TP CORBA n° 2 et n° 3
Invocation statique d'objets distribués
Utilisation d'un serveur de noms

DESS SRO - Module "Objets Distribués"
UFR Sciences et Techniques - Université du Havre

Exercice 1 : Application "moyenne distribuée"

Développer une application client/serveur dont le service a pour tâches :

- de mémoriser des nombres entiers qui lui sont envoyés par différents clients ;
- de renvoyer la moyenne de tous les nombres qu'il a reçu - tous clients confondus - à la demande d'un client ;
- de se désactiver à la demande d'un client.

On écrira une version **C++** et **Java** du servant, du gestionnaire de services et du client.

On testera des appels hétérogènes de l'applicatif (client en **java** et serveur en **C++** ou réciproquement).
Les objets distribués seront référencés par leurs IOR(s) qui seront stockés dans des fichiers et rapatriés par ftp ou scp.
En annnexe, on donne les indications nécessaires pour utiliser une URL afin de récupérer l'IOR des objets distants.

Exercice 2 : Projet "ORoBo", invocation statique des joueurs

Il s'agit d'implanter les requêtes des différents joueurs du jeu "ORoBo", en se basant sur le modèle client/serveur de l'exercice précédent. Le serveur qui gère le damier bidimensionnel du jeu où se déplacent les robots, a pour tâches :

- de gérer les déplacements des robots qui sont demandés par chaque joueur. Deux cas se présentent, soit le déplacement est possible et réalisé car la case cible est vide, soit le déplacement n'est pas réalisé car la case cible est déjà occupée ;
- de renvoyer la confirmation ou l'infirmation du déplacement d'un robot à la demande d'un joueur. Le déplacement en question correspond alors au dernier déplacement sollicité par ce joueur pour ce robot ;
- d'actualiser le damier en détruisant les éventuels robots en prise avec leurs adversaires les encerclant, à la demande d'un joueur ;
- de renvoyer la configuration complète du damier à la demande d'un joueur ;
- de ne renvoyer que la configuration du damier sur un voisinage d'un robot donné, à la demande d'un joueur (cela correspond à la dernière extension possible du jeu énoncé à la fin du TP n°1).

Exercice 3 : “ORoBo” et serveur de noms (TP n° 3)

Modifier l’exercice 2 en utilisant un serveur de noms.

Annexe : Procédé d’accès distant utilisant une URL

Références d’objet

Corba 2 définit un protocole pour l’interopérabilité entre les différents bus CORBA, il s’agit de GIOP (Global Inter-ORB Protocol) dont l’application au niveau TCP/IP donne IIOP (Internet Inter-ORB Protocol). Un objet est alors référencé par son IOR (Interoperable Object Reference), cela correspond à l’hote de l’objet (adresse IP), le port et la clef de l’objet. Cette IOR peut être mis sous la forme d’une chaine de caractères et ensuite utilisée par d’autres objets pour référencer l’objet en question. Nous avons déjà présenté dans différents exemples la façon de faire :

```
....
CORBA::String_var str = orb->object_to_string(compteur);
const char *refFile = "compteur1.ref";
ofstream out(refFile);
out << str << endl;
out.close();
....
```

La “chainification” de l’IOR est ici stockée dans un fichier `compteur1.ref`, maintenant pour un autre objet il suffit de récupérer ce fichier et transformer la chaine en reférence d’objet.

```
....
const char* refFile = "compteur1.ref";
ifstream in(refFile);
char s[2048];
in >> s;
CORBA::Object_var objcompteur = orb->string_to_object(s);
....
```

Si l’objet client est situé sur une autre machine, le fichier doit être rendu accessible par un Filesystem distant ou rapatriement du fichier . . .ou encore l’utilisation d’une URL, c’est ce dernier cas que nous allons présenter dans ce qui suit.

Utilisation d’une URL

Il est parfois difficile ou impossible pour les objets clients d’avoir accès au filesystem de l’objet serveur, pour lire le fichier contenant l’IOR de l’objet serveur. Une façon plus flexible est de mettre à disposition l’IOR dans un fichier qui soit accessible par les objets clients à l’aide d’une URL. Les objets clients peuvent alors utiliser HTTP ou FTP pour obtenir le contenu du fichier, remarquons cependant que cela nécessite de la part des objets clients de connaître l’URL (ce qui n’est pas le cas si l’on utilise un service de noms). Ceci peut être fait très simplement en Java, ainsi par exemple :

```
import java.io.*;
import java.net.*;

....
String location =
       "http://machine8.univ-lehavre.fr/~untel/corba/compteur2/compteur2.ref";
```

```
URL urlWeb = new URL(location);
URLConnection uConnection = urlWeb.openConnection();
BufferedReader in = new BufferedReader(
                        new InputStreamReader(uConnection.getInputStream()));
String s = in.readLine();
in.close();
org.omg.CORBA.Object objcompteur = orb.string_to_object(s);
....
```

En C, C++, cela nécessite soit d'écrire une lecture d'une URL en utilisant les sockets ou d'utiliser une librairie qui permet cela, par exemple `curl` http ://curl.haxx.se/libcurl/

```
#include <curl/curl.h>
#include <string.h>

size_t sauve_ior(void *ptr, size_t size, size_t n, char *ior)
{
  strncpy(ior,(char *)ptr, n);
  ior[n-1] = '\0';
  return n;
}

....

CURL *curl;
const char* url =
       "http://machine8.univ-lehavre.fr/~untel/corba/compteur2/compteur2.ref";
CURLcode res;
char buffer[2048] = "";
curl = curl_easy_init();
if(curl)
  {
   curl_easy_setopt(curl, CURLOPT_URL, url);
   curl_easy_setopt(curl, CURLOPT_FILE, buffer);
   curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, sauve_ior);
   res = curl_easy_perform(curl);
   /* cleanup est obligatoire */
   curl_easy_cleanup(curl);
   }

...
CORBA::Object_var objcompteur = orb->string_to_object(s);
...
```

Pour compiler, on utilise donc la librairie curl `gcc exemple.cc -lcurl .....`