Programmation répartie: Objet distribué

CORBA

(Common Object Request Broker Architectur)

Plan du cours

- Introduction
  - définitions
  - problématiques
  - architectures de distribution
- Distribution intra-applications
  - notion de processus
  - programmation multi-thread
- **Distribution inter-applications et inter-machines**
  - sockets
  - middlewares par appel de procédures distantes (RPC)
  - middlewares par objets distribués (Java RMI, CORBA)
- Conclusion

RPC « à objets »

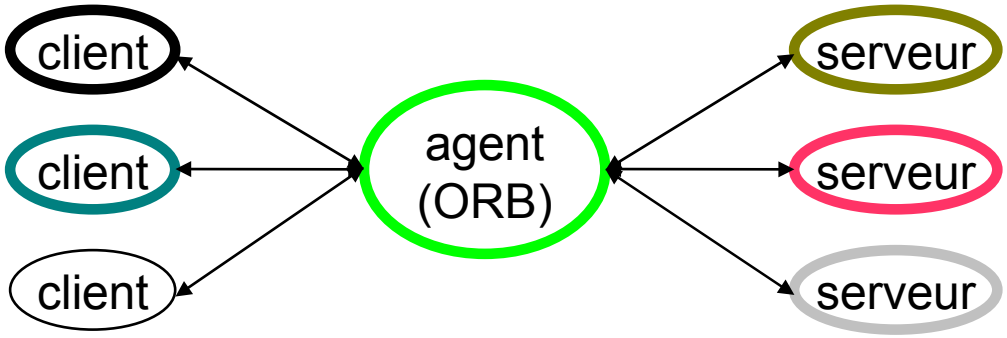
- **Motivations**
  - tirer parti des bonnes propriétés de l'objet (encapsulation, modularité, réutilisation, polymorphisme, composition)
  - objet : unité de désignation et de distribution
- **Objets "langage"**
  - représentation propre au langage : instance d'une classe
  - exemple : Java RMI
- **Objets "système"**
  - représentation "arbitraire" définie par l'environnement d'exécution
  - interopérabilité entre objets écrits dans des langages différents
  - exemple : CORBA

L'Object Management Group

- Objectif : faire émerger des standards pour l'intégration d'applications distribuées hétérogènes et la réutilisation de composants logiciels
- Historique
  - Consortium créé en 1989
  - 90 : Object Management Architecture (OMA) Guide
  - 91 : Common Object Request Broker: Architecture and Specification, version 1.1 (CORBA)
  - 94: CORBA version 2.0

CORBA : principes généraux

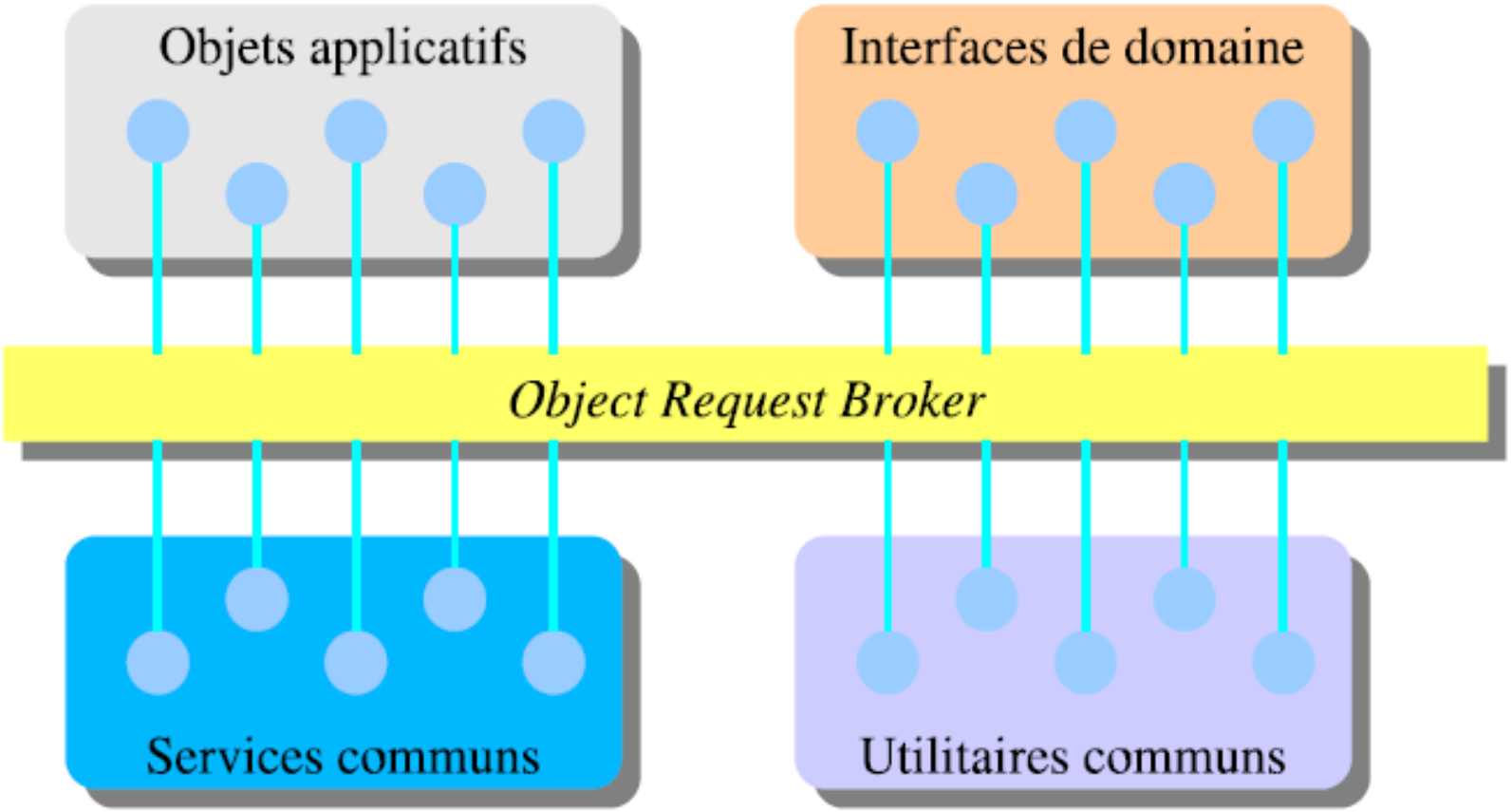
- **Met en oeuvre le modèle client-serveur**
- **Introduit une entité intermédiaire : l'*agent* (*broker*)**
  - isole les clients des serveurs
  - un client peut interagir avec un serveur sans connaître ni son adresse, ni la manière dont il fonctionne
- **Principe de fonctionnement**
  - le client demande à l'agent l'exécution d'un service (demande d'exécution d'une méthode sur un objet)
  - l'agent identifie un serveur capable de fournir le service
  - l'agent transmet la requête au serveur



CORBA : caractéristiques

- **Isolation des clients et des serveurs**
  - ils n'ont pas à se connaître mutuellement
  - permet d'ajouter de nouveaux clients et de nouveaux serveurs sans modifier l'existant
  - seul l'agent intermédiaire connaît l'adresse et les possibilités de chacun
  - ces infos n'ont pas à figurer dans le code des clients et des serveurs
- **Communications synchrones ou asynchrones**
  - choix laissé aux développeurs
  - souvent asynchrone quand enrobage d'applications existantes
- **Client/serveur très dynamique**
  - une même portion de code peut apparaître
    - comme client dans un échange
    - comme serveur dans un autre échange

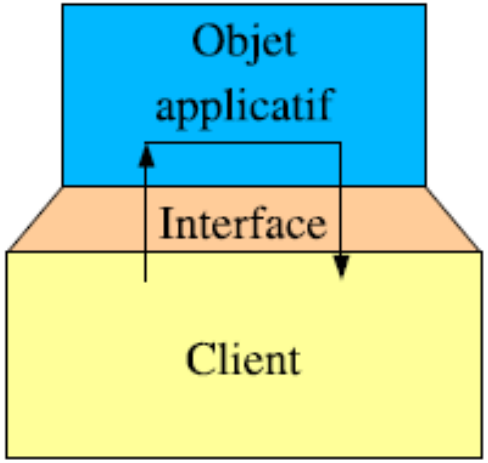
Object Management Architecture (OMA)



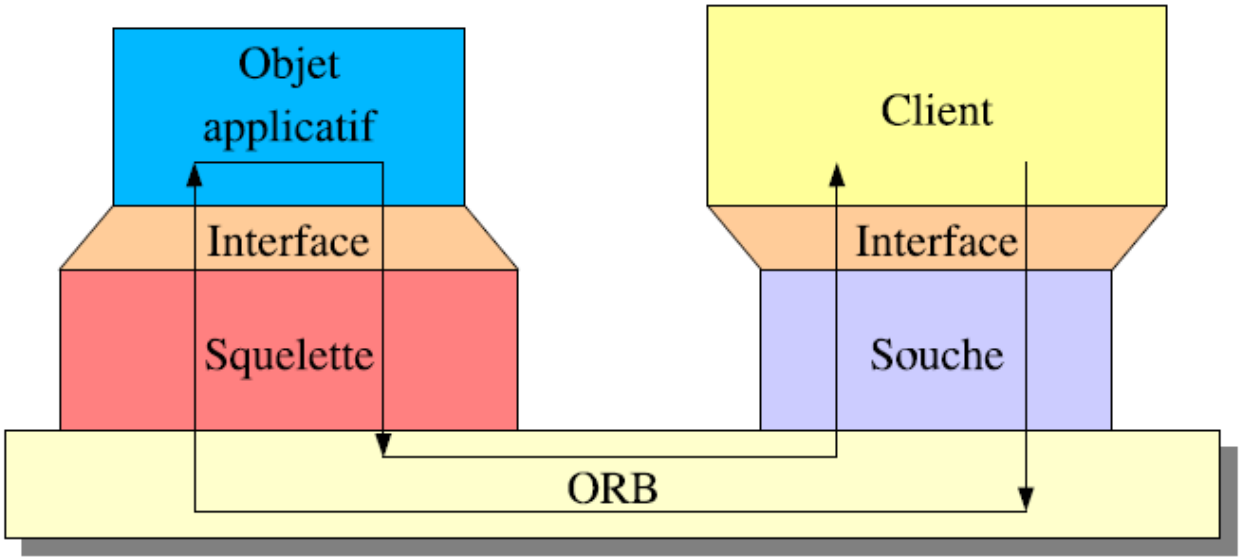
Object Management Architecture (OMA)

- **L'Object Request Broker (ORB ou agent)**
  - permet la portabilité des objets
  - permet leur interopérabilité dans un réseau hétérogène
- **Les services objets (CorbaServices)**
  - services spécifiques à la manipulation d'objets
  - définis dans la Common Object Services Specification (COSS)
    - désignation d'objets, permanence, cycle de vie, transactionnel, notification d'événements, sécurité, accès multiple, relationnel, gestion des licences d'objets)
- **Les services d'application (CorbaFacilities)**
  - ensemble de facilités de haut niveau pour les applications
    - accès aux bases de données, services d'impression, synchronisation
- **Les objets d'applications**
  - bibliothèques d'objets permettant d'accélérer le développement logiciel

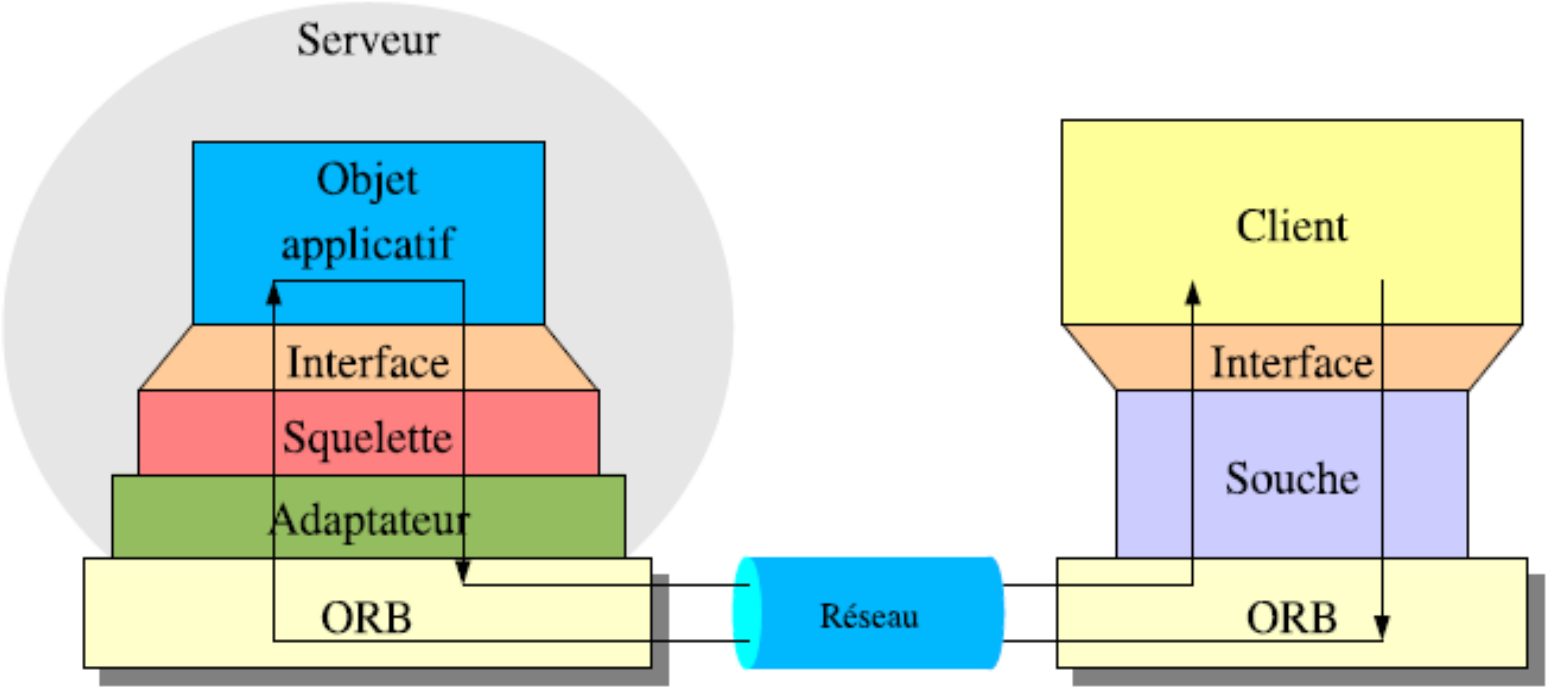
Object Local



Object Distribué



plus précisément

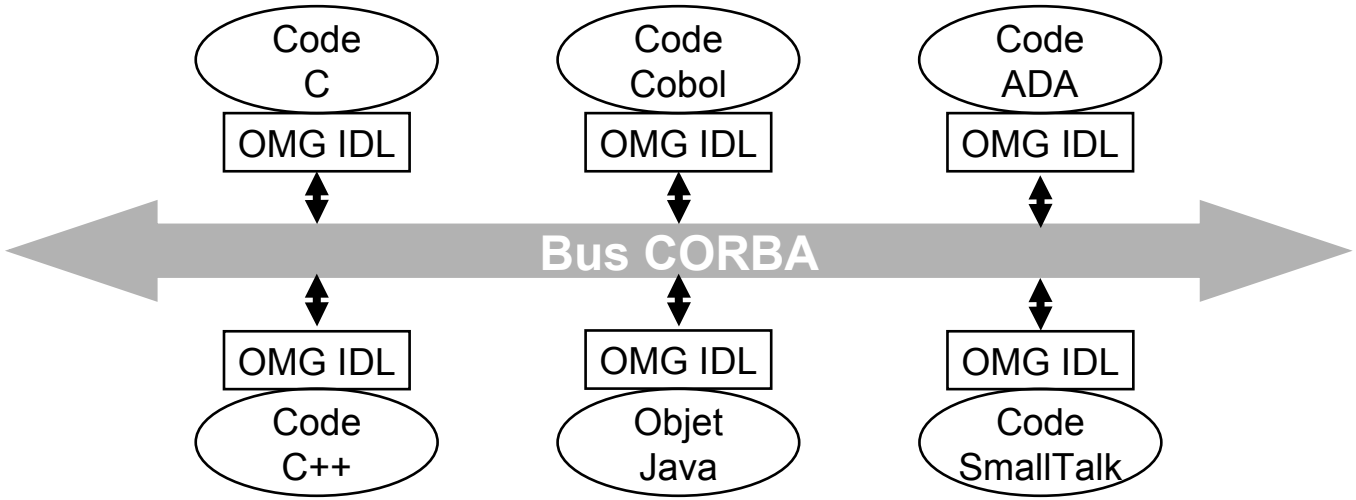


- Souche = *Stub*
- Squelette = *Skeleton*

Quel sera la forme du contrat?

OMG Interface Description Language (IDL)

- **Ce langage a été défini :**
  - pour décrire les interfaces des objets
  - comme langage pivot entre applications
  - pour générer des squelettes de programme dans les langages de programmation des applications
- **Une application = des objets CORBA**
  - décrits par des interfaces OMG IDL
  - dialoguant via le bus CORBA



Les interfaces OMG IDL

- **Une interface OMG IDL**
  - = l'abstraction d'un type d'objets CORBA
  - = l'API à rendre publique
- **... contient les opérations**
  - exportées par l'objet
  - utilisées par les autres objets
  - pas forcément (rarement) toutes les méthodes implantées par la classe de l'objet

Les opérations OMG IDL

- **Une opération OMG IDL**
  - = l'abstraction d'un traitement réalisé par l'objet
- **... est décrite par une signature**
  - le nom de l'opération
  - les paramètres
    - leur nom formel
    - leur type
    - leur mode de passage
  - le type du résultat
  - les cas d'erreurs ou exceptions

Les types de données OMG IDL

- **Les objets sont hétérogènes**
  - différents langages de programmation
  - différents processeurs d'exécution
- **Il faut définir précisément les types de données échangées lors des invocations entre objets**
  - leur nature et leur format binaire
  - pour une gestion automatique de l'hétérogénéité
- **Le langage OMG IDL permet de décrire des types de données et définit exactement leur format binaire**

Les contrats OMG IDL

- **A long terme, beaucoup d'applications CORBA**
  - de nombreux types et interfaces OMG IDL
- **Possibilité de conflits de noms**
  - un type peut avoir un sens différent et une représentation différente selon l'application
- **Aspect génie logiciel**
  - groupement de définitions communes
  - définition du contrat entre applications
- **Pour cela, le langage OMG IDL permet de définir des modules (contrats) réutilisables**

presentation de IDL : un exemple

```
#include <date.idl>
module annuaire {
  typedef string Nom;
  typedef sequence<Nom> DesNoms;
  struct Personne {
    Nom nom;
    string telephone;
    ::date::Date date_naissance;
  };
  typedef sequence<Personne> DesPersonnes;

  readonly attribute string libelle;
  exception ExisteDeja { Nom nom; };
  exception Inconnu { Nom nom; };

  interface AdministrationAnnuaire {
    void ajouterPersonne (in Personne personne) raises(ExisteDeja);
    void retirerPersonne (in Nom nom) raises(Inconnu);
  };

  interface ConsultationAnnuaire {
    Personne obtenirPersonne (in Nom nom) raises(Inconnu);
    DesNoms listerNoms ();
  };
};
```

presentation de IDL : les types primitives

IDL	Java
`void`	`void`
`boolean`	`boolean`
`octet`	`byte`
`char` (1 octet) et `wchar` (2 octets)	`char`
`short` et `unsigned short`	`short`
`long` et `unsigned long`	`int`
`long long` et `unsigned long long`	`long`
`float`	`float`
`double`	`double`
`long double`	
`string`	`String`
`wstring`	`String`

OMG-IDL : Les types utilisateur «structure »

- **Conteneur de données**
- **Comparable aux structures du C**

```
struct Point {
        short x;
        short y;
};
```

- **Traduit en classe Java**

```
public final class Point implements IDLEntity {
        public short x;
        public short y;
public Point () { }
public Point (short x, short y) {
        this.x = x; this.y = y;}
}
```

OMG-IDL : « tableau » et « séquence »

- **Tableaux :** `typedef float Matrice[10][10]`
- **Séquences bornée ou non :**
  - `typedef sequence<string,10> ListeMots`
  - `typedef sequence<string> ListeMots`
- **Traduits en tableaux Java**

OMG-IDL : les autres types

- **Les alias :** `typedef float Nombre` **pas de traduction**
- **Les énumérations :** `enum Ops {Plus, Moins}` **traduites en constantes dans la classe** `Ops` **indexée par des entiers**
- **Les unions :**

```
union Test switch(Ops) {
case Plus:
long l;
default:
short s;
}
```

- **traduites en une unique classe**

OMG-IDL : les types utilisateurs vers Java

- **Implantent `IDLEntity` qui hérite de `Serializable`**
- **Arguments/valeurs de retour recopiées entre client et serveur**
- **Deux autres classes générées par idlj :**
  - <*Type*>Helper pour le transtypage
  - <*Type*>Holder type enveloppe

OMG-IDL : les références

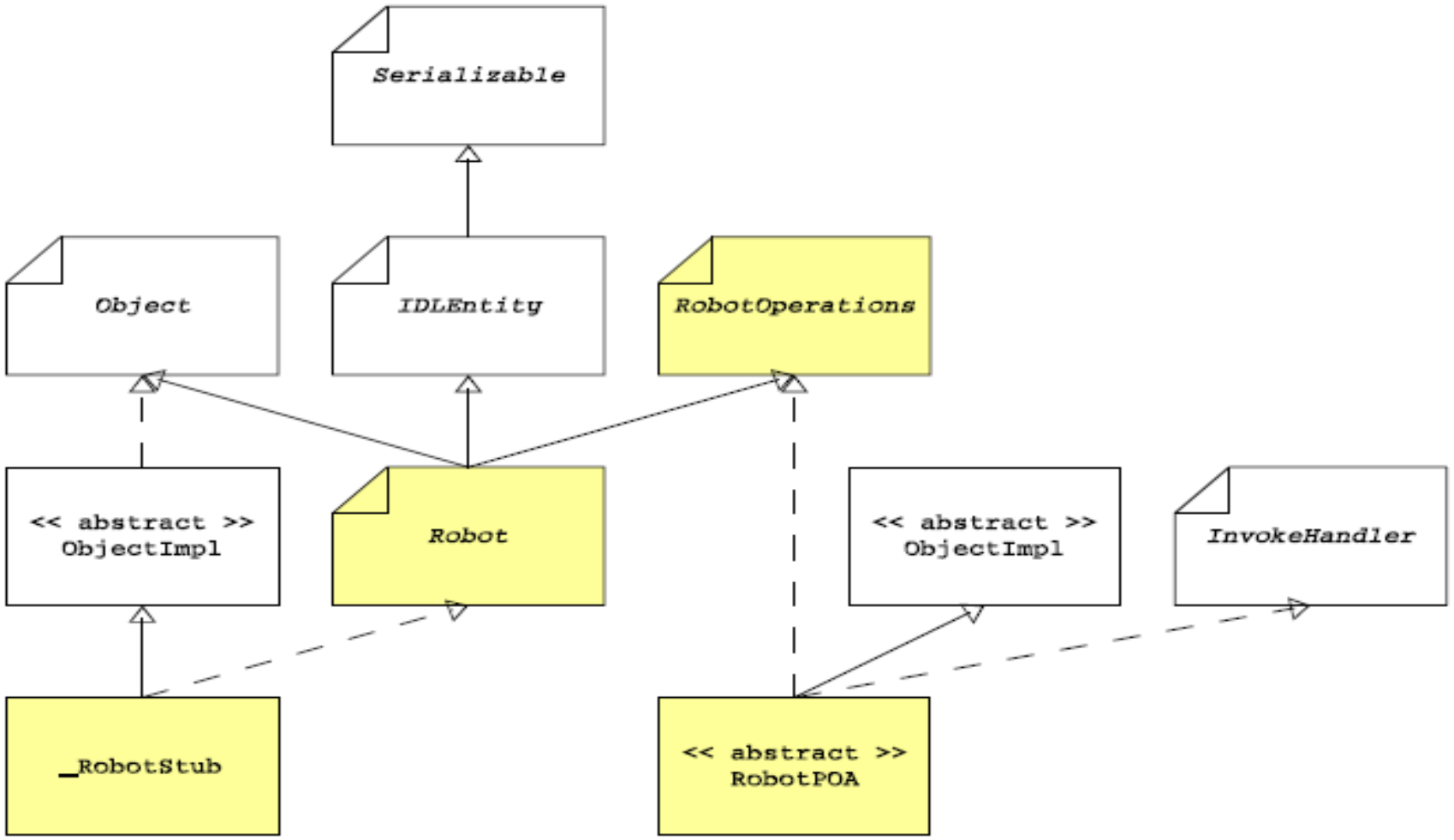
- **Correspond aux interfaces qui peuvent contenir :**
  - des attributs
  - des déclarations de méthodes
  - toute sorte de déclaration de type (struct, enum, ...)
  - des constantes
- **Exemple :**

```
interface Robot {
   struct Point {
     short x;
     short y;
   } ;
attribute Point pos;
  void move(in short dx, in short dy);
   const short def = 1;
};
```

OMG-IDL : les références

- Génération d'un ensemble de classes/interfaces Java
  - Interface `RobotOperations` contient uniquement les opérations définies dans l'IDL
  - Interface `Robot` hérite de `RobotOperations`, plus :
    - `org.omg.CORBA.Object` : interface des objets CORBA
    - `IDLEntity` : interface pour le passage de paramètre/valeur de retour
  - Classes `RobotPOA` et `RobotStub` générées pour, respectivement, le coté serveur et le coté client
  - Classes `RobotHelper` et `RobotHolder` générées pour
    - le transtypage
    - le wrapping

Hiérarchie des Types en Java pour OMG-IDL



Les attribues

- **Précédé du mot clef `attribute` :**
  - `attribute short x;`
- **Traduit en getter/setter**
  - `short x();`
  - `void x(short newX);`
- **Si modificateur `readonly` pas de setter généré**

Les méthodes

- **Profil des méthodes comparables à ceux de Java**
- **Possibilité de préciser que la méthode n'attend pas le résultat : oneway**
- **Précisions sur le mode de passage des paramètres**
  - in : valeur du paramètre initialisée par le client récupérable par le serveur
  - out : valeur du paramètre modifiée par le serveur récupérable par le client
  - inout : les deux !
- **Type in traduit par le type correspondant en Java**
- **Type out ou inout traduit par le type enveloppe du type correspondant en Java**

Les Types `Holder`

- **Dispose d'un champs `value`**
- **Nécessité de construire le `Holder` du coté `client`**
  - Constructeur par défaut pour out
  - Constructeur avec un paramètre pour inout
- **`Holder` copié dans les deux sens**

Les Exceptions

- **Type utilisateur particulier Comparable à une structure**

```
exception MonException {
    string message;
    short val;
};
```

- **Traduit en classe Hhéritant de UserException qui implemente `IDLEntity`**
- **Déclarations de levée d'exception avec le mot clef `raises`**
  - `void method() raises (Ex1, Ex2);`

structuration & visibilité

- **Fichiers générés organisés en paquetages correspondants,plus ou moins, aux accolades « {} » du fichier IDL**
- **Directive `module` définit un paquetage**
- **Structure S définie à l'intérieur d'une interface I, respectivement, dans une autre structure S2, entraîne la génération de la classe I.S, respectivement, S2.S**
- **Visibilité liée aux accolades « {} » comme en Java**
- **Pour éviter les ambiguïtés ou préciser un type particulier,possibilité de donner un chemin d'accès dans l'IDL**
  - chemin relatif l'endroit d'utilisation A::B
  - chemin absolu depuis la « racine » du fichier ::A::B

Les Constantes

- Outre les constantes de preprocessing (#define), qui
- n'auront pas d'équivalent en Java, il est possible de définir des constantes dans l'IDL pour les types primitifs Définies en utilisant le mot clef `const`

```
const short c = 3
```

- Traduit en constante de nom **I.c** si définit dans une interface **I**
- Sinon, traduit en une interface **c** contenant une constante `value`

OMG IDL : Héritage

- **Possibilité d'héritage en IDL**

```
interface Base1 {
   void f();
};
interface Base2 {
   void g();
};
interface Herite : Base1, Base2 {
   void h();
};
```

- **Ni surcharge, ni définitions multiples**

OMG IDL : compilation

- **Mode statique**
  - projection des descriptions OMG IDL vers les langages d'implantation des clients et serveurs
    - production des stubs clients
    - production du skeleton serveur
- **Mode dynamique**
  - instanciation sous forme d'objets CORBA des descriptions OMG IDL dans un référentiel des interfaces commun

OMG IDL : conclusion

- **Langage très complet**
  - permet de définir exhaustivement objets, messages, classes
  - contient toute l'information dont un client a besoin pour utiliser les services qu'elle représente
  - programmes clients écrits en Java, C++, Smalltalk, etc.
  - compilateurs IDL qui automatisent la correspondance en produisant, dans le langage choisi
    - les stubs côté client
    - les skeletons côté serveur

Les étapes de dévellopement d'un objet distribué

1) **Définir l'interface avec le langage IDL (Interface Definition Langage)**
2) **Générer les classes nécessaires à la distribution préprocessing**
3) **Définir le code fonctionnel de l'objet distribué : le servant**
4) **Distribuer l'objet au travers de l'ORB**
   1. Initialiser l'ORB
   2. Enregistrer le servant de l'objet distribué dans l'ORB
   3. Rendre disponible une référence permettant de localiser l'objet distribué
   4. Mettre l'ORB en attente de requêtes

exemple: étape 1> Définir l'interface IDL [Gilles roussel]

Hello.idl

```
module cours1 {
  interface Hello {
    string hello(in string localisation);
  };
};
```

exemple: Compilation et génération des classes

```
# idlj -fall -td generated \
     -pkgPrefix cours1 fr.umlv.ir3.corba Hello.idl
# ls generated/fr/umlv/ir3/corba/cours1
HelloHelper.java      Hello.java
HelloPOA.java         HelloHolder.java
HelloOperations.java  _HelloStub.java
```

exemple: écriture du servant

HelloServant.java

```
package fr.umlv.ir3.corba.cours1;

public class HelloServant extends HelloPOA {
  public String hello(String localisation) {
    return "Hello " + localisation;
  }
}
```

exemple: distribution de l'objet

HelloServer.java

```
package fr.umlv.ir3.corba.cours1;
import org.omg.CORBA.ORB;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;

public class HelloServer {
  public static void main(String[] args) throws Exception {
    ORB orb = ORB.init(args, null);
    HelloServant servant = new HelloServant();
    POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
    byte[] id = rootPOA.activate_object(servant);
    org.omg.CORBA.Object ref = rootPOA.id_to_reference(id);
    String ior = orb.object_to_string(ref);
    System.out.println(ior);
    rootPOA.the_POAManager().activate();
    System.out.println("Server running!");
    orb.run();
  }
}
```

exemple: client

- Connaître l'interface et éventuellement générer les classes correspondantes
- Initialiser l'ORB
- Récupérer une référence de l'objet distant
- Obtenir la souche de l'objet distant de l'ORB
- Appeler la méthode à distance

HelloServer.java

```
package fr.umlv.ir3.corba.cours1;
import org.omg.CORBA.ORB;

public class HelloClient {
  public static void main(String[] args) {
    ORB orb = ORB.init(args, null);
    Hello h = HelloHelper.narrow(orb.string_to_object(args[0]));
    System.out.println(h.hello("World"));
  }
}
```

exemple: autre approche par délégation

- **Permet à un servant d'hériter d'autre chose que du POA**
  - En particulier, utile en cas d'héritage d'interface en IDL
- **Nécessité de générer une classe tie**
  - Option -fallTIE pour la commande idlj
  - Classe tie hérite de la classe POA
  - Les opérations de l'interface sont déléguées à une instance d'une autre classe passée en argument lors de la construction
- **Une instance de la classe tie est activée au sein du POA à la place de la classe d'implantation**

exemple: Exemple revisité

HelloServant.java

```
package fr.umlv.ir3.corba.cours1;

public class HelloServant extends HelloPOA {
  public String hello(String localisation) {
    return "Hello " + localisation;
  }
}
```

```
package fr.umlv.ir3.corba.cours2;

public class HelloServant implements HelloOperations {
  public String hello(String localisation) {
    return "Hello " + localisation;
  }
}
```

exemple: Exemple revisité

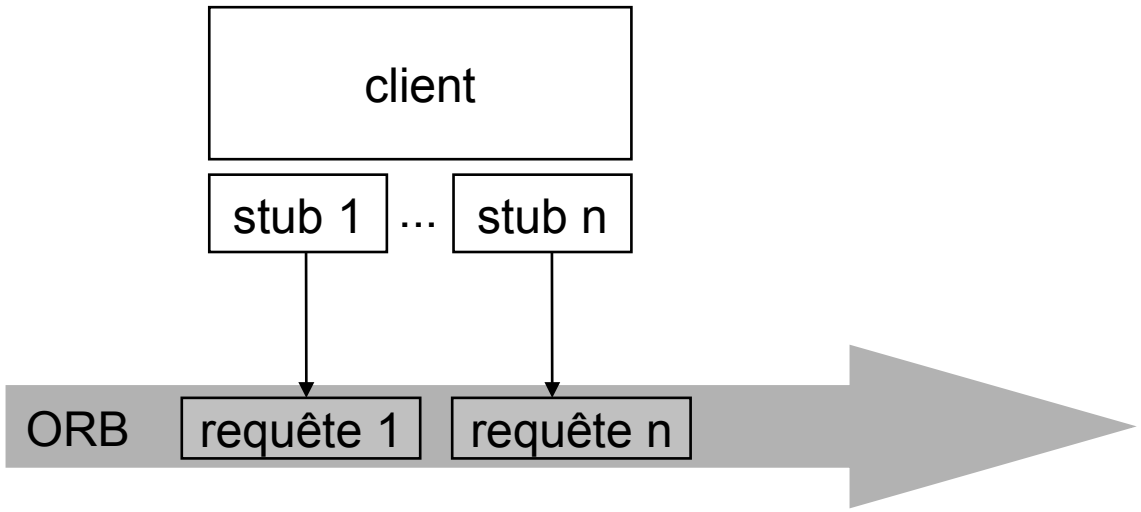
```
package fr.umlv.ir3.corba.cours2;
import org.omg.CORBA.ORB;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;

public class HelloServer {
  public static void main(String[] args) throws Exception {
    ORB orb = ORB.init(args, null);
    org.omg.CORBA.Object o =
            orb.resolve_initial_references("RootPOA")
    POA rootPOA = POAHelper.narrow(o);
    HelloServant servant = new HelloServant();
    HelloPOATie helloTie = new HelloPOATie(servant);
    byte[] id = rootPOA.activate_object(helloTie);
    o = rootPOA.id_to_reference(id);
    String reference = orb.object_to_string(o);
    System.out.println(reference);
    rootPOA.the_POAManager().activate();
    System.out.println("Server running !");
    orb.run();
  }
}
```

L'ORB = infrastructure de communication

- **Un ORB doit fournir les 5 services suivants**
  - la DII (Dynamic Invocation Interface)
    - interface d'invocation dynamique
    - construction dynamique des messages par l'application cliente
  - la DSI (Dynamic Skeleton Interface)
    - interface de skeleton dynamique
    - pour répondre côté serveur à des messages construits dynamiquement
  - l'IR (Interface Repository)
    - banque d'interfaces
  - une interface de programmation pour l'ORB lui-même
  - une implémentation du POA (Portable Object Adapter)

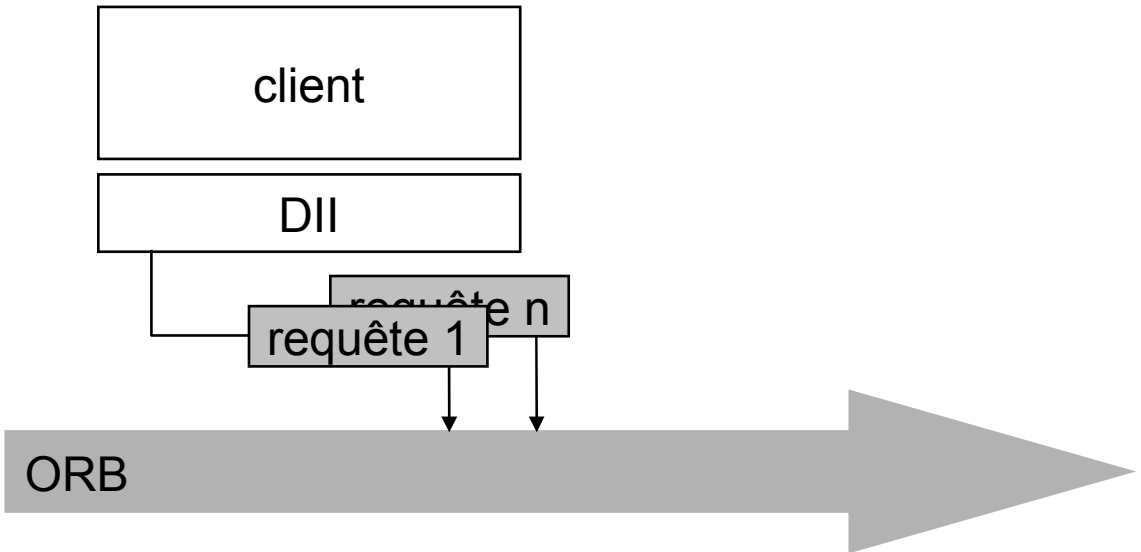
L'ORB côté client : invocation statique

- **Mécanisme similaire à Java RMI**
  - transmission de la requête au bus de communication via des stubs associés aux objets distants



L'ORB côté client : invocation dynamique (1)

- **Envoi de la requête via l'interface DII**
  - envoi de requête vers tous les objets du réseau (y compris ceux pour lesquels le client n'a pas de stub)
  - permet de protéger l'application client de changements d'implantation des objets, changements de programmes, changement de localisation
  - 2 modes d'invocation
    - synchrone : le client attend la réponse du serveur
    - asynchrone : le client n'attend pas la réponse du serveur (il peut la demander plus tard



L'ORB côté client : invocation dynamique (2)

- **L'invocation dynamique est construite en 4 étapes**
  - identifier l'objet destinataire de la requête
    - utilisation des services normalisés par CorbaServices
    - Naming Services et Trader Services
  - récupérer son interface (**`get_interface`**)
  - construire l'invocation (**`create_request`**)

```
ORBStatus create_request (
  in Context ctx,              // le contexte
  in Identifier operation,     // le nom de l'opération
  in NVList param_list,        // la liste des arguments
  inout NamedValue result,     // le résultat et son type
  out Request request,         // la requête construite
  in Flags flags               // options
)
```

  - l'exécuter et recevoir les résultats ou les exceptions éventuelles
    - mode synchrone (**`invoke`**)
    - mode asynchrone (**`send`**)
      - information sur l'état de la requête avec **`get_response`**
      - si résultat positif -> lire le résultat de l'opération dans le paramètre **`result`**

L'ORB : retour sur l'exemple => Initialisation

- **ORB.init(String[] args, Properties props);**
- **permet de créer un objet ORB pour une application la méthode à appeler pour initialiser l'ORB « Démarre » le serveur**
- **args et props permettent de paramétrer l'ORB Parmi les propri´et´es possibles :**
  - org.omg.CORBA.ORBClass et
  - org.omg.CORBA.ORBSingletonClass : classes d'implantation de l'ORB (standard)
  - org.omg.CORBA.ORBInitialHost et org.omg.CORBA.ORBInitialPort : machine et port du service de nommage (standard)
  - com.sun.CORBA.ORBServerHost et com.sun.CORBA.ORBServerPort : machine et port d'attente de l'ORB (spécifique Sun)

L'ORB : retour sur l'exemple => Initialisation

- **Plusieurs façons de paramétrer l'ORB :**
  - via la méthode init()

```
Properties props = new Properties();
props.setProperty("com.sun.CORBA.ORBServerPo
  rt","8080");
String[] argv = { "-ORBInitialPort", "5656" } ;
ORB orb = ORB.init(argv, props);
```

- **via les propriétés systèmes :**

```
java -Dorg.omg.CORBA.ORBInitialPort=8080 ....
```

- **via le fichier orb.properties recherché d'abord dans le répertoire utilisateur, puis dans file:///${java.home}/lib**

L'ORB : les états

- **orb.init(...) démarre l'ORB**
- **orb.run() bloque le processus léger courant tant que l'ORB n'est pas arrêté (plusieurs appels concurrents possibles)**
- **orb.shutdown() permet d'arrêter l'ORB. Un argument booléen précise si l'appel doit attendre la fin de l'ORB avant de retourner (attention au dead-lock)**

L'ORB côté serveur : Object Adapter (1)

- **Rôle**
  - maintenir l'illusion, chez le client, que l'objet est toujours actif et prêt à répondre à ses requêtes
    - préparer les objets à la réception des requêtes
    - informer l'ORB quand ces requêtes sont exécutées
  - définit l'interface entre le serveur et l'ORB
  - responsable de la sauvegarde de l'état des objets quand ils peuvent être désactivés entre 2 invocations
- **Responsable de :**
  - enregistrement des implantations
  - activation/désactivation des serveurs et implantations d'objets
  - sécurité des transactions
  - génération et interprétation des références aux objets
  - invocation des méthodes
- **Se base sur l'ORB et les CorbaServices**

L'ORB côté serveur : Object Adapter (2)

- **Différents types**
  - **BOA** (Basic Object Adapter) : suppose que le serveur est une application séparée de l'application cliente
  - **LOA** (Library Object Adapter) : suppose que le serveur est une DLL qui sera chargée dynamiquement dans l'application cliente
  - **OODA** (Object-Oriented Database Adapter) : suppose que le serveur est une base de données orientée objet
  - **POA** (portable OA) : l'implementation de OA est forunis par le langage de programmation cible.
- **possible de concevoir des Object Adapters spécifiques de besoins particuliers (le PAO on est un)**
- **CORBA ne spécifie que le BOA (sous java on peu utiliser le PAO)**

L'ORB : les états du PAO

- **Le POA racine récupéré par**
  orb.resolve initial reference("RootPOA")
- **est dans l'état suspendu aussi atteint par appel à old_requests() de son manager récupéré par la méthode the_POAmanager()**
- **Passage dans l'´etat actif via la méthode activate() de son manager**
  - L'état rejet est atteint par l'appel à discard_request()
  - L'état inactif est atteint par l'appel à desactivate()

L'ORB : *Interoperable Object Reference*

- **IOR contient toutes les informations nécessaires (encodées en *Common Data Representation*) pour retrouver un objet CORBA**
- **Pour l'implantation au dessus d'IP il contient :**
  - identificateur du type de l'objet
  - numéro IP du serveur
  - port du serveur
  - identificateur de l'objet
- **Format décrit en IDL**
- **IOR d'un objet obtenu par**

```
orb.object_to_string()
```

L'ORB : *Services de nommage*

- **Pour utiliser un objet ii faut obtenir IOR d'un objet ce qui est difficilement gérable**
- **L'idée est de mettre en place un services d'annuaire qui permet d'obtenir un objet distant à partir d'un nom symbolique (comme pour RMI).**
- **Accessible via la référence initiale de nom NameService**
- **Machine et port du service précisés par les propriétés ORBInitialHost et ORBInitialPort**
- **Service démarré par la commande orbd ou tnameserv**
- **Possibilité de préciser un port avec l'option -ORBInitialPort sinon 900**
- **Gère l'association entre des références d'objets CORBA et une hiérarchie de noms**

L'ORB : *Services de nommage*

- **Permet l'utilisation d'une hiérarchie de services de nommage Objet CORBA dont l'interface est d´efinie dans le module CosNaming**
- **Implante l'interface NamingContextExt qui hérite NamingContext**

L'ORB : *services de Nommage : le Contexte*

```
Properties props = new Properties();
props.put("org.omg.CORBA.ORBInitialHost",
  "localhost");
props.put("org.omg.CORBA.ORBInitialPort",
  "1234");
ORB orb = ORB.init(args,props);
org.omg.CORBA.Object o = orb.resolve initial
  references("NameService");
NamingContextExt context =
  NamingContextExtHelper.narrow(o);
```

L'ORB : *services de Nommage : les liaisons*

- **Chaque association entre un nom et un objet CORBA dans la hiérarchie de nommage est appelée une liaison**
- **Deux sortes de liaisons :**
  - les liaisons avec un autre contexte (NamingContext),
  - noeuds internes
  - les liaisons avec d'autres types d'objets CORBA, feuilles
- **Type IDL des noms**

```
typedef string Istring;
struct NameComponent {
Istring id;
Istring kind;
}; typedef sequence<NameComponent> Name;
```

- **Construction du nom à partir d'une chaîne de caractères où / est utilisé comme séparateur : context.to_name() Opération inverse context.to_string()**

L'ORB : *services de Nommage : les liaisons*

- **Quatre opérations :**
  - bind() crée une association si elle n'existe pas déjà
  - rebind() crée une association en effaçant la précédente
  - unbind() supprime l'association précédente
  - resolve(....) récupére la référence de l'objet associée au nom passé en argument
  - resolve_str() permet de récupérer la référence directement `a partir de la chaîne de caractères

L'ORB : *services de Nommage : les liaisons*

- **Coté serveur :**

```
org.omg.CORBA.Object o = ...
NameComponent[] name = context.to_name(str);
try {
   context.bind(name,o);
} catch (AlreadyBound e) {
System.err.println(context.to string(ame)
+ " was already bound !");
context.rebind(name,o);
}
```

- **Coté client :**

```
NameComponent[] name = context.to_name(str);
org.omg.CORBA.Object o =context.resolve(name);
```

ou

```
org.omg.CORBA.Object o = context.resolve str(str);
```

L'ORB : *services de Nommage : l'exemple coté serveur*

```
ORB orb = ORB.init(args, props);
HelloServant servant = new HelloServant();
POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
byte[] id = rootPOA.activate_object(servant);
org.omg.CORBA.Object ref = rootPOA.id_to_reference(id);
String ior = orb.object_to_string(ref);
System.out.println(ior);
rootPOA.the_POAManager().activate();
org.omg.CORBA.Object o = orb.resolve_initial_references("NameService");
NamingContextExt context = NamingContextExtHelper.narrow(o);
NamingContext newContext = context.new_context();
NameComponent[] ctxName = context.to_name("TEST1");
try {
  context.bind_context(ctxName,newContext);
} catch(AlreadyBound e) {
newContext = NamingContextHelper.narrow(context.resolve(ctxName));
}
newContext.rebind(context.to_name("B"),ref);
System.out.println("Server running !");
orb.run();
```

L'ORB : *services de Nommage : l'exemple coté client*

```
ORB orb = ORB.init(args, props);
try{
org.omg.CORBA.Object o = orb.resolve_initial_references("NameService");
NamingContextExt context = NamingContextExtHelper.narrow(o);
NameComponent[] name = context.to_name("TEST1");
NamingContext newContext = (NamingContext) context.resolve(name);
Hello h =(Hello) newContext.resolve(context.to_name("B"));
System.out.println(h.hello("World"));
```

CorbaServices (1)

- **Spécification des services de base d'une architecture informatique distribuée**
  - entièrement défini par des interfaces en IDL
  - utilisés par l'ORB, les objets applicatifs, par les applications
- **Services de la spécification CorbaServices**
  - Naming et Trader Services : nommage et annuaire d'objets
  - Event Services : notifications entre objets
  - Lifecyle et Relationship Services : cycles de vie, relations entre objets
  - Persistent Objet et Externalization Services : stockage et archivage d'objets
  - Transaction et Concurrency Services : transactions et parallélisme entre objets
  - Property et Query Services : administration des objets
  - Security et Licensing Services : sécurité et authentification des objets

CorbaServices (2)

- **Désignation d'objet**
  - établit la correspondance entre le nom logique d'un objet et sa référence interne
  - conçu à partir de services pré-existants (DCE CDS, ISO X.500, Sun NIS+)
- **Permanence**
  - assure la permanence de l'objet au-delà de l'existence du processus qui l'a créé
  - l'état d'un objet est sauvegardé en mémoire permanente (disque) et restitué à la demande
- **Cycle de vie**
  - offre toutes les fonction de gestion d'un objet
    - création
    - destruction
    - déplacement
    - duplication
    - etc.

CorbaServices (3)

- **Transactionnel**
  - assurer la cohérence et l'intégrité d'une transaction entre objets
  - propriétés ACID
    - Atomicity : la transaction est un tout indivisible
    - Consistency : la transaction, une fois effectuée, laisse les données dans un état cohérent
    - Isolation : si 2 transactions ont lieu en parallèle, chacune doit être vue indépendamment de l'autre
    - Durability : les modifications des données effectuées par la transaction sont permanentes
- **Notification d'événements**
  - permet aux objets de s'inscrire en attente d'événements
  - quand l'événement correspondant se produit, les objets qui se sont enregistré sont prévenus

CorbaServices (4)

- **Sécurité**
  - contrôler l'accès à un objet par
    - identification du client
    - gestion d'une liste définissant les clients autorisés
- **Accès multiple**
  - gérer l'accès en parallèle à un ou plusieurs objets
- **Relationnel**
  - exprimer et gérer les associations entre objets
  - ex: objet container et objets contenus
- **Gestion des licences d'objets**
  - contrôler l'utilisation des objets

Compiler et exécuter l'application

- **Compiler l'application client `HelloClient.java`**

```
$ javac HelloClient.java HelloApp\*.java
```

- **Compiler le serveur `HelloServer.java`**

```
$ javac HelloServer.java HelloApp\*.java
```

- **Lancer l'application client-serveur**

1. lancer le serveur de noms

```
$ tnameserv -ORBInitialPort nameserverport
```

`nameserverport` : n° de port à utiliser pour le serveur de noms (900 par défaut)

3. lancer le serveur Hello à partir d'une deuxième console

```
$ java HelloServer -ORBInitialHost nameserverhost
                   -ORBInitialPort nameserverport
```

`nameserverhost` : machine sur laquelle s'exécute le serveur de noms (facultatif si même machine que serveur d'application)

`nameserverport` : n° de port utilisé par le serveur de noms (facultatif si port par défaut)

6. lancer l'application client à partir d'une troisième console

```
$ java HelloClient -ORBInitialHost nameserverhost
                   -ORBInitialPort nameserverport
```

9. le client affiche la chaîne de caractères envoyée par le serveur:

```
Hello world!!
```

- **Arrêter les processus `tnameserv` et `HelloServer`**

Corba 2.0 : l'interopérabilité entre agents

- **Objectif**
  - assurer l'interopérabilité entre ORBs développés par différentes sociétés
  - proposition d'une norme d'interopérabilité
- **Spécifications**
  - une architecture générale d'interopérabilité entre ORB distincts
  - des passerelles entre ORB distincts
  - des protocoles de communication entre ORB
    - GIOP (General Inter-ORB Protocol) / IIOP (Internet Inter-ORB Protocol)
    - ESIOP (Environment Specific Inter-ORB Protocol) / DCE ESIOP

Corba 2.0 : l'interopérabilité entre agents

- **L'architecture d'interopérabilité**
  - canevas général définissant les modes de communication entre ORB dans un système distribué complexe ou hétérogène
  - définit les conventions et termes pour assurer l'interopérabilité efficace entre ORB
  - décrit deux façons d'unifier deux ORB
    - en imposant d'utiliser le même protocole de communication (IIOP)
    - en construisant des passerelles entre les protocoles utilisés par chacun des ORB
    - compromis à trouver entre simplicité (IIOP) et flexibilité (passerelles)

Corba 2.0 : GIOP / IIOP

- **GIOP (General Inter-ORB Protocol)**
  - spécifie
    - une syntaxe standard de transfert de données
    - un format de message pour la communication entre ORB
  - est indépendant du protocole réseau effectivement choisi pour la communication
- **IIOP (Internet Inter-ORB Protocol)**
  - obligatoire pour la conformité à la version 2.0 de la norme
  - spécifie comment les messages GIOP sont échangés sur un réseau TCP/IP
- **GIOP/IIOP**
  - relation similaire à celle qui existe entre l'IDL et un langage de programmation
    - GIOP = abstrait, s'appuie sur n'importe quel protocole réseau
    - IIOP = implantation concrète pour TCP/IP
- **ESIOP**
  - compatibilité avec d'autres standards de communication (DCE)

Les domaines Corba 2.0

- **Constat**
  - même dans des systèmes hétérogènes, il y a des « îlots d'homogénéité » (technologique, structurelle, administrative)
- **Objectif = assurer la connexion entre domaines technologiquement distincts**
  - passage de données et de messages entre un protocole réseau et un autre
  - échange entre deux domaines utilisant le même protocole mais des politiques de communication différentes (réseau local TCP/IP et Internet, séparés par un firewall)
- **2 solutions**
  - passerelles complètes
  - demi-passerelles

Corba 2.0 : passerelles complètes

- **Immediate bridging**
  - relie directement 2 ORB aux caractéristiques techniques différentes
  - traduit les messages du 1er ORB dans le format par les protocoles de communication du 2ème et réciproquement
- **Caractéristiques**
  - autant de passerelles que de paires de domaines
  - rapide et efficace
  - complexe à mettre en oeuvre si beaucoup de domaines

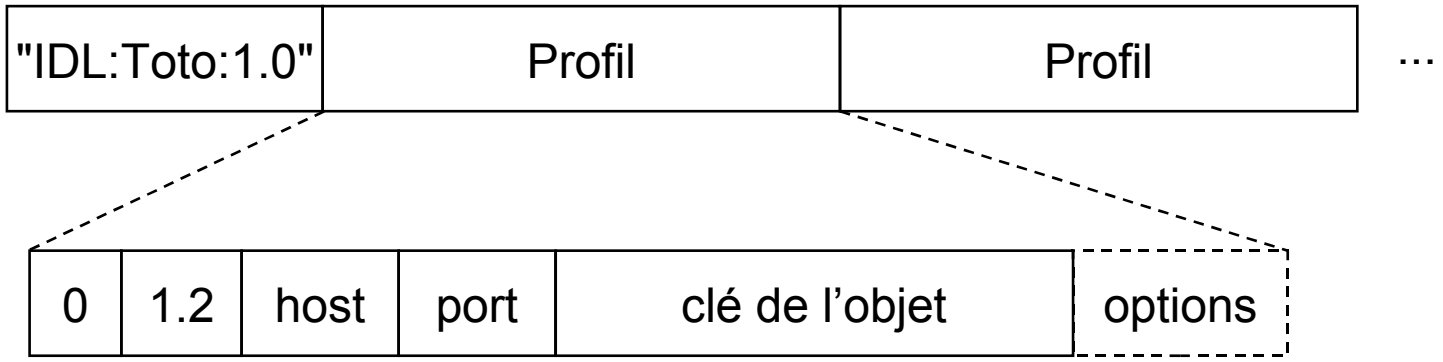
Corba 2.0 : demi-passerelles

- **Mediated bridging**
  - tous les domaines ont une demi-passerelle entre leur protocole de communication et un protocole de communication commun
  - quand un message part du domaine, il est
    - transformé par la demi-passerelle au format commun
    - acheminé vers la demi-passerelle destinataire
    - transformé vers le format du domaine destinataire
- **Caractéristiques**
  - nombre limité de demi-passerelles (une par type de domaine)
  - permet de relier les domaines entre eux plusieurs à plusieurs
  - IIOP constitue le protocole commun
  - 2 traductions au lieu d'une (sauf si l'ORB utilise déjà IIOP pour ses échanges de messages)

Corba 2.0 : IOR (Interoperable Object Reference)

- **Principe**
  - transfert de références entre ORB différents
  - forme que prend une référence lorsqu'elle est transmise comme paramètre ou comme résultat d'une opération
  - contient les informations pour traduire un message d'un ORB à l'autre
    - type de l'objet
    - liste des protocoles que l'ORB ou la passerelle accepte
    - disponibilité des CorbaServices
  - est constitué
    - d'un identifiant
      - le type de l'objet
      - le numéro de version de l'objet
    - d'un ou de plusieurs *profils annotés*
      - sont particuliers à un protocole de transport
      - détiennent les détails sur la localisation de l'objet
    - plusieurs profils
      - pour supporter plusieurs protocoles de transport (IIOP, DCE-ESIOP, autres)
      - pour signaler qu'un objet est disponible sur plusieurs serveurs

Corba 2.0 : IOR / IIOP (Internet IO Protocol)



- le tag : 0 signifie qu'il s'agit d'un profil IIOP
- la version : version de IIOP supportée
- host et numéro de port : adresse de la machine et numéro du port d'écoute du serveur qui contient l'objet
- clé : identification de l'objet : unique dans l'espace d'identification du serveur
- options : par ex., options de sécurité

Corba 2.0 : GIOP / IIOP

- **Spécifications**
  - un format commun de représentation des données (Common Data Representation – CDR)
    - types de base de l'IDL
    - types composés
    - prend en compte les différences d'alignement et d'ordre des octets entre les machines
  - les formats des messages GIOP
    - 7 types de messages différents envoyés
      - 3 par le client (request, cancel-request, locate-request)
      - 3 envoyés par le serveur (reply, locate-reply, close-connection)
      - 1 par le client et par le serveur (message-error)
  - les mécanismes de transport sur lesquels GIOP peut s'appuyer

Corba 2.0 : les messages GIOP (1)

- **`request`**
  - requête habituelle (objet destinataire, opérations, liste de paramètres qualifiés, en-tête avec le n° de requête)
- **`reply`**
  - en-tête avec le n° de la requête dont il est la réponse
  - code de résultat
    - NO_EXCEPTION : exécution correcte de la requête; le reste du message contient le résultat de l'opération et les valeurs des paramètres
    - USER_EXCEPTION ou SYSTEM_EXCEPTION : échec de la requête, le reste du message contient l'exception correspondante
    - LOCATION_FORWARD : réémission de la requête vers la bonne destination ; le reste du message contient l'IOR de la nouvelle localisation de l'objet
- **`cancel-request`** : annuler une requête déjà envoyée

Corba 2.0 : les messages GIOP (2)

- **`locate-request`** : demander à l'ORB distant s'il est responsable de l'objet destinataire de la requête ou non ; récupérer l'information équivalente à un LOCATION_FORWARD
- **`locate-reply`** : réponse au message **`locate-request`** avec les informations pour re-router les requêtes
- **`close-connection`** : envoyé par les serveurs pour informer les ORB de la fermeture de la connexion
- **`message-error`** : signaler une erreur d'interprétation des messages GIOP

Références

- **Ce cours est basé sur le cours de Gilles Roussel disponible en ligne sur l'adresse**
  http://igm.univ-mlv.fr/~roussel/CORBA/
  - Recommendation : pour des aspects avancés de corba consulter les cours 5 et 6 de l'adresse spécifié.
- **Merci également à Guillaume Hutzler qui m' a fournit son support qui a servi de base a ce cours**