intervient au niveau du gestionnaire de service qui doit enregistrer le ou les objets créés, ainsi qu'au niveau du client pour l'interroger et obtenir une référence de l'objet à invoquer.

5.1.3 Mise en œuvre en C++

Quelques opérations principales de `NamingContext`

- Ajouter une association : `bind`, `bind_new_context`
- Résoudre une association : `resolve`
- Détruire une association : `unbind`
- Lister le contenu : `list`
- Détruire le contexte : `destroy`

5.1.3 Mise en œuvre en C++

Enregistrement d'un servant dans le serveur de noms

- On commence par récupérer l'objet *notoire* correspondant au serveur de noms :

```
CORBA::Object_var objet = orb->resolve_initial_references("NameService");
CosNaming::NamingContext_var ns = CosNaming::NamingContext::_narrow(objet);
```

- Pour le serveur, on construit deux objets de type `CosNaming::Name`
  - Le premier permet de définir un contexte où on range l'objet.
  - Le second permet de ranger l'objet service dans ce contexte.
- Un objet de type `CosNaming::Name` contient un tableau d'identificateurs pouvant être classé. On le définit en précisant :
  - La taille du tableau avec la méthode `length()` ;
  - L'identificateur de chaque élément du tableau avec le champ `id` ;
  - La classe à laquelle appartient cet élément avec le champ `kind`.
- Il faut ensuite l'insérer dans le serveur de nom avec
  - la fonction `bind_new_context`, si cet objet décrit un contexte ;
  - la fonction `bind`, si cet objet contient un service.
- Dans ce dernier cas, il faut donc que cette insertion contienne aussi l'adresse (ou référence) du service qui pourra être invoqué par un client.

5.1.3 Mise en œuvre en C++

Enregistrement d'un servant dans le serveur de noms

Dans l'exemple qui suit, on gère également la désactivation du serveur en retirant les objets supprimés du serveur de noms. Cette opération se fait grâce à la fonction `unbind` qui est appelée deux fois pour retirer le service et le contexte précédemment créé dans le serveur de noms.

5.1.3 Mise en œuvre en C++

Enregistrement d'un servant dans le serveur de noms

```
//*************************************
//* fichier serveur_compteur.cpp
//*************************************
#include <OB/CORBA.h>
#include <OB/CosNaming.h>
#include <fstream>
#include "compteur_impl.h"
using namespace std;

void main(int argc, char **argv)
{
 CORBA::ORB_var orb =
  CORBA::ORB_init(argc, argv);
 CORBA::Object_var obj =
  orb->resolve_initial_references(
                          "RootPOA");
 PortableServer::POA_var poa =
  PortableServer::POA::_narrow(obj);

 //Instanciation et activation d'un servant
 C_compteur_impl*  servant_compteur =
  new C_compteur_impl(orb);
 M_compteur::I_compteur_var compteur =
  servant_compteur->_this();

 obj =
  orb->resolve_initial_references(
                       "NameService");
 CosNaming::NamingContext_var ns =
  CosNaming::NamingContext::_narrow(obj);

 CosNaming::Name nomcontexte;
 nomcontexte.length(1);
 nomcontexte[0].id = CORBA::string_dup("CTX");
 nomcontexte[0].kind = CORBA::string_dup("");
 CosNaming::NamingContext_var nvctx =
  ns->bind_new_context(nomcontexte);

 CosNaming::Name nom;
 nom.length(1);
 nom[0].id = CORBA::string_dup("Compteur");
 nom[0].kind = CORBA::string_dup("");
 nvctx->bind(nom,compteur);

 // Activation du POA manager
 PortableServer::POAManager_var manager =
  poa->the_POAManager();
 manager->activate();

 // On accepte les requetes
 orb->run();

 // arret du serveur on supprime Compteur et
 // son contexte du serveur de nom
 nvctx->unbind(nom);
 ns->unbind(nomcontexte);
}
```

5.1.3 Mise en œuvre en C++

Accès aux objets distants par le client via le serveur de noms

- L'accès au serveur de noms se fait de manière analogue à ce qui a été fait pour le servant.
- On devra d'abord contruire un `CosNaming::Name` puis le passer en paramètre de la méthode `resolve` que l'on invoquera sur l'adresse du serveur de nom. Cette invocation retourne alors la référence de l'objet sur l'ORB.

5.1.3 Mise en œuvre en C++

```
//************************************
//* fichier "client.cpp"
//************************************
#include <OB/CORBA.h>
#include <OB/CosNaming.h>
#include <iostream.h>
#include <fstream.h>

#include "compteur.h"

void main(int argc, char**argv) {
 int i;
 char reponse;
 CORBA::ORB_var orb;

 try {
  cout <<"initialisation de l'ORB\n";
  orb = CORBA::ORB_init(argc, argv);
 }
 catch(...) {
  cerr << "Impossible d'initial. l'ORB\n";
  return 1;
 }

 CORBA::Object_var objet =
  orb->resolve_initial_references("
                      NameService");
 CosNaming::NamingContext_var ns =
  CosNaming::NamingContext::_narrow(objet);

 CosNaming::Name nom;
 nom.length(2);
 nom[0].id = CORBA::string_dup("CTX");
 nom[0].kind = CORBA::string_dup("");
 nom[1].id = CORBA::string_dup("Compteur");
 nom[1].kind = CORBA::string_dup("");
 CORBA::Object_var objcompteur =
  ns->resolve(nom);

 M_compteur::I_compteur_var compteur =
  M_compteur::I_compteur::_narrow(objcompteur);

 cout << "somme a 0" << endl;
 compteur->somme((CORBA::Long) 0);

 for(i=0; i<100; i++) compteur->increment();
 cout << "resultat du traitement : "
      << compteur->somme() << endl;
 cout << "Arret du serveur (o/n) ? : ";
 cin >> reponse;
 try {
  if (reponse == 'o' || reponse == 'O') {
    cout << "arret du serveur demande\n'';
    compteur->desactivation();
  }
 }
 catch (...) {}
 return 0;
}
```

5.1.3 Mise en œuvre en C++

Configuration et lancement des applications

On procède maintenant à une véritable mise en place d'une architecture client/serveur distribuée, dans laquelle on spécifie la localisation du serveur de noms qui se chargera d'invoquer les objets sur leur machines d'implantation, via l'ORB.

- Compilation des applications : on procède comme dans le chapitre précédent en ajoutant un lien supplémentaire vers la librairie dynamique d'Orbacus qui est spécifique au service de nommage...
  on ajoutera donc l'option `-lCosNaming` aux commandes de création des exécutables.

5.1.3 Mise en œuvre en C++

- Lancement du serveur de noms distant sur une machine donnée et accessible à partir de l'ORB.
  1. Création d'un fichier de configuration "serveur.conf" décrivant la localisation du serveur de noms.

```
/* fichier ‘‘ob.config’’ */
/***************************/
# initialisation des services
ooc.orb.service.NameService=corbaloc::unemachine.univ-lehavre.fr:5000/NameService
```

     On vérifiera que le port 5000 est ouvert (`/etc/services`).
  2. Lancement du serveur de noms sur la machine décrite dans le fichier précédent

```
nameserv -OAport 5000&
```

- Lancement du gestionnaire de service sur une machine quelconque

```
serveur -ORBconfig ob.config &
```

- Lancement du client sur une machine quelconque

```
client -ORBconfig ob.config
```

5.1.4 Implémentation en Java

```
// serveur_compteur.java
package M_compteur;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import java.io.*;
import org.omg.CosNaming.*;
import
  org.omg.CosNaming.NamingContextPackage.*;

public class serveur_compteur {
 public static void main(String args[]) {
  // desactivation de l'orb du jdk
  // pour le remplacer par celui d'orbacus
  java.util.Properties prop =
   System.getProperties();
  prop.put("org.omg.CORBA.ORBClass",
           "com.ooc.CORBA.ORB");
  prop.put("org.omg.CORBA.ORBSingletonClass",
           "com.ooc.CORBA.ORBSingleton");
  ORB orb = null;

  try {
   orb = ORB.init(args, prop);
   org.omg.CORBA.Object obj =
    orb.resolve_initial_references(
                          "RootPOA");
   POA rootPOA = POAHelper.narrow(obj);
   POAManager manager =
    rootPOA.the_POAManager();

   C_compteur_impl servant_compteur =
    new C_compteur_impl();

   I_compteur compteur =
    servant_compteur._this(orb);

   obj = orb.resolve_initial_references(
                              "NameService");
   NamingContext ns =
    NamingContextHelper.narrow(obj);

   NameComponent[] nomcontexte =
    new NameComponent[1];
   nomcontexte[0] = new NameComponent();
   nomcontexte[0].id = "CTX";
   nomcontexte[0].kind = "";
   NamingContext nvctx =
    ns.bind_new_context(nomcontexte);

   NameComponent[] nom =
    new NameComponent[1];
   nom[0] = new NameComponent();
   nom[0].id = "Compteur";
   nom[0].kind = "";
   nvctx.bind(nom, compteur);

   manager.activate();
   orb.run();

   nvctx.unbind(nom);
   ns.unbind(nomcontexte);

  } catch(Exception erreur) {System.exit(1);}
 }
}
```

5.1.4 Implémentation en Java

```
// client_compteur.java
package M_compteur;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import java.io.*;
import org.omg.CosNaming.*;
import
  org.omg.CosNaming.NamingContextPackage.*;

public class client_compteur {
 public static void main(String args[]) {
  int i;
  String rep;
  // desactivation de l'orb du jdk
  java.util.Properties prop =
   System.getProperties();
  prop.put("org.omg.CORBA.ORBClass",
           "com.ooc.CORBA.ORB");
  prop.put("org.omg.CORBA.ORBSingletonClass",
           "com.ooc.CORBA.ORBSingleton");
  ORB orb = null;

  try {
   orb = ORB.init(args, prop);
   org.omg.CORBA.Object objet =
    orb.resolve_initial_references(
                      "NameService");
   NamingContext ns =
    NamingContextHelper.narrow(objet);

   NameComponent[] nom =
    new NameComponent[2];
   nom[0] = new NameComponent();
   nom[0].id = "Compteur";
   nom[0].kind = "";
   nom[1] = new NameComponent();
   nom[1].id = "Compteur";
   nom[1].kind = "";
   org.omg.CORBA.Object objcompteur =
    ns.resolve(nom);

   I_compteur compteur =
    I_compteurHelper.narrow(objcompteur);

   System.out.println("somme a 0");
   compteur.somme(0);
   for (i=0; i<100; i++) compteur.increment();
   System.out.println(
          "resultat du traitement :"
          + compteur.somme());
   System.out.println(
    "arret du serveur (o/n) ? : ");
   BufferedReader in =
    new BufferedReader(
     new InputStreamReader(System.in));
   rep  = in.readLine();
   if (rep.equals("o")||rep.equals("O")){
     System.out.println("arret du serveur deman
     compteur.desactivation();
   }
  } catch(Exception erreur) {
     System.out.println("Exception declanchee")
     System.exit(1);
   }
 System.exit(0);
}}
```

5.1.4 Implémentation en Java

Compilation et lancement des applications

Pour la compilation avec `javac`, pas de modifi cation par rapport aux exemples du chapitre précédents.
On utilise la commande vue auparavant pour lancer le serveur de noms. Puis les applicatifs sont lancés avec l'option `-ORBconfig`.

```
nameserv -OAport 5000 &

java M_compteur.serveur_compteur -ORBconfig ob.config &
java M_compteur.client_compteur  -ORBconfig ob.config
```

5.2 POA : adaptateur d'objets

5.2.1 Fonction du POA
5.2.2 Fonctionnement arborescent
5.2.3 Notion de POA manager
5.2.4 Etats et transitions du gestionnaire POA
5.2.5 Notion de persistence

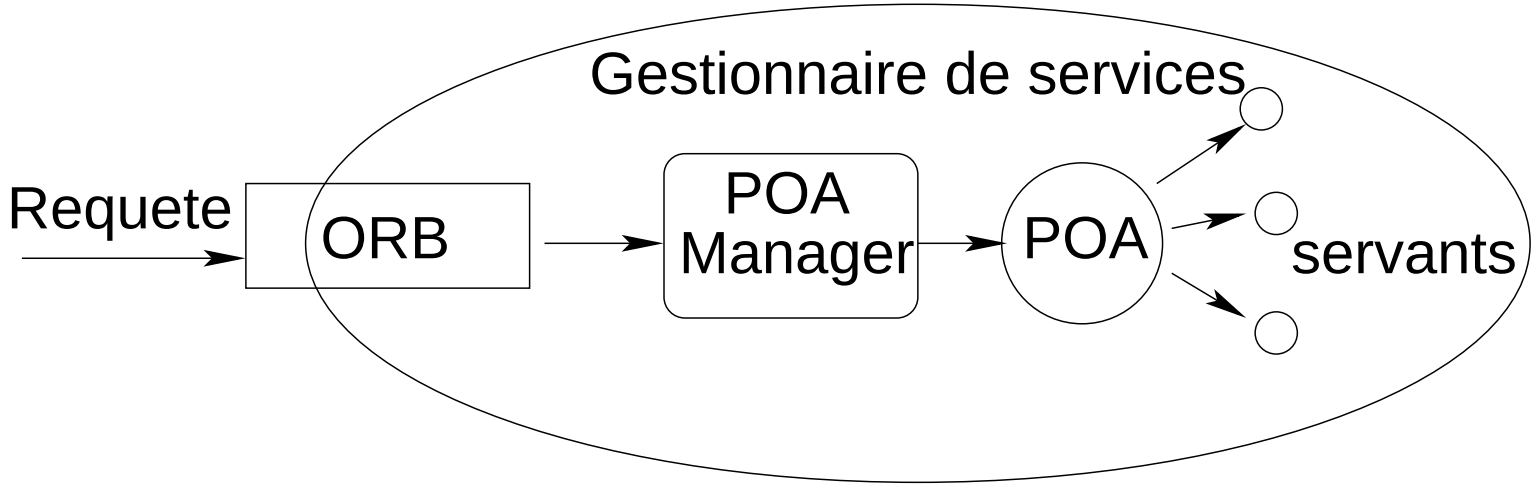
5.2.1 Fonction du POA

- POA : version “portable” d’un adaptateur d’objet. L’ancienne spécifi cation standard, le BOA, était insuffi samment spécifi ée $\rightarrow$ implémentations spécifi ques à chaque éditeur d’ORB.
- Fonction principale : fournir une transition entre la notion abstraite d’objet CORBA et la réelle implémentation du comportement de l’objet sous la forme d’un servant.
- Chaque servant actif sur l’ORB est rattaché à un POA qui lui fournit un espace de noms (namespace).
- Un POA peut gérer plusieurs servants.

5.2.2 Fonctionnement arborescent du POA

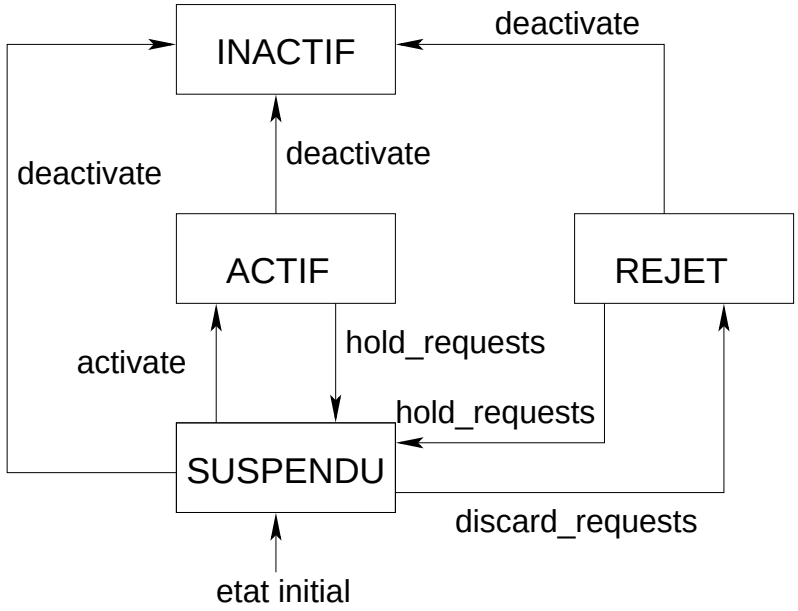
- Par défaut un serveur doit commencer par se lier à un POA racine, appelé `RootPOA`.
- A partir de cette racine, on peut créer, si nécessaire, de nouveaux POA qui vont devenir des fi ls de `RootPOA`.
- On a donc une organisation arborescente des différents POA qui peuvent être créer dans un programme.
- L'intérêt de défi nir plusieurs POA est que l'on peut attribuer des règles (`policies` ... contrôlant les caractéristiques d'implémentation des servants : persistance, multi-threading, ...) spécifi ques à chaque POA.

5.2.3 Notion de POA manager



- Chaque POA est associé à un gestionnaire de POA, `POAManager`, qui peut gérer plusieurs POA et contrôle leur état de traitement.
- Ces différents états sont décrits dans le schéma ci-après avec les méthodes associées permettant d'en changer.

5.2.4 Etats et transitions du gestionnaire POA



Un POA gère une file d'attente des requêtes qu'il doit traiter. Son état décrit la manière dont il gère cette file :

- actif (active) : le POA Manager empile les requêtes qui sont traitées dès que possible par le POA cible.
- inactif (inactive) : état temporaire précédent la destruction du POA.
- suspendu (holding) : Les requêtes s'empilent et ne sont pas traitées jusqu'à ce que le POA revienne dans un état actif.
- rejet (discarding) : Les requêtes sont rejetées tant que le POA est dans cet état.

5.2.5 Notion de persistence

- Il est possible de créer des POA persistants qui continuent à fonctionner même si le serveur s'arrête et redémarre.
- On doit construire et associer des *policies* spécifi ques et activer les servants avec une méthode utilisant directement l'adresse ID de l'objet ...

5.3 Le référentiel d'interfaces (IFR)

5.3.1 Principe de l'invocation dynamique
5.3.2 Rôles de l'IFR
5.3.3 Contexte d'utilisation
5.3.4 Contenants et contenus
5.3.5 Interfaces du référentiel
5.3.6 Exemple de consultation du référentiel
5.3.7 Exemple de consultation de l'interface d'un objet

5.3.1 Principe de l'invocation dynamique

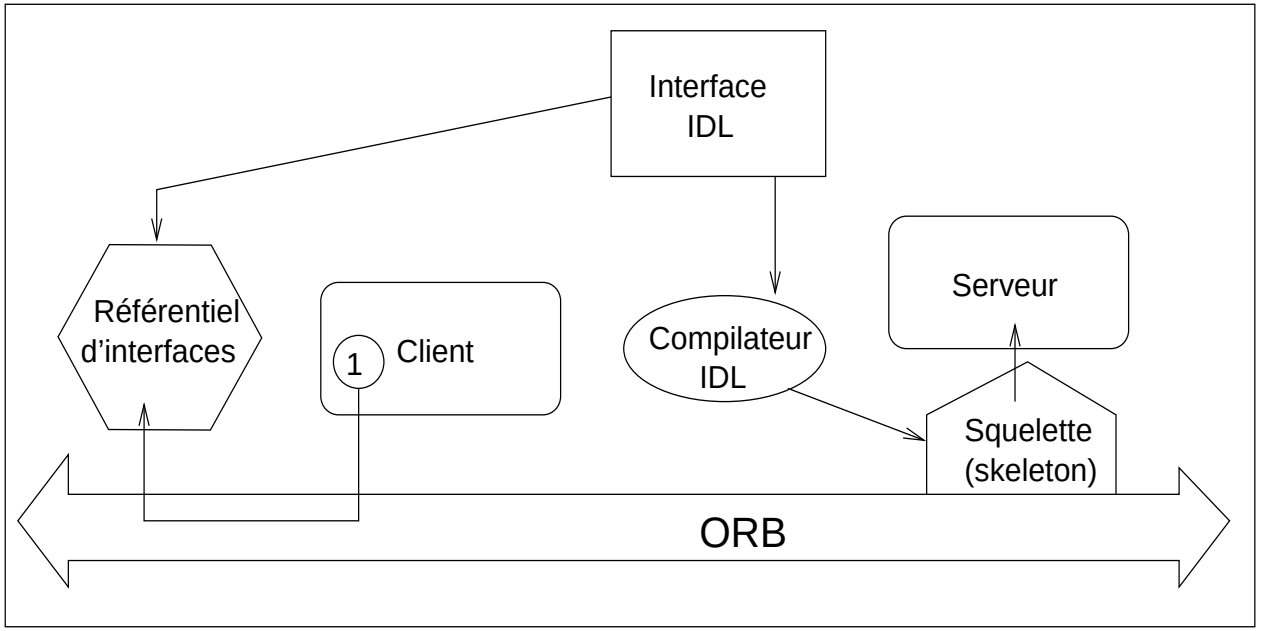
Ici, l'interface IDL du service invoqué n'est pas connu du client

- qui consulte et découvre l'interface en interrogeant le référentiel d'interfaces (IFR) ;
- qui construit dynamiquement sa requête en cours d'exécution, puis en fait l'invocation.

5.3.1 Principe de l'invocation dynamique

Schéma de fonctionnement de l'invocation dynamique

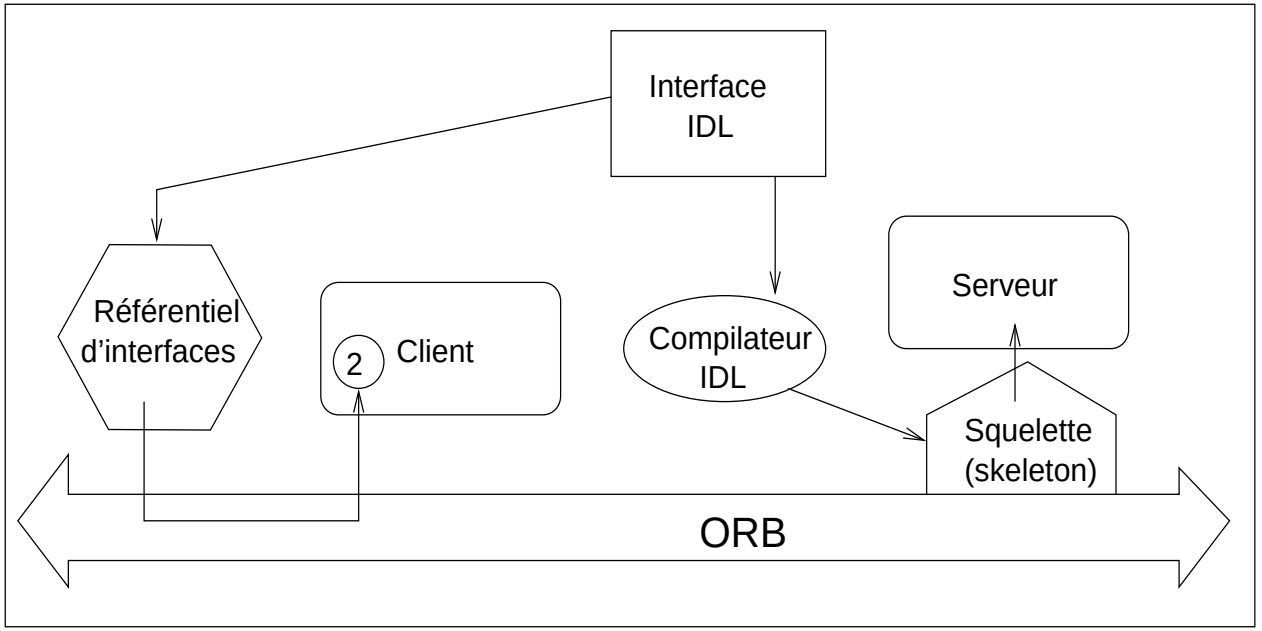
Requêtes à invocation dynamique (DII)



5.3.1 Principe de l'invocation dynamique

Schéma de fonctionnement de l'invocation dynamique

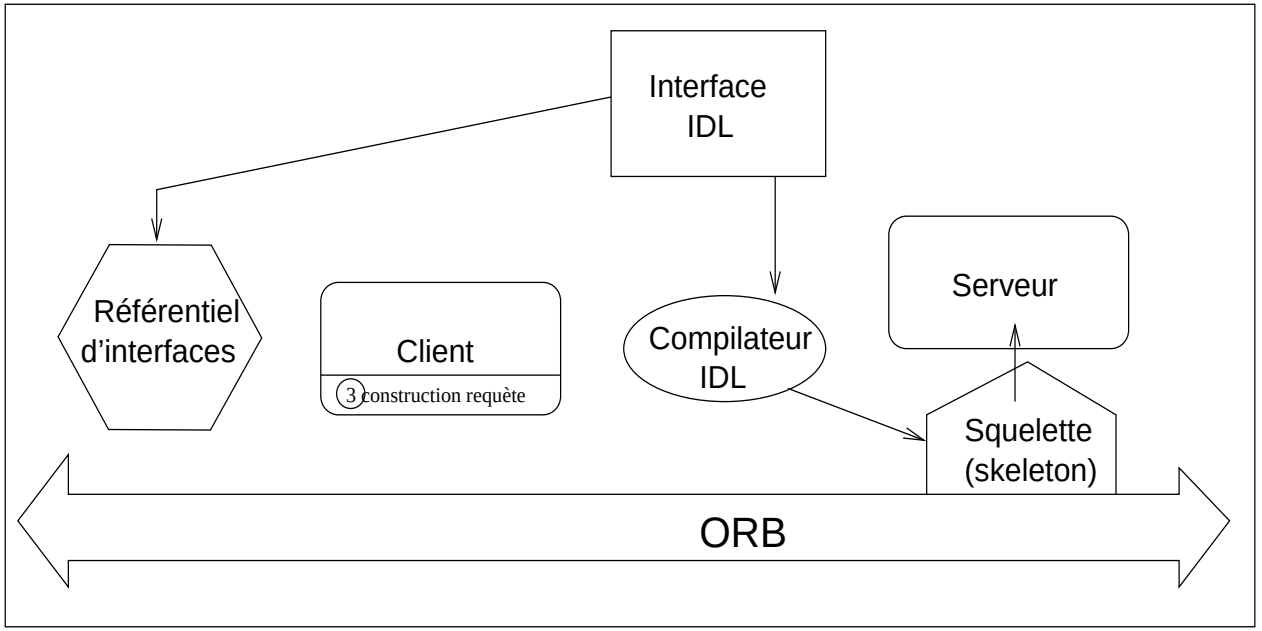
Requêtes à invocation dynamique (DII)



5.3.1 Principe de l'invocation dynamique

Schéma de fonctionnement de l'invocation dynamique

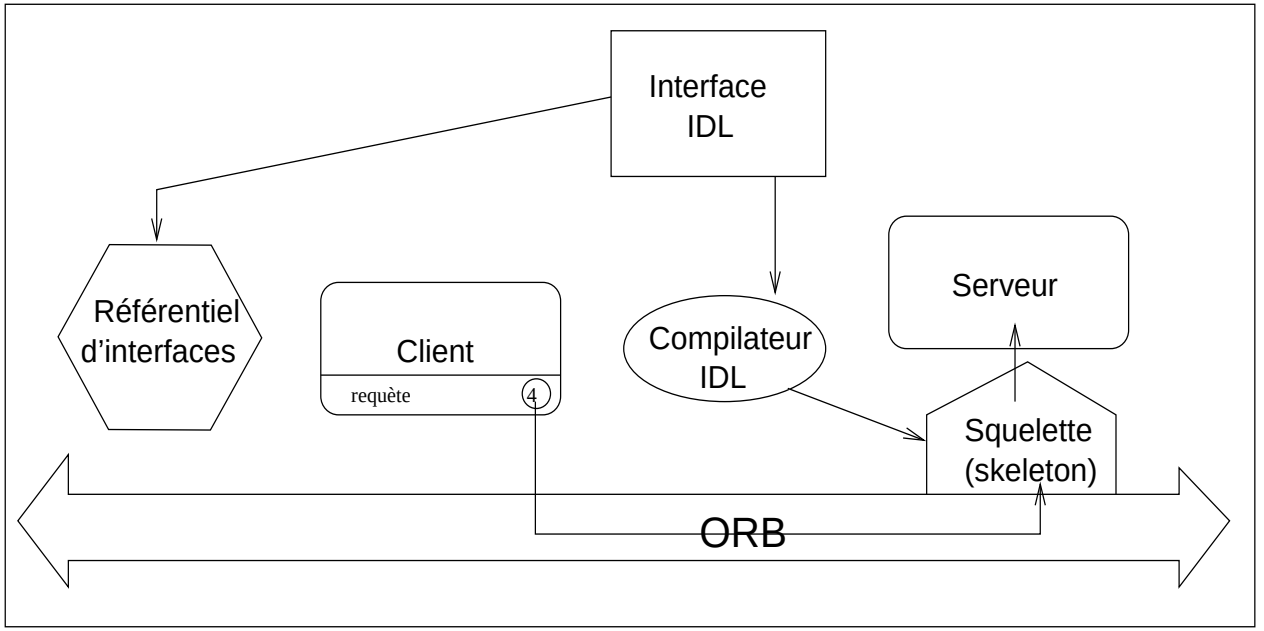
Requêtes à invocation dynamique (DII)



5.3.1 Principe de l'invocation dynamique

Schéma de fonctionnement de l'invocation dynamique

Requêtes à invocation dynamique (DII)



5.3.2 Rôle de l'IFR

- Permet de stocker des archives d'interfaces IDL. L'IFR est accessible depuis l'ORB grâce à sa propre interface IDL (masquage de sa réelle implantation).
- Outils de manipulation : consultation, modifi cation et navigation.
- Possibilité d'IFR multiples coopératifs et communiquants.
- La norme laisse une liberté totale sur le système d'implantation du référentiel.

5.3.3 Quelques contextes d'utilisation de l'IFR

- Distribution, installation et accessibilité des interfaces IDL pour permettre la génération des souches chez les clients ;
- Vérifi cations diverses des spécifi cations des objets (absence de cycle d'héritage, signature des opérations, ...) ;
- Fédération de plusieurs BUS Corba qui s'échangent les interfaces grâce aux IFR ;
- Exécution et invocation dynamique : c'est notre propos principal ici !

5.3.4 Contenant et contenu

Dans le référentiel d'interfaces, on a 2 catégories d'éléments :

- les conteneurs (*Container*) : il peut contenir plusieurs autres types, comme des modules qui peuvent contenir des interfaces, des défi nitions de types, ...
- les contenus (*Contained*) : ce sont des éléments d'un conteneur comme un attribut ou une opération contenus dans une interface

Certains éléments peuvent être à la fois conteneur et contenu comme une interface, contenu dans un module et contenant des attributs.

5.3.5 Interfaces du référentiel

Le référentiel contient des Métadonnées décrivant les interfaces IDL de tous les objets Corba : chaque type IDL est donc associé à une interface du référentiel. Par exemple, un module est asocié à l'interface `ModuleRef`.

Il existe ainsi 9 interfaces : Repository, ModuleDef, InterfaceDef, AttributeDef, OperationDef, ParameterDef, ExceptionDef, TypedefDef et ConstantDef.

Commentaire

Les 9 principales interfaces sont :

- Repository : interface décrivant l'objet racine du référentiel à partir duquel on pourra accéder aux autres objets ;
- ModuleDef : interface associée aux modules IDL et permettant de les découvrir, d'en ajouter ou d'en supprimer ;
- InterfaceDef : interface associée aux interfaces IDL et permettant de découvrir leurs opérations et attributs ;
- AttributeDef : interface associée aux attributs IDL et permettant de connaître le type et leur mode d'accès (consultation ou modifi cation) ;
- OperationDef : interface associée aux opérations IDL et permettant d'obtenir la liste des paramètres et des exceptions de chaque opération ;
- ParameterDef : interface associée aux paramètres de opérations IDL en permettant la consultation de leur type, nom et mode de passage ;
- ExceptionDef : interface associée aux exceptions IDL en permettant d'obtenir la liste de leur champs ;
- TypedefDef : interface associées aux types de données IDL en permettant de fournir les informations associées à leur défi nition (champs des structures, dimensions de tableaux, type de séquence, ...) ;
- ConstantDef : interface associée aux constantes IDL en permettant de découvrir leur type et leur valeur.

5.3.7 Exemple de consultation du référentiel

On utilise ici le langage C++.

1 Obtenir la référence sur l'objet notoire IFR :

```
CORBA::Object_var objet =
   orb->resolve_initial_references(
                    "DefaultRepository");
```

2 Convertir cette référence vers le type réel de l'objet :

```
CORBA::Repository_var referentiel =
     CORBA::Repository::_narrow(objet);
```

5.3.7 Exemple de consultation du référentiel

3 Rechercher une définition par son nom ou son identifiant :

```
CORBA::Contained_var definition =
     referentiel->lookup(
            "MonModule::MonInterface");
definition = referentiel->lookup_id(
     "IDL:MonModule/MonInterface:1.0");
```

4 Convertir vers un `Container` :

```
CORBA::Container_var conteneur =
     CORBA::Container::_narrow(definition);
```

- Obtenir la liste de tous les objets contenus :

```
CORBA::ContainedSeq_var contenus =
   conteneur->contents(CORBA::dk_all, CORBA::TRUE);;
```

5 Parcourir la séquence des objets contenus et consulter le nom de chaque objet contenu :

```
for (CORBA::ULong i=0; i< contenus->length(); i++)
{
   CORBA::String_var nom = contenus[i]->name();
   cout << " " << nom << endl;
}
```

5.3.7 Exemple de consultation de l'interface d'un objet

1 Obtenir l'interface d'un objet (ici, `objcompteur`) :

```
CORBA::InterfaceDef_var objcompteur_interface =
   objcompteur->_get_interface();
```

2 Obtenir la description complète de l'interface :

```
CORBA::InterfaceDef::
   FullInterfaceDescription_var intdesc =
   objcompteur_interface->describe_interface();
```

5.3.7 Exemple de consultation de l'interface d'un objet

3 Affi cher les informations générales :

```
cout << "Nom  = " << intdesc->name << endl;
cout << "ID   = " << intdesc->id << endl;
cout << "Dfini dans  = " << intdesc->defined_in
                         << endl;
cout << "Version = " << intdesc->version << endl;
```

4 Affi chage des opérations :

```
cout << "Oprations" << endl;
for (i=0; i<intdesc->operations.length(); i++)
{ CORBA::OperationDescription opi =
        intdesc->operations[i];
  cout << " " << opi.name << endl;
  for (j=0; j<opi.parameters.length(); j++)
  { CORBA::ParameterDescription paramj =
        opi.parameters[j];
    cout << " nom du parametre "
        << j << " : " << paramj.name << endl;
  }
}
```

5.3.7 Exemple de consultation de l'interface d'un objet

5 affi chage des attributs :

```
cout << "Attributs" << endl;
for (i=0; i<intdesc->attributes.length(); i++)
{ CORBA::AttributeDescription ati =
       intdesc->attributes[i];
  cout << "  " << ati.name << endl;
  cout << "  Type : ";
  cout << ((ati.type == CORBA::_tc_long) ?
            "long" : "inconnu") << endl;
  cout << "  Mode : ";
  cout << ((ati.mode == CORBA::ATTR_NORMAL) ?
            "Lect/Ecr" : "Lect") << endl;
}
```

5.3.7 Exemple de consultation de l'interface d'un objet

Traduction en Java

Depuis les dernières versions d'Orbacus, l'appel de la méthode `_get_interface()` pour obtenir l'interface d'un objet, doit être utilisée en C++ mais elle doit être remplacée, en Java, par les 2 appels suivants :

```
org.omg.CORBA.Object defObj =
   objcompteur._get_interface_def();
org.omg.CORBA.InterfaceDef objcompteur_interface =
   org.omg.CORBA.InterfaceDefHelper.narrow(defObj);
```