Objets répartis - Partie 2
CORBA

© Olivier Caron

Polytech'Lille

Merci à

- √ Jean-Marc Geib, Christophe Gransart, Philippe Merle
  "Corba, des concepts à la pratique"
  InterEditions
- √ CorbaScript (Christophe Gransart, Philippe Merle)
  `http://corbaweb.lifl.fr`
- √ JIDLScript (Jean-Francois Roos)
  `http://www.lifl.fr/~roos`

Genèse de CORBA :Objectifs

- √ assurer l'interopérabilité
  - Des OS différents (ex : clients windows et serveur unix)

Genèse de CORBA :Objectifs

√ assurer l'interopérabilité

- Des OS différents (ex : clients windows et serveur unix)
- Des langages différents (COBOL, Java, Smalltalk,. . .)

Genèse de CORBA :Objectifs

√ assurer l'interopérabilité

- Des OS différents (ex : clients windows et serveur unix)
- Des langages différents (COBOL, Java, Smalltalk,. . .)
- Faciliter la programmation réseau

Genèse de CORBA :Objectifs

- √ assurer l'interopérabilité
  - Des OS différents (ex : clients windows et serveur unix)
  - Des langages différents (COBOL, Java, Smalltalk,. . .)
  - Faciliter la programmation réseau
- √ Intégration d'applications patrimoines (legacy applications)

Genèse de CORBA :Objectifs

- √ assurer l'interopérabilité
  - Des OS différents (ex : clients windows et serveur unix)
  - Des langages différents (COBOL, Java, Smalltalk,. . .)
  - Faciliter la programmation réseau
- √ Intégration d'applications patrimoines (legacy applications)
- √ Bénéficier de l'approche objet (description, progr.,. . .)

Une des applications : les Systèmes d'information fédérés

√ Caractéristiques : autonomie des BD, sécurité, règles de partages, SGBD différents, modèle de représentation différents,. . .

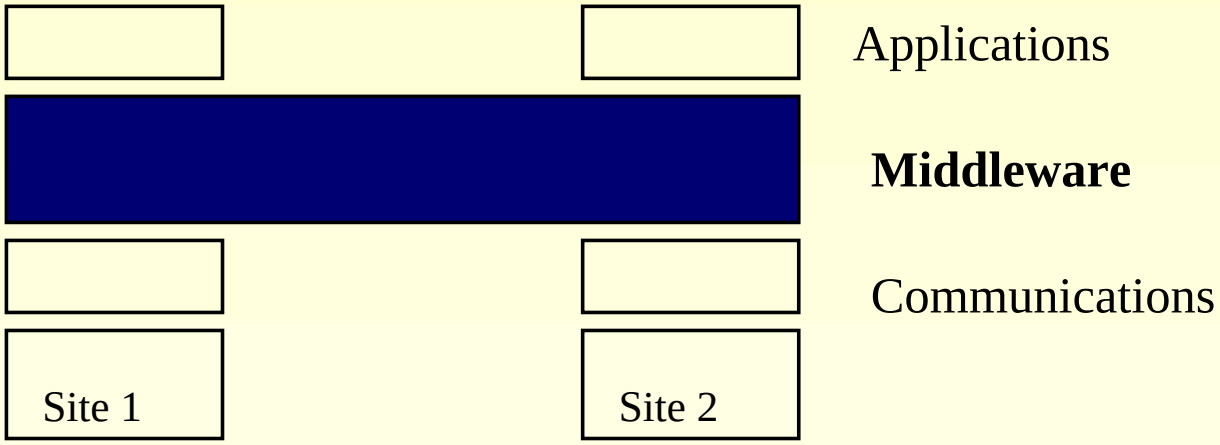
Une des applications : les Systèmes d'information fédérés

- √ Caractéristiques : autonomie des BD, sécurité, règles de partages, SGBD différents, modèle de représentation différents,. . .
- √ Au sein d'une entreprise.
  Ex : SI conception, SI fabrication, SI commercial,. . .

Une des applications : les Systèmes d'information fédérés

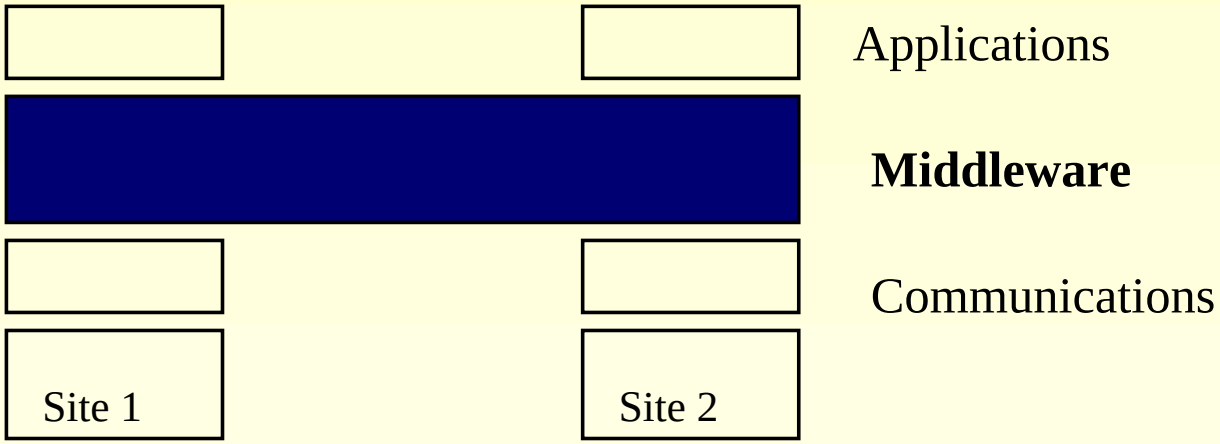
- √ Caractéristiques : autonomie des BD, sécurité, règles de partages, SGBD différents, modèle de représentation différents,. . .
- √ Au sein d'une entreprise.
  Ex : SI conception, SI fabrication, SI commercial,. . .
- √ Entre plusieurs entreprises
  Sécurité, protocoles réseaux

Objectif : Normaliser le middleware



√ On parle également d'intergiciels.

Objectif : Normaliser le middleware



√ On parle également d'intergiciels.

√ Le programmeur dispose d'APIs pour accéder au services distants.

Corba en quelques lignes (1/3)

√ C'est une norme (aucune implémentation)

Corba en quelques lignes (1/3)

- √ C'est une norme (aucune implémentation)
- √ Fournie par l'Object Management Group

  `http://www.omg.org`

Corba en quelques lignes (1/3)

- √ C'est une norme (aucune implémentation)
- √ Fournie par l'Object Management Group
  `http://www.omg.org`
- √ Consortium réunissant plus de 800 constructeurs
  Sun, Oracle, IBM, Bull, Microsoft, LIFL,. . .

Corba en quelques lignes (2/3)

- √ Définition d'un bus logiciel ORB (à l'instar d'un bus matériel où l'on vient enficher des nouveaux composants)

Corba en quelques lignes (2/3)

- √ Définition d'un bus logiciel ORB (à l'instar d'un bus matériel où l'on vient enficher des nouveaux composants)
- √ Définition d'un langage de spécification objet IDL

Corba en quelques lignes (2/3)

- √ Définition d'un bus logiciel ORB (à l'instar d'un bus matériel où l'on vient enficher des nouveaux composants)
- √ Définition d'un langage de spécification objet IDL
  - ▶ génération automatique de souches réseaux

Corba en quelques lignes (2/3)

- √ Définition d'un bus logiciel ORB (à l'instar d'un bus matériel où l'on vient enficher des nouveaux composants)
- √ Définition d'un langage de spécification objet IDL
  - ▶ génération automatique de souches réseaux
  - ▶ mécanismes d'introspection.

Corba en quelques lignes (2/3)

- √ Définition d'un bus logiciel ORB (à l'instar d'un bus matériel où l'on vient enficher des nouveaux composants)
- √ Définition d'un langage de spécification objet IDL
  - ▶ génération automatique de souches réseaux
  - ▶ mécanismes d'introspection.
- √ Mapping vers de multiples langages

Corba en quelques lignes (3/3)

$\surd$ Définition de nombreux services : transaction, sécurité, persistence,. . .

Corba en quelques lignes (3/3)

- √ Définition de nombreux services : transaction, sécurité, persistence,. . .
- √ De nombreuses implémentations existent
  MICO, orbix, orbacus (`http://www.ooc.com`)

Corba en quelques lignes (3/3)

- √ Définition de nombreux services : transaction, sécurité, persistence,. . .
- √ De nombreuses implémentations existent
  MICO, orbix, orbacus (`http://www.ooc.com`)
- √ Des produits existent (l'environnement GNOME).

Fonctionnement de l'OMG

√ Basé sur les RFP (Request For Proposal)

Fonctionnement de l'OMG

√ Basé sur les RFP (Request For Proposal)

1. Identification d'un problème

Fonctionnement de l'OMG

√ Basé sur les RFP (Request For Proposal)

1. Identification d'un problème
2. Propositions de solutions

Fonctionnement de l'OMG

√ Basé sur les RFP (Request For Proposal)

1. Identification d'un problème
2. Propositions de solutions
3. Vote jusqu'à consensus. . .

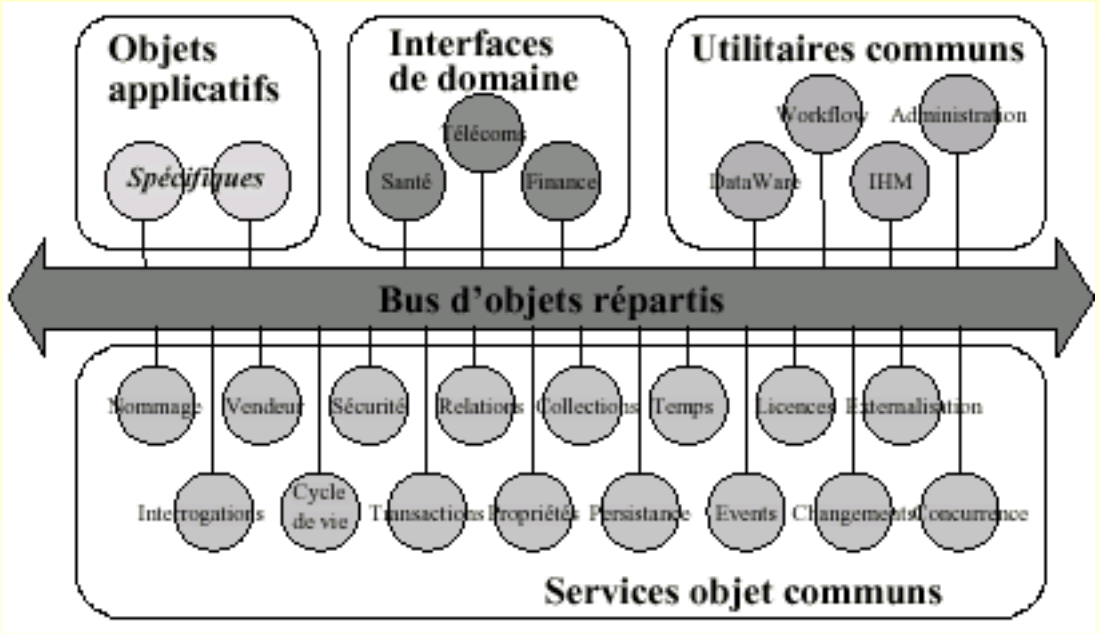
Fonctionnement de l'OMG

√ Basé sur les RFP (Request For Proposal)

1. Identification d'un problème
2. Propositions de solutions
3. Vote jusqu'à consensus. . .

√ Processus long !

Object Management Architecture



OMA : les services communs (1/2)

√ Services de recherche d'objets

- Service Nommage (Naming)
  - pour retrouver un objet par un nom
  - structuration arborescente (répertoire)

OMA : les services communs (1/2)

√ Services de recherche d'objets

- Service Nommage (Naming)
  - pour retrouver un objet par un nom
  - structuration arborescente (répertoire)
- Service Vendeur (Trader)
  - pour retrouver un objet par des caractéristiques
    → une caractéristique peut être différent d'une méthode ou attribut de l'objet serveur implémenté

OMA : les services communs (2/2)

- √ Services de communications asynchrones
  - Evènements, notification

OMA : les services communs (2/2)

- √ Services de communications asynchrones
  - Evènements, notification
- √ Services de sûreté de fonctionnement
  - Sécurité, transactions, concurrence

OMA : les services communs (2/2)

- √ Services de communications asynchrones
  - Evènements, notification
- √ Services de sûreté de fonctionnement
  - Sécurité, transactions, concurrence
- √ Services concernant la vie des objets
  - Cycle de vie, relations, persistance, requête, collections, "versioning", time, license

OMA : les objets métiers

√ Spécifications officielles de l'OMG pour les canevas verticaux :

- Commerce Electronique
- Santé
- Télécommuncations
- Transports
- Finances/Assurances
- . . .

CORBA : premiers constats (1/2)

√ Les plus :

- Puissance de l'architecture, des possibilités (langages, OS, LAN, WAN (IIOP))
- Pas d'égal :
  - DCOM : mono OS !
  - Java RMI : mono langage

CORBA : premiers constats (1/2)

√ Les plus :

- Puissance de l'architecture, des possibilités (langages, OS, LAN, WAN (IIOP))
- Pas d'égal :
  - DCOM : mono OS !
  - Java RMI : mono langage
- Des passerelles existent vers les deux autres approches

  Ex : des composants DCOM communiquent avec des composants CORBA

CORBA : premiers constats (2/2)

√ Les moins :

- Tout n'est pas défini ! (API)
  Ex : les objets métiers (consensus :-))

CORBA : premiers constats (2/2)

√ Les moins :

- Tout n'est pas défini ! (API)
  Ex : les objets métiers (consensus :-))
- Tout n'est pas implémenté
  Ex : modèle de composants (CORBA 3.0)

CORBA : premiers constats (2/2)

√ Les moins :

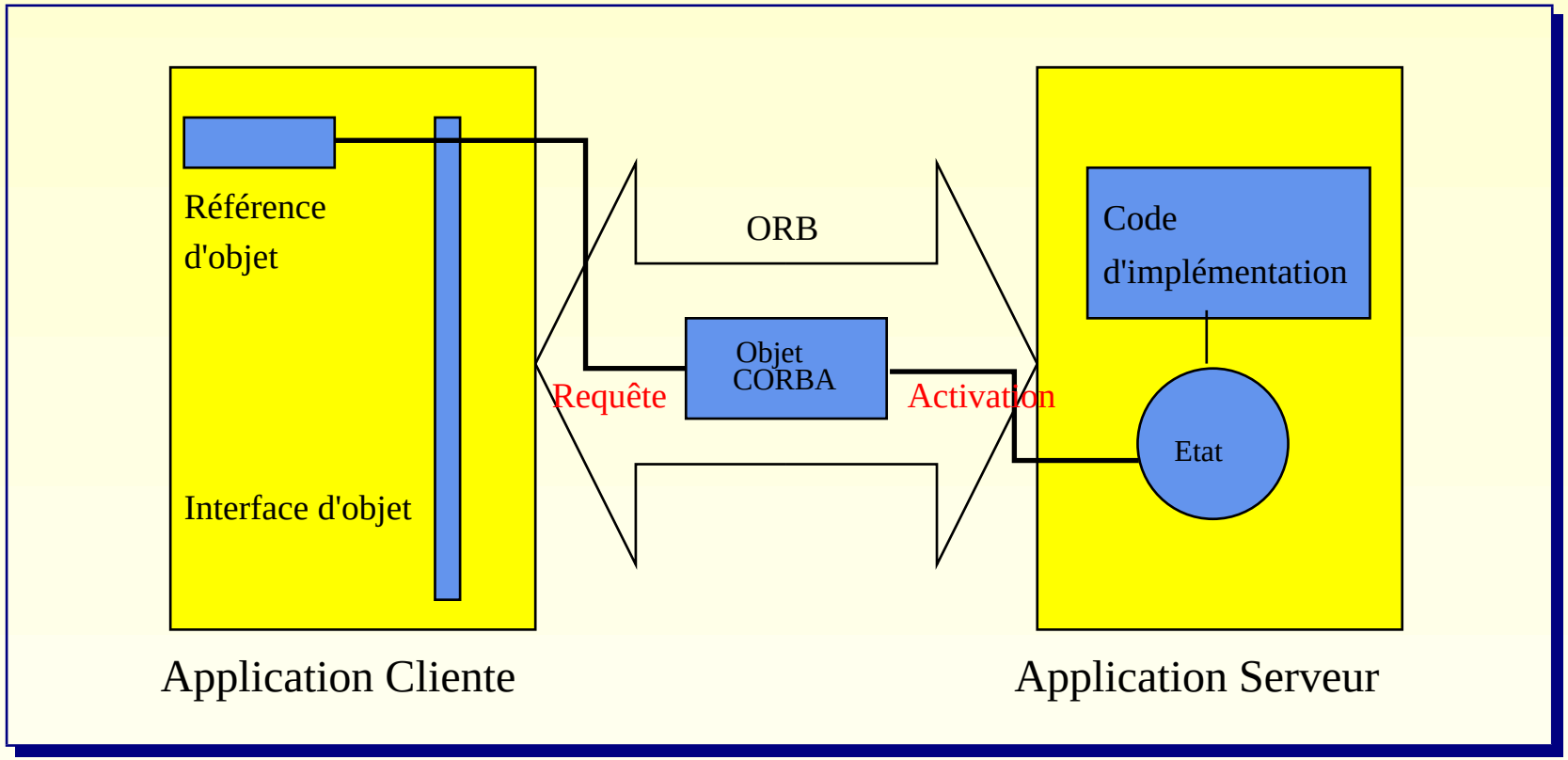
- Tout n'est pas défini ! (API)
  Ex : les objets métiers (consensus :-))
- Tout n'est pas implémenté
  Ex : modèle de composants (CORBA 3.0)
- Assimiler les APIs !

CORBA : premiers constats (2/2)

√ Les moins :

- Tout n'est pas défini ! (API)
  Ex : les objets métiers (consensus :-))
- Tout n'est pas implémenté
  Ex : modèle de composants (CORBA 3.0)
- Assimiler les APIs !
- Manque d'outils

le modèle Objet CORBA



Un objet CORBA

- √ Décrit par une interface OMG IDL unique
- √ Implanté par un objet langage
- √ Désigné par des références CORBA
- √ Attention objet CORBA != objet langage :
  - car activation contrôlable
  - l'objet d'implantation peut varier au cours du temps
  - une référence peut désigner un objet non présent en mémoire

Le langage OMG IDL

- √ C'est le langage pivot entre les applications CORBA
- √ Il masque leur hétérogénéité
- √ sert à décrire les interfaces des objets CORBA
- √ Notation partagée par les membres de l'OMG (description de tous les services,. . .)

Illustration avec un exemple (1/2)

```
module Annuaire {
  typedef string Nom ;
  typedef sequence<Nom> DesNoms;
  struct Personne {
    Nom nom ;
    string status ;
    string telephone ;
    string email ;
    string url ;
  } ;
```

Illustration avec un exemple (2/2)

```
interface Repertoire {
  readonly attribute string description ;
  exception ExisteDeja { Nom nom; } ;
  exception Inconnu { Nom nom; } ;
  void ajouterPersonne(in Personne personne)
     raises(ExisteDeja) ;
  void retirerPersonne(in Nom nom) raises (Inconnu) ;
  void modifierPersonne(in Nom nom, in Personne p)
     raises(Inconnu) ;
  Personne obtenirPersonne(in Nom nom)
     raises(Inconnu) ;
  DesNoms listerNoms() ;
};
};
```

OMG IDL : quelques constructions

- √ module
- √ types de base au format binaire normalisé
- √ nouveaux types : typedef, enum, struct, union, array, sequence
- √ interface (avec héritage multiple sans surcharge)
- √ opérations avec ou sans exceptions
- √ paramètre (in, out, inout)
- √ attribut (possibilité de lecture seule)
- √ types de méta-données (TypeCode, any)
- √ objets passés par valeur

OMG IDL n'est pas un langage de programmation

- √ lexicalement proche du C/C++
- √ avec macros et préprocesseur C++
- √ pas de généricité/template C++
- √ pas de surcharge ni de redéfinition des opérations
- √ problèmes de mapping :
  - exceptions en C
  - passage *out* en Java

La configuration ORB de l'E.P.U.L. (1/2)

- √ orb de ORBACUS et CorbaScript
- √ JDK 1.3, 1.4
- √ Configuration des variables d'environnement :

```
export CLASSPATH=.:/usr/local/corba/lib/OB.jar
export CLASSPATH=${CLASSPATH}:/usr/local/corba/lib/OBNaming.jar
export CLASSPATH=${CLASSPATH}:/usr/local/corba/lib/OBUtil.jar

source /usr/local/corba/bin/envi.ORB.sh
```

La configuration ORB de l'E.P.U.L. (1/2)

√ Lancement ORB, serveur de noms, référentiel d'interfaces :

`cssh_install` ...

`cssh_deinstall`

√ Dans `~/CS_for_ORBacus` :

- Fichier `NameService.IOR` : contient l'IOR du serveur de noms
- Fichier `InterfaceRepository` : contient l'IOR du référentiel d'interfaces

CORBA : le mode statique

- √ Projection des descriptions OMG IDL vers les langages d'implantation des clients et serveurs
- √ Implantation par les clients et serveurs des descriptions OMG IDL communes.
- √ Vérification du typage des invocations à la compilation
- √ Les souches générées encapsulent l'utilisation du bus, l'activation et la distribution des composants et l'hétérogénéité de l'architecture

L'application Hello World avec CORBA !

√ Le source IDL `helloWorld.idl`

```
module helloWorld {
  interface Hello {
    string sayHello() ;
  } ;
} ;
```

HelloWorld : compilation IDL (1/2)

√ Lancement du compilateur de Orbacus :

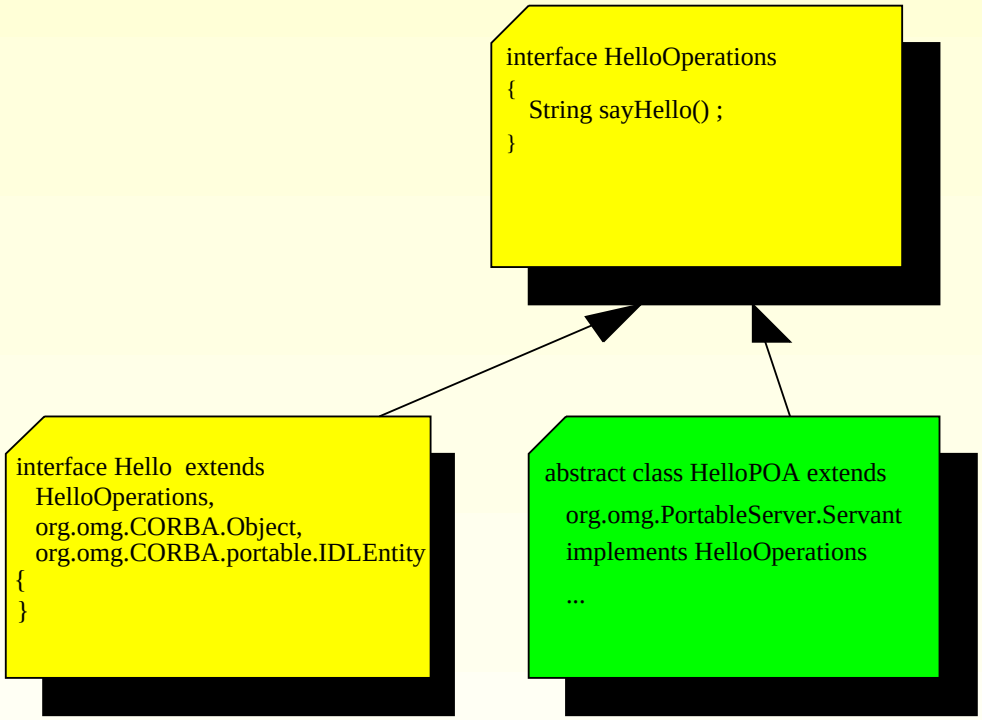
```
jidl helloWorld.idl
```

HelloWorld : compilation IDL (1/2)

- √ Lancement du compilateur de Orbacus :
  `jidl helloWorld.idl`
- √ Génération du module helloWorld :
  - ▶ Contient les interfaces distantes
  - ▶ Contient les classes réseaux (souches)
- √ `Hello.java, HelloHolder.java, HelloPOA.java, HelloHelper.java, HelloOperations.java, _HelloStub.java`

HelloWorld : compilation IDL (2/2)

√ Génération de l'interface distante `Hello` :



HelloWorld : implémentation de l'objet serveur

- √ Implémenter les méthodes générées par le compilateur IDL (`HelloOperations`).

HelloWorld : implémentation de l'objet serveur

- √ Implémenter les méthodes générées par le compilateur IDL (`HelloOperations`).
- √ Les attributs deviennent des méthodes accesseurs

HelloWorld : implémentation de l'objet serveur

- √ Implémenter les méthodes générées par le compilateur IDL (`HelloOperations`).
- √ Les attributs deviennent des méthodes accesseurs
- √ Etendre la classe abstraite `HelloPOA`

```
import helloWorld.* ;

public class HelloImpl extends HelloPOA {
  public String sayHello() {
    return "Bonjour le monde !" ;
  }
}
```

HelloWorld : Connecter l'objet à l'ORB (1/5)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)

HelloWorld : Connecter l'objet à l'ORB (1/5)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA

HelloWorld : Connecter l'objet à l'ORB (1/5)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA
3. Accéder au POA

HelloWorld : Connecter l'objet à l'ORB (1/5)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA
3. Accéder au POA
4. Créer une instance de Hello

HelloWorld : Connecter l'objet à l'ORB (1/5)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA
3. Accéder au POA
4. Créer une instance de Hello
5. Activer l'objet

HelloWorld : Connecter l'objet à l'ORB (1/5)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA
3. Accéder au POA
4. Créer une instance de Hello
5. Activer l'objet
6. Obtenir et stocker son 'IOR'

HelloWorld : Connecter l'objet à l'ORB (1/5)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA
3. Accéder au POA
4. Créer une instance de Hello
5. Activer l'objet
6. Obtenir et stocker son 'IOR'
7. Activation du POA par son manager

HelloWorld : Connecter l'objet à l'ORB (1/5)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA
3. Accéder au POA
4. Créer une instance de Hello
5. Activer l'objet
6. Obtenir et stocker son 'IOR'
7. Activation du POA par son manager
8. Lancer et attendre les requêtes

HelloWorld : Connecter l'objet à l'ORB (2/5)

```
import helloWorld.* ;
import org.omg.CORBA.* ;
import org.omg.PortableServer.* ;
import java.io.* ;

public class HelloWorldServeur {
  public static void main(String args[]) {
      java.util.Properties props = System.getProperties();
      props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
      props.put("org.omg.CORBA.ORBSingletonClass",
                "com.ooc.CORBA.ORBSingleton");
      try {
          // 1: Initialisation de l'ORB
          ORB orb = org.omg.CORBA.ORB.init(args, props);
```

HelloWorld : Connecter l'objet à l'ORB (3/5)

```
// 2: obtenir une référence de l'adaptateur d'objet
org.omg.CORBA.Object objPoa=null ;
org.omg.PortableServer.POA rootPOA = null;
try {
  objPoa = orb.resolve_initial_references("RootPOA") ;
} catch ( org.omg.CORBA.ORBPackage.InvalidName ex ) {}
// 3: obtenir référence objet
  rootPOA = org.omg.PortableServer.POAHelper.narrow(objPoa);
// 4: Instancier l'objet Hello
HelloImpl helloImpl = new HelloImpl();
```

HelloWorld : Connecter l'objet à l'ORB (4/5)

```
try {
  //5: Activer l'objet
  Hello hello = helloImpl._this(orb);
  //6: conserver l'IOR de l'objet
  String ref = orb.object_to_string(hello);
  FileOutputStream fic=new FileOutputStream("Hello.ref") ;
  PrintWriter out = new PrintWriter(fic);
  out.println(ref); out.close();
} catch(IOException exIO) {
  System.err.println("Erreur E/S intervenue...") ;
  System.exit(1) ;
}
```

HelloWorld : Connecter l'objet à l'ORB (5/5)

```
            //7: Activer le POA par son manager
            rootPOA.the_POAManager().activate();
            System.out.println("Le serveur est pret...") ;
            //8: lancer et attendre les requetes
            orb.run();
        }  catch(Exception ex) {
            System.err.println("Une erreur est intervenue") ;
            ex.printStackTrace();
        }
    }
}
```

L'adresse d'objet IOR

√ Version texte d'une référence d'objet (équivalent adr IP)

L'adresse d'objet IOR

- √ Version texte d'une référence d'objet (équivalent adr IP)
- √ Deux méthodes (API) existent :

```
String object_to_string(Object obj)
Object string_to_object(String ior)
```

L'adresse d'objet IOR

- √ Version texte d'une référence d'objet (équivalent adr IP)
- √ Deux méthodes (API) existent :

```
String object_to_string(Object obj)
Object string_to_object(String ior)
```

- √ permet de diffuser un objet (ex : par E-mail)

L'adaptateur d'objet (POA)

- √ défini dans la norme CORBA 2.3
- √ mécanisme trés puissant :
  - garantit que le code serveur est portable vers d'autres ORBs.
  - possibilité d'avoir plusieurs POA dans une même application
  - un POA peut être responsable de plusieurs objets.
  - les objets regroupés par un POA ont les mêmes caractéristiques (activation, persistence,. . .)
  - possibilité d'avoir plusieurs **POA managers** contrôlant les requêtes vers un groupe de POAs
  - managers et POA permettent de gérer des millions d'objets

HelloWorld : l'application cliente (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)

HelloWorld : l'application cliente (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir l'IOR'

HelloWorld : l'application cliente (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir l'IOR'
3. Obtenir l'objet CORBA

HelloWorld : l'application cliente (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir l'IOR'
3. Obtenir l'objet CORBA
4. Obtenir l'objet hello

HelloWorld : l'application cliente (2/3)

```
import helloWorld.* ;
import org.omg.CORBA.* ;
import java.io.* ;
public class HelloWorldClient {
  public static void main(String args[]) {
      org.omg.CORBA.Object obj=null ;

      java.util.Properties props = System.getProperties();
      props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
      props.put("org.omg.CORBA.ORBSingletonClass",
                "com.ooc.CORBA.ORBSingleton");
      try {
          //1: Initialisation de l'ORB
          ORB orb = org.omg.CORBA.ORB.init(args, props);
```

HelloWorld : l'application cliente (3/3)

```
 try {
  String ref=null ;  //2 : obtenir l'IOR
  BufferedReader in=new BufferedReader(new FileReader("Hello.ref"));
  ref=in.readLine() ;
  obj=orb.string_to_object(ref) ; //3 : obtenir la ref distante
 } catch(IOException e) {
  System.err.println("Erreur E/S intervenue") ;
  System.exit(1) ;
 }
 Hello p = HelloHelper.narrow(obj) ; //4: referencer l'objet
 System.out.println(p.sayHello()) ;
} catch(org.omg.CORBA.SystemException ex) {
 System.err.println("une erreur CORBA est intervenue...") ;
}}}
```

HelloWorld : Execution (1/2)

√ Coté Serveur :

```
source ~ocaron/public/config.env

cssh_install

java HelloWorldServeur &
Le serveur est pret...
```

HelloWorld : Execution (2/2)

√ Coté Client (sur une autre machine :-) :

```
source ~ocaron/public/config.env

java HelloWorldClient
Bonjour le monde !
```

HelloWorld : premier constat

√ Programmation : pas si simple !

HelloWorld : premier constat

√ Programmation : pas si simple !
√ IOR : bof !

HelloWorld : premier constat

√ Programmation : pas si simple !

√ IOR : bof !

solution : Utilisez le service de noms

HelloWorld 2 : coté serveur (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)

HelloWorld 2 : coté serveur (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA

HelloWorld 2 : coté serveur (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA
3. Accéder au POA

HelloWorld 2 : coté serveur (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA
3. Accéder au POA
4. Créer une instance de Hello

HelloWorld 2 : coté serveur (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA
3. Accéder au POA
4. Créer une instance de Hello
5. Activer l'objet

HelloWorld 2 : coté serveur (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA
3. Accéder au POA
4. Créer une instance de Hello
5. Activer l'objet
6. Recherche du service de noms
7. construction chemin pour Hello et lier au service

HelloWorld 2 : coté serveur (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA
3. Accéder au POA
4. Créer une instance de Hello
5. Activer l'objet
6. Recherche du service de noms
7. construction chemin pour Hello et lier au service
8. Activation du POA par son manager

HelloWorld 2 : coté serveur (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Obtenir une référence de l'adaptateur d'objets POA
3. Accéder au POA
4. Créer une instance de Hello
5. Activer l'objet
6. Recherche du service de noms
7. construction chemin pour Hello et lier au service
8. Activation du POA par son manager
9. Lancer et attendre les requêtes

HelloWorld 2 : coté serveur (2/3)

```
// recherche du serveur de noms
org.omg.CORBA.Object obj = null;
org.omg.CosNaming.NamingContext naming = null;
try {
  obj = orb.resolve_initial_references("NameService");
  naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
} catch ( org.omg.CORBA.ORBPackage.InvalidName name ) {
  System.out.println("nom de service invalide");
  System.exit(1);
}
if ( naming == null ) {
  System.out.println("Service de noms non trouve");
  System.exit(1);
}
```

HelloWorld 2 : coté serveur (3/3)

```
// 7. construire le chemin pour le sevice Hello et le lier au service
org.omg.CosNaming.NameComponent []name=
   new org.omg.CosNaming.NameComponent[1];
name[0] = new org.omg.CosNaming.NameComponent();
name[0].id = "HelloWorld"; name[0].kind = "Exemple";
try {
  naming.bind(name,hello);
} catch (org.omg.CosNaming.NamingContextPackage.NotFound ex ) {
  System.out.println("Object not found"); System.exit(1);
} catch (org.omg.CosNaming.NamingContextPackage.AlreadyBound ex ){
  System.out.println("Already bound"); System.exit(1);
} catch (org.omg.CosNaming.NamingContextPackage.InvalidName ex ){
  System.out.println("Invalid name"); System.exit(1);
} catch (org.omg.CosNaming.NamingContextPackage.CannotProceed ex ){
  System.out.println("Cannot proceed"); System.exit(1); }
```

HelloWorld 2 : coté client (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)

HelloWorld 2 : coté client (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Recherche du service de noms

HelloWorld 2 : coté client (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Recherche du service de noms
3. Construction du chemin

HelloWorld 2 : coté client (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Recherche du service de noms
3. Construction du chemin
4. Obtenir l'objet CORBA

HelloWorld 2 : coté client (1/3)

1. Initialisation de l'ORB (utilisez les classes de Orbacus plutot que Sun JDK)
2. Recherche du service de noms
3. Construction du chemin
4. Obtenir l'objet CORBA
5. Obtenir l'objet hello
6. Utiliser l'objet

HelloWorld 2 : coté client (2/3)

```
// 2: recherche du service de noms
org.omg.CORBA.Object obj = null;
org.omg.CosNaming.NamingContext naming = null;
try {
  obj = orb.resolve_initial_references("NameService");
  naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
} catch ( org.omg.CORBA.ORBPackage.InvalidName name ) {
  System.out.println("service de noms non trouvé");
  System.exit(1);
}
```

HelloWorld 2 : coté client (3/3)

```
// 3: construction du chemin
org.omg.CosNaming.NameComponent [] name =
  new org.omg.CosNaming.NameComponent[1];
name[0] = new org.omg.CosNaming.NameComponent();
name[0].id = "HelloWorld"; name[0].kind = "Exemple";
// 4 : recherche de référence hello du service de noms
try {
  obj = naming.resolve(name);
} catch (org.omg.CosNaming.NamingContextPackage.NotFound ex ) {
  System.err.println("Objet non trouvé"); System.exit(1);
} catch (org.omg.CosNaming.NamingContextPackage.CannotProceed ex ){
  System.err.println("Cannot proceed"); System.exit(1);
} catch (org.omg.CosNaming.NamingContextPackage.InvalidName ex ){
  System.err.println("nom non correct"); System.exit(1);
}
```

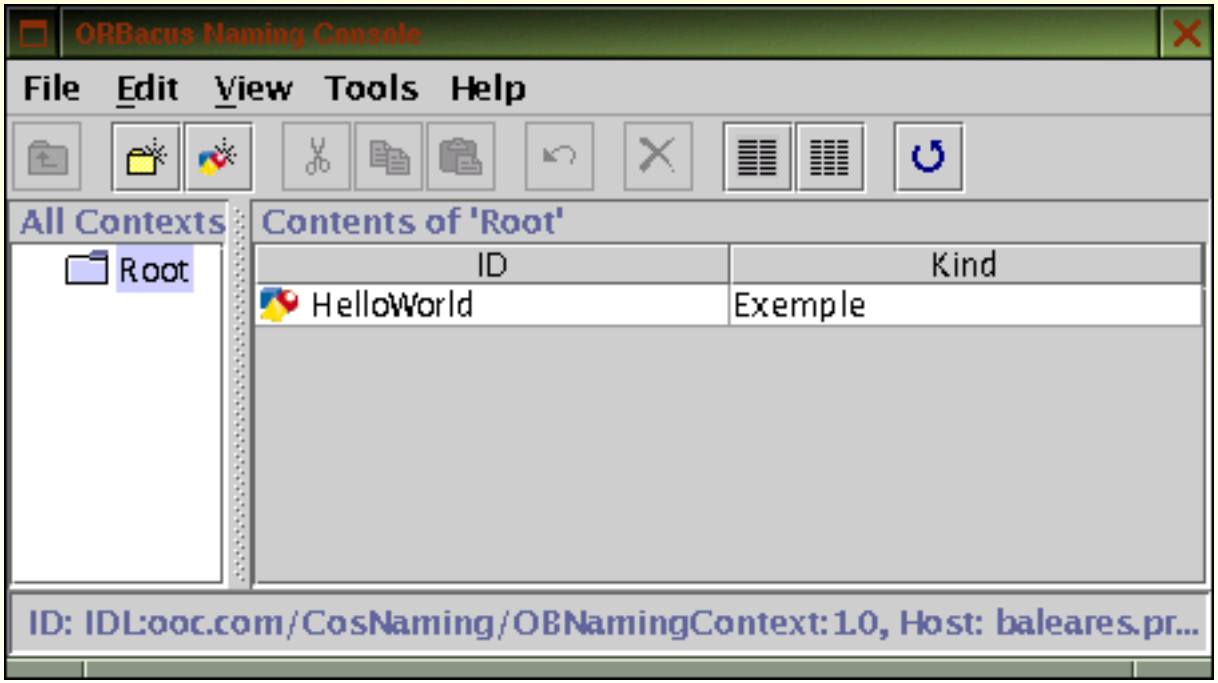
HelloWorld 2 : Execution (1/3)

√ Coté Serveur :

```
source ~ocaron/public/config.env
cssh_install
java HelloWorldServeur
    -ORBnaming `cat ~/CS_for_ORBacus/NameService.IOR` &
Le serveur est pret...
```

HelloWorld 2 : Execution (2/3)

```
java com.ooc.CosNamingConsole.Main
     --file ~/CS_for_ORBacus/NameService.IOR
```



HelloWorld 2 : Execution (3/3)

√ Coté Client (sur une autre machine :-) :

```
source ~ocaron/public/config.env
java HelloWorldClient
     -ORBnaming ‘cat ~/CS_for_ORBacus/NameService.IOR‘
Bonjour le monde !
```

HelloWorld 2 : second constat

√ Service de noms mieux qu'IOR

HelloWorld 2 : second constat

- √ Service de noms mieux qu'IOR
- √ Programmation : toujours pas simple !

HelloWorld 2 : second constat

- √ Service de noms mieux qu'IOR
- √ Programmation : toujours pas simple !

  solution :
  - ▶ Utiliser des langages de scripts

HelloWorld 2 : second constat

√ Service de noms mieux qu'IOR

√ Programmation : toujours pas simple !

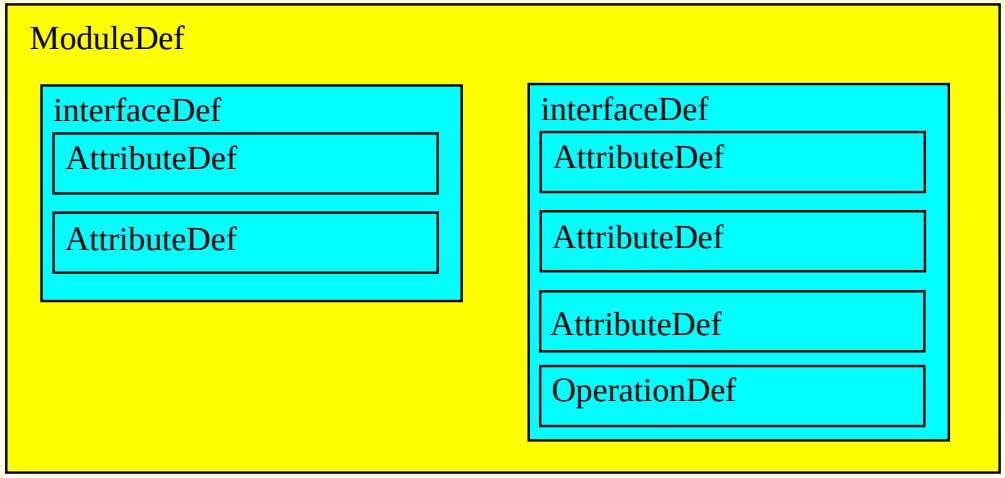
solution :

- Utiliser des langages de scripts
- Utiliser le mécanisme DII

Le mécanisme DII (1/2)
Dynamic Invocation Interface

- √ Un nouvel objet notoire : le référentiel d'interfaces (IR)
- √ Possibilité de consulter le référentiel : introspection
- √ Possibilité de construire une requête

Exemple de structure du référentiel d'interface



Le mécanisme DII (2/2)
Scénario d'utilisation

1. Trouver la référence d'objet

Le mécanisme DII (2/2)
Scénario d'utilisation

1. Trouver la référence d'objet
2. Obtenir l'interface de l'objet

Le mécanisme DII (2/2)
Scénario d'utilisation

1. Trouver la référence d'objet
2. Obtenir l'interface de l'objet
3. Obtenir la description de l'opération à invoquer

Le mécanisme DII (2/2)
Scénario d'utilisation

1. Trouver la référence d'objet
2. Obtenir l'interface de l'objet
3. Obtenir la description de l'opération à invoquer
4. Construire la liste des arguments à transmettre

Le mécanisme DII (2/2)
Scénario d'utilisation

1. Trouver la référence d'objet
2. Obtenir l'interface de l'objet
3. Obtenir la description de l'opération à invoquer
4. Construire la liste des arguments à transmettre
5. Créer la requête

Le mécanisme DII (2/2)
Scénario d'utilisation

1. Trouver la référence d'objet
2. Obtenir l'interface de l'objet
3. Obtenir la description de l'opération à invoquer
4. Construire la liste des arguments à transmettre
5. Créer la requête
6. Invoquer la requête

Le mécanisme DII (2/2)
Scénario d'utilisation

1. Trouver la référence d'objet
2. Obtenir l'interface de l'objet
3. Obtenir la description de l'opération à invoquer
4. Construire la liste des arguments à transmettre
5. Créer la requête
6. Invoquer la requête
7. Traiter les résultats retournés

CorbaScript

√ Un résultat de la thèse de Philippe Merle (Goal-LIFL)

CorbaScript

√ Un résultat de la thèse de Philippe Merle (Goal-LIFL)

√ Première implémentation de la norme IDLScript :-)

CorbaScript

- √ Un résultat de la thèse de Philippe Merle (Goal-LIFL)
- √ Première implémentation de la norme IDLScript :-)
- √ Version Java : JIDLScript

CorbaScript

- √ Un résultat de la thèse de Philippe Merle (Goal-LIFL)
- √ Première implémentation de la norme IDLScript :-)
- √ Version Java : JIDLScript
- √ `http://corbaweb.lifl.fr`
- √ `http://www.lifl.fr/~roos`

CorbaScript en tant que client

```
ir_load /usr/local/corba/CorbaScript-1.3.2/idl/CosNaming.idl
ir_load helloWorld.idl
cssh
CorbaScript 1.3.2 (Apr  7 2000) for ORBacus 4.0b2 for C++
Copyright 1996-99 LIFL, France
>>> NS = CORBA.ORB.resolve_initial_references ("NameService")
>>> object=NS.resolve([["HelloWorld","Exemple"]])
>>> println("le resultat est ",object.sayHello())
le resultat est Bonjour le monde !
>>> ^D
```

CorbaScript : langage complet (1/2)

√ Fichier corbascript, lancement `cssh helloName.cs`

```
# fichier helloName.cs
NS = CORBA.ORB.resolve_initial_references ("NameService")
try {
  object = NS.resolve ([["HelloWorld", "Exemple"]])
} catch (CosNaming.NamingContext.NotFound exc) {
  println ("\nERREUR, serveur ‘HelloWorld’ non lancé")
  return    # fin exécution
}
# accès à l’IOR
theIORString = object._ior
println ("\nIOR vaut : ", theIORString)
object = helloWorld.Hello (theIORString)
println(object.sayHello())
```

CorbaScript : langage complet (2/2)

√ Fichier corbascript, lancement `cssh_server helloServeur.cs`

```
# fichier helloServeur.cs
class HelloImpl {
  proc __HelloImpl__(self) {}
  proc sayHello(self) { return "Bonjour le monde !" }
}

hello=HelloImpl()
println(hello.sayHello())
CORBA.ORB.connect(hello, helloWorld.Hello)
NS=CORBA.ORB.resolve_initial_references("NameService")
NS.rebind([["HelloWorld","Exemple"]],hello._this)
println("\n CTRL-C pour sortir")
CORBA.ORB.run()
```

Mapping Java - héritage, données (1/3)

√ Un exemple :

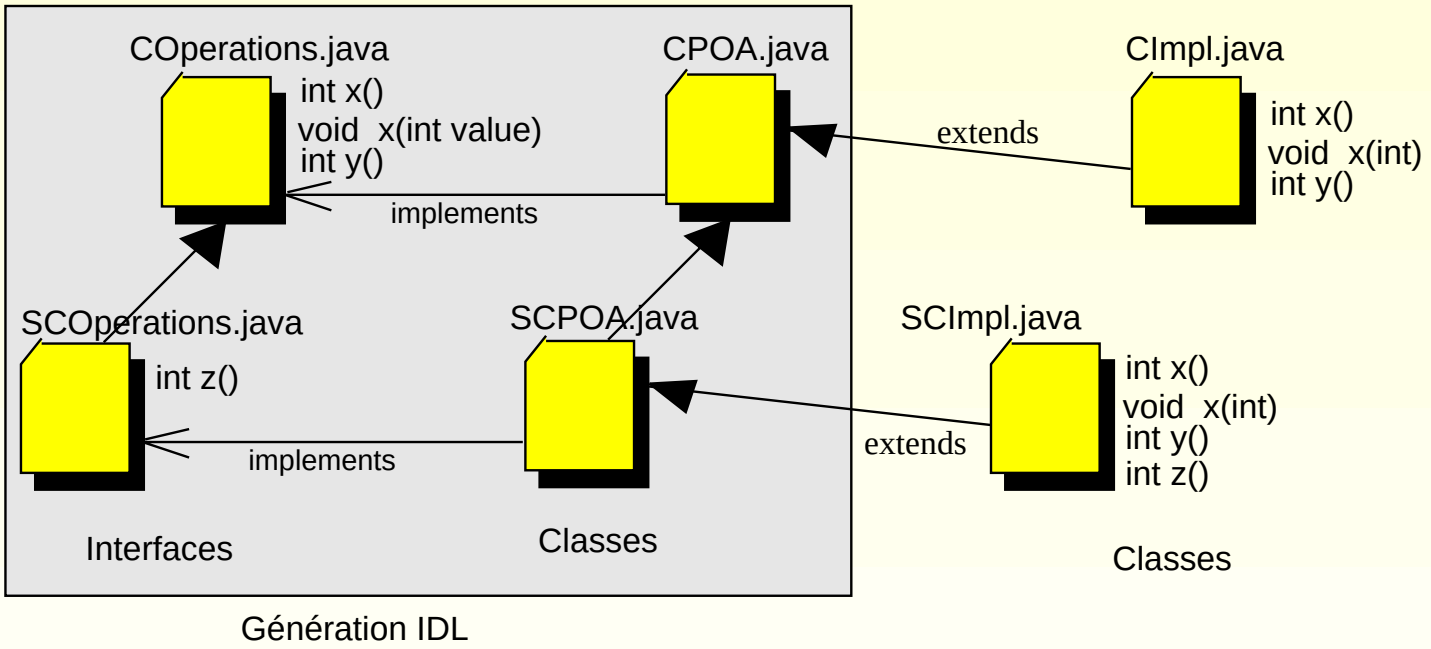
```
module exemple {
  interface C {
    attribute long x ;
    readonly attribute long y ;
  } ;
  interface SC : C {
    readonly attribute long z ;
  } ;
} ;
```

Mapping Java - héritage, données (2/3)

√ Fonctions accesseurs pour attributs IDL

Mapping Java - héritage, données (2/3)

- √ Fonctions accesseurs pour attributs IDL
- √ Héritage simple Java : réécrire plusieurs procédures !



Mapping Java - héritage, données (3/3)

√ Mécanisme de délégation (tie)

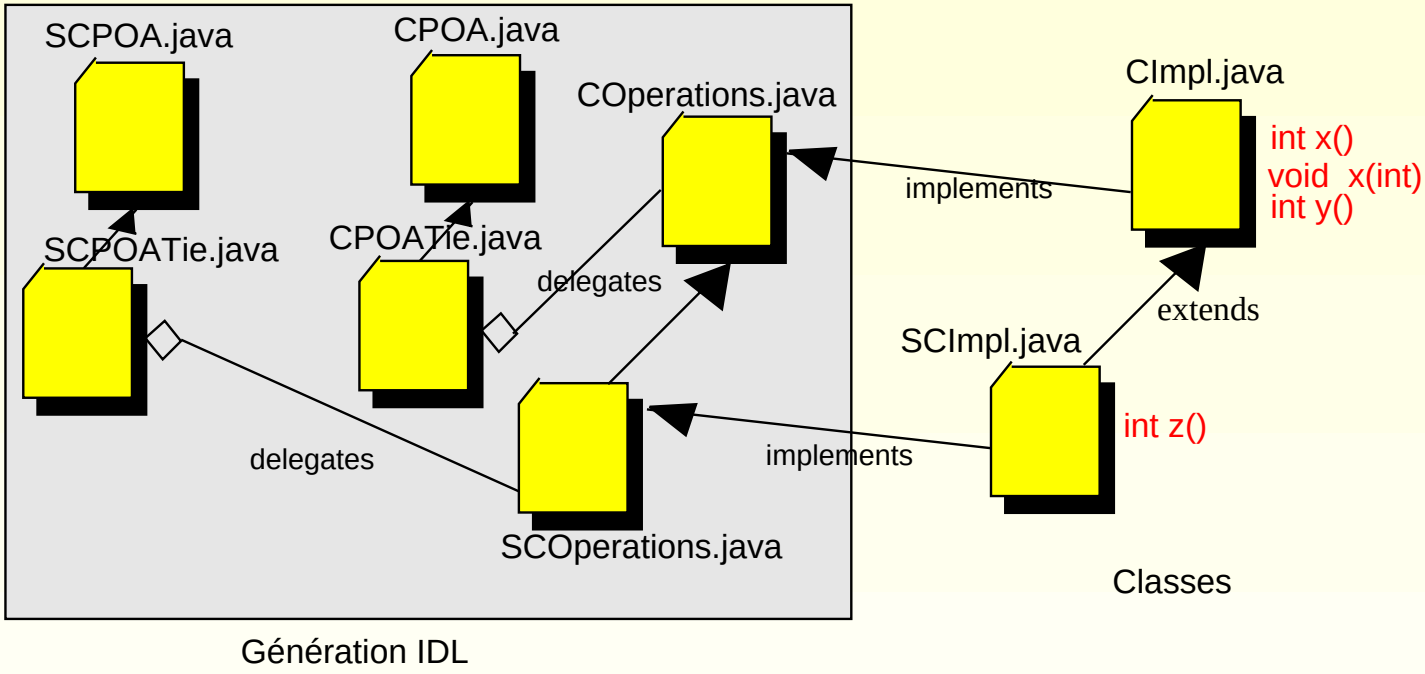
Mapping Java - héritage, données (3/3)

- √ Mécanisme de délégation (tie)
  - `jidl --tie exemple.idl`

Mapping Java - héritage, données (3/3)

√ Mécanisme de délégation (tie)

- `jidl --tie exemple.idl`



Conclusion

- √ Les moins :
  - Programmation difficile

Conclusion

√ Les moins :

- Programmation difficile
- de nombreux services (API) à assimiler

Conclusion

- √ Les moins :
  - Programmation difficile
  - de nombreux services (API) à assimiler
  - Encore beaucoup de choses à réaliser : conception, modèle de composants, service métier, déploiement,. . .
- √ Les plus :
  - Programmation langage de script facile
    CorbaScript, JIDLScript (API Java et CorbaScript), Gnome-python,. . .

Conclusion

√ Les moins :
- Programmation difficile
- de nombreux services (API) à assimiler
- Encore beaucoup de choses à réaliser : conception, modèle de composants, service métier, déploiement,. . .

√ Les plus :
- Programmation langage de script facile
  CorbaScript, JIDLScript (API Java et CorbaScript), Gnome-python,. . .
- Puissance du système

Conclusion

√ Les moins :
- Programmation difficile
- de nombreux services (API) à assimiler
- Encore beaucoup de choses à réaliser : conception, modèle de composants, service métier, déploiement,. . .

√ Les plus :
- Programmation langage de script facile
  CorbaScript, JIDLScript (API Java et CorbaScript), Gnome-python,. . .
- Puissance du système
- de nombreux services implémentés (trader, transactions, . . .)

Conclusion

√ Les moins :
- Programmation difficile
- de nombreux services (API) à assimiler
- Encore beaucoup de choses à réaliser : conception, modèle de composants, service métier, déploiement,. . .

√ Les plus :
- Programmation langage de script facile
  CorbaScript, JIDLScript (API Java et CorbaScript), Gnome-python,. . .
- Puissance du système
- de nombreux services implémentés (trader, transactions, . . .)