Chapitre 7

CORBA

Présentation de la norme CORBA.
Les principes de base de la programmation CORBA.
Mise en œuvre de CORBA avec JacORB.

1. Les logiciels

1.1. Java (JEE)

http://www.oracle.com/technetwork/java/javaee/downloads/index.html
java_ee_sdk-6u1-web-windows.exe
contient la version 7 de java :
C:\glassfish3\jdk7\bin
PATH ajout de C:\glassfish3\jdk7\bin

Documentation java
http://www.oracle.com/technetwork/java/javase/documentation/java-se-7-doc-download-435117.html
jdk-7u15-apidocs.zip

1.2. JacORB

Tous les exemples de code qui suivent sont faits avec la version 2.2.3 de Jacorb.
(Ne pas prendre une version plus récente car ces exemples de codes ont été créés il y a quelques années et les procédures de compilation demanderaient à être adaptées).

Le site de téléchargement :

http://www.jacorb.org/download.html

[Top | Features | Documentation | License | Download | Support]
[Authors | Customers | Sponsors | How to Contribute | Mailing lists]

JacORB 2.2.4

Binary version | Full source code

JacORB 2.2.3

- Binary version: tar.gz | zip
- Source version: tar.gz | zip

JacORB 2.2.2

- Compact version: gzipped tar-archive, binary only
- Compact version: zip-Archive, binary only
- Full version with source code.(gzipped tar)
- Full version with source code.(Zip format)

JacORB 2.2.1

On obtient le fichier zip suivant
JacORB-2.2.3-binary.zip

Vous le dézippez dans C:
C:/JacORB-2.2.3

Ne pas l'installer ailleurs que dans C: car les scripts .du répertoire bin comme jaco.bat ont le path en dur. (Dans la dernière version 3.3 ces scripts utilisent la variable $JACORB_HOME).

Puis, maj des variables d'environnement :
Ordinateur/Propriétés systemes/Paramètres systèmes avancés/
Variable d'environnement
JACORB_HOME=C:/JacORB-2.2.3 (Elle est utilisée dans les fichiers build.xml utilisés par ant pour compiler).

Mettre dans le PATH :
C:\JacORB-2.2.3\bin

1.3. Ant

Pour compiler les applications Jacorb, vous devez télécharger Ant qui est un outil de compilation, d'exécution et de déploiement.

Le site de téléchargement :
http://ant.apache.org/bindownload.cgi

Vous obtenez le zip suivant :
apache-ant-1.8.4-bin.zip

Vous le dézippez dans D: parexemple :
D:/apache-ant-1.8.4

Mettre dans le PATH :
D:\apache-ant-1.8.4\bin

2. Introduction

2.1. Définitions

CORBA, acronyme de Common Object Request Broker Architecture, est une architecture logicielle, pour le développement de composants et d'Object Request Broker ou ORB.

Ces composants,

- sont assemblés afin de construire des applications complètes,
- peuvent être écrits dans des langages de programmation distincts,
- être exécutés dans des processus séparés

- être déployés sur des machines distinctes.

Corba est un standard maintenu par l'Object Management Group
http://www.omg.org/spec/CORBA/3.3/
Pour plus d'information, vous pouvez consulter les 2 pdf téléchargés qui se trouvent sur le site (à côté du présent cours).

Historique :

- **Corba est une norme** créée en 1992, initiée par différents constructeurs et éditeurs dont Sun, Oracle, IBM,… regroupés au sein de l'Object Management Group.
- C'est avec la version 2 de Corba (fin 95) qu'est apparu le protocole standard *IIOP* et l'Interface description language (*IDL*).
- La version 2.3 rend interopérables Corba et RMI.
- La version 3 de Corba spécifie 16 types de services (nommage et annuaire des objets, cycle de vie, notification d'événements, transaction, relations et parallélisme entre objets, stockage, archivage, sécurité, authentification et administration des objets, gestion des licences et versions,…) mais tous ne sont pas mis en œuvre dans les ORB du marché.

Tous ne sont pas gratuits. Leurs prix sont plutôt élevés

- VisiBroker (C++/Java)
- ObjectBroker (C++)
- Orbix (C++/Java)
- JavaOrb (Java) (gratuit)
- Jacorb (Java) (gratuit)
- Tao (C++) (gratuit)

Chacun de ces logiciels n'implémente pas toutes les fonctionnalités de la norme.

Ces logiciels sont des Middleware qui appartiennent à la famille des ORB (Object Request Broker).

2.2. Un choix de conception

La technologie Corba adopte une approche essentiellement **orientée objet**.

Chaque composant est décrit sous la forme d'une interface écrite en langage IDL.

Une correspondance a été spécifiée entre le langage IDL et différents langages de programmation.

Des précompilateurs dédiés permettent de générer automatiquement le squelette de l'interface IDL dans un langage donné, en produisant aussi le code qui assure l'appel de fonctions distantes et le traitement des résultats. Ce code porte le nom de ***stub*** du côté client et de ***skeleton*** du côté serveur.

3. L'ORB

3.1. Présentation

ORB est le sigle de Object Request Broker (traduction littérale : courtier de requêtes objet).

Un ORB est un "bus logiciel" par lequel les objets envoient et reçoivent des requêtes et des réponses, de manière transparente : il s'agit de l'activation ou de l'invocation par un objet, et à distance, de la méthode d'un autre objet (dit "distribué") - en pratique ces objets invoqués sont souvent des services.

Un ORB s'apparente à une tuyauterie permettant les échanges de messages entre objets. Les ORB appartiennent à la famille des middlewares ou intergiciels. La plupart des ORB (hormis la technologie COM/DCOM de Microsoft) s'appuient sur la norme CORBA édictée par l'OMG.

Deux ORB peuvent communiquer entre eux au travers du protocole IIOP (Internet Inter-Orb Protocol, voir (en) General Inter-ORB Protocol)

Les rôles d'un ORB sont nombreux :

- localiser les objets distants.
- établir la relation entre tous les éléments présents dans cette architecture.
- prendre en charge le dialogue entre les objets serveurs et les différents clients qui s'y connectent.
- rendre ce dialogue transparent (ou presque) quel que soient les plates-formes, systèmes d'exploitation ou langages.
- coordonner (intégration système) les produits du développement, les outils et les composants du système
- gérer les erreurs et la destruction des objets
- répartir la charge CPU entre différentes machines
- générer automatique (précompilateurs d'IDL) les éléments informatiques permettant la mise en oeuvre de la communication entre un client et un serveur
- permettre la description des interfaces (le langage IDL)

Avec Internet et le besoin constant de développer des applications de plus en plus complexe, la technologie CORBA est utilisée par de nombreux serveurs d'application pour mettre en œuvre l'architecture intranet des services.
Exemple JBOSS : http://www.jboss.org/jbossiiop

3.2. Les services CORBA

Les services CORBA sont préconisés par l'OMG et sont développés en partie par les éditeurs.

On peut citer les services suivants :

- **Naming** Nommage des objets lors de leurs créations
- **Events** Production des événements asynchrones au niveau des serveurs.

- **Life Cycle**(Cycle de vie) Ensemble d'outils pour la gestion d'un objet durant toute sa durée de fonctionnement, création d'objets standardisés
- **Persistence** Stockage et restauration des objets
- **RelationShips** Gérer les relations inter-objets.
- **Externationalization** (et internationalization) Gère le propagation de l'objet dans un flux de données, éventuellement entre des ORB différents.
- **Transaction** Garantir l'intégrité des différents objets du système quels que soient les problèmes survenant en cours de traitement.
- **Concurrency** Gère les problèmes de concurrence d'accès à des ressources partagées.
- **Licensy** Facturation des clients en fonction de l'utilisation réelle des objets.
- **Query** Requêtes d'accès aux caractéristiques des objets par un autre objet.
- **Collection** Outils pour parcourir une collection d'objets grâce à des itérateurs.
- **Properties** Ajout dynamique et temporaire d'une propriété supplémentaire à un objet.
- **Security** Authentification des clients, cryptage des données, ...
- **Time** Synchroniser les clients et le serveur.
- **Trader** *(Vendeur)* Localiser des objets en fonction de critères d'interrogation précis (équivalent des pages jaunes).

3.3. Les composants de base d'un ORB

Un ORB est composé au moins de 3 parties :

IDL (Interface Definition Language)

- langage de définition des interfaces d'une architecture CORBA
- trouver un langage commun et unifiant les concepts de définition des données et des objets
- les précompilateurs

POA (Portable Object Adaptater)

- gère les objets distribués (ou servant)
- définit des règles de comportement des objets distribués (méthodes distantes)
- agit sur le cycle de vie des objets distribués
- on parle de "serveur CORBA"

Un principe de nommage : l'IOR (Interoperable Object Référence)

- norme d'encodage unique de la référence d'un objet distribué permettant sa reconnaissance et sa nature sur le réseau
- il contient
  - le nom complet de l'interface IDL correspondant à l'objet ;
  - l'adresse IP de la machine hôte de l'objet ;

- le port du serveur de l'objet ;
- une clé désignant l'objet. Son format dépend de l'implantation du bus.

- les IOR sont des chaînes de caractères hexadécimales
- ces IOR deviennent transparents si on utilise un adaptateur d'objet (ou service de nommage)

3.4. Le POA

Le rôle du POA est de déterminer quel servant doit être utilisé quand une requête cliente est reçue. Le POA transmet alors la requête au servant.

Chaque servant est identifié par une référence unique dite IOR Interoperable Object Reference. C'est le POA qui assigne les IOR aux objets CORBA.

Un POA appelé rootPOA est crée a l'instanciation de l'ORB. Ce POA n'est pas paramétrable et il est utilisé uniquement pour créer des POA fils qui peuvent être, eux, paramétrés.

Ces POA fils sont gérés par un POAManager qui peut notamment les détruire quand ils ne sont plus utiles. Il peut également les mettre en état d'attente ce qui est utile en cas d'arrêt temporaire d'une partie de l'application.

Un servant peut être activé soit au lancement du POA soit à la demande. Quand un servant est activé auprès du POA son IOR est stocké dans l'Active Object Map.

Le POA gère un "pool" de thread (Serveur de Socket) :
Le pool est défini par un minimum et un maximum

- le minimum est le nombre de thread créés à la création du POA
- le maximum est le nombre maximal de thread pouvant être créés par le POA

```
# thread pool configuration for request processing
jacorb.poa.thread_pool_max=20
jacorb.poa.thread_pool_min=5
```

A chaque requête sur un servant, le POA utilise un thread libre pour traiter la requête.

Si il n' y a pas de thread libre, il en crée un.

A moins que le nombre maximum soit atteint, alors la requête est bloqué (mise en attente d'un thread)

Les threads au dessus du nombre minimum sont détruits après le traitement de leurs requêtes.

La priorité d'exécution des threads est par défaut la plus haute sauf si la priorité est modifiée.

```
# if set, request processing threads in thePOA
# will run at this priority. If not set or invalid,
# MAX_PRIORITY will be used.
#jacorb.poa.thread_priority=
```

Cette valeur doit être entre les valeurs de Java: Thread.MIN_PRIORITY et Thread.MAX_PRIORITY

Tout servant (ou Objet Distribué) hérite de la classe org.omg.PortableServer.**Servant**

Exemple:

- dans l'dl : interface InterfaceDeviseOD …
- dans le code : public class DeviseOD extends InterfaceDeviseODPOA
- dans le code généré par l'idl : public abstract class InterfaceDeviseODPOA

  extends ***org.omg.PortableServer.Servant***
  implements InterfaceDeviseODOperations

La classe Servant est une classe abstraite dont tous les servants doivent héritées.

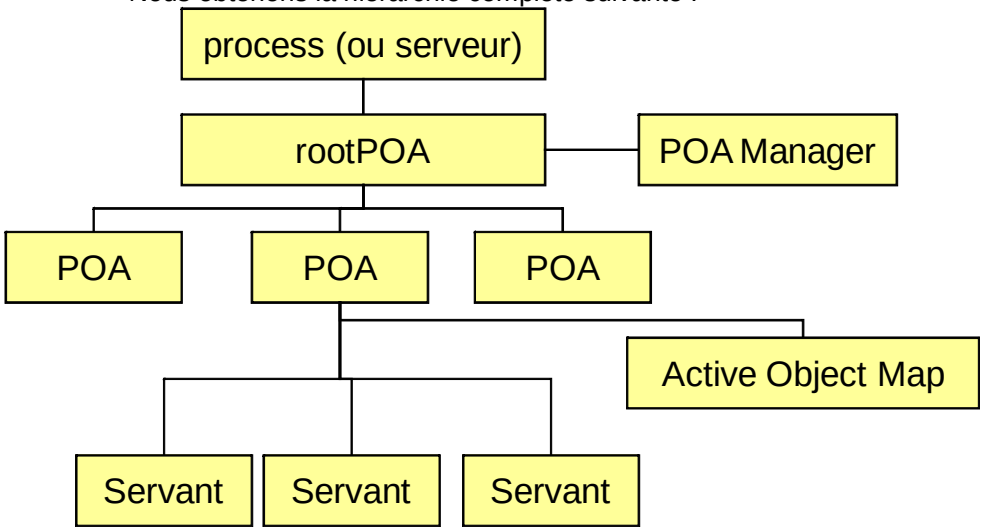
Remarque :
Le xxxxPOA généré par l'IDL correspond en RMI à la classe UnicastRemoteObject.

3.5. La hiérarchie des composants d'un serveur CORBA

Ainsi une application serveur CORBA est composé de différents éléments qu'il est nécessaire de connaitre pour différentes raisons :

- il faut créer soi-même ces composants dans nos programmes même si cela se fait souvent toujours de la même façon
- on peut tuner ces composants avec la mise à jour de leurs propriétés et donc connaître au moins leur existence et leur nom
- il est important de connaître leur rôle afin de maitriser nos interventions de maintenance sur le code de l'application

Nous obtenons la hiérarchie complète suivante :



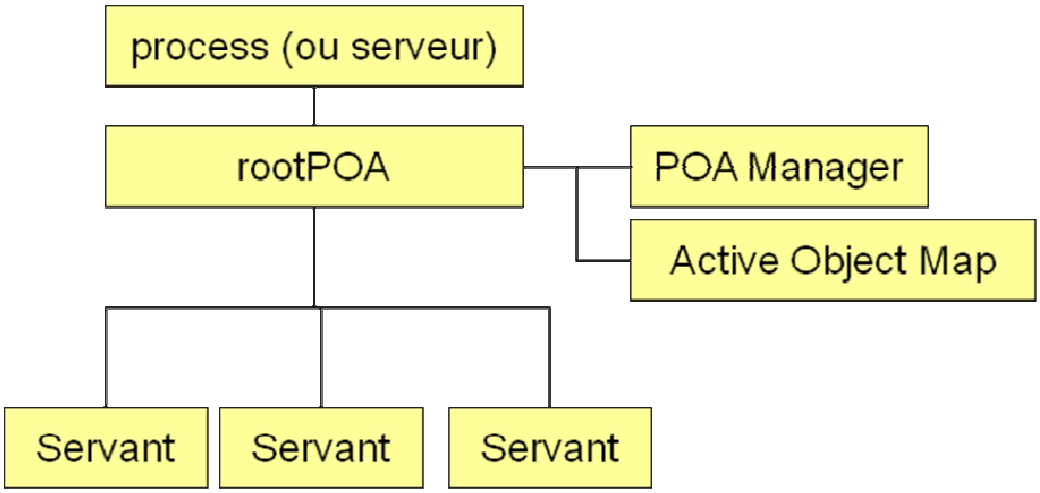
L'application "CORBA" est un process (ou classe main Java).

Le rootPOA est un objet qui réunit le POAManager et les POA créés (dans la plupart des cas 1 seul POA suffit). Le rôle du POAManager est de gérer tous les POA de l'application.

Chaque POA contient les servants et la table des IOR des servants (ActiveObjectMap).

La plupart du temps, nous ne gérons pas des types de comportement des servants différents (priorités différentes, …).

Nous simplifions donc cette hiérarchie en ne créant pas de POA particulier et en utilisant directement le rootPOA.



Nous verrons cela dans les exemples qui suivent.

4. Mise en œuvre d'une application CORBA

4.1. Les API et Library

L'API JAVA (1.7) contient les packages :
org.omg.CORBA. *
org.omg. *

Ces packages contiennent deux grandes familles d'éléments :
- les interfaces de définition des services CORBA dont l'instanciation et l'implémentation sont réalisés par l'ORB de l'éditeur (dans nos exemples: JacORB)
- les éléments réels Java de projection de l'IDL
- des classes diverses utilitaires

Comme nous allons le voir, on peut créer une application CORBA avec ou sans service de nommage.

Précisons déjà que tout servant (ou objet distribué) est identifié et référençable par son IOR indépendamment de tout service de nommage.

4.2. Notre exemple

Voir sur le site **http://jacques.laforgue.free.fr/SITE_NSY102/Site/Exemples.phtml**

les exemples **ExempleCh07_01_xxx**

ExempleCh07_01_cas0_Corba : description de l'exemple : Voir **ExempleCh07_01.pdf**

Nous utilisons cet exemple pour illustrer les principes de mise en œuvre des applications distribués avec CORBA.

Cet exemple contient 6 exemples d'utilisation de CORBA :

- ExempleCh07_01_cas1_hello
- ExempleCh07_01_cas2_hello_with_ns
- ExempleCh07_01_cas3_devise
- ExempleCh07_01_cas4_devise_avec_Tie
- ExempleCh07_01_cas5_event_push
- ExempleCh07_01_cas6_event_push_ref_corba

Prenons le premier exemple : **cas1_hello**
Il correspond à l'utilisation d'un objet distribué sans utiliser un service de nommage.

4.3. L'interface de l'objet distribué

```
module hello {
  interface GoodDay {

     string hello_world( in string msg );

     attribute long compteur;
  };
};
```

Le compilateur IDL de JacORB prend cette interface en entrée et génère les fichiers :

GoodDay.java
GoodDayHelper.java
GoodDayHolder.java
GoodDayOperations.java
GoodDayPOA.java
GoodDayPOATie.java
_GoodDayStub.java

4.4. Le Servant (ou Objet distribué)

Le fichier GoodDayImpl.java

```
package hello;

import org.omg.CORBA.*;

public class GoodDayImpl extends GoodDayPOA
{
    private String location;
    private int compteur ;

    public GoodDayImpl( String location )
    {
        this.location = location;
        this.compteur = 0;
    }

    public String hello_world(String wide_msg)
    Cette méthode est celle déclarée dans l'interface idl. Pour connaitre
précisément la syntaxe il faut aller voir la projection java.
    {
```

```
            compteur++;
            System.out.println("The message is  : " + wide_msg );
            return "Hello : Mr " + wide_msg+" "+location+" "+compteur;
        }

        public int compteur()
        {
            return compteur;
        }

        public void compteur(int c)
        {
            compteur = c;
        }
    }
```

En CORBA un objet distribué (ou servant) est une classe qui **hérite d'un POA** généré par l'DL.

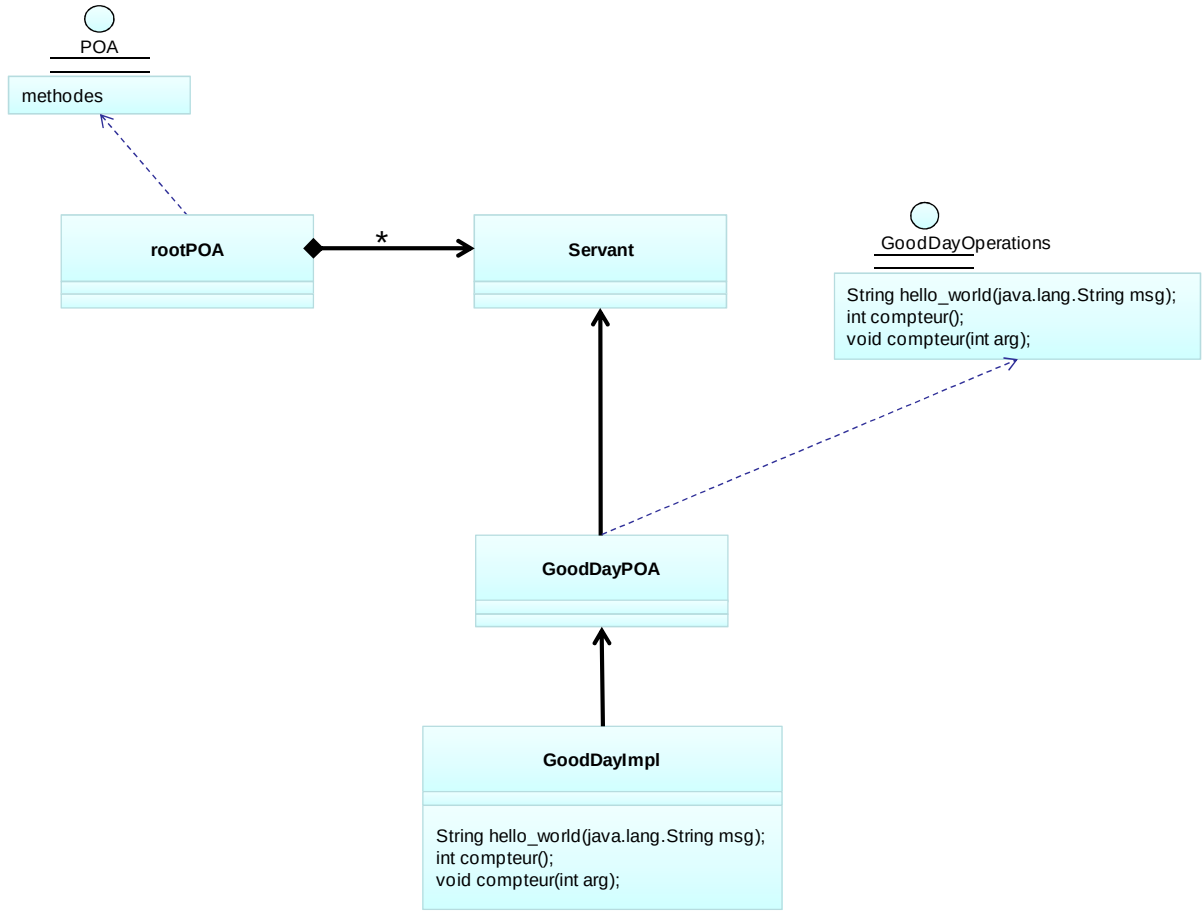
Cette classe générée hérite de Servant et implémente l'interface contenant les méthodes déclarées dans le fichier idl.

Ainsi la classe GoodDayImpl doit i**mplémenter** toutes les méthodes de l'interface **GoodDayOperations**.

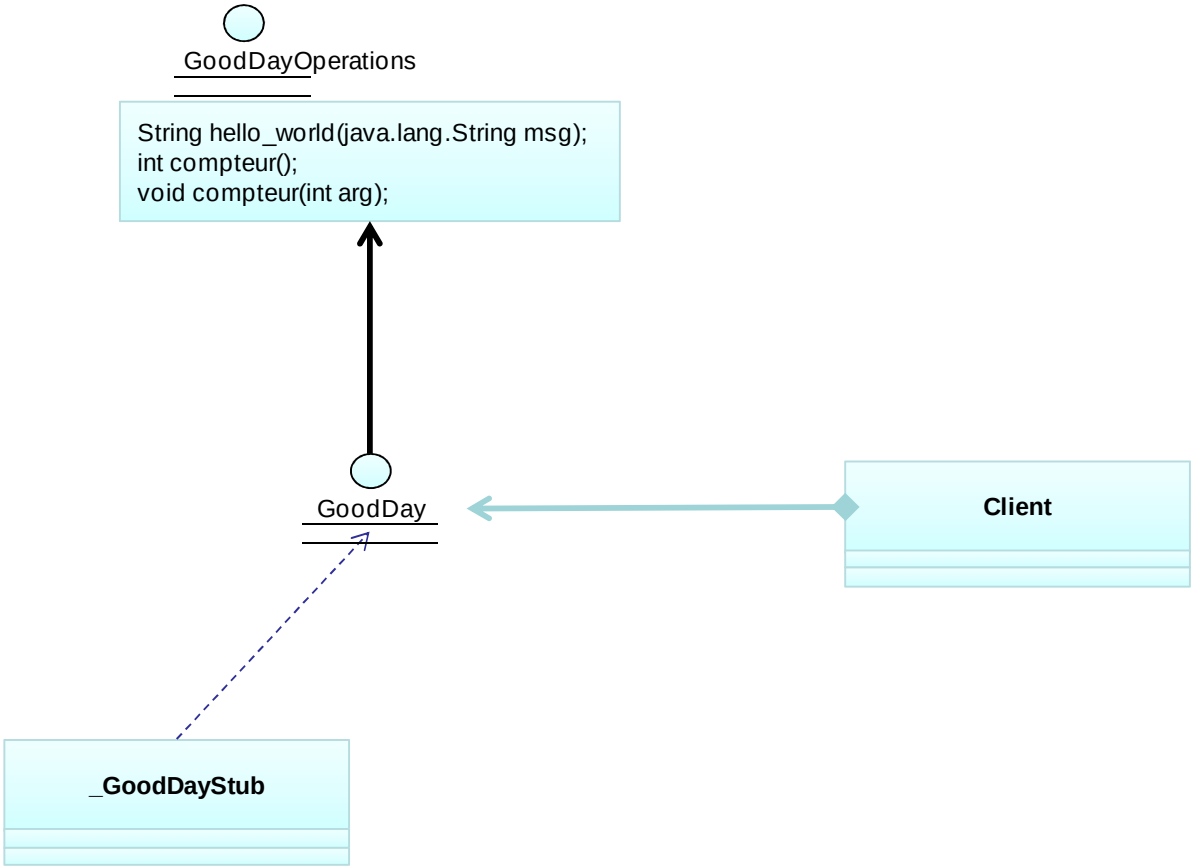
Dans notre exemple :

GoodDayImpl hérite de GoodDayPOA

GoodDayPOA est une classe abstraite qui implémente l'interface GoodDayOperations et qui hérite de Servant.



Côté client :



Le client utilise l'objet distribué à travers l'interface GoodDay qui est implémenté par le stub _GoodDayStub.
L'interface GoodDay est vide mais hérite de GoodDayOperations.

4.5. Le Serveur (sans NS)

Les étapes pour créer un "serveur" CORBA dans lequel un servant est créé sont :

- initialisation de l'ORB
- création du root POA
- activation du manager du root POA
- création du servant
- création de la référence CORBA du servant (IOR)
- écrire l'IOR du servant dans un fichier
- mise en attente des requêtes clientes

Le fichier Server.java

```
package hello;
import java.io.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;

public class Server {
```

```
      public static void main(String[] args)  {
          if( args.length != 1 ) {
        System.out.println("Usage: jaco hello.Server <ior_file>");
        System.exit( 1 );
      }
          try {
        //initialisation de l'ORB
        ORB orb = ORB.init( args, null );
        Initialisations de l'ORB standard en utilisant les arguments du
programme: important pour passer les propriétés à l'ORB

        //Création du POA
        POA poa =
          POAHelper.narrow(
                     orb.resolve_initial_references( "RootPOA" ));
        On récupère le root POA comme un servant prédéfini dans l'ORB


        // Activation du POA manager
        poa.the_POAManager().activate();


        // Création du servent
        GoodDayImpl goodDayImpl = new GoodDayImpl( "CNAM NFA032" );
        Le servant est créé comme n'importe quel objet Java. Jusque là ce
n'est pas encore un servant utilisable

        // Création de la référence CORBA (IOR)
        org.omg.CORBA.Object obj =
          poa.servant_to_reference( goodDayImpl );
        La référence du servant est créée par le POA. A partir de là le
servant est un objet distribué avec son IOR calculé par le POA

        // Ecrire l'IOR dans un fichier
        PrintWriter pw =
          new PrintWriter( new FileWriter( args[ 0 ] ));
        pw.println( orb.object_to_string( obj ));
        Le IOR du servant est écrit dans un fichier afin que le client
puisse le lire pour récupérer l'IOR et donc utiliser le servant à distance


        pw.flush();
        pw.close();

        // Attentes des requetes
        orb.run();
        Attente des requêtes du client. Cette instruction est bloquante. Si
vous voulez que le programme réalise un traitement en // il faut démarrer
un thread avant.

      }
          catch( Exception e )
              {
                  System.out.println( e );
              }
      }
  }
```

4.6. Le Client (sans NS)

Les étapes pour créer un client sont :

- initialisation de l'ORB
- récupération de l'IOR CORBA du servant en lisant le fichier
- création de la référence CORBA du servant distant
- connexion au servant distant (interface)
- utilisation des méthodes distantes

Le fichier client.java

```
package hello;

import java.io.*;
import org.omg.CORBA.*;

public class Client {
    public static void main( String args[] ) {
    if( args.length != 1 ) {
            System.out.println( "Usage: jaco hello.Client <ior_file>" );
            System.exit( 1 );
        }
        try {
            File f = new File( args[ 0 ] );

            //check if file exists
            if( ! f.exists() ) {
                      System.out.println("File " + args[0] +
                                         " does not exist.");

                System.exit( -1 );
            }

            //check if args[0] points to a directory
            if( f.isDirectory() ){
                 System.out.println("File " + args[0] +
                                    " is a directory.");

                System.exit( -1 );
            }

            // Initialisation de l'ORB
            ORB orb = ORB.init( args, null );

            // Lecture de l'IOR
            BufferedReader br =
                new BufferedReader( new FileReader( f ));

            // Récupération de la référence CORBA à partir de l'IOR
            org.omg.CORBA.Object obj =
                orb.string_to_object( br.readLine() );
            br.close();

            // Récupération du servant
            GoodDay goodDay = GoodDayHelper.narrow( obj );
            Cette classe fabrique le stub de l'objet distribuée en
fonction de sa reference CORBA.
```

```
              // Exemple de l'utilisation du servant
              goodDay.compteur(100);
              while(true)
                  {
                      // invoke the operation again and print the wide
string result
                      String str = goodDay.hello_world( "J. Laforgue ");
                      System.out.println(str);
                      try{Thread.sleep(1000);}catch(Exception ex){};
                  }

          }
          catch( Exception ex )
  {
              ex.printStackTrace();
          }
      }
  }
```

4.7. La classe ORB

La classe ORB est dans le package org.omg.CORBA

Classe de l'API Java permettant de créer un ORB :

- soit par défaut
- soit à partir d'un ORB du commerce

L'ORB fourni un script de lancement des applications CORBA : jaco.bat (en windows):

Exemple sur jacorb, le script d'exécution **jaco.bat** :

java -Djava.endorsed.dirs="C:\JacORB-2.2.3\lib"
-classpath "C:\JacORB-2.2.3\lib\jacorb.jar;
C:\JacORB-2.2.3\lib\logkit-1.2.jar;
C:\JacORB-2.2.3\lib\avalon-framework-4.1.5.jar;%CLASSPATH%"
-Djacorb.home=C:\JacORB-2.2.3
-Dorg.omg.CORBA.ORBClass=org.jacorb.orb.ORB
-**Dorg.omg.CORBA.ORBSingletonClass=org.jacorb.orb.ORBSingleton** %*

```
  ORB orb = ORB.init():
```

Singleton d'instanciation de l'ORB

```
  orb.run()
```

Et de nombreuses autres méthodes ….. (nous les verrons en avançant dans le cours)

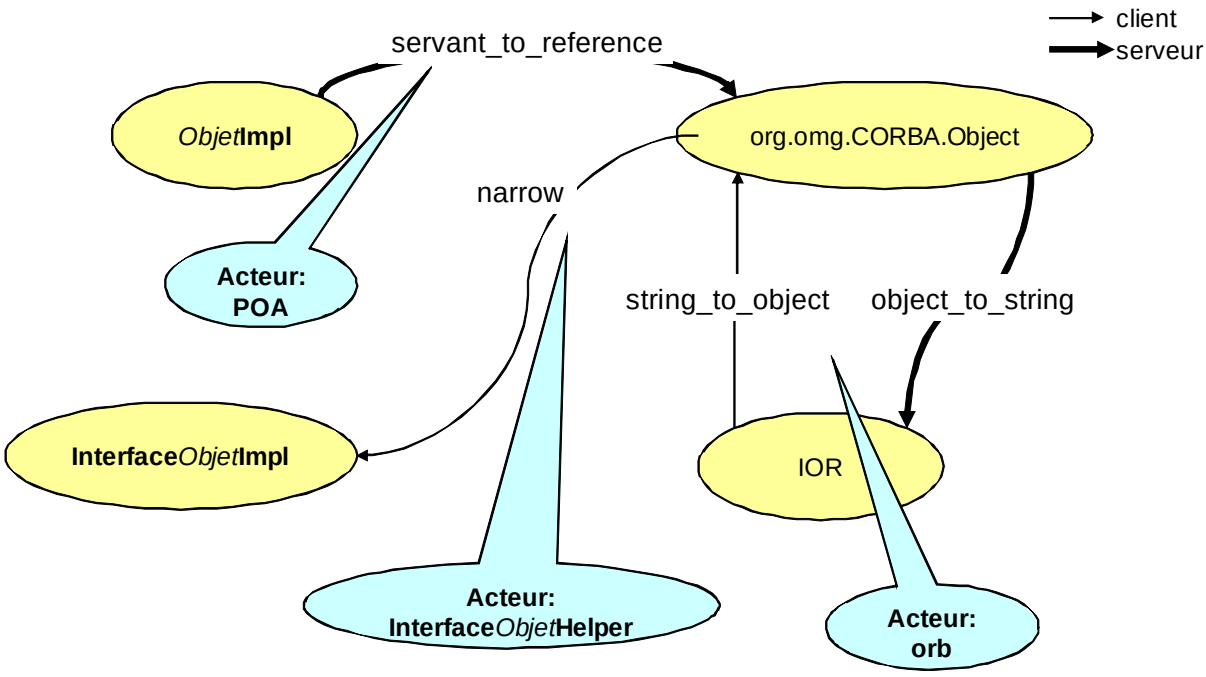
Cette classe assure les fonctions suivantes :

- initialise l'implémentation du ORB à partir de propriétés et de paramètres
- obtenir une référence initiale d'un service prédéfini CORBA comme le *NameService* en utilisant la méthode *resolve_initial_references*
- convertir une référence d'objet CORBA en String (IOR) et l'inverse
- créer des objets comme :
  - TypeCode
  - Any
  - NamedValue
  - Context

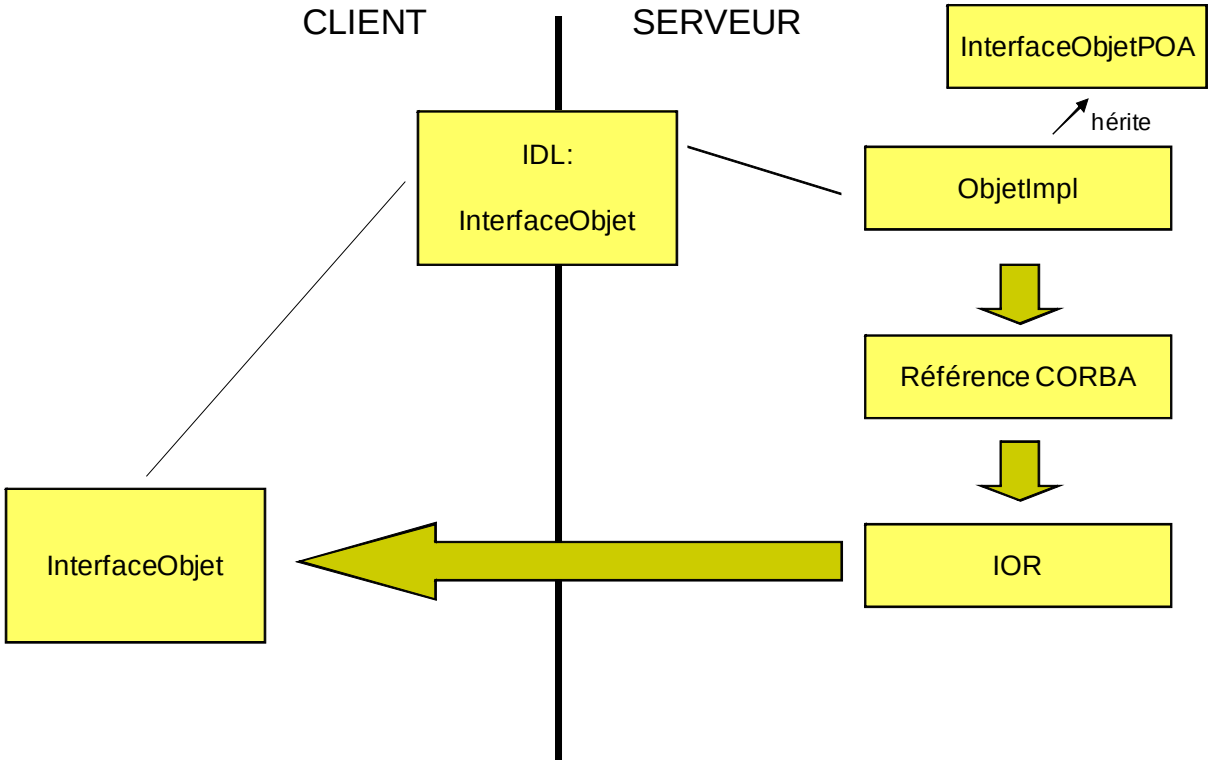
- Environment
- lists (comme une NVList) contenant ces objets

- envoyer de multiple messages sends multiple messages avec le DII (Dynamic Invocation Interface)
- La classe ORB peut être utilisée pour obtenir des références des objets implémentés n'importe où sur le réseau.

4.8. Synthèse (sans Naming Services)



Commentaires en cours



4.9. Notre exemple

Ce deuxième exemple est réalisé en utilisant un service de nommage.

Il correspond au cas : **cas2_hello_with_ns**

Voir sur le site http://jacques.laforgue.free.fr l'exemple
ExempleCh07_01_cas2_ hello_with_ns
Voir aussi ExempleCh07_01.pdf joint à l'exemple.

4.10. Le service de nommage

4.10.1. Présentation

Dans cet exemple au lieu de se débrouiller à transmettre l'IOR de l'objet distribué par ses propres moyens, nous pouvons utiliser un service de CORBA : le **Naming Services**.

Ce service est unique sur le réseau.

Il centralise et rend transparent l'usage des IOR.

Le NS permet la recherche des objets distribués en fonction de noms symboliques leur ayant été associés. Par exemple, on recherchera l'objet de nom « répertoire ». C'est donc l'équivalent des « pages blanches » de l'annuaire téléphonique.

Le NS est prédéfini. On le récupère ainsi :

```
NamingContextExt nc = NamingContextExtHelper.narrow(
                        orb.resolve_initial_references("NameService"))
```

4.10.2. Utilisation

Le service de nommage stocke les IOR des servants.

Pour enregistrer un objet dans l'annuaire :

```
nc.bind(   nc.to_name( " le nom de l'objet ") ,
          la référence CORBA de l'objet );
```

nc.to_name : est une méthode qui décide de la syntaxe d'accès.

Pour récupérer la référence d'un objet CORBA à partir de l'annuaire :

```
org.omg.CORBA.Object obj = nc.resolve(nc.to_name("HELLO"));
```

Le service de nommage stocke les IOR dans un fichier qui est créé sur la machine au lancement du service (par défaut C:/NS_Ref.). Voir la propriété : jacorb.naming.ior
Le service de nommage s'exécute sur un port indéterminé (le premier libre) ou sur un port donné. Voir la propriété OAPort.

L'exécution du Naming Service se fait par la commande : ***ns***

- exécuter dans le répertoire contenant le fichier jacorb.properties
- ns [-Djacorb.naming.ior filename=<filename>] [-DOAPort=port] [-Djacorb.naming.time out=<timeout>]
- jaco jacorb.naming.NameServer [-Djacorb.naming.ior_filename=<filename>] [-DOAPort=port] [-Djacorb.naming.time_out=<timeout>]
- Exemple: ns -Djacorb.naming.ior_filename=/home/me/public/NS_Ref

La commande "ns" est propre à Jacorb.

4.11. Le Serveur (avec NS)

Les étapes pour créer un "serveur" CORBA dans lequel un servant est créé sont :

- initialisation de l'ORB
- création du root POA
- activation du manager du root POA
- **Connexion au service de nommage**
- création du servant
- création de la référence CORBA du servant (IOR)
- **Enregistrer la référence CORBA du servant dans l'annuaire**
- mise en attente des requêtes clientes

La classe de définition de l'objet distribué GoodDayImpl reste inchangée.

Le fichier Server.java

```
package hello_with_ns;

import java.io.*;
```

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;

import org.omg.CosNaming.*;

public class Server
{
    public static void main(String[] args)
    {
        try
            {
                //Initialisation de l'ORB
                ORB orb = ORB.init( args, null );

                //Création du POA
                POA poa =
                    POAHelper.narrow(
                       orb.resolve_initial_references( "RootPOA" ));

                //Activation du POA
                poa.the_POAManager().activate();

                //On récupère le Naming Services comme un servant
                //  prédéfini dans l'ORB
                // Récupération du Naming contexte root
                NamingContextExt contextRoot =
                    NamingContextExtHelper.narrow(
                       orb.resolve_initial_references("NameService"));

                // Création d'un sous contexte dans le contexte root
                NameComponent[] name = new NameComponent[1];
                name[0] = new NameComponent("sub","context");
                NamingContextExt subContext =
                    NamingContextExtHelper.narrow(
                       contextRoot.bind_new_context( name ));

                //Création du servant dans le contexte root
                GoodDayImpl goodDayImpl = new GoodDayImpl( "Somewhere" );

                //Création de la référene de l'objet
                org.omg.CORBA.Object obj =
                    poa.servant_to_reference( goodDayImpl );

                //Enreistrement de la référence du servant dans
                //  l'annuaire
                contextRoot.bind( contextRoot.to_name("HELLO"), obj);


                // Pour faire un test
                //Création d'un autre servant dans le sous contexte
                //"sub.context"
                GoodDayImpl goodDayImpl2 = new GoodDayImpl( "Somewhere2"
);

                //Création de la référene de l'objet
                org.omg.CORBA.Object obj2 =
                    poa.servant_to_reference( goodDayImpl2 );

                //Enreistrement de la référence du servant dans
                //l'annuaire
```

```
                subContext.bind( subContext.to_name("HELLO"), obj2);
                // Le même nom peut être utilisé dans deux contextes
                // différents

                //Attente des requêtes clientes
                orb.run();
            }
        catch( Exception e )
            {
                System.out.println( e );
            }
    }
}
```

4.12. Le Client (avec NS)

Les étapes pour créer un client sont :

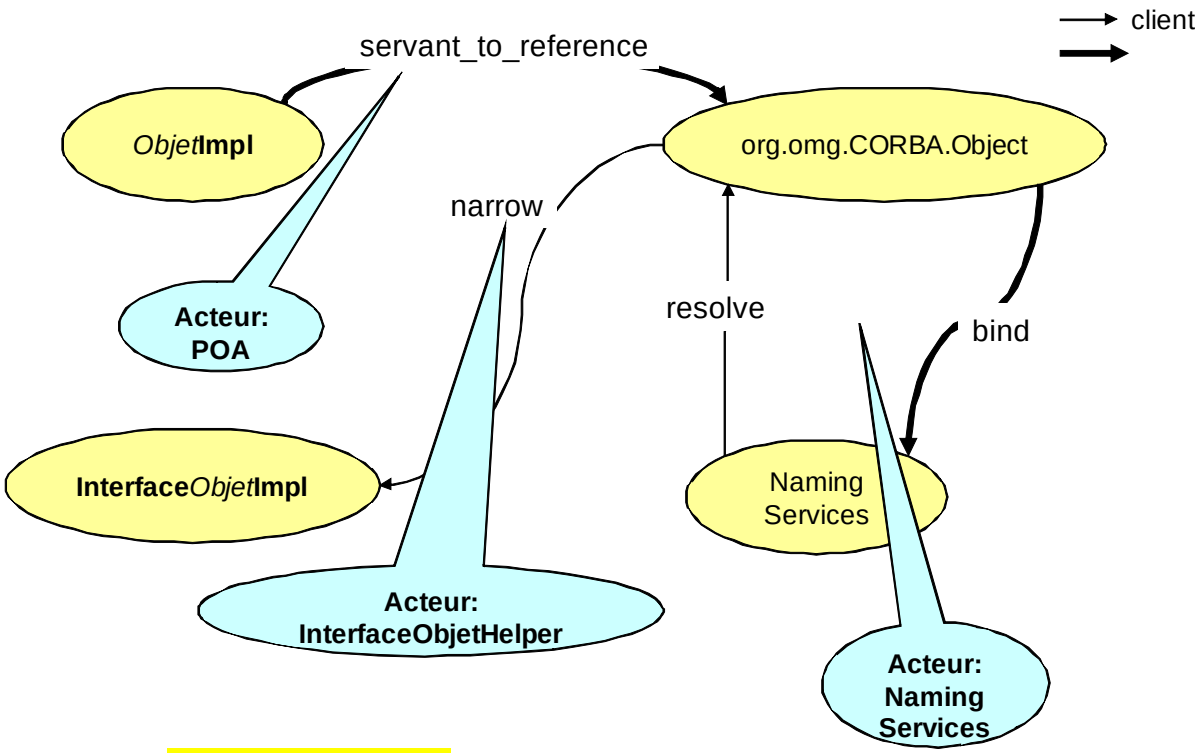
- initialisation de l'ORB
- **récupération de la référence du service de nommage**
- **récupération de la référence CORBA du servant via le service de nommage**
- connexion au servant distant (interface)
- utilisation des méthodes distantes

Le fichier Client.java

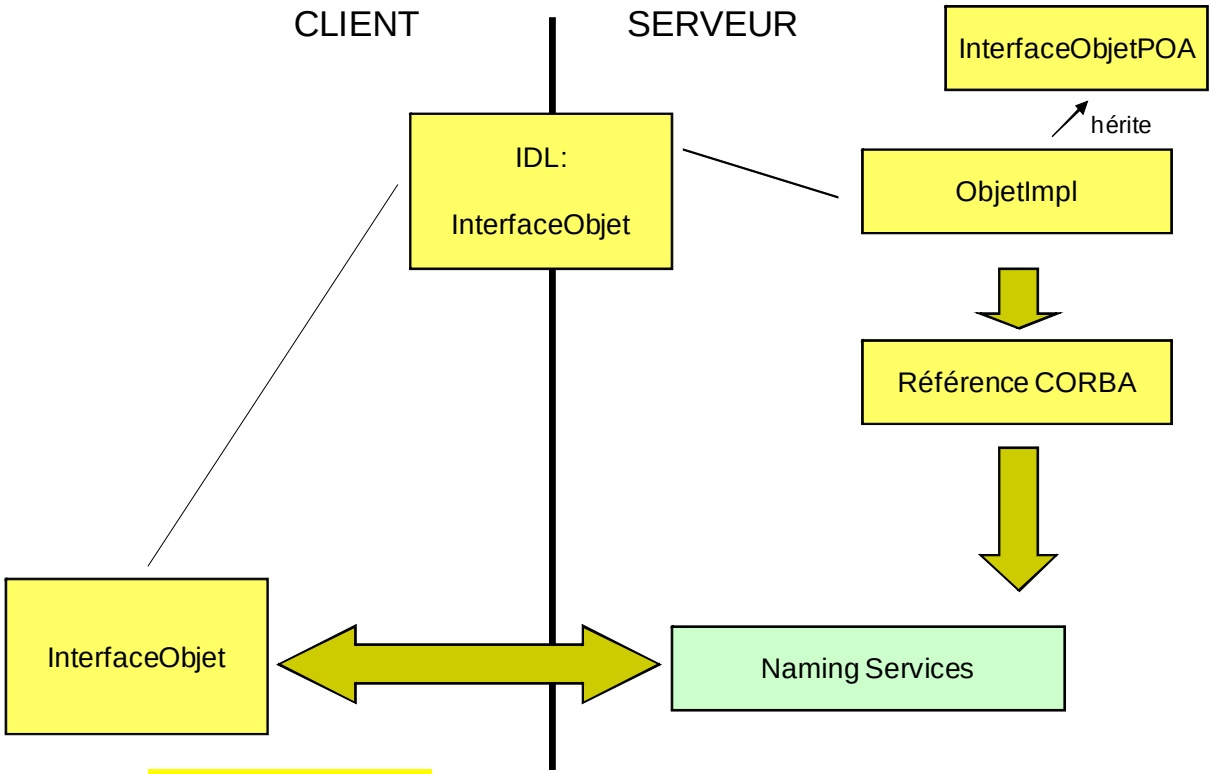
```
package hello_with_ns;

import java.io.*;
import org.omg.CORBA.*;

import org.omg.CosNaming.*;

public class Client
{
    public static void main( String args[] )
    {
        try
            {
                //Initialisation de l'ORB.
                ORB orb = ORB.init( args, null );

                // Récupération du Naming Service
                NamingContextExt nc =
                    NamingContextExtHelper.narrow(
                      orb.resolve_initial_references("NameService"));

                // Récupération de la référence CORBA à partir du NS
                org.omg.CORBA.Object obj = nc.resolve(
                                            nc.to_name("HELLO"));

                // Récupération du servant CONTEXT ROOT
                GoodDay goodDay =
                    GoodDayHelper.narrow(obj);

                // Utilisation du servant
                System.out.println( goodDay.hello_world( "CNAM"));

                NamingContextExt subContext =
```

```
                    NamingContextExtHelper.narrow(
                       nc.resolve( nc.to_name("sub.context") ));

                // Récupération de la référence CORBA à partir du NS
                org.omg.CORBA.Object obj2 = subContext.resolve(
                                        subContext.to_name("HELLO"));

                // Récupération du servant  SOUS-CONTEXTE
                GoodDay goodDay2 =
                    GoodDayHelper.narrow(obj2);

                // Untilisation du servant
                goodDay2.compteur(100);
                while(true)
                    {
                        // invoke the operation again and print
                        //    the string result
                        String str = goodDay2.hello_world( "J. Laforgue
");
                        System.out.println(str);
                        try{Thread.sleep(1000);}catch(Exception ex){};
                    }

            }
        catch( Exception ex )
            {
                ex.printStackTrace();
            }
    }
}
```

4.13. Synthèse (avec NS)



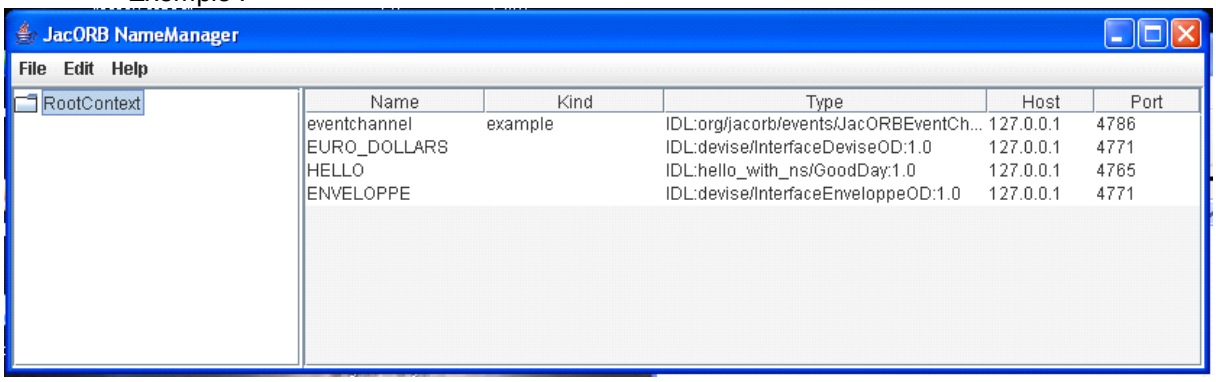
Commentaires en cours



Commentaires en cours

4.14. Hiérarchie de Name Spaces

Jacorb a un utilitaire permettant de visualiser le contenu du service de nommage :
La commande **nmg** (même paramètres que ns)
Exécuté dans le répertoire etc (par exemple)
Exemple :



En reprenant le même exemple.

Construire une hiérarchie de Name Spaces :

```
  // Récupération du Naming contexte root
  NamingContextExt contextRoot =
   NamingContextExtHelper.narrow(orb.resolve_initial_references("NameService
"));

  // Création d'un sous contexte dans le contexte root
  NameComponent[] name = new NameComponent[1];
```

```
name[0] = new NameComponent("sub","context");
NamingContextExt subContext =
          NamingContextExtHelper.narrow(
 contextRoot.bind_new_context( name ));
```

Création d'un servant dans le sous-contexte :

```
//Création d'un autre servant dans le sous contexte "sub.context"
GoodDayImpl goodDayImpl2 = new GoodDayImpl( "Somewhere2" );

//Création de la référene de l'objet
org.omg.CORBA.Object obj2 =
poa.servant_to_reference( goodDayImpl2 );

//Enreistrement de la référence du servant dans l'annuaire
subContext.bind( subContext.to_name("HELLO"), obj2);
// Le même nom peut être utilisé dans deux contextes différents
```

Utilisation par le client :

```
NamingContextExt subContext =
       NamingContextExtHelper.narrow( nc.resolve(
         nc.to_name("sub.context") ));

        // Récupération de la référence CORBA à partir du NS
        org.omg.CORBA.Object obj2 =
                  subContext.resolve(subContext.to_name("HELLO"));

        // Récupération du servant
        GoodDay goodDay2 =
          GoodDayHelper.narrow(obj2);

        // Utilisation du servant
        System.out.println( goodDay2.hello_simple() );
```

5. Les DP d'un OD avec CORBA

A FAIRE PLUS TARD

6. Exemple Ch07_01 : cas3_devise

Voir sur le site http://jacques.laforgue.free.fr l'exemple
ExempleCh07_01_cas3_devise
Voir aussi ExempleCh07_01.pdf joint à l'exemple.

Ce cas contient les cas fonctionnels suivants :

- un objet DeviseOD est un servant qui permet de convertir une monnaie
  - méthodes distantes de conversion de la monnaie
- un objet DesDevisesOD est un servant qui permet de stocker plusieurs DeviseOD
  - méthodes distantes qui permettent de créer une nouvelle devise et qui retourne la liste des devises gérées
- "mappage" de l'objet devise en une donnée de l'IDL

- le client utilise les servants créés par le serveur afin de mettre en évidence les différents cas.

Pour exécuter ce cas :
runNS.bat
runServeur.bat
runClient.bat

Le fichier server.idl

Le module contient les interfaces des 2 OD :
DeviseInt
Devises

```
module devise
{
  // Les informations de la devise distante
  struct DeviseData {
    string   nom1;
    string   nom2;
    double   taux;
  };

  // L'interface de la DEVISE
  interface DeviseInt
  {
       double convertir1(in double val);
       double convertir2(in double val);
       string getNom1();
       string getNom2();
       double getTaux();

       DeviseData getDeviseData();
  };

   //Type de la sequence de devise
   typedef sequence<DeviseData> DesDevises;

   // L'interface de la collection de devise
   interface Devises
   {
     DesDevises getDevises();
   };

};
```

Le fichier DeviseOD.java

Implémente les méthodes de l'interface **DeviseIntOperations**
hérite de **DeviseIntPOA**

```
package devise;

import org.omg.CORBA.*;


public class DeviseOD extends DeviseIntPOA
{
    private String nom1;
    private String nom2;
    private double taux;

    public DeviseOD( String nom1, String nom2, double taux )
    {
        this.nom1 = nom1;
        this.nom2 = nom2;
        this.taux = taux;
    }

  public double convertir1(double val)
  {
         return(val*taux);
  }

  public double convertir2(double val)
  {
         return(val/taux);
  }

    public String getNom1()
    {
        return(nom1);
    }

    public String getNom2()
    {
        return(nom2);
    }

    public double getTaux()
    {
        return(taux);
    }

  public DeviseData getDeviseData()
  {
    return new DeviseData(this.nom1,
                this.nom2,
                this.taux);
  }


}
```

Le fichier DesDevisesOD.java

implémente les méthodes de l'interface **DevisesOperations**
hérite de **DevisesPOA**

```
package devise;

import java.util.*;
import org.omg.CORBA.*;

import devise.DeviseData;

public class DesDevisesOD extends DevisesPOA
{
    ArrayList<DeviseData> devises;

    public DesDevisesOD()
    {
        this.devises = new ArrayList<DeviseData>();
    }

    public void ajouterDevise(String nom1, String nom2, double taux)
    {

        devises.add(new DeviseData(nom1,nom2,taux));
    }

  public DeviseData[] getDevises()
  {
        DeviseData[] r = new DeviseData[devises.size()];
        return(devises.toArray(r));
  }
}
```

Le fichier Server.java

- se connecte au gestionnaire de devise
- crée deux devises et les ajoute au gestionnaire de devise

```
package devise;

import java.io.*;

import org.omg.CORBA.*;
import org.omg.PortableServer.*;

import org.omg.CosNaming.*;

import devise.*;

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

        //init POA
        POA poa =
          POAHelper.narrow(
              orb.resolve_initial_references( "RootPOA" ));

        poa.the_POAManager().activate();

        NamingContextExt nc = NamingContextExtHelper.narrow(
               orb.resolve_initial_references("NameService"));


        // ===================================
        // Création de la table des devises
        DesDevisesOD lesDevises = new DesDevisesOD();
        nc.bind( nc.to_name("DEVISES"),
              poa.servant_to_reference(lesDevises) );


        // ===================================
        // Creation de la premiere devise
        DeviseOD deviseod = new DeviseOD( "EURO","DOLLAR",1.23 );

        // create the object reference
        org.omg.CORBA.Object obj =
          poa.servant_to_reference( deviseod );

        // Enregistrement dans l'annuaire
        System.out.println("Enregistrement de la devise EURO_DOLLAR");
        nc.bind( nc.to_name("EURO_DOLLAR") , obj);

        // Mise à jour du tableau des devises
        lesDevises.ajouterDevise("EURO","DOLLAR",1.23);

        // ===================================
        // Creation de la deuxième devise
        String nom1 = "EURO";
```

```
          String nom2 = "FRANC";
          double taux = 6.5;
          DeviseOD devise = new DeviseOD(nom1,nom2,taux);
          nc.bind( nc.to_name(nom1+"_"+nom2),
              poa.servant_to_reference(devise) );

          // Mise à jour du tableau des devises
          lesDevises.ajouterDevise(nom1,nom2,taux);


          // wait for requests
          System.out.println("ORB> run");
          orb.run();
        }
      catch( Exception e )
        {
          System.out.println( e );
        }
    }
  }
```

Le fichier Client.java

- se connecte au gestionnaire de devises
- récupère la liste des devises gérées
- se connecte aux devises pour utiliser les méthodes distantes de conversion de monnaie

```
package devise;

import java.io.*;
package devise;

import java.io.*;
import org.omg.CORBA.*;

import org.omg.CosNaming.*;

import devise.*;


public class Client
{
  public static void main( String args[] )
  {
    try {
        // Flot de lecture du clavier
        java.io.DataInputStream in =
          new java.io.DataInputStream(System.in);

              // initialize the ORB.
              ORB orb = ORB.init( args, null );

              // Récupération du Naming Service
              NamingContextExt nc =
                  NamingContextExtHelper.narrow(
           orb.resolve_initial_references("NameService"));


        // Acces à l'OD qui contient la liste des devises et
        // leurs caractéristiques
        Devises lesdevises = DevisesHelper.narrow(
                    nc.resolve(nc.to_name("DEVISES")));
        DeviseData[] tabdevises = lesdevises.getDevises();

        while (true)
          {
            // Les devises possibles
            //
            for(int i=0;i<tabdevises.length;i++)
              {
                System.out.println(String.format
                  ("%d : %10s vers %10s (taux:%.2f)",
                    i,tabdevises[i].nom1,
                    tabdevises[i].nom2,
                    tabdevises[i].taux));
              }

            // Choix de la devise
            System.out.print("Choix de la devise:");
            int num = Integer.parseInt(in.readLine());
```

```
              DeviseData devisedata = tabdevises[num];

              // Accès au servant de la devise
              DeviseInt devise =
                DeviseIntHelper.narrow(
                  nc.resolve(nc.to_name(
                   devisedata.nom1+"_"+devisedata.nom2)));

              // Les informations de la devise
              DeviseData infos = devise.getDeviseData();
              System.out.println( "Conversion de " +
                        infos.nom1 + " vers " +
                        infos.nom2 + " avec un taux de:" +
                        infos.taux);
              System.out.println( "Conversion de " +
                        devise.getNom1() + " vers " +
                        devise.getNom2() + " avec un taux de:" +
                        devise.getTaux() );

              double val,resultat;

              System.out.print(devisedata.nom1+" : ");
              System.out.flush();
              val = Double.parseDouble(in.readLine());
              resultat = devise.convertir1(val);
              System.out.println(devisedata.nom2+" : "+resultat);

              System.out.print(devisedata.nom2+" : ");
              System.out.flush();
              val = Double.parseDouble(in.readLine());
              resultat = devise.convertir2(val);
              System.out.println(devisedata.nom1+" : "+resultat);

            }
          }
      catch( Exception ex )
              {
                  ex.printStackTrace();
              }
    }
  }
```

7. Interface Definition Language (IDL)

7.1. Le contrat IDL

Le contrat IDL permet la coopération entre les fournisseurs et les utilisateurs de services.
Il sépare l'interface et l'implantation des objets
Il masque les divers problèmes liés à l'interopérabilité l'hétérogénéité et la localisation des objets

Un contrat IDL spécifie les types manipulés par les applications réparties,

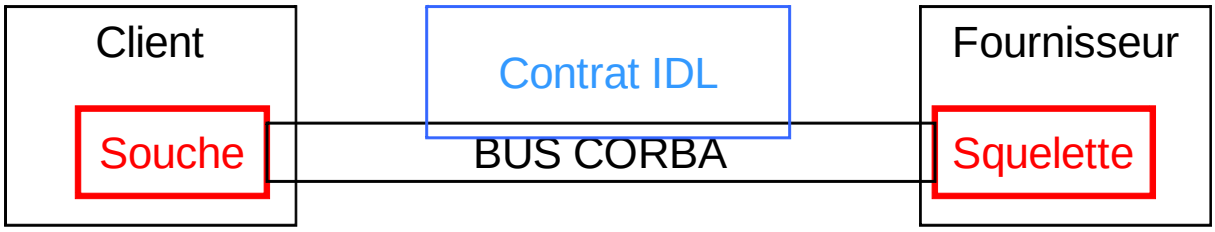
- c'est-à-dire les types d'objets (ou interfaces IDL) et
- les types de données échangés entre les objets.

Les contrats IDL sont projetés en souches IDL (ou interface d'invocations statiques SII) dans l'environnement de programmation du client et en squelettes IDL (ou interface de squelettes statiques SSI) dans l'environnement de programmation du fournisseur.

Le client invoque localement les souches pour accéder aux objets.

Les souches IDL construisent des requêtes, qui vont être transportées par le bus, puis délivrées par celui-ci aux squelettes IDL qui les délégueront aux objets.

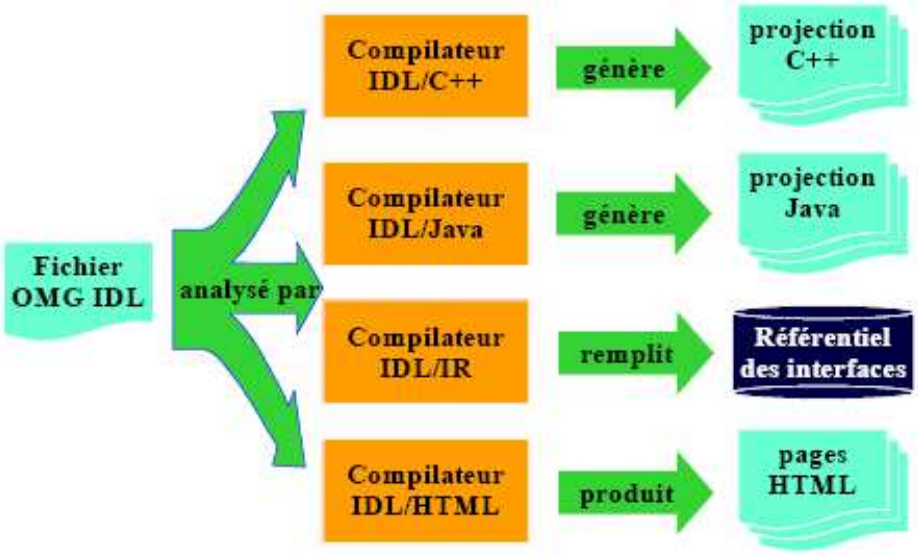
Ainsi le langage OMG-IDL est la clé de voûte du bus d'objets répartis CORBA.



7.2. La compilation IDL

Projections vers les langages informatiques

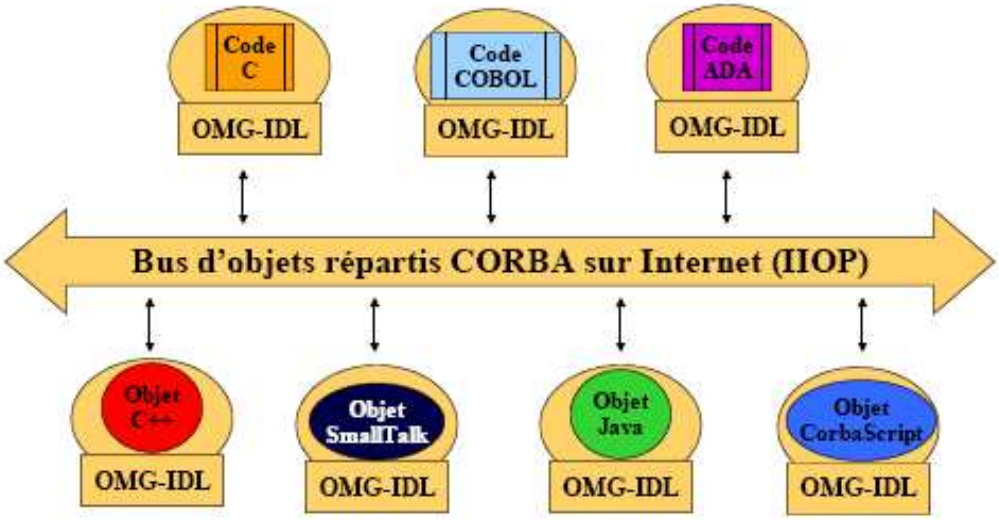
- en standard : C, Smalltalk, C++, Ada, COBOL, Java
- autres : Eiffel, Common Lisp, JavaScript, Python, Tcl, Perl



7.3. Objectif de la projection

Interopérabilité entre les langages via le bus CORBA:

- Définition des règles de programmation;
- Portabilité du code des applications CORBA;
- Indépendance par rapport :
  - Implémentation du bus CORBA utilisé;
  - Machines, systèmes d'exploitation et compilateurs.



7.4. La projection vers un langage de programmation

Une projection est la traduction d'une spécification OMG-IDL dans un langage d'implantation. Pour permettre la portabilité des applications d'un bus vers un autre, les règles de projection sont normalisées et fixent précisément la traduction de chaque construction IDL en une ou des constructions du langage cible et les règles d'utilisation correcte de ces traductions. Actuellement, ces règles existent pour les langages C, C++, SmallTalk, Ada, Java et Cobol orienté objet. Nous ne détaillons pas ici ces règles.

La projection est réalisée par un pré-compilateur IDL dépendant du langage cible et de l'implantation du bus CORBA cible. Ainsi, chaque produit CORBA fournit un pré-compilateur IDL pour chacun des langages supportés.

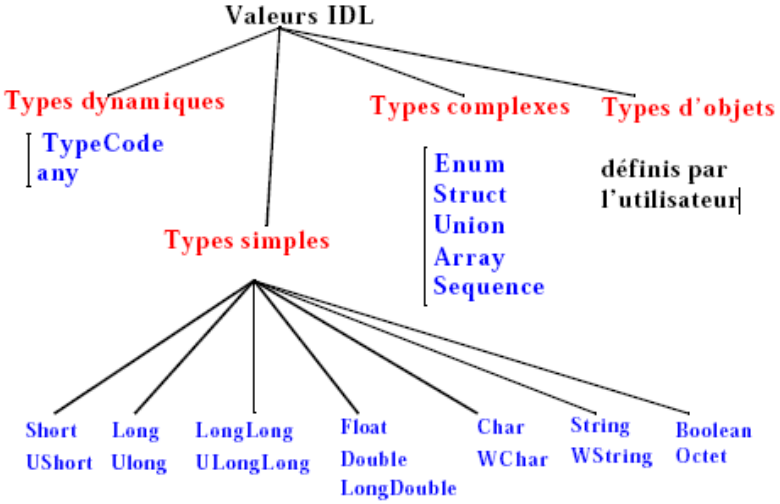
Le code des applications est alors portable d'un bus à un autre car les souches/squelettes générés s'utilisent toujours de la même manière quel que soit le produit CORBA. Par contre, le code des souches et des squelettes IDL n'est pas forcément portable car il dépend de l'implantation du bus pour lequel ils ont été générés.

7.5. Le langage IDL

Langage de description orienté "données"

Les éléments du langage est une mise en commun des types de données des langages structurés

Les éléments sont "projetés" dans le langage de programmation destinataire (C++, Java, …)



Les **types de données de base** sont ceux couramment rencontrés *void*, *short*, *unsigned, short*, *long*, *unsigned long*, *long long* (64 bits), *unsigned long long*, *float*, *double*, *long double* (128 bits), *boolean*, *octet*, *char*, *string*, *wchar* et *wstring* (format de caractères international) et *fixed* pour les nombres à précision fixe. Le format binaire de ces types est défini par la norme afin de régler les problèmes d'échanges de données entre environnements hétérogènes

Les **types de méta-données** (*TypeCode* et *any*) sont une composante spécifique à l'OMG-IDL et nouvelle par rapport à d'autres langages IDL. Le type *TypeCode* permet de stocker la description de n'importe quel type IDL. Le type *any* permet de stocker une valeur IDL de n'importe quel type en conservant son *TypeCode*. Ces méta-types permettent de spécifier des contrats IDL génériques indépendants des types de données manipulés, e.g. une pile d'*any* stocke n'importe quelle valeur

Une **constante** se définit par un type simple, un nom et une valeur évaluable à la compilation (e.g. const double PI = 3.1415 ;).

Un **alias** (ou *typedef*) permet de créer de nouveaux types en renommant des types déjà définis.

Une **énumération** (ou *enum*) définit un type discret via un ensemble d'identificateurs.

Une **structure** définit une fiche regroupant des champs. Cette construction est fortement employée car elle permet de transférer des structures de données composées entre objets CORBA.

Une **union** juxtapose un ensemble de champs, le choix étant arbitré par un discriminant de type simple (entiers, caractère, booléen ou énumérations). Toutefois, cette construction est très rarement utilisée et on préférera plutôt utiliser le type *any.*

Un **tableau** sert à transmettre un ensemble de taille fixe de données homogènes. Mais cette construction est rarement utilisée car on lui préfère la suivante.

Une **séquence** permet de transférer un ensemble de données homogènes dont la taille sera fixée à l'exécution et non à la définition comme pour un tableau.

Une **exception** spécifie une structure de données permettant à une opération de signaler les cas d'erreurs ou de problèmes exceptionnels pouvant survenir lors de son invocation.

Une **interface** décrit les opérations fournies par un type d'objets CORBA

Une **opération** se définit par une signature qui comprend le type du résultat, le nom de l'opération, la liste des paramètres et la liste des exceptions éventuellement déclenchées lors de l'invocation.

Un paramètre se caractérise par un mode de passage, un type et un nom formel. Les modes de passages autorisés sont ***in*, *out* et *inout***

Le résultat et les paramètres peuvent être de n'importe quel type exprimable en IDL. Par défaut, l'invocation d'une opération **est synchrone.**

- Pour une communication asynchone, on utilise les services d'évènement, de notification ou *oneway* en préfixe des opérations

Par défaut toutes les opérations sont synchrones.

Un **attribut** est une manière raccourcie d'exprimer une paire d'opérations pour consulter et modifier une propriété d'un objet CORBA. Il se caractérise par un type et un nom.

- De plus, on peut spécifier si l'attribut est en lecture seule (*readonly*) ou consultation/modification (mode par défaut).
- Il faut tout de même noter que le terme IDL attribut est trompeur, l'implantation d'un attribut IDL peut être faite par un traitement quelconque.

Opérations asynchrones (oneway) :

- Communication par messages;
- Pas de résultat, ni d'argument ***out*** ou ***inout***, ni d'***exception*** utilisateur;
- Pas de garantie de livraison définies par l'OMG (seul un *best effort*).
- Exemple :
  - **oneway** void envoyer_un_message(in long d);

7.6. Les limites de l'IDL

Les types de données de base sont limités en nombre et l'utilisateur de CORBA ne peut pas en ajouter facilement. Cependant, l'OMG ne s'interdit pas d'introduire de nouveaux types au fur et à mesure des besoins, e.g. les entiers 64 bits, les réels 128 bits, les caractères internationaux et les nombres à précision fixe n'étaient pas présents dans les premières versions de la norme CORBA.

L'impossibilité de spécifier des types intervalles

L'impossibilité de sous-typer/d'étendre la définition d'une structure ou d'une union

L'interdiction de conflits de noms à l'intérieur d'un module ou d'une interface impliquant l'interdiction de surcharger des opérations, c'est-à-dire définir deux opérations ayant le même nom mais des signatures différentes.

Le passage par valeur de structures de données et non d'objets. Il serait préférable comme avec Java RMI de pouvoir passer l'objet graphe par valeur. L'OMG travaille actuellement sur une extension de l'OMG-IDL pour exprimer le passage d'objets par valeur, cela sera disponible dans CORBA 3.0

7.7. La projection d'une interface IDL

Compiler une IDL dans laquelle est définie une interface, de nom *InterfaceObjet* génère 7 fichiers java :

- _InterfaceObjet**Stub**.java
  - l'amorce utilisée par le client
- InterfaceObjet.java
  - l'interface utilisée par le client pour invoquer les méthodes distantes
- InterfaceObjet**Helper**.java
  - classe qui instancie l'amorce à travers l'implémentation de la méthode ***narrow***
- InterfaceObjet**Holder**.java
  - classe qui contient l'interface (passage par objet)
- Interface**Operations**.java
  - interface qui décrits les méthodes de l'interface
- Interface**POA**.java
  - la classe abstraite qui hérite de *org.omg.PortableServer.Servant* et "implements" l'interface *InterfaceOperations* sans les définir, dont le servant
- Interface**POATie**.java
  - classe permettant d'instancier un servant qui ne peut pas hériter de la classe POA

Ne pas faire de return null dans une méthode d'une interface CORBA. Cela n'est pas supporté dans tous les langages.

7.8. La projection d'une structure IDL

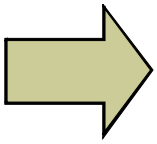
Voir sur le site **http://jacques.laforgue.free.fr/SITE_NSY102/Site/Exemples.phtml**

ExempleCh07_01_cas7_projections_autres

Compiler une IDL dans laquelle est définie une structure, de nomIDL dans laquelle est définie une structure, de nom *StructData* génère 3 fichiers java:

- StructData.java
  - classe de définition de la donnée (attributs publics, constructeurs avec en param les attributs
- StructDataHelper.java
  - classe de récupération d'une interface (méthode narrow)
- StructDataHolder.java
  - la classe "container" de la classe StructDataHelper

```
struct DeviseData {
    string   nom;
    double   taux;
    SensTaux sens;
};
```

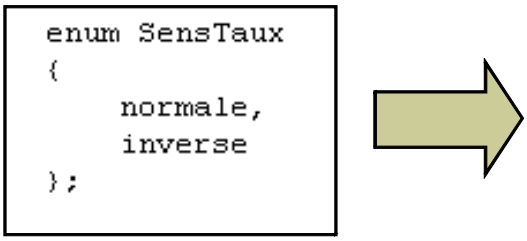


```
package devise;

/**
 *  Generated from IDL definition of struct "DeviseData"
 *  @author JacORB IDL compiler
 */

public final class DeviseData
  implements org.omg.CORBA.portable.IDLEntity
{
  public DeviseData(){}
  public java.lang.String nom = "";
  public double taux;
  public devise.SensTaux sens;
  public DeviseData(java.lang.String nom, double taux, devise.SensTaux sens)
  {
    this.nom = nom;
    this.taux = taux;
    this.sens = sens;
  }
}
```

7.9. La projection d'une énumération

Voir sur le site **http://jacques.laforgue.free.fr/SITE_NSY102/Site/Exemples.phtml**

ExempleCh07_01_cas7_projections_autres




```
public final class SensTaux
        implements org.omg.CORBA.portable.IDLEntity
{
        private int value = -1;
        public static final int _normale = 0;
        public static final SensTaux normale = new SensTaux(_normale);
        public static final int _inverse = 1;
        public static final SensTaux inverse = new SensTaux(_inverse);
        public int value()
        {
                return value;
        }
        public static SensTaux from_int(int value)
        {
                switch (value) {
                        case _normale: return normale;
                        case _inverse: return inverse;
                        default: throw new org.omg.CORBA.BAD_PARAM();
                }
        }
        protected SensTaux(int i)
```

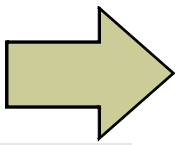
```
                {
                        value = i;
                }
                java.lang.Object readResolve()
                throws java.io.ObjectStreamException
                {
                        return from_int(value());
                }
    }
```

7.10. La projection d'une séquence

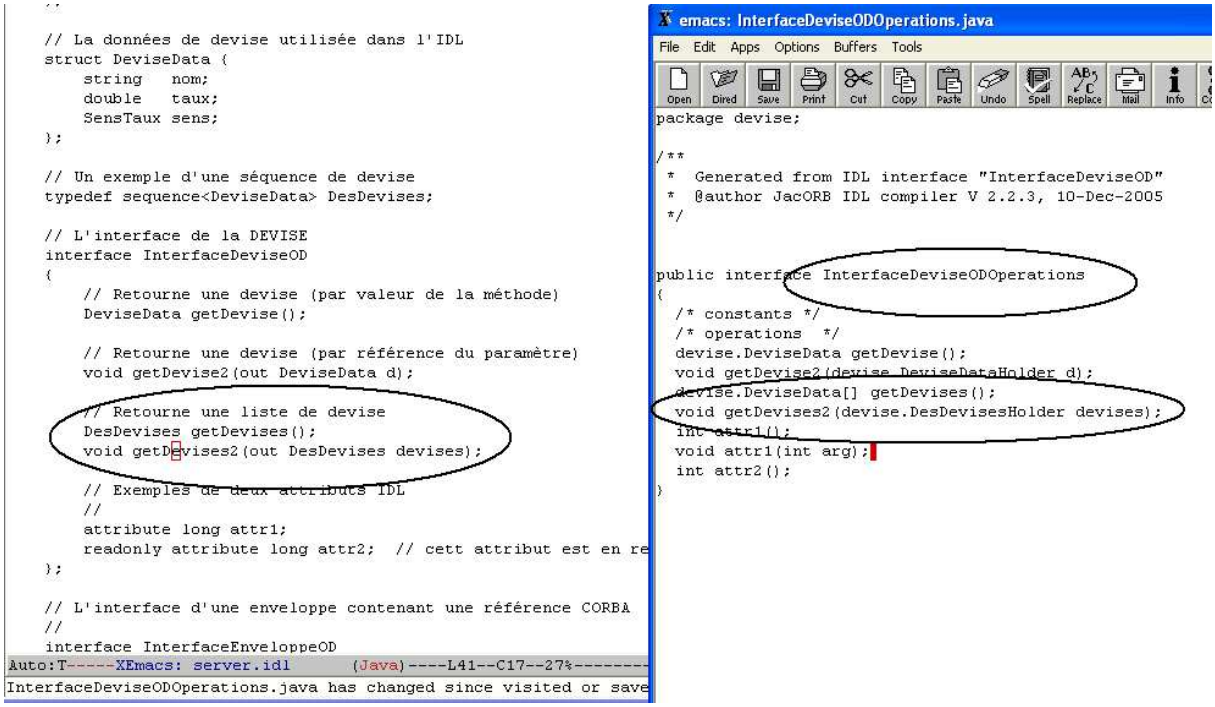
Voir sur le site **http://jacques.laforgue.free.fr/SITE_NSY102/Site/Exemples.phtml**

ExempleCh07_01_cas7_projections_autres

```
typedef sequence<DeviseData> DesDevises;
```



```
public final class DesDevisesHolder
implements org.omg.CORBA.portable.Streamable
{
public devise.DeviseData[] value;

public DesDevisesHolder ()
{
}
public DesDevisesHolder (final devise.DeviseData[] initial)
{
 value = initial;
}
public org.omg.CORBA.TypeCode _type ()
{
 return DesDevisesHelper.type ();
}
public void _read (final org.omg.CORBA.portable.InputStream in)
{
 value = DesDevisesHelper.read (in);
}
public void _write (final org.omg.CORBA.portable.OutputStream out)
{
 DesDevisesHelper.write (out,value);
}
}
```



7.11. Les attributs

Voir sur le site **http://jacques.laforgue.free.fr/SITE_NSY102/Site/Exemples.phtml**

ExempleCh07_01_cas7_projections_autres

La déclaration d' "attributs" correspond en la génération des getteurs et setteurs des attributs de l'objet distribué.

```
module hello {
  interface InterfaceGoodDay {
     wstring hello_world( in wstring msg );

        attribute long compteur;
  };
};
```

```
public interface InterfaceGoodDayOperations
{
/* constants */
/* operations  */
java.lang.String hello_world(java.lang.String msg);
int compteur();
void compteur(int arg);
}
```

```
public class GoodDayImpl extends InterfaceGoodDayPOA
{
    private String location;
    private int compteur ;

    public GoodDayImpl( String location )
```

```
    {
        this.location = location;
        this.compteur = 0;
    }

    public String hello_world(String wide_msg)
    {
        compteur++;
        System.out.println("The message is  : " + wide_msg );
        return "Hello : Mr " + wide_msg+" "+location+" "+compteur;
    }

    public int compteur()
    {
        return compteur;
    }

    public void compteur(int c)
    {
        compteur = c;
    }
}
```

7.12. Any et TypeCode

Le type **Any** est en CORBA: **org.omg.CORBA.Any**

il correspond à un meta-type en CORBA.

Il peut contenir n'importe quelle valeur IDL

- any x; ➔ org.omg.CORBA.Any x;

Pratique pour définir des méthodes "génériques"
Any contient :

- une valeur (int, char, long, …, object)
- le type de la valeur : TypeCode

Il existe de nombreuses méthodes pour insérer et extraire la valeur dans la donnée

8. La référence d'un objet CORBA

La référence d'un objet CORBA identifie de manière unique un servant du bus CORBA.

Elle a été créée par un POA.

Si le servant est déplacé, la référence doit être recréée.

A tout moment, on peut obtenir l'interface de l'objet distribué à partir de la référence (**narrow**) et donc utiliser l'objet distribué.

On peut stocker ces références, s'échanger ces références en paramètre des méthodes distantes d'autres servant.
Cela permet de créer et gérer des **Factory**.

Dans un IDL, cette référence est de type : **Object**
Dans le code java, cette référence est de type : **org.omg.CORBA.Object**

9. Le mécanisme Tie

La classe d'implémentation d'un objet distribué doit hériter de la classe *Interface*POA. Cela n'est pas toujours possible

Pour cela, on réalise une conception par délégation :

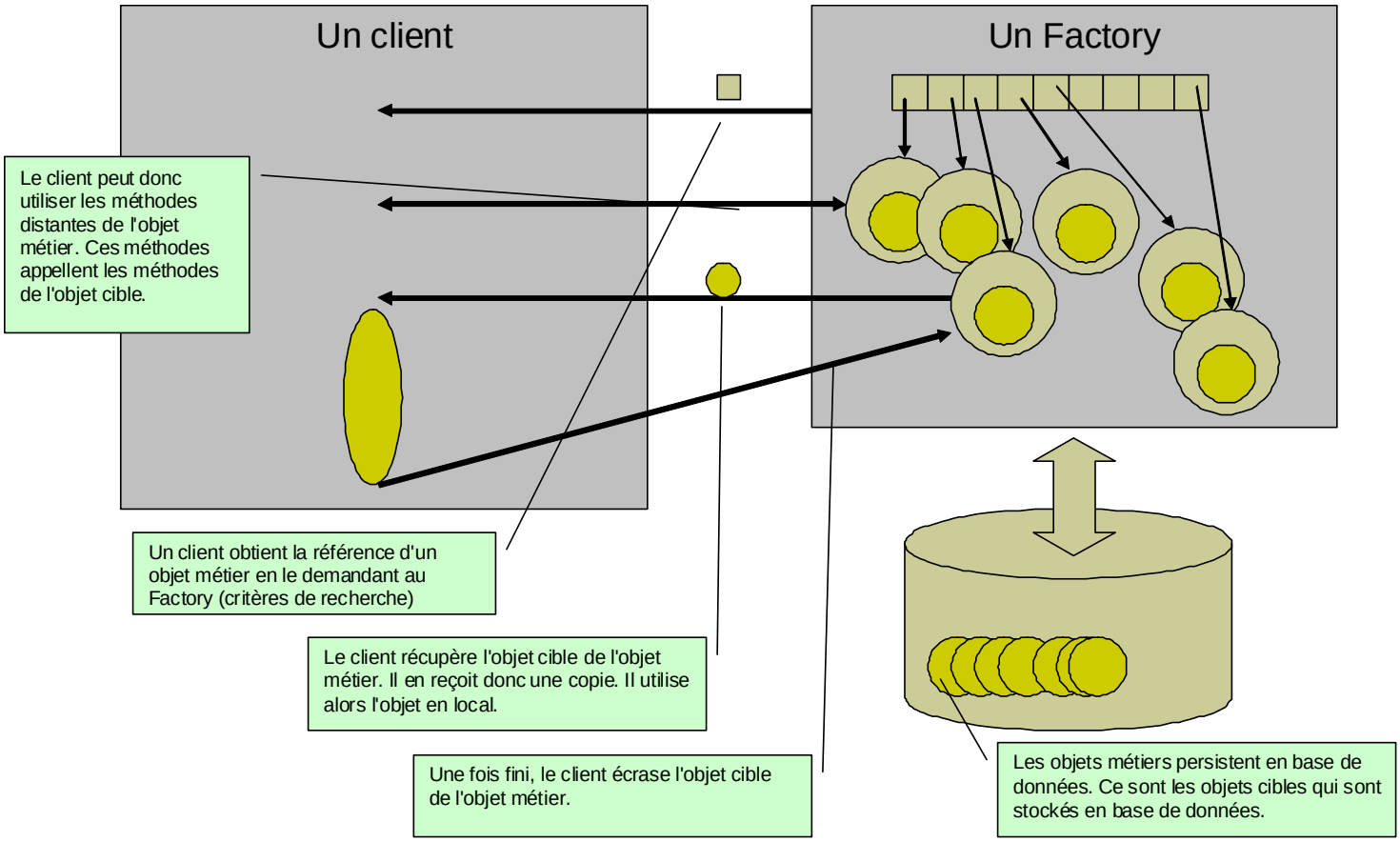
- la classe d'implémentation de l'objet distribué doit *implements* l'interface *Interface*Operations
- instanciation de *Interface*POATie en passant en paramètre l'objet distribué

La classe de délégation *Interface*POATie est générée par l'IDL.

Voir sur le site **http://jacques.laforgue.free.fr/SITE_NSY102/Site/Exemples.phtml**

ExempleCh07_01_cas4_devise_avec_Tie

10. Le factory



11. Le canal d'évènement

Voir sur le site http://jacques.laforgue.free.fr les exemples
ExempleCh07_01_cas5_event_push
ExempleCh07_01_cas6_event_push_ref_corba

Voir aussi Exemple10.pdf joint à l'exemple.

Nous avons vu dans chapitre 7 la notion de canal d'évènement.

Nous voyons ici la mise en pratique d'un tel canal d'évènement en CORBA.