6

Appel de méthodes à distance CORBA

Objets répartis : Motivations

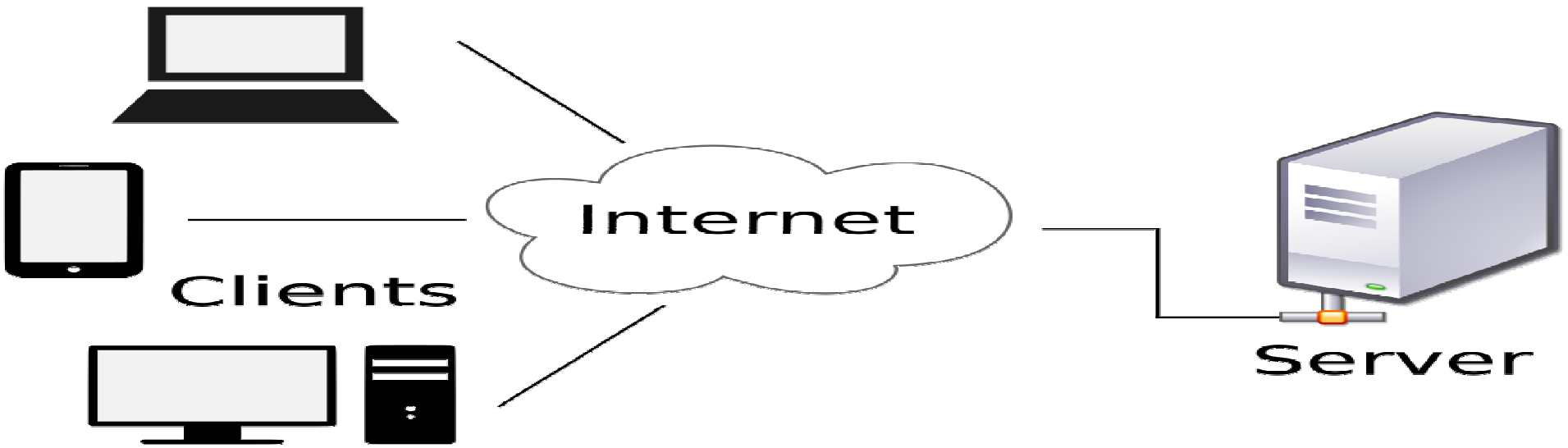
- **Objet : unité de désignation et de distribution**
  - **Objets "langage"**
    - ✓ représentation propre au langage : instance d'une classe
    - ✓ exemple : Java RMI (Remote Method Invocation)
  - **Objets "système"**
    - ✓ représentation "arbitraire" définie par l'environnement d'exécution
    - ✓ interopérabilité entre objets écrits dans des langages différents
    - ✓ exemple : CORBA

Objets répartis : Motivations

- **RPC + Orienté Objet** : tirer parti des bonnes propriétés de l'objet
  - Encapsulation
    - ✓ L'interface (méthodes + attributs) est la seule voie d'accès a l'état interne, non directement accessible
  - Classes et instances
    - ✓ Mécanismes de génération d'exemplaires conformes a un même modèle
  - Héritage
    - ✓ Mécanisme de spécialisation : facilite récupération et réutilisation de l'existant
  - Polymorphisme
    - ✓ Mises en œuvre diverses des fonctions d'une interface
    - ✓ Remplacement d'un objet par un autre si interfaces "compatibles"
    - ✓ Facilite l'évolution et l'adaptation des applications

Objets répartis : Motivations

- L'appel de procédure à distance permet d'exploiter une procédure distante.
- L'appel de méthode à distance permet d'exploiter un objet distant et d'invoquer une méthode sur cet objet
  - RPC + Orienté Objet
  - Exemple :
    - ✓ Java RMI
    - ✓ **CORBA**
      - **Intergiciel indépendant du langage de programmation**
    - ✓ Microsoft DCOM/COM+
    - ✓ SOAP
      - RMI on top of HTTP



CORBA®

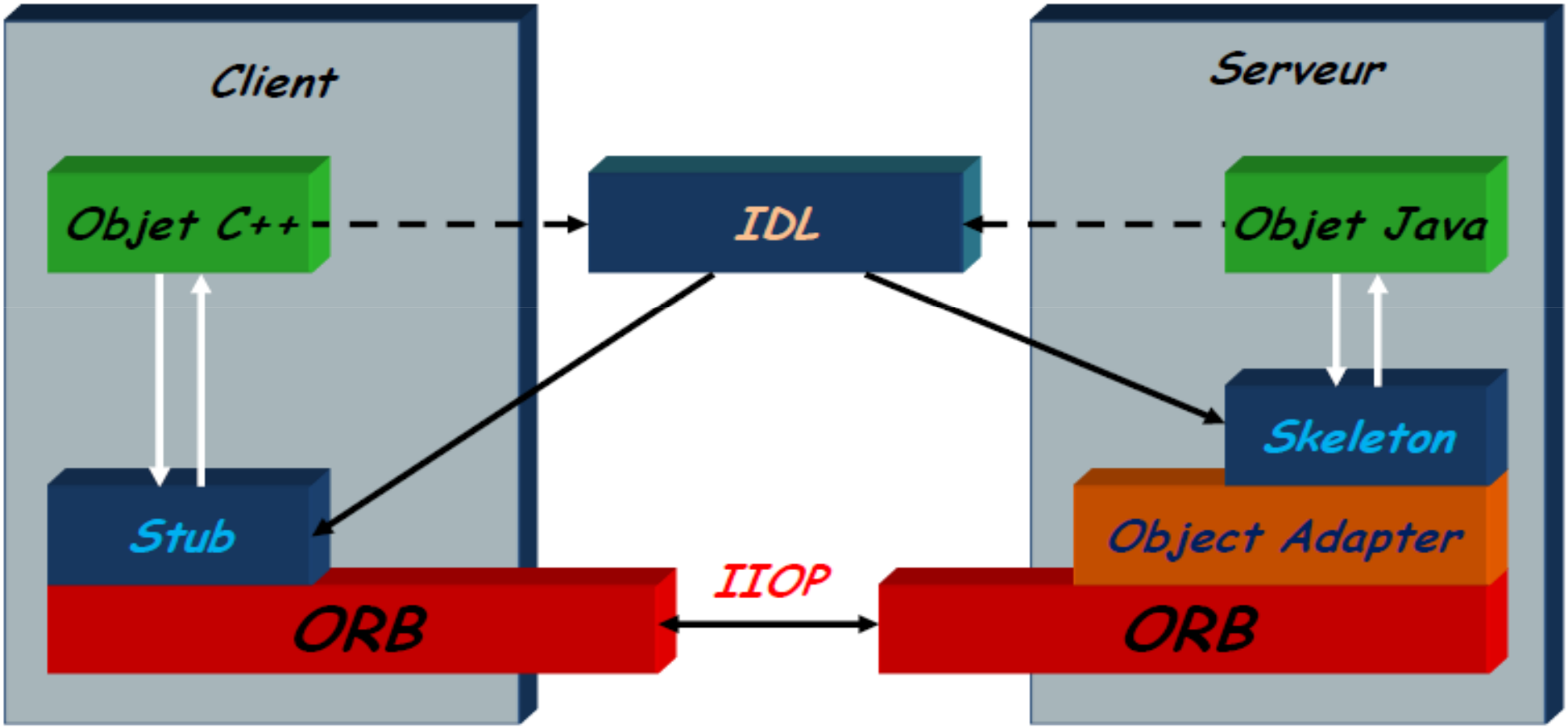
OMG®

OBJECT MANAGEMENT GROUP®

CORBA : Introduction

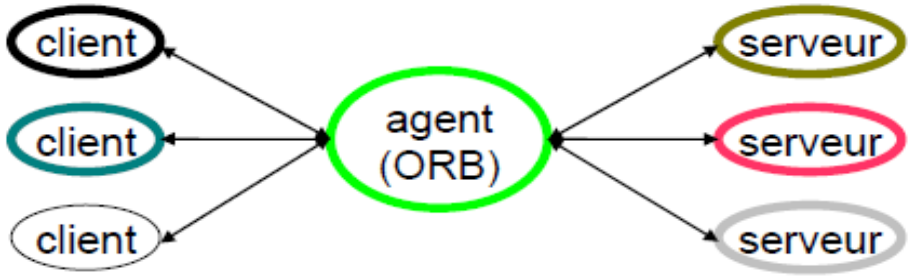
- Objectif
  - fournir un standard pour la gestion d'objets distribués.
- Une spécification proposée, en 1991, par l'Object Management Group, une organisation composée de plus de 800 Membres
- Architecture permettant de développer des applications distribuées :
  - standardisées
  - dans des environnements hétérogènes indépendant des langages de programmation et des systèmes d'exploitation;
  - orientées objet.

CORBA : Le minimum à connaître



CORBA : Le minimum à connaître

- **Introduire 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



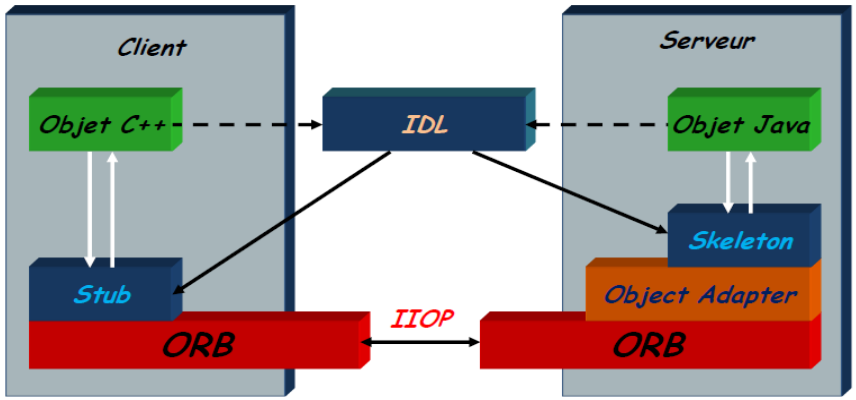
CORBA : caractéristiques

- **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 informations n'ont pas à figurer dans le code des clients et des serveurs
- **Communications synchrones ou asynchrones**
  - choix laissé aux développeurs
- **Client/serveur très dynamique**
  - une même portion de code peut apparaître
    - ✓ comme client dans un échange
    - ✓ comme serveur dans un autre échange

CORBA : les composants

- **ORB (Object Request Broker)** : noyau de CORBA
  - Bus logiciel de communication (passage de message).
- **IIOP:** Internet Inter-ORB Protocol
- **Object Adapter :** enregistrement des objets
- **Stub** : partie cliente
- **Skeleton** : partie serveur

générés automatiquement



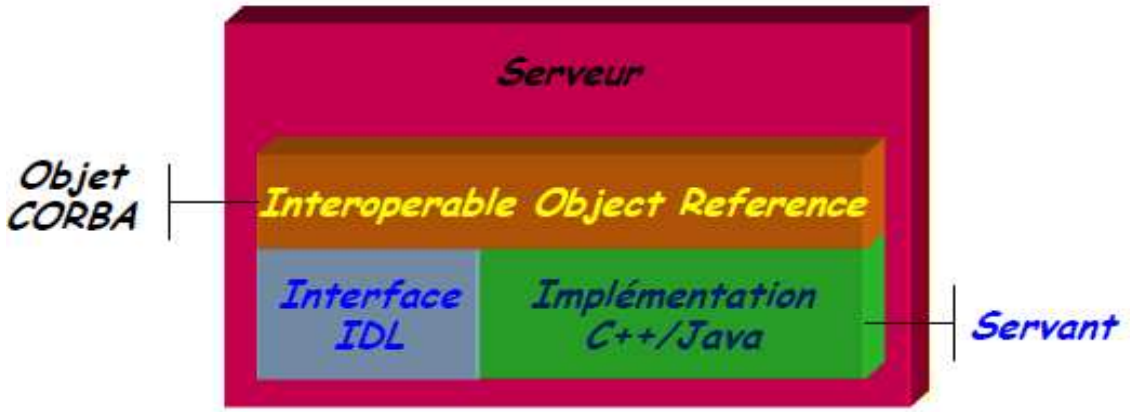
CORBA: L'objet CORBA

Définition

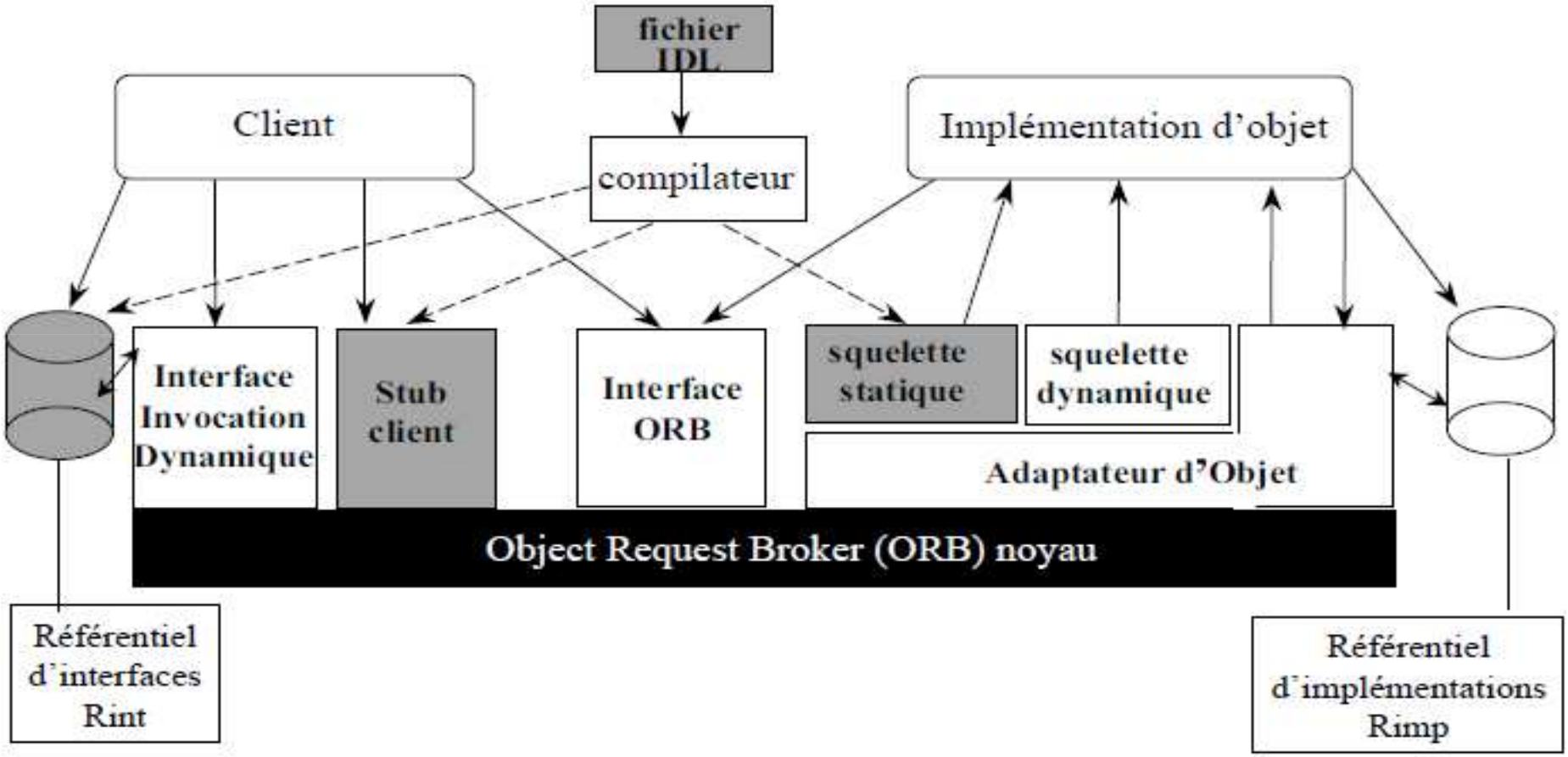
- entité logicielle désignée par une référence et recevant les requêtes émises par les applications clientes

Un objet CORBA comporte

- une référence ou IOR, localisant l'objet sur le réseau
- une interface IDL
- une implémentation (le servant)

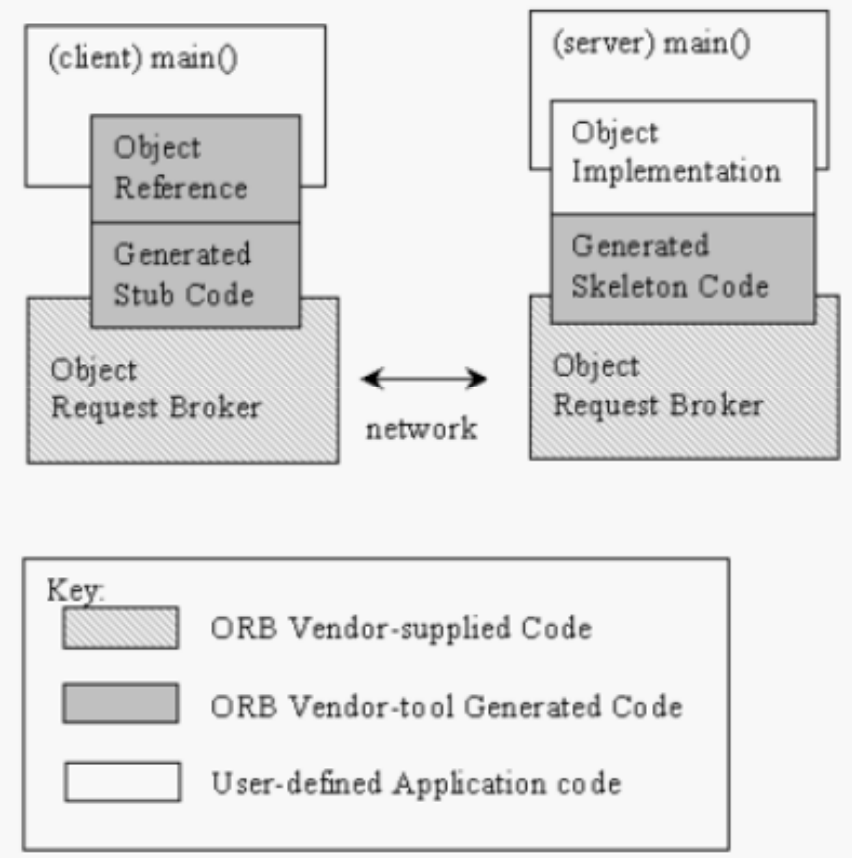


CORBA : Architecture générale



CORBA : l'ORB

- assure la localisation du serveur qui héberge l'objet,
- assure l'activation du serveur (si besoin),
- attribue la partie serveur de la référence d'objet,
- assure l'acheminement de la requête vers le serveur, puis vers l'adaptateur d'objets.



CORBA : Quelques ORBs

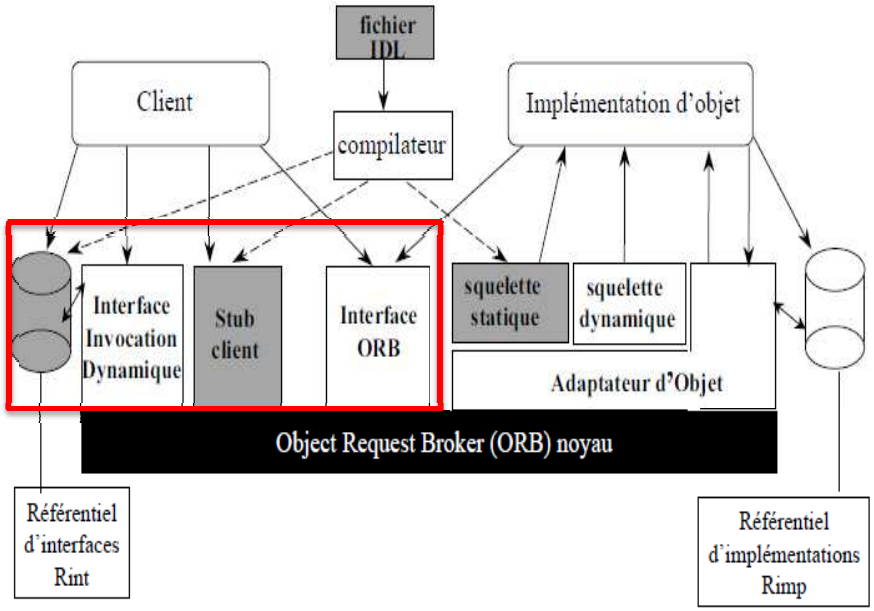
- **Logiciel libre**
  - MICO : Université de Francfort
  - TAO : Université de Washington
  - OmniORB : AT&T
  - Jonathan : ObjectWeb group (né à FranceTelecom R&D)
  - ORBit : RHAD Labs
- **Commerciaux**
  - Orbix : IONA
  - VisiBroker : Inprise
  - M3 : BEA
  - ORBacus : Object Oriented Concepts

L’ORB (coté client)

- Les stubs fournissent les interfaces statiques des services objet du serveur. Pour chaque objet distant, le client doit posséder le stub correspondant à l’objet. Grâce à ce stub, l’encodage (et le décodage) de l’appel d’une méthode (avec ses paramètres) pourra être réalisé et envoyé au serveur

- Les appels dynamiques se font grâce aux interfaces à appel dynamique (DDI), qui permettent donc de découvrir la méthode à exécuter au moment de l’exécution

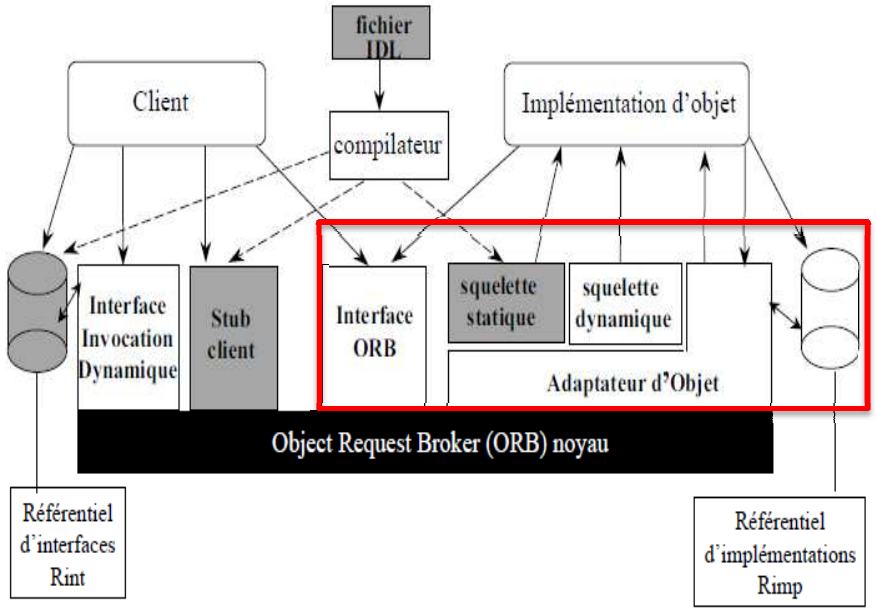
- Le Référentiel d’interfaces est une «base de données» qui contient toutes les versions exécutables (versions binaires) des interfaces définies en IDL. Des API de CORBA permettent d’accéder à ces métadonnées (données décrivant un composant)



- Les interfaces de l’ORB sont des API permettant d’accéder à certains services de l’ORB (transformation d’une référence d’objet en son nom en clair par exemple)

L'ORB (coté serveur)

- Les squelettes (skeletons) du serveur ont le même rôle que les stubs: elles décrivent en IDL les services fournies par le serveur
- Les appels dynamiques se font via les interfaces de squelettes dynamiques (DSI) : elles permettent l'appel de méthodes pour des composants serveur qui ne possèdent pas de skeletons
- Le référentiel d'implémentation contient l'ensemble des classes supportées par le serveur, les objets en mémoire et leurs OIDs (Object IDentifier)
- Les interfaces de l'ORB ont le même rôle que chez le client



- L'adaptateur d'objet(BOA, POA, …) se situe au dessus du noyau de l'ORB : c'est lui qui va traiter les demandes d'appels de méthodes. Pour ce faire, il fournit un environnement d'exécution pour créer les instances des objets sur le serveur. L'adaptateur va également enregistrer les objets dans le Référentiel d'implémentations

CORBA : Protocole IIOP

- Communications inter-ORBs :
  - **GIOP** (General Inter-ORB Protocol)
    - ✓ spécification du protocole générique de transport de CORBA
    - ✓ représentation commune des données (CDR)
    - ✓ format des références d'objet (IOR)
    - ✓ transport des messages (transparence réseaux)
- **IIOP** (Internet Inter-ORB Protocol): ***GIOP on TCP/IP***
  - implantation de GIOP au dessus de TCP/IP
  - GIOP/IIOP permet l'interopérabilité entre des ORBs provenant de "vendeurs" différents

CORBA : Object Adapter

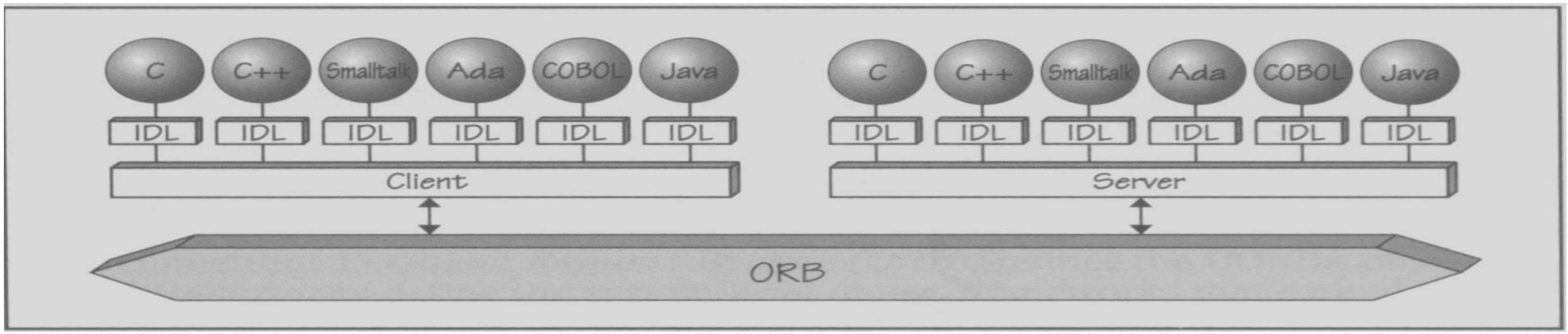
- Aiguillage des requêtes des clients
- Activation et désactivation des objets
  - Association entre objet CORBA et servant
  - Création et lancement de processus ou threads
  - Arrêt ou destruction des processus
- Génération des références d'objets (IORs)
- Gestion du référentiel des implémentations
- Authentification du client / contrôle d'accès

CORBA : Object Adapter

- **Différents types :**
  - **BOA ou Basic Object Adapter** : les structures d'accueil des implémentations d'objets CORBA sont matérialisées par des processus systèmes.
    - ✓ **Le BOA est devenu obsolète depuis CORBA 2.2 remplacé par le POA.**
  - **OODA ou Object-Oriented Database Adapter** : la structure d'accueil est une base de données orientée objet.
  - **LOA ou Library Object Adapter** : le code d'implémentation des objets est stocké dans des bibliothèques chargées dans l'espace des applications clientes.
  - **COA ou Card Object Adapter** : l'implémentation des objets est stockée dans une carte à microprocesseur.
  - **POA ou Portable Object Adapter** : l'implémentation des objets est réalisée par des objets fournis par un langage de programmation.

CORBA : IDL

- Description des interfaces
- Indépendant du langage d'implémentation
- Indépendant de la plateforme client
- Indépendant de la plateforme serveur
- Syntaxe proche du C++ (Java)



CORBA : Les étapes de développement d'un objet distant

1) Définir l'interface avec le langage IDL (Interface Definition Langage)

2) Générer les classes nécessaires à la distribution

3) Définir le code fonctionnel de l'objet distant : le servant

4) Distribuer l'objet au travers de l'ORB

4.1. Initialiser l'ORB

4.2. Enregistrer le servant de l'objet distant dans l'ORB

4.3. Rendre disponible une référence permettant de localiser l'objet distant

4.4. Mettre l'ORB en attente de requêtes

Le langage IDL

- Trois éléments hiérarchiques principaux
  - **Module** : espace de définition possédant un nom
    - ✓ Hiérarchie de modules (peuvent être imbriques)
  - **Interface** : regroupement de services
    - ✓ Définition de l'objet serveur accessible a distance
  - **Méthode** : définition des fonctionnalités du serveur
- Eléments Secondaires d'un fichier IDL
  - Types complexes (structures et unions), constantes, exceptions, attributs d'interface
- Exemple :

```
module banque { //module
   interface Compte {//interface
         public float solde(); //méthode} ;};
```

Le langage IDL

- Langage de définition des services proposés par un objet serveur CORBA
  - Définit les méthodes qu'un client peut invoquer sur le serveur
  - Définit les champs accessibles à distance du serveur (get/set)
- Génération des souches et squelettes à partir de l'interface IDL
  - **Génération pour plusieurs langages (Java, C++…, C…)**
    - ✓ Si le langage est non objet, la projection respecte une sémantique objet
  - Le client et le serveur ne sont pas forcement écrits dans le même langage!

Le langage IDL

- **Types simples normalises et projetés pour chaque langage/OS**
  - **Boolean** :
    - ✓ boolean (2 valeurs : TRUE et FALSE)
  - **Octet** :
    - ✓ Octet (1 octet, signé ou non signé, dépend du langage)
  - **Nombres signés** :
    - ✓ short (2 octets), long (4 octets), long long (8 octets)
  - **Nombres non signés** :
    - ✓ unsigned short (2 octets), unsigned long (4 octets), unsigned long long (8 octets)
    - ✓ (projeté vers l'équivalent signe en Java!)
  - **Nombres à virgule flottante** :
    - ✓ float (4 octets), double (8 octets), long double (16 octets, pas de projection vers Java!))
  - **Caractère et chaine de caractères** :
    - ✓ char (1 octet, ISO Latin1), string (chaine de char), string<n> (à taille fixe)
    - ✓ wchar (2 octets, unicode), wstring (chaine de wchar),

Le langage IDL

Types construits : structures, énumération, définition, tableaux

- **Structure** : regroupement d'attributs (struct C ou tuple SQL)
  - ✓ **struct *identificateur { liste d'attributs types; };***
  - ✓ **Exemple: struct Personne { string nom; long age; };**
- **Enumération** : liste de valeur pour un type (enum C)
  - ✓ **enum *identificateur { liste de valeurs, };***
  - ✓ **Exemple: enum couleur { rouge, vert, bleu };**
- **Définition de type** : nom symbolique (typedef C)
  - ✓ **typedef type *identificateur***
  - ✓ **Exemple: typedef string<16> tprenom;**
- **Tableau mutli-dimensionnel de taille fixe** :
  - ✓ type *identificateur* ***[taille]+;***
  - ✓ **Exemple:** long matrice**[32][16];**
- **Tableau uni-dimensionnel de taille quelconque**:
  - ✓ **sequence<type> identificateur; ou sequence<type, max> identificateur;**
  - ✓ **Exemple: sequence<long> vecteur; ou sequence<long, 16> vecteur;**
- **Union** : type variable en fonction d'un discriminant (de base ou énumère)
  - ✓ **union identificateur switch(enumIdentificateur | typeBase) {**
    case valeur : type identificateur; +
    default: type identificateur; [0,1]
    };
  - ✓ **Exemple:**
    union Foo switch(long) { case 0 : float toto; default: double tata; };
    union Bar switch(couleur) {
    case rouge: long r; case vert: boolean v; case bleu: long long b; };

Le langage IDL

- **Autres types primitifs :**
  - **Type Object Corba** : tout objet Corba (passage d'un objet par référence)
    - ✓ void f(in **Object o); ou struct st { Object x; };**
  - **Type indifférencié** : n'importe quel type (de base ou construit)
    - ✓ void f(in **any o); ou struct st { any x; };**
  - **Type vide** : spécifie qu'une méthode ne renvoie rien
    - ✓ **void f(in Foo f);**

Le langage IDL

- **Modules** : espaces de définition de symboles
  - Types, constantes, exceptions, modules, interface
  - Hiérarchie de modules pour structurer les applications
  - Operateur de résolution de portée : "::"
  - Visibilité respecte l'inclusion

- Exemple

```
module mod {
    typedef wstring<32> tadresse;
    module mod1 {
        typedef wstring<16> tnom;
    };
    module mod2 {
        struct pom {
            mod1::tnom nom; /* ou mod::mod1::tnom */
            tadresse adresse;
        };
    };
};
```

Le langage IDL

- **Interfaces** : points d'accès aux objets CORBA
  - Contiennent types, constantes, exceptions, attributs, méthodes
  - Peuvent hériter (multiple) d'autres interfaces (*interface X : Y, Z*)
  - Contenues dans un module
  - Autorise les prédéfinition (*interface X; interface Y {uses X}; interface X {uses Y})*

- Exemple

```
module pim {
struct Person { /*equivalent a la declaration
      d'interface Serializable en RMI*/
wstring name;
wstring adress;
};
interface Pom { /*équivalent a la déclaration
            d'interface Remote en RMI*/
Person find(in wstring name);
};
};
```

Le langage IDL

- **Méthodes** : des services proposés par un objet CORBA
  - Sont nommées par un identificateur
  - Peuvent recevoir des arguments et en renvoyer
  - Peuvent lever une (ou plusieurs) exceptions

    typeRetour identificateur([paramètres]*) [raises [exceptions]+];

    Paramètre = mode type identificateur

    Mode : **in** en entrée, **out** en sortie, **inout** en entrée et sortie

    Type : **tout type de base ou construit (tableau et union) avec un typedef**

    ~~*void f(in sequence<string> arg);*~~ ***interdit***

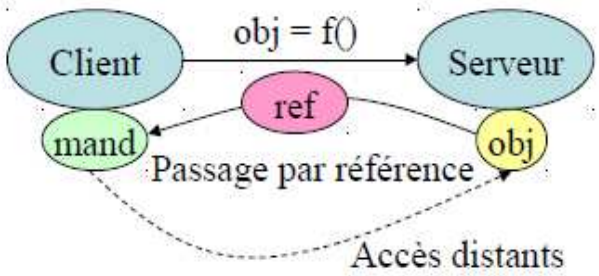
    typdef *sequence<string> tsa; void f(in tsa arg);* ***autorisé***

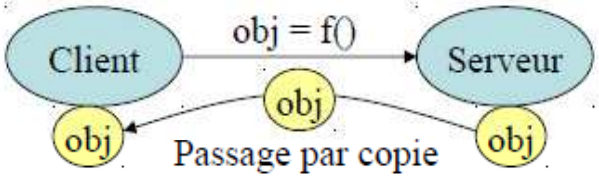
  - Surcharge interdite (pas deux méthodes ayant le même nom)

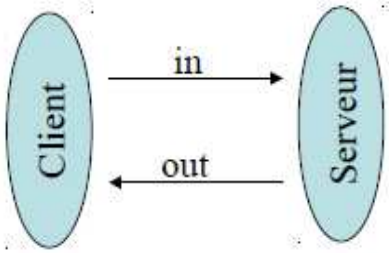
Le langage IDL

Passage par référence ou par valeur

- Par référence si
  - ✓ Object CORBA (interface et Objet)
- Passage par copie si
  - ✓ Types simples (float, long, double…)
  - ✓ Types construit (struct, sequence…)
- Paramètre in : le client fournit la valeur
  - ✓ Si le serveur la modifie, le client n'est pas mis a jour
- Paramètre inout : le client fournit la valeur
  - ✓ Si le serveur la modifie, le client est mis a jour
- Paramètre out : le serveur fournit la valeur
  - ✓ Le client est mis a jour

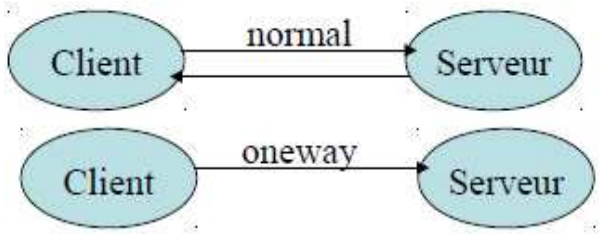






Le langage IDL

- **Méthode oneway** : pas d'attente du retour
  - Appel normal : attend le retour
    - ✓ Exécution de l'appel garanti
  - Appel oneway : pas d'attente
    - ✓ Exécution de l'appel non garanti
    - ✓ Utilisable uniquement pour méthodes
      - Sans paramètre de retour (void)
      - Sans paramètre d'entrée inout ou out




```
module banque {
interface Compte {
oneway void setTitulaire(in Person p);
        Person getTitulaire(); // oneway impossible car renvoie une structure Person
}; }; // dans cet exemple, Person est un struct => passe par copie
```

Le langage IDL

- **Constantes** : variable typées dont la valeur est fixe
  - **const type *identificateur* = *valeur;***
  - Exemple **: const long *taille* = *16*2; typedef wstring<taille> tnom;***
    - ✓ Uniquement type de base
    - ✓ Operateurs autorises : +, *, /, -, %, &, |, ^, <<, >>
- **Exception** : structure qui signale une situation exceptionnelle
  - **exception *Identificateur { [attributs types]* };***
  - Exemple : **exception *Overflow { double limite; };***

    interface Math { double exp(in double x) raises(Overflow); }
- **Attributs** : propriétés typées attachées aux interfaces
  - Propriétés variables, accessible avec des méthodes **getter/setter**
  - [readonly] **attribute type identificateur;**
  - Exemple : interface X { attribute float lim; readonly attribute float maxLim; };

Le langage IDL

- **Héritage d'interface**
  - Simple : interface X : Y { … };
  - Multiple : interface X : Y, Z { … };
- Une sous-interface hérite tous les éléments de ses super-interfaces
  - Peut ajouter de nouveaux éléments
  - Peut redéfinir les types, constantes, exceptions
  - Ne peut pas redéfinir les attributs et les opérations!
- **Graphe d'héritage**
  - Possibilité d'héritage en losange (B ← A, C ← A, D ← B,C)
  - Cycle dans le graphe d'héritage interdit (A ← B, B ← A **interdit**)

Développement JAVA

IDL--> JAVA : Types primitifs

IDL	*Java*	*IDL*	*Java*
octet	byte		
short	short	unsigned short	short
long	int	unsigned long	int
long long	long	unsigned long long	long
float	float	char, wchar	char
double	double	string, wstring	String
long double	pas de correspondance		

Développement JAVA

IDL--> JAVA : Structure

```
module mod {
struct Foo {
…
}; };
```

- Foo.java : une classe Java contenant les champs de la structure
  - ✓ Herite de IDLEntity
  - ✓ Deux constructeurs (vide et avec un paramètre par champs de la structure)
- FooHelper.java : classe de gestion des objets Foo (voir 44)
- FooHolder.java : classe utilitaire pour la gestion des out/inout (36-37)

```
module pim {
 struct Person {
  string nom;
  float age;
 };
};
```

Traduction →

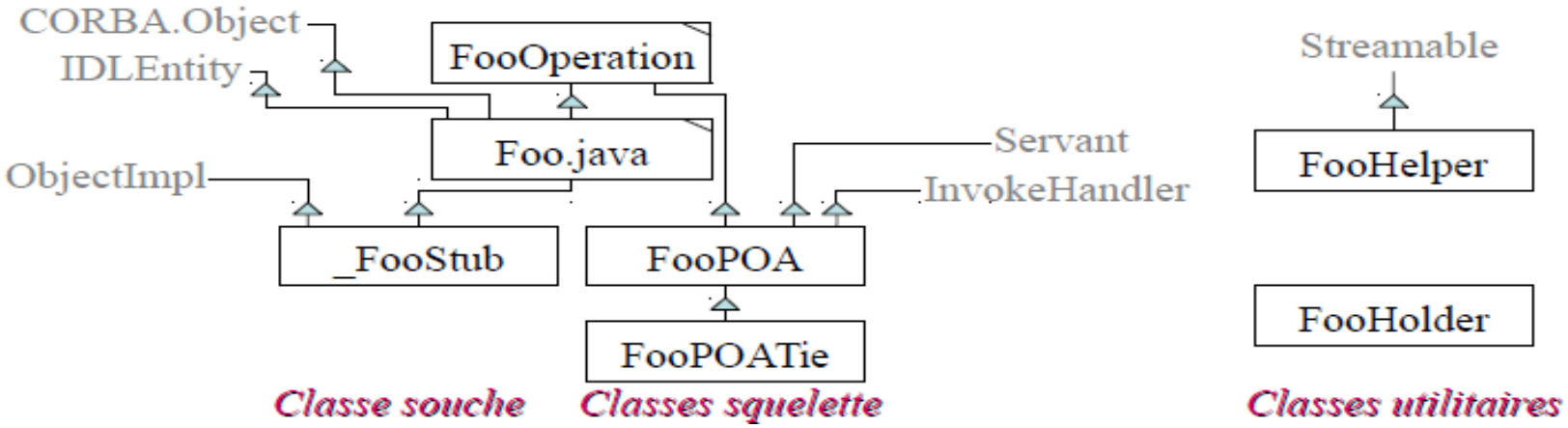
```
package pim;
public class Person implements IDLEntity{
 public String nom;
 public float age;

 Person() {}
 Person(String nom, float age) {
  this.nom = nom;  this.age = age;
 }
}
```

Développement JAVA

IDL--> JAVA : Interface

- FooOperation.java : interface Java traduisant l'interface IDL
- Foo.java : interface de **l'objet Corba** traduisant l'interface IDL
- _FooStub.java : souche cliente
- FooPOA.java : squelette du serveur
- FooHelper.java : classe de gestion des objets implantant Foo (voir 44)
- FooHolder.java : classe utilitaire pour la gestion des out/inout (voir 36-37)

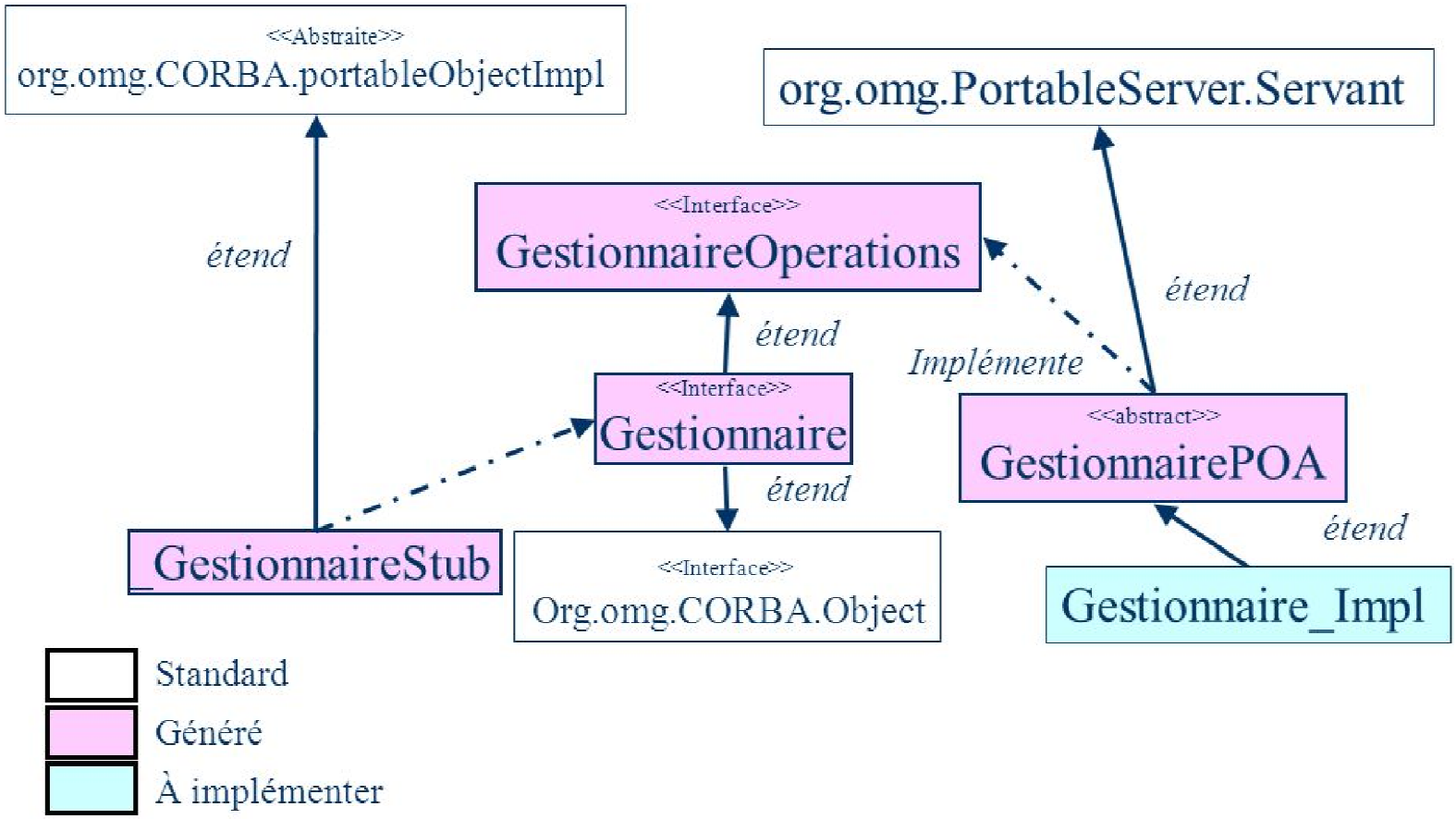
```
module mod {
interface Foo {
…
}; };
```



Développement JAVA

IDL--> JAVA : Interface

```
module mod {
interface Gestionnaire {
…
}; };
```



Développement JAVA

IDL--> JAVA : les paramètres des méthodes

- Problème : les paramètres out et inout doivent être modifiés chez le client

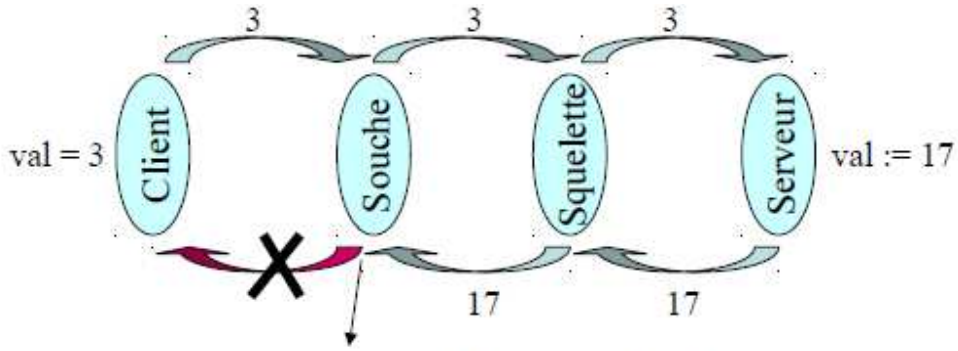
  interface X { void f(**inout** long val); }

  val doit être mis à jour chez l'appelant après l'appel de f

- Impossible à le faire de manière transparente

  Java : void caller(X x) { int val = 3; x.**f(val);** /* ici, val = 3 */ }

  val est copié lors de l'appel : sa valeur n'est donc pas modifiée



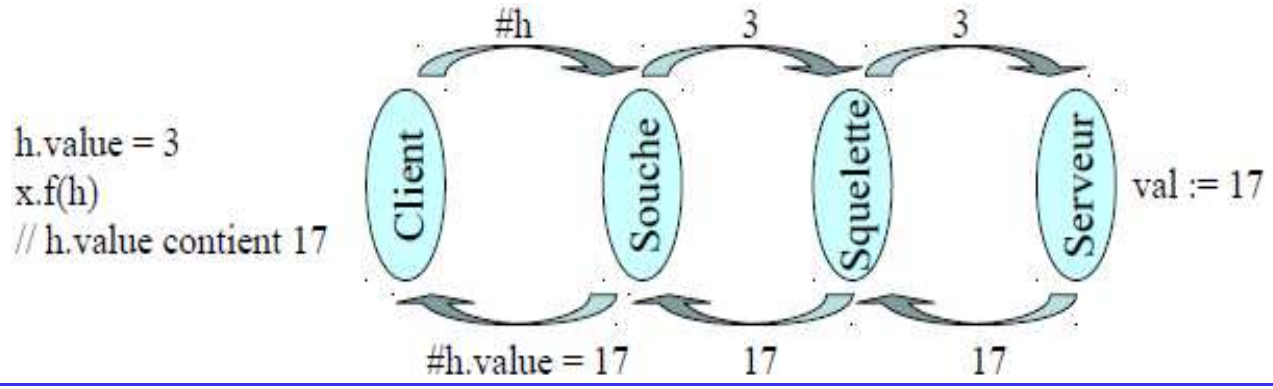
La souche ne connaît que la valeur 3, elle n'a pas de référence vers val
Et ne peut donc pas mettre la variable à jour

Développement JAVA

IDL--> JAVA : les paramètres des méthodes

- Utilisation de Holder (conteneurs)
  - ✓ Classe encapsulant une autre classe
    - Un champs publique du type contenu
    - Un constructeur avec un paramètre du type contenu
    - Une par type primitif (IntHolder, StringHolder…)
    - Une par type complexe générée lors de la traduction (struct, enum, interface, …)

```
void caller(X x) {
IntHolder h = new IntHolder(3);
x.f(h); System.out.println("valeur: " + h.value); // nouvelle valeur
}
```



Développement JAVA

IDL--> JAVA : les interfaces

- **Attribut IDL** --> méthode getter et **setter (si !readonly)**

```
interface X {
  attribute string nom;
  readonly attribute float solde;
};
```

Traduction →

```
public interface XOperation {
  public void nom(String n);
  public String nom();
  public float solde();
}
```

- **Méthode IDL** --> méthode Java
  - ✓ Paramètre in : utilisation directe du type
  - ✓ Paramètre out et inout : utilisation de Holder

```
interface X {
  float f(out long l, in Person p);
  void g(in long l, inout Person p);
};
```

Traduction →

```
public interface XOperation {
  public float f(IntHolder l, Person p);
  public void g(int l, PersonHolder p);
}
```

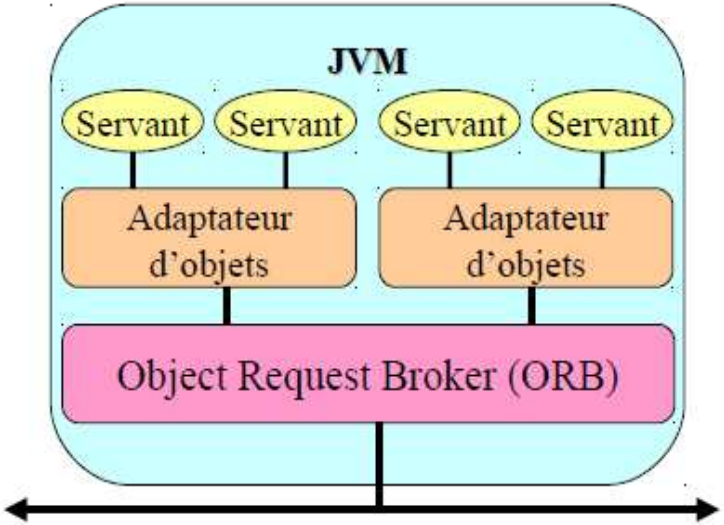
Développement JAVA

Trois entités fondamentales

- **Servant** : objet d'un langage de programmation implantant un service Corba
- **Adaptateur d'Objets (POA)** : entité chargée de gérer des servants (activation, transmission de requêtes, …)
- **Le bus à objet (ORB)**: entité chargée d'acheminer les requêtes entre machines

Ecriture du programme serveur :

1. Création d'une ou plusieurs instances de servant
2. Enregistrement des servants dans l'adaptateur d'objets



ORB, le bus à objets CORBA : connexion avec l'extérieur

- Trois méthodes fondamentales

```
class org.omg.CORBA.ORB {
static ORB init(String[] args, Properties props); // initialisation l'ORB (un par JVM)
void run(); // lancement de l'ORB
// aucun objet ne reçoit de requête avant l'appel a run()
org.omg.CORBA.Object resolve_initial_references(String name);
// trouve un objet initial a partir de son nom
}
```

- Les objets initiaux sont
  - ✓ Le RootPOA : le père de tout adaptateur d'objets
  - ✓ Eventuellement le service de résolution de noms

Développement JAVA

ORB, le bus à objets CORBA : exemple coté serveur

```
public static void main(String[] args) {
    try {

        ORB orb = ORB.init(args, null);
        org.omg.CORBA.Object rootobj=orb.resolve_initial_references("RootPOA");
        POA poa = POAHelper.narrow(rootobj);
        poa.the_POAManager().activate();

        Carnet_CommandesImpl cc = new Carnet_CommandesImpl(poa);

        org.omg.CORBA.Object od = poa.servant_to_reference(cc);

        Context ctx = new InitialContext();
        ctx.rebind("Carnet1", od);

        orb.run();

    } catch (InvalidName | AdapterInactive | ServantNotActive | WrongPolicy | NamingException ex) {
        Logger.getLogger(CORBA_Cmd_Server.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Développement JAVA

- **Le POA : gère un ensemble de servants**
  - Tout servant est associe à un POA
  - Existence d'un POA Racine (RootPOA) dans l'ORB
  - A chaque POA est associé un POAManager qui contrôle l'état du POA
    - ✓Activé, mode tampon, désactivé
    - ✓Activation d'un POA : poa.the_POAManager().activate();

Développement JAVA

Le POA : gère un ensemble de servants

```
public static void main(String[] args) {
    try {

        ORB orb = ORB.init(args, null);
        org.omg.CORBA.Object rootobj=orb.resolve_initial_references("RootPOA");
        POA poa = POAHelper.narrow(rootobj);
        poa.the_POAManager().activate();

        Carnet_CommandesImpl cc = new Carnet_CommandesImpl(poa);

        org.omg.CORBA.Object od = poa.servant_to_reference(cc);

        Context ctx = new InitialContext();
        ctx.rebind("Carnet1", od);

        orb.run();

    } catch (InvalidName | AdapterInactive | ServantNotActive | WrongPolicy | NamingException ex) {
        Logger.getLogger(CORBA_Cmd_Server.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

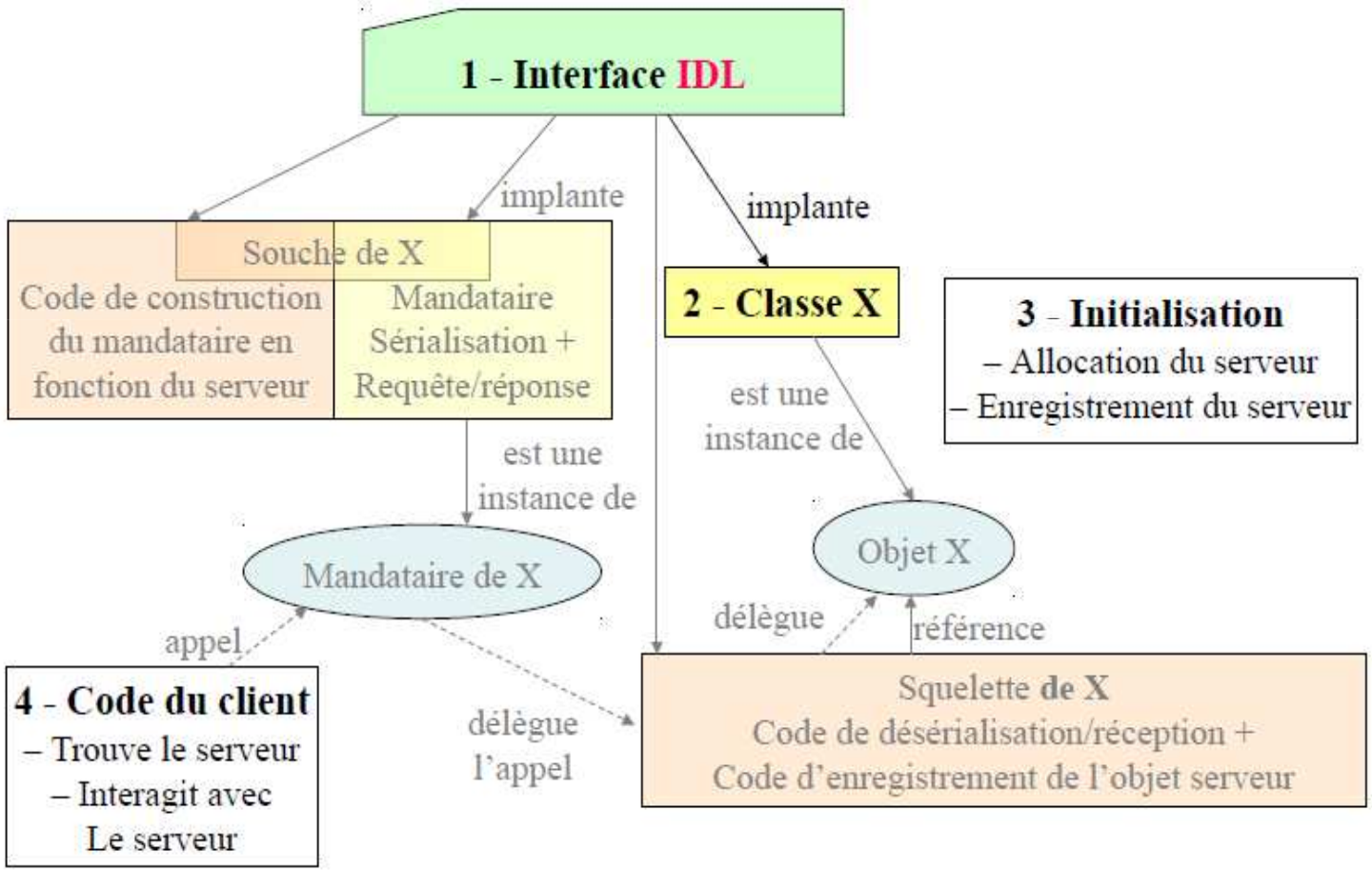
Développement JAVA

Objet CORBA, Servant et Mandataire

- Un servant est référencé par objet CORBA
  - ✓ Objet CORBA = référence distante (CORBA) vers un servant
  - ✓ Exp : org.omg.CORBA.Object rootref = orb.resolve_initial_references("RootPOA");
    - **Un Objet CORBA n'est pas un mandataire!**
    - Rootref est une référence vers le RootPOA
- Construction des mandataires à partir d'un objet CORBA
  - ✓ À toute interface CORBA est associée une classe *Helper*
    - CompteHelper, POAHelper, CosNamingHelper…
  - ✓ Methode narrow dans les classes *Helper pour la construction du mandataire*
  - ✓ Exp : static org.omg.PortableServer.POA narrow(org.omg.CORBA.Object)
    - ➔ convertit org.omg.CORBA.Object en un mandataire du POA
- **Attention : ne jamais essayer d'obtenir un mandataire a partir d'un objet CORBA en utilisant le cast Java**
  - ✓ Compte cpt = (Compte)obj; ➔ plantage

Développement JAVA

Objet CORBA, Servant et Mandataire



Développement JAVA

IOR : Interoperable Object References

- Identifiant unique représentant un objet Corba
  - ✓ Référence distante vers un objet Corba
  - ✓ Contenu dans un objet du type CORBA.Object
- Peut être représenté sous la forme d'une chaine de caractères
  - ✓ org.omg.CORBA.Object obj = …;
  - ✓ String s = orb.object_to_string(obj);
    - ➔s est une chaine qui contient l'IOR
- Une référence CORBA peut être construite à partir d'une chaine
  - ✓ String ior = …;
  - ✓ org.omg.CORBA.Object obj = orb.object_to_string(ior);
- Utile pour échanger une référence distante entre un serveur et un client

Développement JAVA

Écriture du servant : deux méthodes

Par héritage

public class **Impl1** extends **FooPOA** {
// implantation des méthodes de FooOperation
}

- ✓ Une instance de **Impl1** est directement un servant Corba (car hérite de Servant)
- ✓ **Impl1** ne peut pas hériter d'une autre classe Java (car héritage simple en Java)

Par délégation

public class **Impl2** implements **FooOperation** {
// implantation des méthodes de FooOperation
}

- ✓ Une instance de **Impl2** n'est pas un objet Servant, le Servant est construit par new FooPOATie(new Impl2());
- ✓ Utilisation possible de l'héritage simple Java

Développement JAVA

Écriture du serveur

1. Initialiser le bus CORBA : obtenir l'ORB
2. Initialiser l'adaptateur d'objets : obtenir le POA
3. Créer les objets distants
4. Enregistrer les objets distants par l'adaptateur
5. Diffuser leurs références (IOR)
   - ✓ Plusieurs méthodes (par le service des noms JNDI (Java Naming and Directory Interface, une interface unique pour utiliser différents services de nommages) ou par l'enregistrement du IOR dans un fichier à partager avec le client)
6. Attendre des requêtes venant du bus
7. Destruction du Bus