

TP N°8



Objectifs :

- 1) Maîtriser le concept de CORBA
- 2) Mettre en œuvre CORBA
- 3) Développer des applications réparties avec CORBA

Ressources utilisées :

- 1) Packages : org.omg.CORBA.ORB, javax.naming
- 2) Outils : javac, java, idlj, orbd.
- 3) Langages : IDL, Java and others (avec extensions respectives .idl, .java, etc.)

Travail à faire :

- 1) Utiliser google pour apprendre plus sur les implémentations de CORBA suivantes :
 -  Commerciales : IONA ORBacus, Orbix, Borland VisiBroker (pour C++ et Java)
 -  Gratuits : Red Hat ORBit pour C, MICO pour C++, AT&T omniORB pour C++, TAO pour C++, Jonathan pour Java, Robin pour Java, etc.
- 2) Faire les exercices suivants :

Exercice 1 – [2]

L'objectif de cet exercice et du suivant est de concevoir un programme qui va utiliser un objet distribué pour dire "Bonjour". Cet exercice que nous allons réaliser ensemble nous permettra d'appréhender la démarche de conception d'un système distribué basé sur l'utilisation de la norme CORBA.

Pour rendre plus facilement disponible un objet distribué CORBA, il est possible d'utiliser l'un des services CORBA appelé Naming Service. On doit alors donner la référence de cet objet au Naming Service. Il s'agit d'enregistrement de l'objet avec un nom ce qui permet ensuite à un client de faire appel à cet objet en l'appelant par son nom.

Néanmoins, l'utilisation du Naming Service n'est pas obligatoire comme pour tous les autres services CORBA. Nous allons donc voir lors de la conception de notre programme HelloWorld les deux possibilités :

1. Conception du programme HelloWorld sans le Naming Service.
2. Conception du programme HelloWorld avec le Naming Service.

Dans les deux versions, il s'agira du même fichier IDL (Hello.idl) permettant de déclarer l'interface de l'objet distribué.

```
{  
    void sayHello();  
};
```

Exercice 2 – [2]

Pour générer les Stub et Skeleton nécessaires, il vous faut employer la commande suivante : `idlj -fall Hello.idl`.

Il vous faut maintenant donner l'implémentation de l'objet distribué en JAVA (Hello_impl.java). Cet objet doit hériter de la classe abstraite HelloPOA qui a été générée automatiquement lors de l'étape précédente.

```
public class Hello_impl extends HelloPOA {  
    public void sayHello () {  
        System.out.println ("Bonjour le monde !");  
    }  
}
```

Ensuite, il vous faut écrire le server. Le passage de la référence de l'objet en l'absence de naming va alors se faire au moyen d'un fichier "Hello.ref". Si nous devons faire communiquer le serveur avec des clients distants, il faudrait alors transférer le fichier "Hello.ref" au niveau de chaque client.

```

public class HelloServer{
    public static void main (String args[]){
        java.util.Properties props = System.getProperties ();
        int status = 0;
        org.omg.CORBA.ORB orb = null;
        try{
            orb = org.omg.CORBA.ORB.init (args,props);
            status = run (orb);
        } catch (Exception ex){
            ex.printStackTrace ();
            status = 1;
        }
        System.exit (status);
    }

    static int run(org.omg.CORBA.ORB orb)
        throws org.omg.CORBA.UserException {
        org.omg.PortableServer.POA rootPOA =
            org.omg.PortableServer.POAHelper.narrow(
                orb.resolve_initial_references("RootPOA"));
        org.omg.PortableServer.POAManager manager = rootPOA.the_POAManager();

        Hello_impl helloImpl = new Hello_impl();
        Hello hello = helloImpl._this(orb);

        try {
            String ref = orb.object_to_string(hello);
            String refFile = "Hello.ref";
            java.io.PrintWriter out =
                new java.io.PrintWriter(new java.io.FileOutputStream(refFile));
            out.println(ref);
            out.close();
        } catch(java.io.IOException ex){
            ex.printStackTrace();
            return 1;
        }

        manager.activate();
        orb.run();
        return 0;
    }
}

```

Ensuite il vous faut écrire le client :

```

public class HelloClient {
    public static void main(String args[]) {
        java.util.Properties props = System.getProperties ();
        int status = 0;
        org.omg.CORBA.ORB orb = null;
        try{
            orb = org.omg.CORBA.ORB.init (args,props);
            status = run (orb);
        } catch (Exception ex){
            ex.printStackTrace ();
            status = 1;
        }
        System.exit (status);
    }
}

```

```

static int run(org.omg.CORBA.ORB orb) {
    org.omg.CORBA.Object obj = null;
    try {
        String refFile = "Hello.ref";
        java.io.BufferedReader in =
            new java.io.BufferedReader(new java.io.FileReader(refFile));
        String ref = in.readLine();
        obj = orb.string_to_object(ref);
    }
    catch(java.io.IOException ex) {
        ex.printStackTrace();
        return 1;
    }

    Hello hello = HelloHelper.narrow(obj);
    hello.sayHello();
    return 0;
}
}

```

Maintenant, vous pouvez compiler l'ensemble des sources du programme au moyen de la commande `javac -cp . *.java`.

L'exécution du côté du serveur se fera par le biais de l'appel `java HelloServer` alors que du côté client, vous devrez utiliser la commande `java HelloClient`.

Exercice 3 – [2]

Pour générer les Stub et Skeleton nécessaires, il vous faut employer la commande suivante : `idlj -fall -oldImplBase Hello.idl`.

Ensuite, il vous faut écrire le server qui nous servira aussi à implémenter l'objet distribué.

```

import org.omg.CORBA.ORB;
import javax.naming.*;
import java.util.*;

public class HelloServer implements HelloOperations {
    public void sayHello () {
        System.out.println ("Bonjour le monde");
    }

    public static void main(String args[]) {
        try {
            ORB orb = ORB.init(args, null);

            HelloServer hello = new HelloServer ();
            Hello helloRef = new Hello_Tie(hello);
            orb.connect(helloRef);

            Properties env = System.getProperties();
            env.put("java.naming.factory.initial", "com.sun.jndi.cosnaming.CNCtxFactory");

            Context ctx = new InitialContext(env);
            ctx.rebind("Hello", helloRef);

            while(true) Thread.sleep(1000);
        } catch (Exception e) {
            System.err.println("ERROR: " + e);
            e.printStackTrace(System.out);
        }
    }
}

```

Il vous faut ensuite écrire le client :

```

import org.omg.CORBA.ORB;
import javax.naming.*;
import java.util.*;

public class HelloClient{

    public static void main(String args[]) throws Exception {
        Hello objDistant;
        ORB orb = ORB.init(args, null);

        Properties env = System.getProperties();
        env.put("java.naming.factory.initial", "com.sun.jndi.cosnaming.CNCtxFactory");
        env.put("java.naming.provider.url", "iiop://localhost:900");
        Context ctx = new InitialContext(env);
        objDistant = (Hello)ctx.lookup("Hello");

        objDistant.sayHello();
    }
}

```

Maintenant, vous pouvez compiler l'ensemble des sources du programme au moyen de la commande `javac -cp . *.java`.

Avant de pouvoir exécuter vos programme, il vous faut activer le naming service au moyen de la commande `orbd`.

L'exécution du côté du serveur se fera par le biais de l'appel `java HelloServer` alors que du côté client, vous devrez utiliser la commande `java HelloClient`.

Exercice 4 – [2]

Il s'agit de reprendre le programme HelloWorld avec l'utilisation du Naming Service puis de l modifier de façon à envoyer votre prénom au serveur puis à récupérer en retour un "Bonjour VotrePrénom !". Vous affichez alors du côté du client le résultat de retour. Prévoir, contrairement au programme précédent, la possibilité pour que le client soit sur une autre machine que le serveur. Il faudra donc que le client puisse préciser en ligne de commande le nom du serveur.

Exercice 5 – [2]

L'objectif de se programme est de mettre à disposition un objet distribué qui sera capable d'effectuer les quatres opérations élémentaires puis de retourner les résultats aux clients qui en feront la demande.

Références :

1. Livre du cours : Java Network Programming, 2nd Edition, O'Reilly
2. Univ-LeHavre/CNAM, NFP 111, TP sur CORBA (C. Duvallet)
3. Introduction à CORBA, Sylvain BARTHELEMY

Annexe

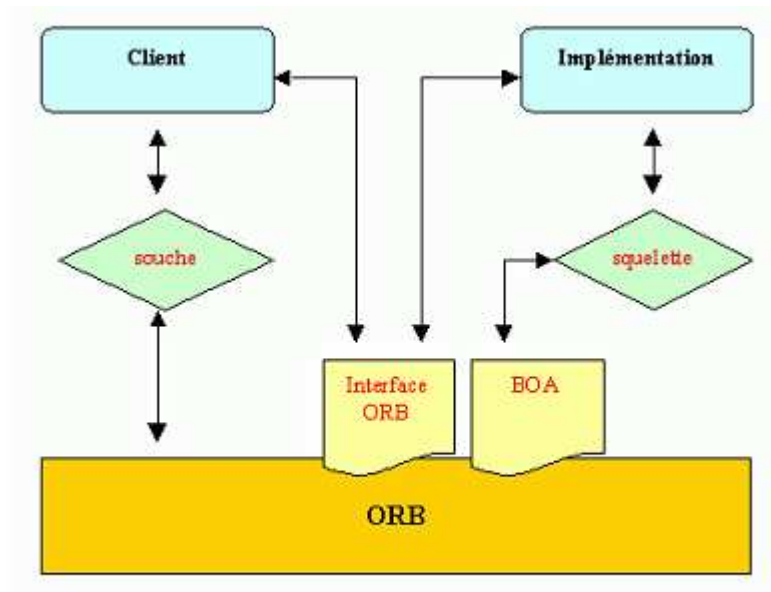
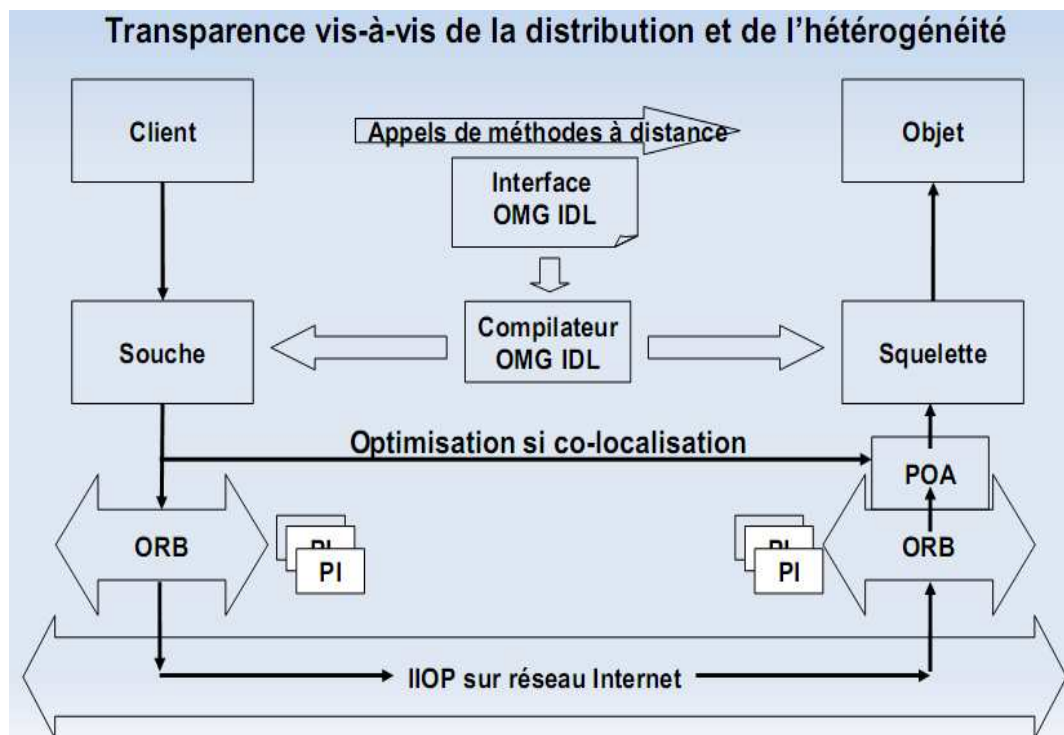
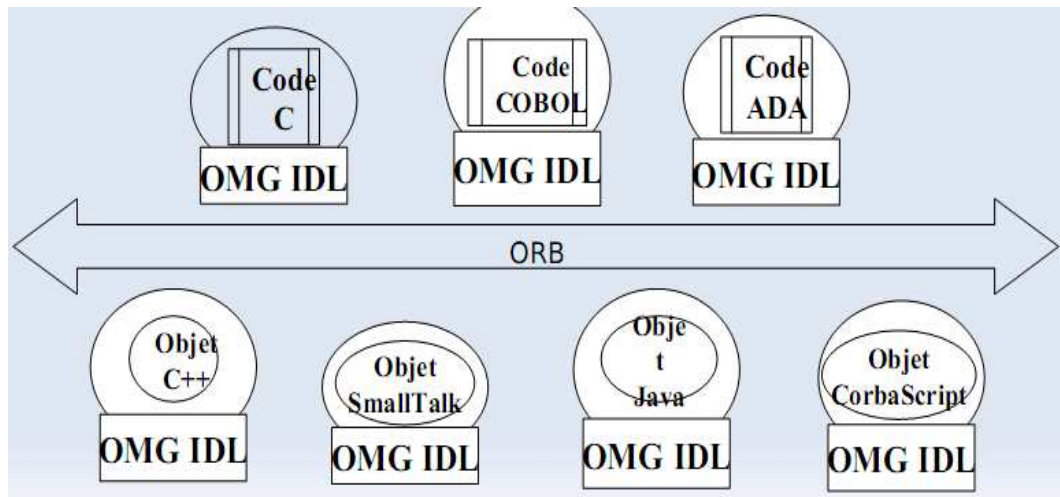


Figure 1: Représentation simplifiée du bus CORBA

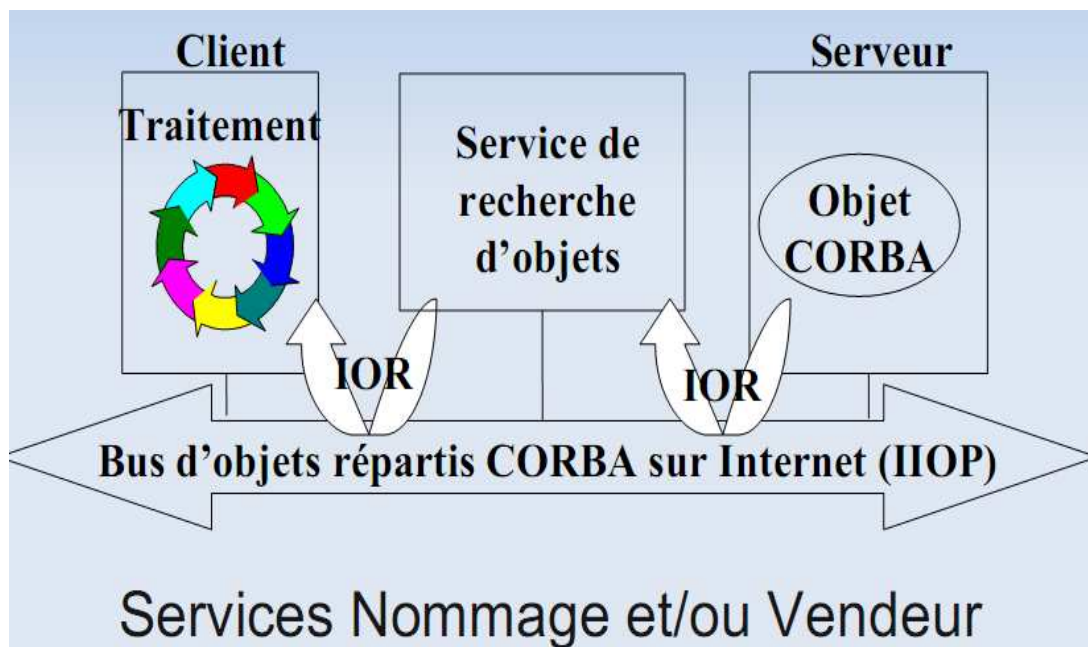
[3]



[4]



[4]



[4]