UNIVERSITÉ DE BOURGOGNE
UFR Sciences et Techniques

Cours: BD et Environnement Distribuées

TP 2 - Les objets distribués CORBA

Préparé par:

MATTA Elie et al.

Table des matières

Introduction

Dans nos temps, les technologie s'évolue d'une façon rapide.

Cette évolution a notamment porté sur les réseaux qui ont évolués à la fois quantitativement et qualitativement. Ceci à conduit progressivement à une organisation moins centralisée des ressources informatiques. D'où la nécessité de mettre en disposition un system qui fait communiquer les différents systèmes, comme par exemple CORBA.

CORBA est un standard d'objets distribués. Il permet à une application de demander à un objet distribué d'effectuer une opération et d'en obtenir le résultat. L'application communique avec l'objet qui effectue l'opération. C'est une fonctionnalité clients/serveur basique où un client envoie une requête à un serveur et le serveur répond au client. Les données peuvent passer du client au serveur et sont associées à une opération particulière sur un objet particulier. Les données sont ensuite retournées au client sous la forme d'une réponse.(1)

A l'inverse d'autres architectures logicielles, CORBA n'est pas un outil ou un ensemble d'outils créés par un seul éditeur de logiciels. CORBA est une norme, un ensemble de spécifications et de recommandations rédigées par un groupe de travail nommé OMG (Object Management Group), auquel appartiennent la plupart des acteurs informatiques majeurs.(2)

- CORBA définit
  - Un langage de spécification d'interfaces (IDL)
  - Des interfaces pour invoquer des méthodes à distance
  - Un ensemble de services pour manipuler aisément les objets

- CORBA simplifie
  - La localisation des objets
  - L'invocation des méthodes distantes

On va distinguer les caractéristiques du CROBA dans ce TP, tout en prenant compte la comparaison entre CORBA et Java-RMI, utilisation du Pont RMI-CORBA et d'autre caractéristiques et testant l'activation.

Exercices 3: Invocation dynamique

A) L'invocation dynamique

L'interface d'invocation dynamique d'un ORB en CORBA autorise la création et l'invocation dynamiques de requêtes. Un client utilisant cette interface pour envoyer une requête à un objet, obtient la même sémantique qu'un client utilisant l'opération stub générer à partir de la spécification de type IDL.

L'invocation statique ne permettant pas d'accéder aux nouveaux objets qui ont été ajoutés au système plus tard, l'interface d'invocation dynamique fournit cette capacité. Cette interface permet à un client, à l'exécution de :

- ✓ découvrir de nouveaux objets
- ✓ découvrir leurs interfaces
- ✓ retrouver les définitions d'interfaces
- ✓ construire et distribuer des invocations
- ✓ recevoir les réponses

L'interface d'invocation dynamique est donc une interface de l'ORB qui comprend des routines autorisant le client et l'ORB de travailler ensemble. Il permet de construire et invoquer des opérations sur tout objet disponible à l'exécution.

B) Le service CosNaming

Le service de nommage fournit le principal mécanisme à travers lequel la plupart des objets d'un système basé ORB situent les objets qu'ils veulent utiliser. Pour cela, il utilise des noms. Un nom est une séquence ordonnée de composants. Chaque composant sauf le dernier nomme un contexte, le dernier dénote les objets liés par le nom. Un composant est composé de deux attributs:

- ✓ Attribut *identifier*: identifiant
- ✓ Attribut *kind*: type du composant

Un contexte est un jeu de liens *noms-objets* dans lequel chaque nom est unique. Le service de nommage permet :

- ✓ De lier un nom un objet, ceci dans un contexte de nommage: *NamingContext*
- ✓ Déterminer l'objet associé au nom dans un contexte donné: *BindingIterator*

Puisque les valeurs des attributs des noms ne sont ni assignées ni interprétée par ce service, les logiciels de plus haut niveau n'ont pas de contraintes en terme de politique sur l'utilisation ou la gestion de ces valeurs.

Les noms peuvent être indépendants de la représentation, permettant ainsi à cette représentation d'évoluer sans requérir de changements du coté client.

C) L'implémentation

i. Serveur de nom: nameserv

La première étape consiste à créer le serveur de nom. Pour cela, il faut tenir compte des propriétés du serveur (nom du serveur et port d'écoute). Le serveur de nom sera utilisé par le serveur et le client pour retrouver la référence de l'objet. Pour cela, il faut juste lancer l'instruction suivant:

```
nameserv -OAport 9999 &
```

Le nombre 9999 précise le numéro de port disponible pour accéder au serveur de nom. Cependant il faut s'assurer que les variables d'environnement nécessaires sont bien fixés.

ii. Interface IDL et son implémentation

Nous avons travaillé avec l'interface *Hello.idl* qui contient l'unique méthode *say_hello*. Ceci à l'aide d'une syntaxe spécifique au langage de définition d'interfaces.

```
interface Hello {
    void say_hello();
};
```

Figure 1 - L'interface Hello

Nous avons ainsi implémenté en la classe *Hello_imp.cpp* et la classe *Hello_imp.h*. Ces classes implémenté au niveau serveur la méthode *say_hello()* de l'interface *Hello.idl* d'une manière concrète.

L'instruction suivante permet de générer les fichiers nécessaires à l'application CORBA en C++:

```
idl Hello.idl
```

iii. Serveur C++

Pour implémenter le serveur, nous avons initialisé l'orb, crée l'objet en question, enregistrer la référence de l'objet dans un fichier, se connecter au service de nommage pour récupérer une référence, construire le nom de l'objet, enregistrer le nom au service de nommage et retrouver ce nom à l'aide du service de nommage. Tous ces étapes sont implémenté dans le fichier *server.cpp* annexe et en particulier dans la méthode *run().*

(1) Initialisation de l'ORB et du POA

Pour pouvoir gérer les transfères entre les clients et le serveur plus tard, la première chose à faire est l'initialisation du bus CORBA puis initialiser l'adaptateur d'objets racine POA. Ceci à travers les instructions suivantes :

```
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
CORBA::Object_var poaobj = orb->resolve_initial_references("NameService ");
PortableServer::POA_var poa =PortableServer::POA::_narrow(poaobj);
```

Figure 2 - Initialisation de l'ORB et du POA

(2) Création de l'objet

Dans notre cas, l'objet en question est *Hello_impl*. Il faut donc la crée. Et ceci à travers l'instruction suivante :

```
Hello_impl* helloImpl = new Hello_impl();
```

(3) Enregistrement de la référence

Le serveur doit sauvegarder la référence dans un fichier. Nous avons choisie de l'enregistrer sous le nom *Hello.ref*. Pour ce faire, on a procédé comme suit :

```
CORBA::String_var s = orb -> object_to_string(Hello);
  const char* refFile = "refSUN/Hello.ref";
  ofstream out(refFile);
```

Figure 3 - Enregistrement de la référence - Hello.ref

(4) Récupération de la référence de service de nommage

Pour pouvoir utiliser le serveur de nom décrit précédemment, il faut arriver à récupérer le contexte initial. Pour cela nous avons ajouté les lignes suivantes dans notre code :

```
CORBA::Object_var nsobj = orb->string_to_object("corbaloc
:localhost:9999/NameService");
CosNaming::NamingContext_var nc = CosNaming::NamingContext::_narrow( nsobj
);
```

Figure 4 - Récupération de la référence

La clé "*NameService*" permet d'obtenir une référence du contexte du *serveur* dans le *serveur de nom* (serveur d'amorçage) et *9999* est le port ouvert par le serveur d'amorçage. *Localhost* est l'url de notre objet.

(5) Construction d'un nom et Enregistrement du service de nommage

Le service *CosNaming* permet de créer des noms indépendants. La fonction *bind()* disponible dans *CosNaming* permet d'enregistrer le nom et l'objet correspondant qu'on vient de créer dans serveur de nom.

```
CosNaming::Name name;
name.length( 1 );
name[0].id = CORBA::string_dup( "myHello" );
name[0].kind = CORBA::string_dup( "" );


nc->bind( name, helloImpl );
```

Figure 5 - CosNaming et bind

(6) Activation de l'objet pour attendre les invocations

```
poa->impl_is_ready(CORBA::ImplementationDef::_nil() );
orb->run();
```

Figure 6 - Activation de l'objet

iv. Client JAVA

Le client JAVA doit pouvoir invoquer des objets Pour pouvoir attendre un objet référencé dans un service de nommage, les étapes suivantes sont nécessaires:

(1) Initialisation de l'ORB

Comme au niveau serveur, pour pouvoir gérer les transfères entre les clients et le serveur, nous devons initialiser le *bus CORBA* (*l'ORB*) grâce à les instructions suivantes :

```
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, props);
```

(2) Récupération de la référence du service de nommage

Le client doit ensuite récupérer la référence du service de nommage qui lui permettra d'obtenir le contexte initial. C'est ce contexte qui lui permettra par la suite de créer et d'initialiser l'instance ORB.

```
org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
```

Figure 7 - Récupération de la référence

(3) Construction du nom de l'objet

La référence du service de nommage étant effectué, nous pouvons maintenant créér une instance ORB du client qui nécessite :

- ✓ La création d'un objet propriétés
- ✓ Le paramétrage de la propriété de classe ORB sur la classe ORB produit
- ✓ Le paramétrage des propriétés de référence initiales

```
Properties props = new Properties();
props.put("org.omg.CORBA.ORBInitialPort", "9999");
props.put("org.omg.CORBA.ORBInitialHost", "localhost");
ORB orb = ORB.init(args, props);
```

Figure 8 - Construction du nom de l'objet

(4) Execution

Une fois toutes ces étapes effectuées, le client peut invoquer la méthode say_Hello() du serveur C++. Ceci à l'aide donc du serveur de nom. Pour cela, à l'exécution, le client doit appelle la méthode init() de l'ORB avec en paramètre: le nom du serveur et son port d'écoute comme suit :

```
java client -ORBInitialPort 9999
```

Le code complet est inclus dans la **Section 1** de l'annexe.

D) Conclusion

Après implémentation du serveur de nom, du serveur du client et des classes (fichiers) associés, on dispose maintenant d'un outil permettant donc d'invoquer dynamiquement des opérations à l'aide des services de nommage. Et ceci sans se soucier du langage de programmation utilisé par le client (java, C++).

Exercice 4 : Activation

Dans cet exercice, on va expérimenter le mécanisme d'activation dans CORBA avec le principe de téléchargement à l'aide d'un serveur web. Nous nous somme référés à la documentation d'Orbacus 4.3.4[3] pour implémenter l'activation sur l'exemple 1. Et dans un deuxième temps, on va comparer ce mécanisme avec la technologie Java-RMI.

A) Orbacus Implementation Repository (IMR)

IMR donne la possibilité de la liaison indirecte des références d'objets persistent.
En d'autre terme, cette liaison vas nous permettre de desserrer la conjonction entre le client et le serveur de façon que l'emplacement du serveur change sans que le client s'affecte.

Pour implémenter cette approche, on fournit pour chaque client, un Interoperable Object Reference (IOR)[4] qui contient les détails du client pour communiquer avec CORBA et se référer à l'IMR, alors le client ne vas plus se connecter au serveur, mais plutôt à l'IMR qui offre aussi la possibilité de lancer des serveurs à la demande à l'aide de l'Object Activation Daemon (OAD) qui sera responsable pour l'activation des serveurs.

B) Comment ça marche

D'abord, le client envoie la demande en utilisant une référence à l'objet cible (servant).
Cette demande est ensuite reçue par l'ORB, qui transmettra la demande au POA.
Le POA envoie la demande au servant, ce dernier prend cette demande puis renvoie le résultat au POA, à l'ORB, et enfin au client.
Notre programme peut avoir plusieurs PAO, alors l'ORB va utiliser une clé d'objet pour qu'il puisse envoyer les demandes nécessaires aux PAO relatifs. Cette clé d'objet est un identificateur qui est une partie de la demande et sera conservé dans la référence de l'objet.

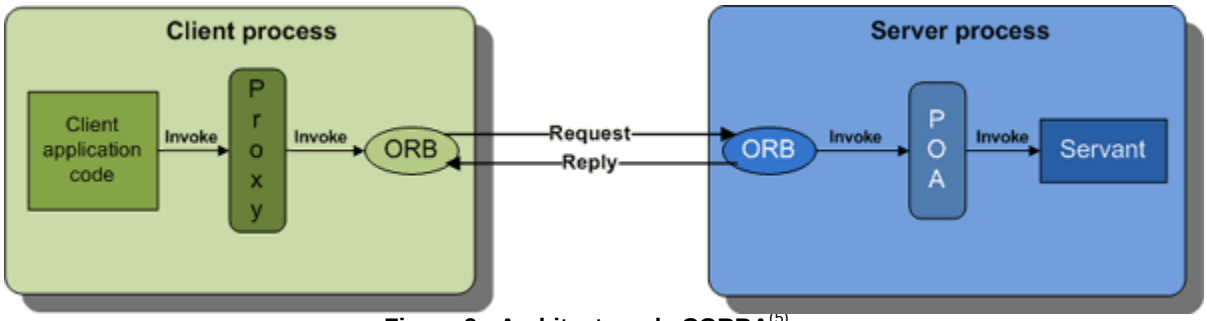

Figure 9 - Architecture de CORBA[5]