CORBA : des concepts à la pratique

L'application « Hello World »

J-M. Geib, C. Gransart, P. Merle

LIFL

1998-99

Au sommaire

- L'environnement ORBacus
- L'application « Hello World »
- Les interfaces de base du bus CORBA
- L'implantation en C++
- L'implantation en Java
- L'implantation en CorbaScript
- Conclusion

L'environnement ORBacus

- Conçu par la société Object-Oriented Concepts, Inc.
  - gratuit pour utilisation non commerciale
  - les sources sont entièrement disponibles
- Environnement CORBA 2.x ouvert
  - langages : C++, Java et CorbaScript
  - OS : Solaris, Linux, Windows 95/NT, AIX, SGI et HP-UX
- De très bonne facture
  - robuste, efficace, léger et adaptable
- Disponibilité : *http://www.ooc.com*

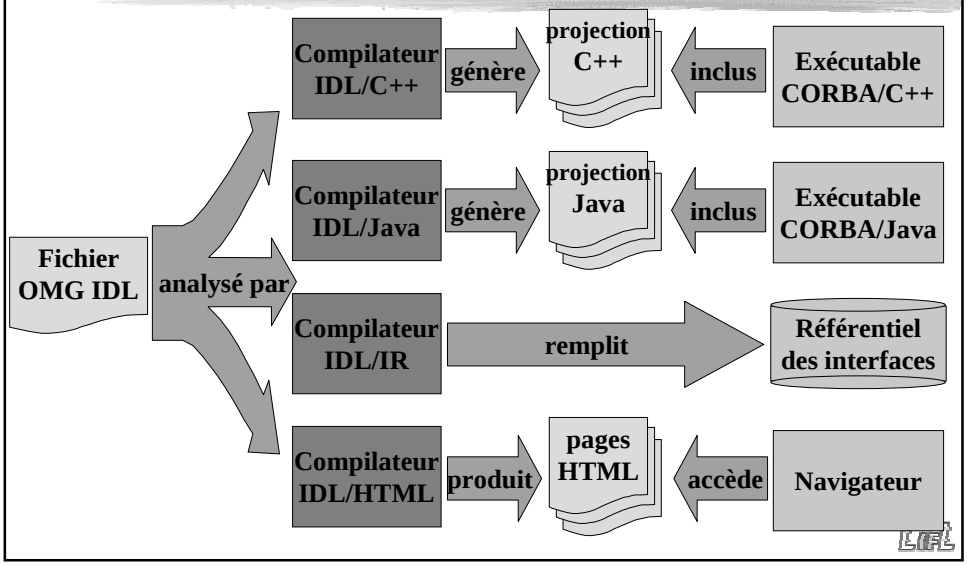
ORBacus : quelques caractéristiques techniques (1/2)

- Une projection pour les langages C++ et Java
  - 1 compilateur OMG IDL générant les souches
  - 1 bibliothèque implantant le bus CORBA
- Les mécanismes dynamiques
  - le référentiel des interfaces, le DII et le DSI
- Quelques services standards :
  - Nommage, Evénements, Propriétés, Vendeur et Sécurité
- IIOP en natif
  - interopérabilité avec d'autres bus !

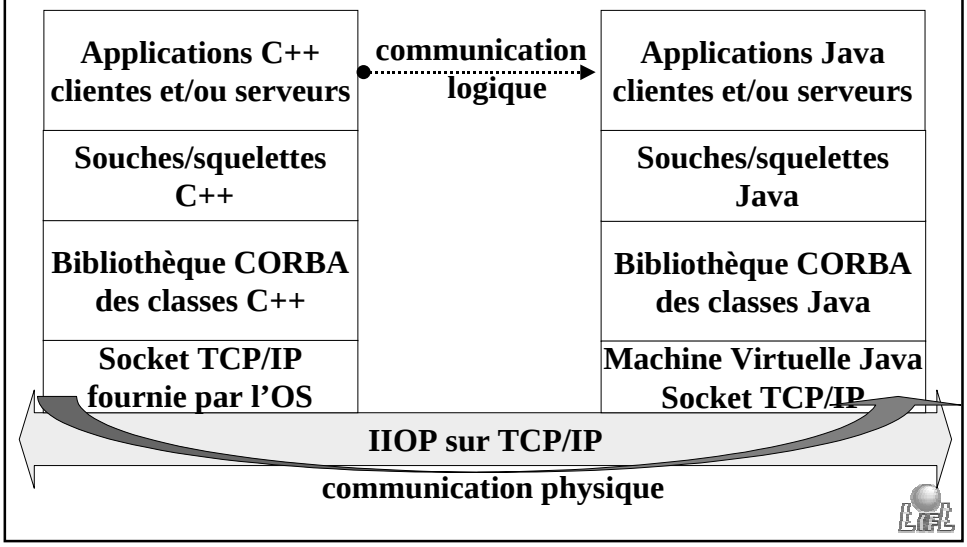
ORBacus : quelques caractéristiques techniques (2/2)

- L'interface Open Communication Interface (OCI)
  - « plugger » de nouveaux protocoles de transport
    - UDP, Multicast IP et réseaux spécialisés
  - proposition pour un prochain standard OMG
    - utilisateurs OUI / fournisseurs NON !
- A venir :
  - le Portable Object Adapter
  - un démon d'activation des processus serveurs
  - les prochaines évolutions de la norme CORBA !

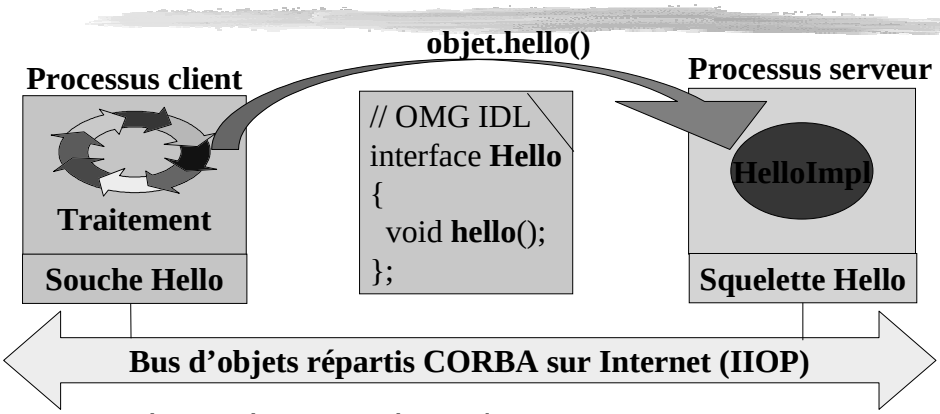
ORBacus : divers compilateurs OMG IDL



ORBacus : des bibliothèques et des processus répartis

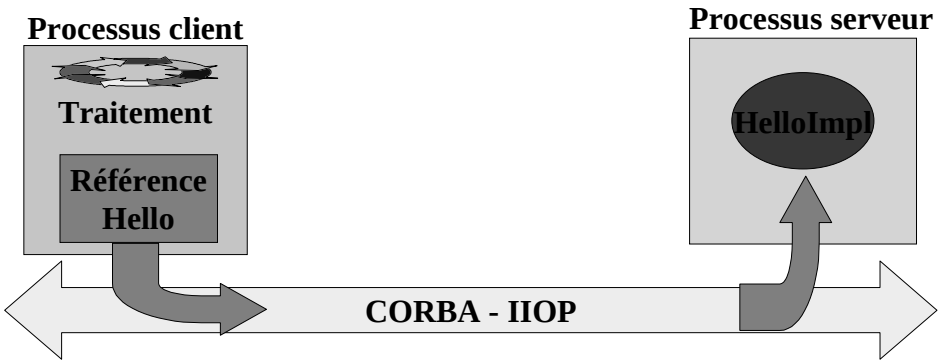


L'application « Hello World »



- Simple application client / serveur mais
  - multi langages, multi plates-formes, multi ORBs
  - invocation d'opérations sur objets distants
  - localisation / communication « transparente » via le bus

L'application « Hello World »



- Création de la référence Hello ?

Comment établir la liaison entre le client et le serveur ?

- Problème
  - comment le processus client connaît-t-il l'objet du processus serveur ?
- Solution
  - via une référence d'objet CORBA
- Problème
  - comment le processus client obtient-t-il la référence de l'objet CORBA ?
- Solution
  - via un service de désignation (voir cours suivant)

La notion de référence d'objet CORBA

- Sert à désigner / localiser un objet CORBA
  - nom de l'interface OMG IDL
  - protocole de communication (ex. IIOP)
  - processus serveur (ex. machine et port IP)
  - clé unique pour retrouver l'objet dans le serveur
- Représentation interne au bus
  - langage non OO : structure opaque
  - langage OO : référence sur 1 instance d'1 classe souche
- Représentation externe
  - chaîne IOR : *IOR:010000...345...2345*

L'interface `CORBA::Object`

```
// OMG IDL
module CORBA {
  interface Object {
    // Teste si une référence ne dénote aucun objet.
    boolean _is_nil ();
    // Teste si un objet n'existe plus.
    boolean _non_existent();
    // Teste si 2 références désignent le même objet.
    boolean _is_equivalent(in Object obj);
    // Calcule une clé de hachage.
    long _hash (in long maximum);
    // et bien d'autres opérations . . .
}; };
```

- Implicitement héritée par tout objet CORBA

La notion de bus CORBA

- Le bus CORBA est omniprésent et réparti !
- Chaque exécutable CORBA est lié à une bibliothèque pour utiliser le bus CORBA
- A l'exécution, un objet représente le bus dans chaque exécutable
  - c'est un objet local (pas d'IOR)
  - donc non accessible à distance
  - son rôle est d'assurer le contrôle local du bus
  - défini par l'interface `CORBA::ORB`

L'interface `CORBA::ORB`

```
// OMG IDL
module CORBA {
  // interface de l'objet local représentant le bus
  interface ORB {
    // représentation interne vers chaîne IOR
    string object_to_string (in Object obj);
    // chaîne IOR vers représentation interne
    Object string_to_object (in string str);
    // et bien d'autres opérations . . .
  };
  // Initialiser et obtenir localement l'ORB.
  ORB ORB_init ( . . . );
};
```

La notion d'adaptateur d'objets

- Intermédiaire entre le bus et les possibles supports physiques d'implantation des objets
  - BOA pour processus
  - OODA pour SGBDOO
  - LOA pour bibliothèques
  - COA pour carte à microprocesseur
  - POA de CORBA 2.2
- ORBacus fournit un BOA minimal
  - pas d'activation automatique des processus serveurs
- Prochainement le Portable Object Adapter

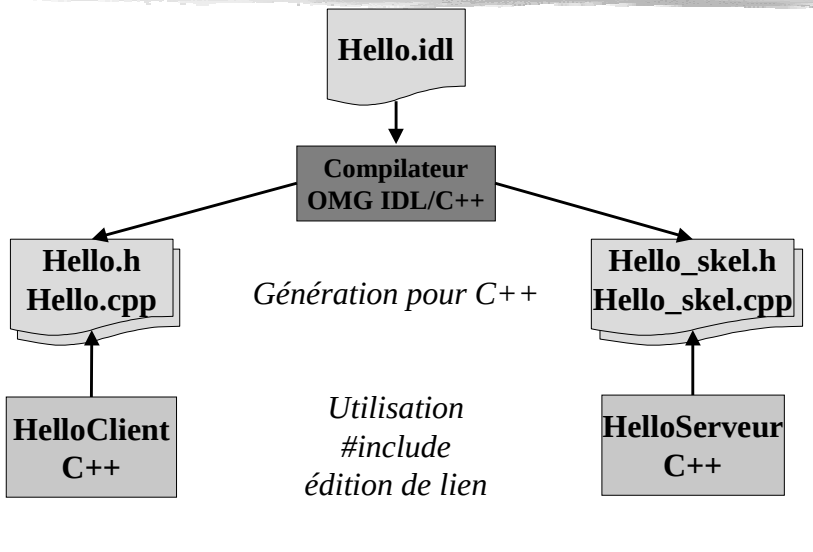
L'interface CORBA::BOA

```
module CORBA {
  // non standardisé par l'OMG.
  interface ImplementationDef { . . . };

  interface BOA {
    // Se mettre en attente des requêtes aux objets.
    void impl_is_ready (in ImplementationDef impl);
  };

  interface ORB {
    typedef sequence<string> arg_list;
    typedef string OAid;
    // Obtenir l'adaptateur d'objets.
    BOA BOA_init (inout arg_list argv, in OAid oa_identifier);
}; };
```

La génération des souches C++ pour le fichier `Hello.idl`



La classe C++ souche `Hello`

- **Hello.h** : définition de la classe C++ souche

```
class Hello;
typedef Hello* Hello_ptr;
class Hello_var { Hello_ptr ptr; ... };

class Hello : virtual public CORBA_Object {
  . . . partie spécifique à l'ORB . . .
public:
  virtual void hello();

  static Hello_ptr _narrow(CORBA_Object_ptr);
};
```

- **Hello.cpp** : implantation de la souche
  - spécifique à chaque ORB !

La classe C++ squelette `Hello`

- **Hello_skel.h** : définition de la classe squelette

```
#include <Hello.h>

class Hello_skel : virtual public Hello,
                   virtual public CORBA_Object_skel {
public:
  virtual void hello() =0;

protected:
  Hello_skel();
  void dispatcher (...);
};
```

- **Hello_skel.cpp** : implantation du squelette
  - spécifique à chaque ORB !

L'implantation C++ de `Hello`

```
// Importation du squelette C++ de l'interface OMG IDL Hello.
#include <Hello_skel.h>

// Implantation par héritage du squelette OMG IDL.
class HelloImpl : public Hello_skel {
public:
  // Constructeur.
  HelloImpl () : Hello_skel()
  {
    cout << "C++ HelloImpl::HelloImpl() : instanciation" << endl;
  };

  // Implantation de l'opération OMG IDL void hello()
  virtual void hello()
  {
    cout << "C++ HelloImpl::hello() : Hello World!" << endl;
} };
```

La hiérarchie C++ des classes souches / squelettes

Localisation	Classe	Rôle
HelloClient HelloServeur	**CORBA_Object**	Bibliothèque CORBA contient l'IOR
HelloClient HelloServeur	**Hello** void hello()	Souche C++ émission requête
HelloServeur	**Hello_skel** dispatcher()	Squelette C++ réception requête
HelloServeur	**HelloImpl** void hello()	Réalisation C++ affichage écran

Le programme HelloServeur en C++ (1/2)

```
// Importation de l'implantation C++ de l'interface Hello.
#include <HelloImpl.cpp>

int main(int argc, char* argv[], char*[])
{
  try {
    // Initialisation du bus CORBA pour un processus serveur.
    CORBA_ORB_var orb = CORBA_ORB_init (argc, argv);
    CORBA_BOA_var boa = orb -> BOA_init (argc, argv);

    // Création de l'objet d'implantation Hello.
    Hello_var hello = new HelloImpl ();

    // Diffuser la référence de l'objet.
    CORBA_String_var chaineIOR = orb->object_to_string (hello);
    cout << "IOR de l'objet : " << chaineIOR << endl;
```

Le programme HelloServeur en C++ (2/2)

```
    // Mettre le serveur en attente des requêtes CORBA.
    boa -> impl_is_ready (CORBA_ImplementationDef::_nil());

  // en cas de problème lié à l'utilisation de CORBA.
  } catch(CORBA_SystemException& ex) {
    OBPrintException (ex);
    return 1;
  }

  return 0;
}
```

- Classes **_var** pour la libération automatique de la mémoire
- Portabilité : CORBA: : à la place de CORBA_ !

Le programme HelloClient en C++ (1/2)

```
// Importation de la souche Hello générée par le compilateur IDL.
#include <Hello.h>

int main(int argc, char* argv[], char*[])
{
  try {
    // Initialisation du bus CORBA pour un processus client.
    CORBA_ORB_var orb = CORBA_ORB_init (argc, argv);

    // Instanciation d'une souche C++ référençant l'objet CORBA.
    CORBA_Object_var obj = orb -> string_to_object(argv[1]);
    assert(!CORBA_is_nil(obj));

    // Conversion vers une référence Hello.
    Hello_var hello = Hello::_narrow(obj);
    assert(!CORBA_is_nil(hello));
```

Le programme HelloClient en C++ (2/2)

```
    // Invocation de l'objet distant.
    hello -> hello();

  // en cas de problème lié à l'utilisation de CORBA.
  } catch(CORBA_SystemException& ex) {
    OBPrintException(ex);
    return 1;
  }

  return 0;
}
```

- Localisation réseau « masquée » lors des invocations !

Compilation et exécution en C++ de l'application « Hello World »

- Génération des souches/squelettes C++

```
> <répertoire_ORBacus>/bin/idl Hello.idl
```

- Compilation C++ des sources applicatifs et souches/squelettes

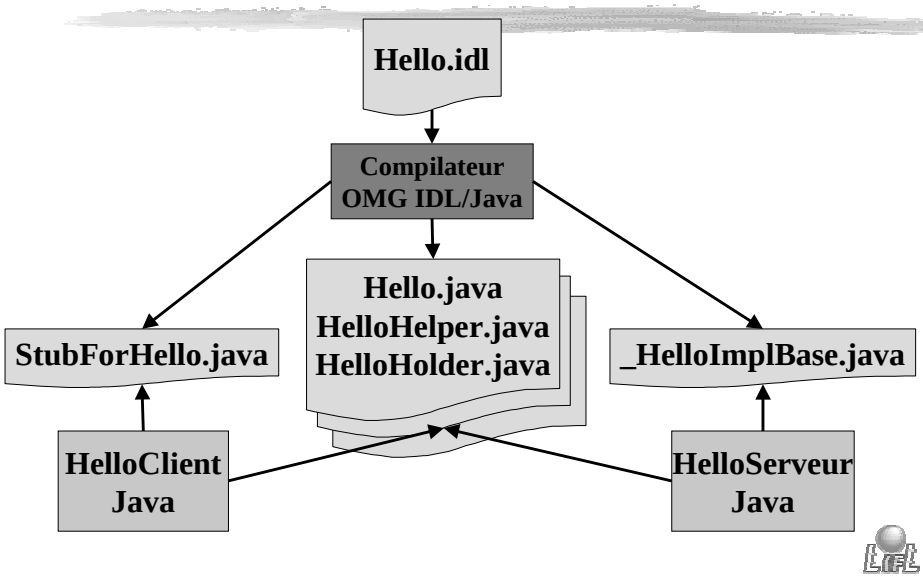
```
> gcc -o Hello.o -c Hello.cpp            <flags C++ / ORBacus>
> . . .
> gcc -o HelloServeur HelloServeur.o HelloImpl.o Hello_skel.o
                                  Hello.o <bibliothèques ORBacus>
> gcc -o HelloClient HelloClient.o Hello.o
                                        <bibliothèques ORBacus>
```

  - => besoin d'un Makefile

- Exécution des programmes

```
> ./HelloServeur
> ./HelloClient IOR: . . .
```

La génération des souches Java pour le fichier `Hello.idl`



L'interface et les classes `Hello` en Java

- **Hello.java** : interface pour le type Hello

```
public interface Hello extends org.omg.CORBA.Object {
  public void hello();
}
```

- **HelloHelper.java** : classe utilitaire pour le type Hello

```
final public class HelloHelper {
  public static Hello narrow(org.omg.CORBA.Object obj) {...}
  ... méthodes insert, extract, type ,id, read, write ...
}
```

- **HelloHolder.java** : pour le passage *out* et *inout*

```
final public class HelloHolder implements ... {
  public Hello value;
  ... méthodes constructeurs, _read, _write, _type …
}
```

Les classes Java souche et squelette `Hello`

- **StubForHello.java** : implantation de la souche
  - sert à envoyer les requêtes
  - invisible pour les programmeurs
  - instanciée automatiquement par **HelloHelper** (narrow)
  - utilise le DII pour assurer la portabilité binaire
- **_HelloImplBase.java** : implantation du squelette
  - reçoit et décode les requêtes
  - doit être héritée par l'implantation
  - utilise le DSI pour assurer la portabilité binaire

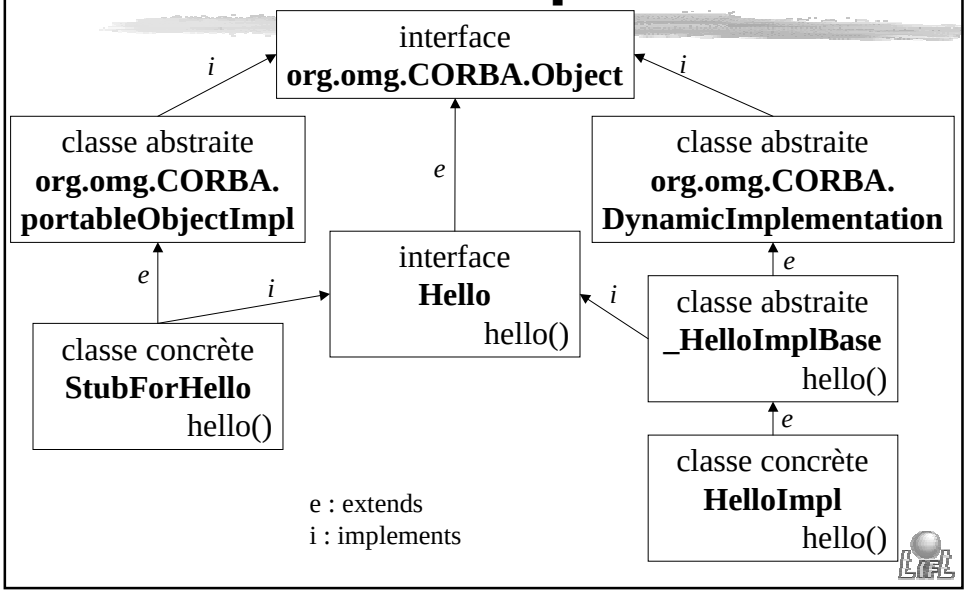
L'implantation Java de `Hello`

```
// Classe HelloImpl implante l'interface IDL Hello et hérite donc
// du squelette _HelloImplBase génére par le compilateur IDL.

  public class HelloImpl extends _HelloImplBase
  {
    public HelloImpl ()
    {
      System.out.println ("JAVA HelloImpl.HelloImpl() : instanciation");
    }

    // Implantation de l'opération OMG IDL void hello()
    public void hello()
    {
      System.out.println("JAVA HelloImpl.hello() : Hello World!");
    }
  }
```

La hiérarchie Java des classes souches/squelettes



Le programme HelloServeur en Java (1/2)

```
public class HelloServeur
{ // Fonction principale implantant un processeur serveur CORBA.
  public static void main(String args[])
  {
    try {
      // Initialisation du bus CORBA pour un processus serveur.
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args,null);
      org.omg.CORBA.BOA boa = orb.BOA_init (args,null);

      // Création de l'objet d'implantation Hello.
      HelloImpl hello = new HelloImpl();

      // Diffusion de la référence de l'objet.
      String chaineIOR = orb.object_to_string (hello);
      System.out.println ("IOR de l'objet : " + chaineIOR);
```

Le programme HelloServeur en Java (2/2)

```
      // Mettre le serveur en attente des requêtes CORBA.
      boa.impl_is_ready(null);

      System.exit(0);

    // en cas de problème lié à l'utilisation de CORBA.
    } catch(org.omg.CORBA.SystemException ex) {
      System.err.println(ex.getMessage());
      ex.printStackTrace();
      System.exit(1);
    }
  }
}
```

Le programme HelloClient en Java (1/2)

```
public class HelloClient
{
public static void main(String args[])
{
  try {
    // Initialisation du bus CORBA pour un processus client.
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args,null);

    // Création de la souche Java référençant l'objet Hello.
    org.omg.CORBA.Object obj = orb.string_to_object(args[0]);
    Hello hello = HelloHelper.narrow(obj);

    // Invocation de l'objet distant.
    hello . hello();
```

Le programme HelloClient en Java (2/2)

```
      System.exit(0);
    // en cas de problème lié à l'utilisation de CORBA.
    } catch(org.omg.CORBA.SystemException ex) {
      System.err.println(ex.getMessage());
      ex.printStackTrace();
      System.exit(1);
    }
  }
}
```

- Similaire au code HelloClient en C++
- Sources et binaires Java portables sur tous les ORB/Java !

Compilation et exécution en Java de l'application « Hello World »

- Quelques variables d'environnement (Unix C Shell)

```
> setenv CLASSPATH <RépertoireORBacus>/lib/OB.jar: ... :classes
> setenv PATH <RépertoireORBacus>/bin:<JDK>/bin:$PATH
```

- Génération des souches/squelettes Java

```
> mkdir hello
> jidl Hello.idl --package hello
```

- Compilation des sources et souches/squelettes

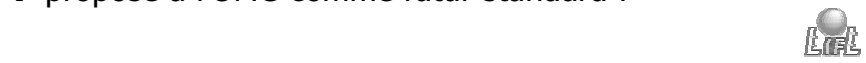
```
> mkdir classes
> javac -d classes *.java hello/*.java
```

- Exécution du serveur Hello en Java

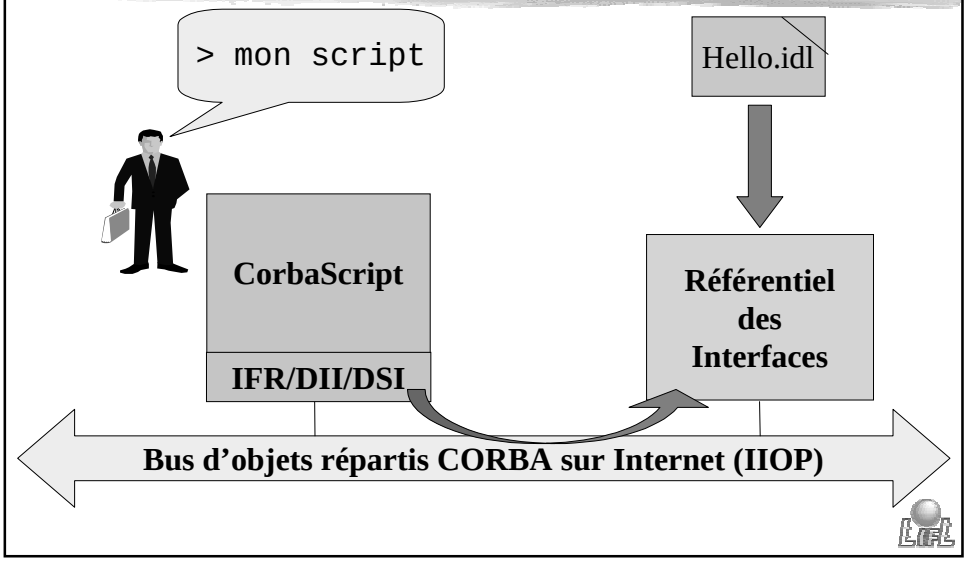
```
> java HelloServeur
> java HelloClient IOR: . . .
```

CorbaScript

- Langage de scripts pour CORBA 2.0
  - interactif, interprété et orienté objet
  - accès au système de typage OMG IDL (via IFR)
  - invocation d'objets CORBA (via DII)
  - implantation d'objets CORBA (via DSI)
- Projection IDL/CorbaScript vraiment très simple !
  - pas de génération de souches et de squelettes
- Conçu au LIFL
  - thèse de Philippe Merle
  - proposé à l'OMG comme futur standard !



Le chargement du fichier `Hello.idl` dans le référentiel des interfaces



L'implantation CorbaScript de l'interface Hello

```
class HelloImpl
{
  # Constructeur.
  proc __HelloImpl__ (self)
  {
    println ("CorbaScript HelloImpl.HelloImpl() : instanciation")
  }

  # Implantation de l'opération OMG IDL void hello()
  proc hello (self)
  {
    println ("CorbaScript HelloImpl.hello() : Hello World!")
  }
}
```

Le programme HelloServeur en CorbaScript

```
# Importation de l'implantation HelloImpl.
import HelloImpl

# Création de l'objet d'implantation.
hello = HelloImpl.HelloImpl()

# Rendre disponible l'objet sur le bus
CORBA.ORB.connect(hello, Hello)

# Diffusion de la référence de l'objet.
println ("IOR de l 'objet : ", hello._this._ior)

# Le serveur attend les requêtes CORBA.
CORBA.ORB.run ()
```

Le programme HelloClient en CorbaScript

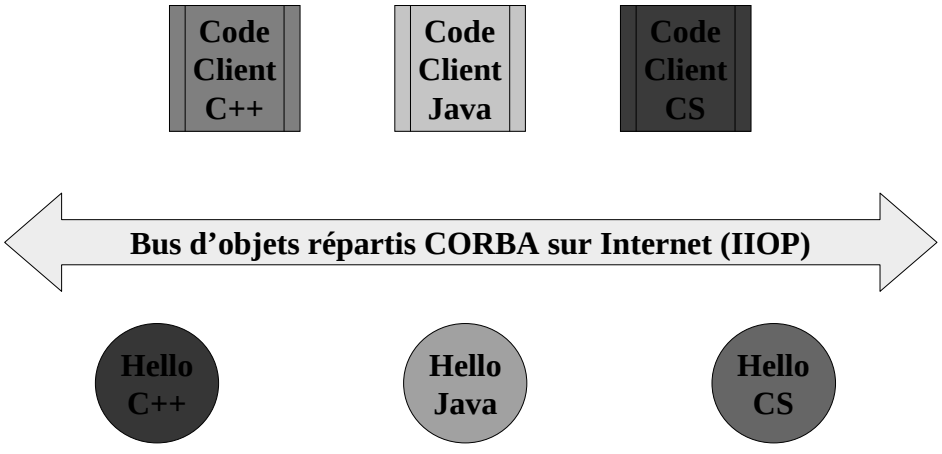
```
# Initialisation du bus implicite.

# Connexion à l'objet distant.
hello = Hello («IOR:...»)
# Narrow implicite.

# Invocation de l'objet distant.
hello.hello()
```

- Interactivement ou bien dans un fichier script
- CorbaScript est un « shell » pour les objets CORBA

L'interopérabilité entre les implantations



Quel langage choisir ?

- CORBA avec C++
  - - projection complexe à utiliser => erreurs
  - + mais efficace et déjà utilisé
- CORBA avec Java
  - + simple et portable
  - - mais pas encore efficace
- CORBA avec CorbaScript
  - + simple : pas de souches/squelettes, accès à tout CORBA
  - + aussi rapide que Java et 3 fois plus lent que C++ pour la partie CORBA :-)

Conclusion

- ORBacus : un sympathique CORBA 2.0
  - + multi plates-formes, multi langages et IIOP
  - + gratuit et sources disponibles
- Ce n'est pas très compliqué d'écrire une application CORBA !
  - il faut suivre quelques règles et schémas de conception
- « CORBA c'est bien, CorbaScript c'est mieux ! »