

# JAVA, CORBA et RMI

objectif :

développer des applications client/serveur incluant des objets répartis

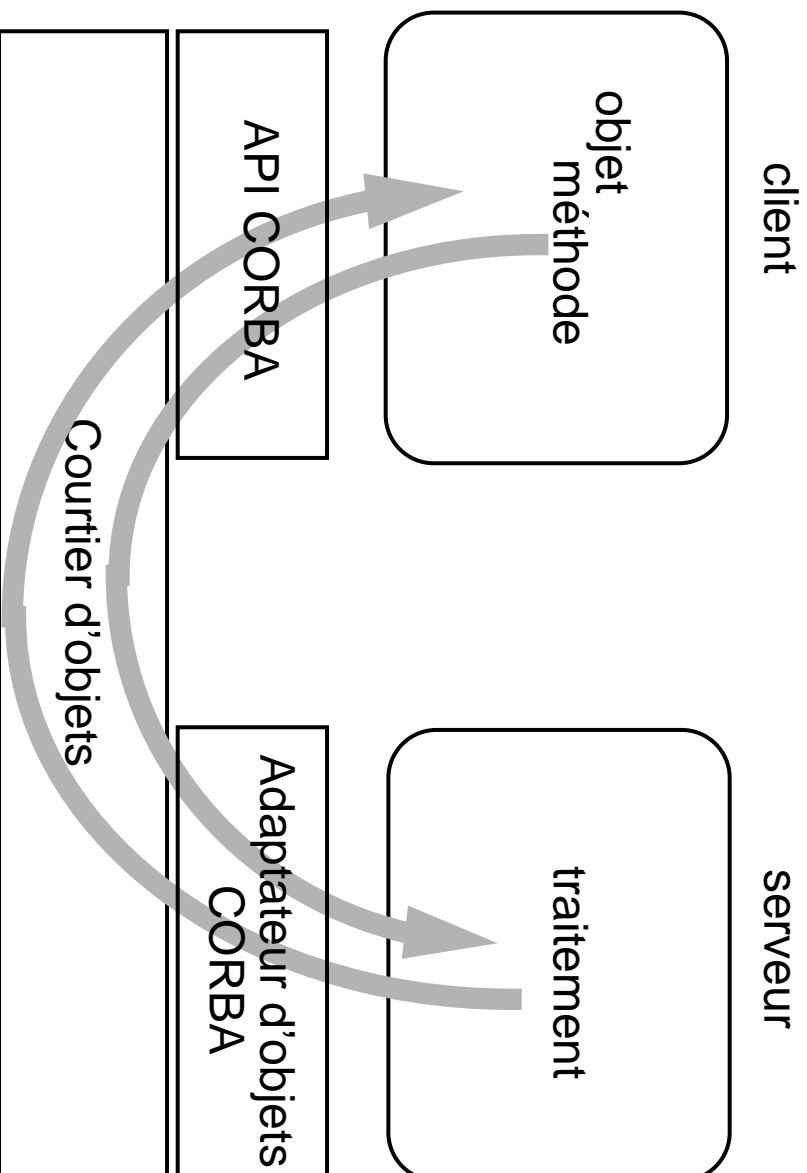
Java / CORBA : client/serveur hétérogènes

Java / RMI : client/serveur homogènes

plan :

- l'architecture CORBA
- le développement d'applications avec CORBA
- applets Java / CORBA
- applets Java /RMI

# CORBA : une introduction



# CORBA : une introduction

issu de l'OMG (Object Management Group)

(+ de 500 entreprises)

objectif :

créer une infrastructure à objets **ouverte**

CORBA (Common Object Request Broker Architecture)

- bus à objets
- architecture définit la forme des composants et leur mode d'interopérabilité (IDL)

# CORBA : une introduction

## IDL (Interface Definition Language)

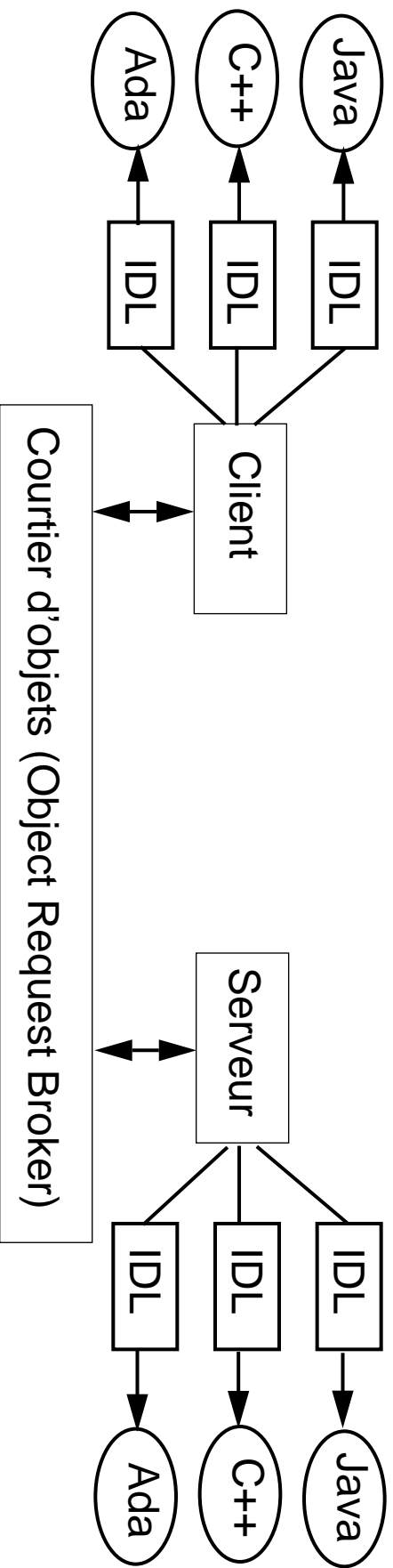
- langage neutre de spécification d'interface
- définit les interfaces contractuelles d'un composant avec ses clients potentiels.

## composants portables :

- entre langages,
- entre outils,
- entre systèmes d'exploitation,
- entre réseaux

# Objet réparti CORBA

- composant logiciel accessible par des clients éloignés via des appels de méthode.
- les clients connaissent l'interface de l'objet.
- son mode de fabrication est transparent pour ses clients potentiels
- la machine et son système d'exploitation ne sont pas connues des clients



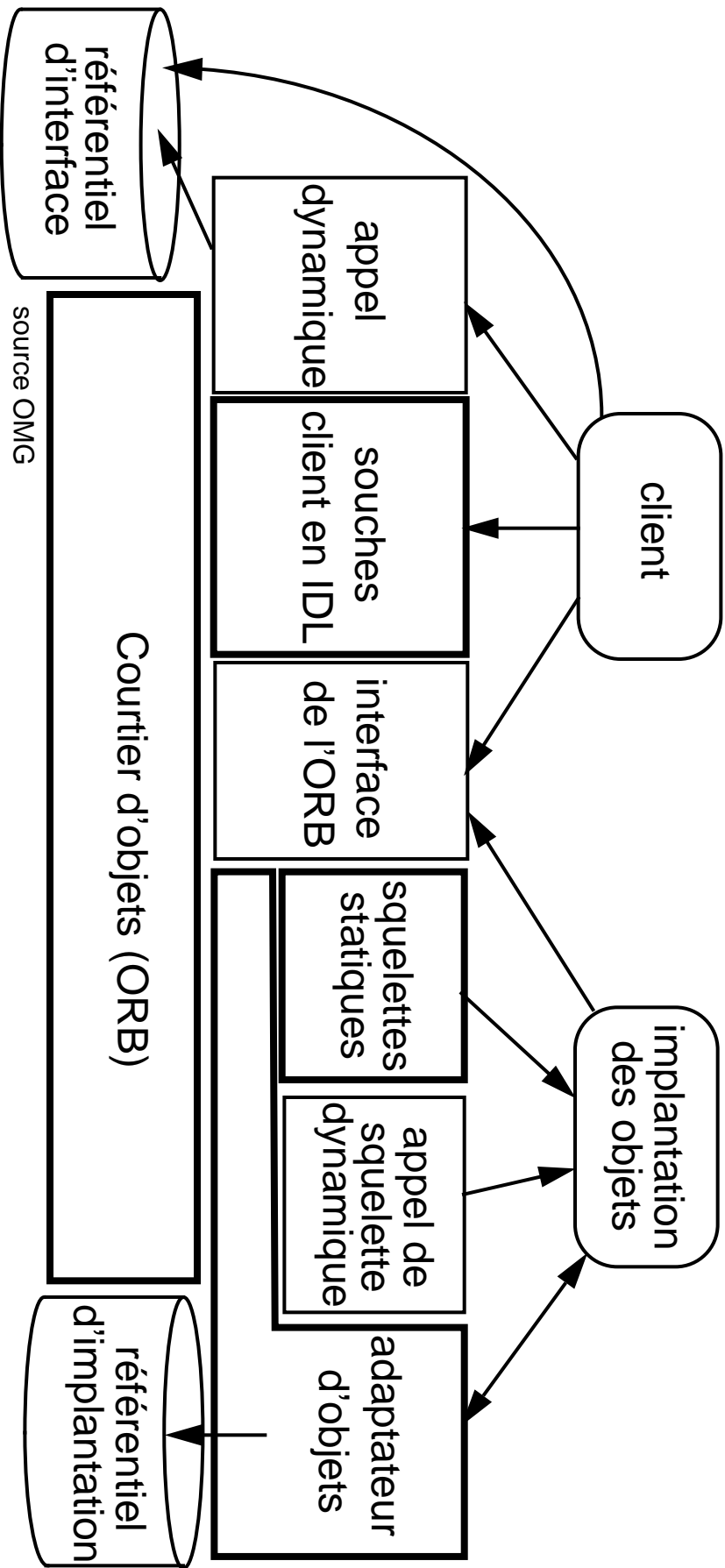
## **Interface Definition Language**

- définition contractuelle de l'interface d'un composant avec ses clients potentiels
- aucun détail d'implantation (langage, système d'exploitation)
- méthodes spécifiées en IDL peuvent être définies par tout langage fournissant des liens avec CORBA
- méthodes sont utilisées selon les constructions du langage (procédure, méthode)

# Exemple d'interface

```
module Gestion_de_compte {  
    interface comptedistant {  
        float depot(float amount);  
        float retrait(float amount);  
        float litSolde();  
    };  
}
```

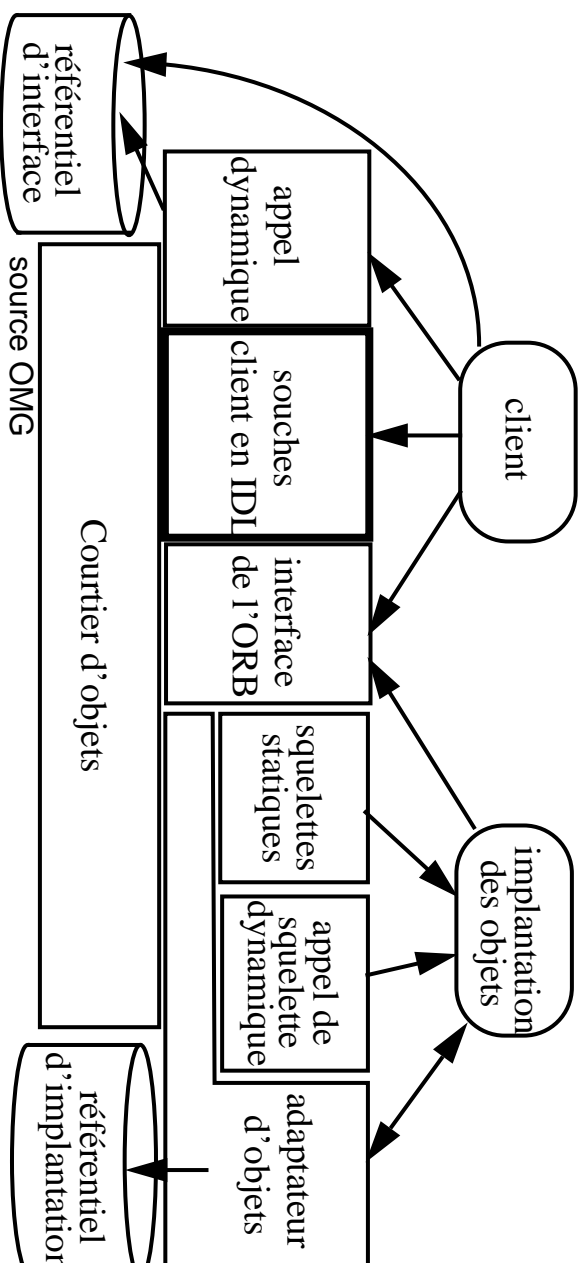
# Architecture d'un courtier CORBA



- l'ORB établit des relations client/serveur entre des objets.
- l'ORB a la responsabilité de trouver un objet implantant la requête

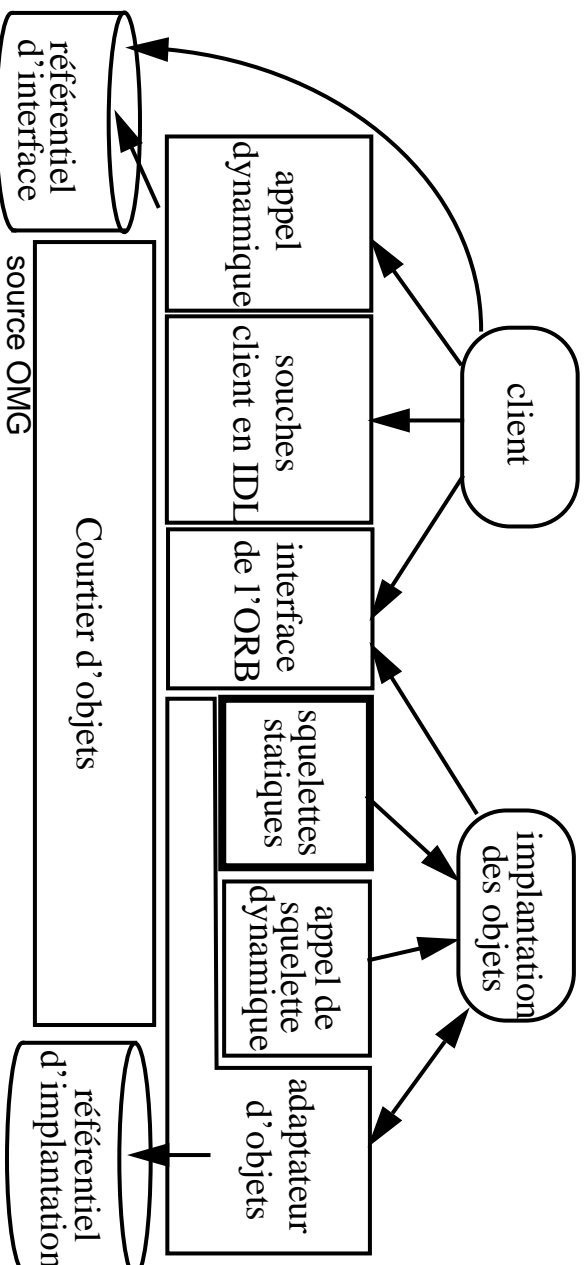


# Souche statique client IDL



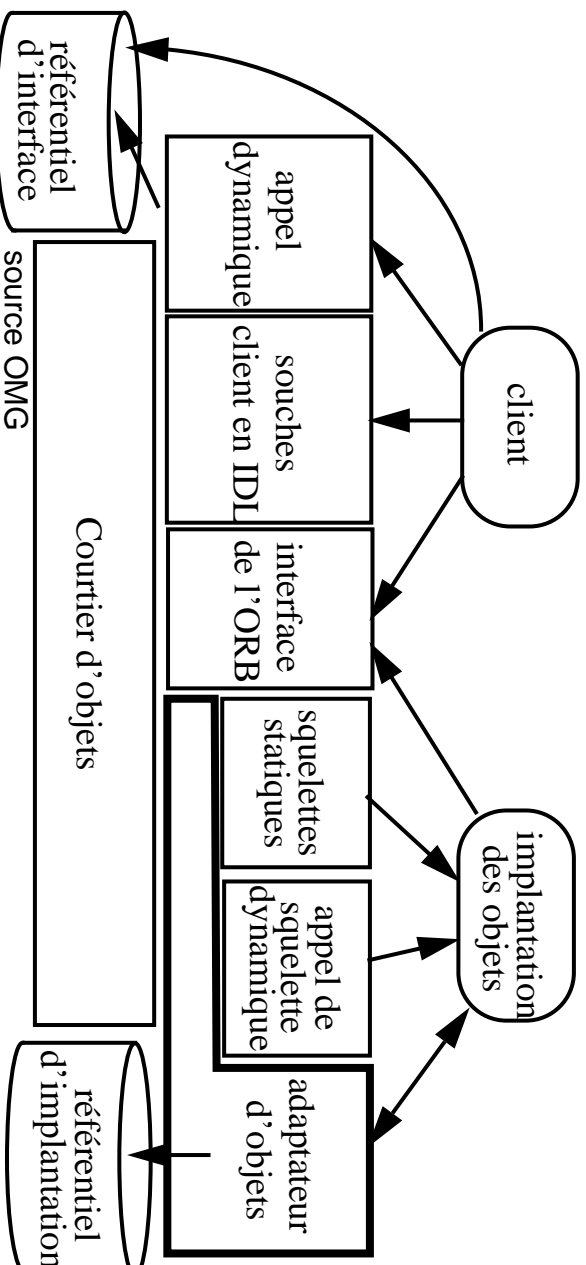
- interface précompilée statique aux services objets
- mandataire (proxi) local pour un objet serveur
- du point de vue du client, la souche agit comme un appel local
- le client possède une souche IDL pour chaque interface

# Souche IDL du serveur (Squelette)



- interface statique à chaque service exporté par le serveur.
- souche créée par compilation de l'IDL

# Adaptateur d'objets

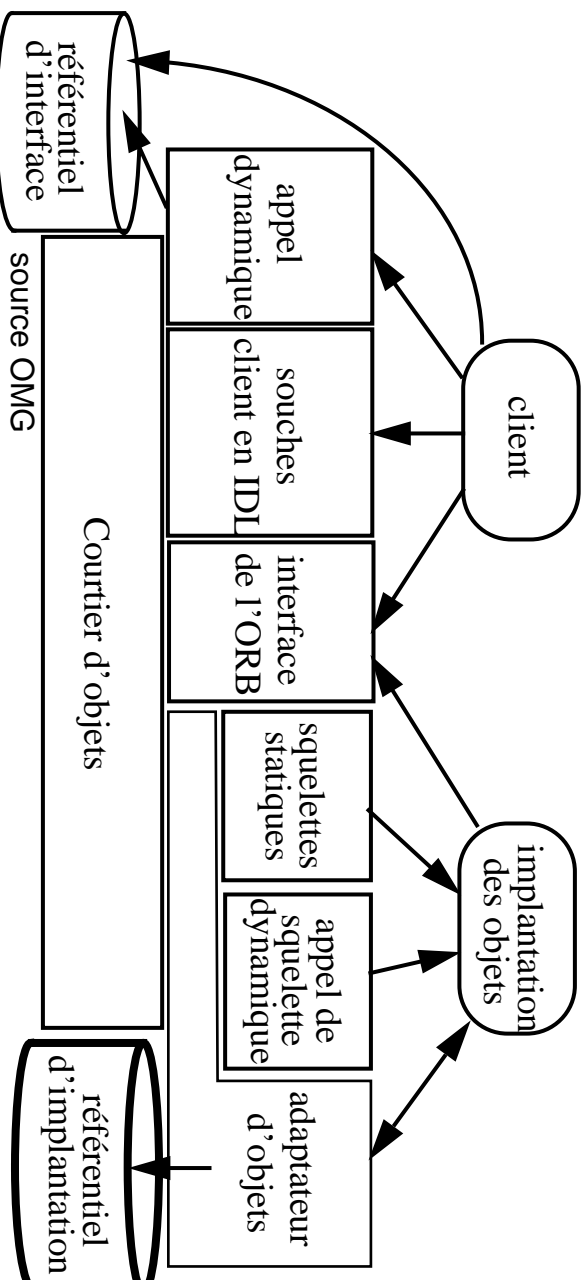


- mécanisme permettant à l'implantation d'un objet d'accéder aux services de l'ORB
- fournit un environnement d'exécution complet pour une application serveur

# Des services de l'adaptateur d'objets

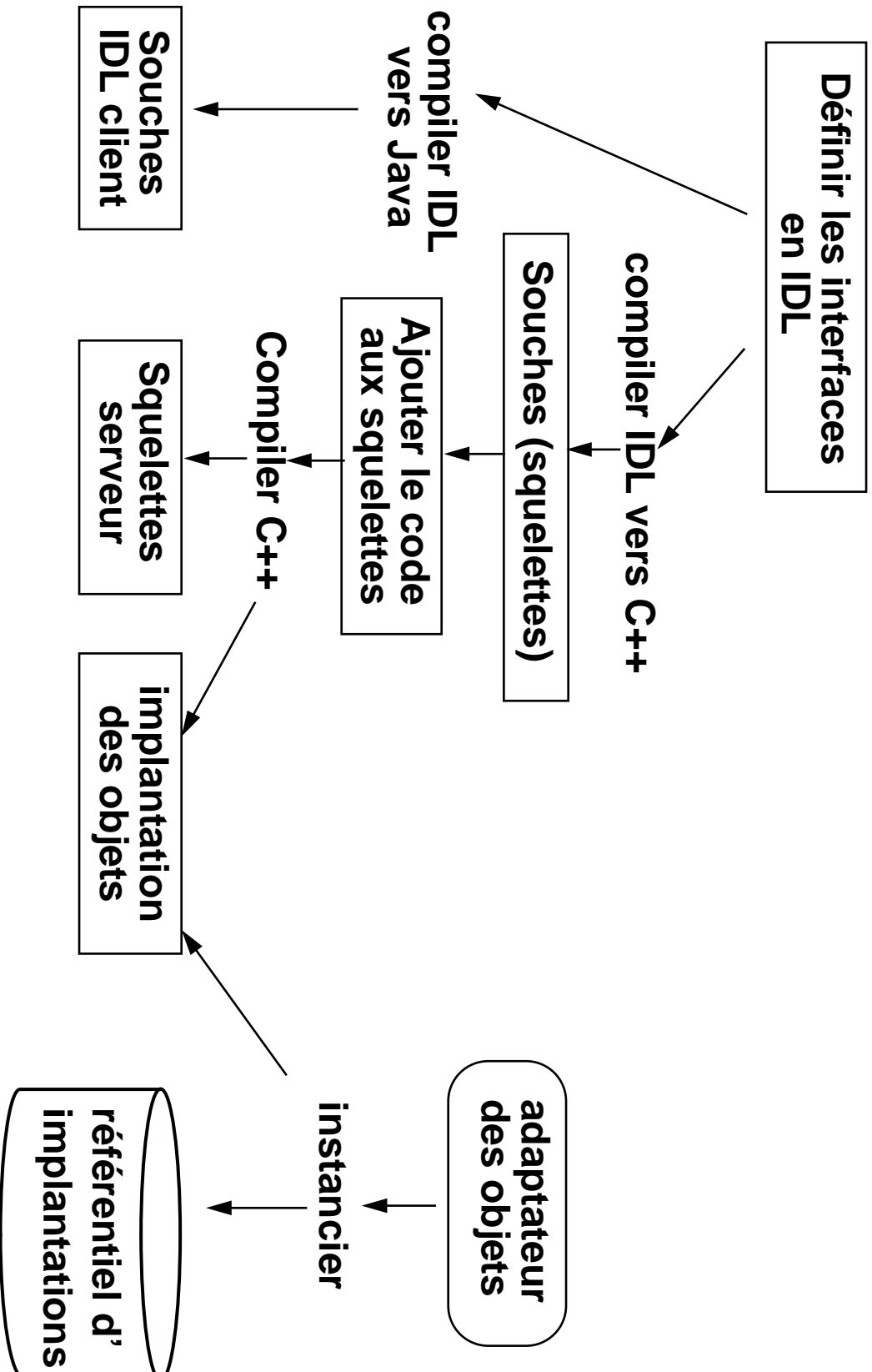
- enregistrement des classes serveur dans le réf. d'implantation
- création des objets durant l'exécution
- gestion des références d'objet (correspondance entre les références propres de l'ORB et celles de l'implantation)
- signal de la présence des serveurs d'objet
- traitement des requêtes client entrantes

# Référentiel d'implantation



- gestion des informations sur les classes, les objets instanciés et leurs ID.
- il n'est utilisé que lors de l'exécution

# mise en œuvre d'objets répartis (CORBA)



# Définition des interfaces

- terminologie OMG :
  - interface <-> classe
  - opération <-> méthode
- syntaxe proche du C++

## **fichier IDL à compiler selon les langages cible**

- module : ensemble de définitions d'interfaces
- interface : ensemble d'attributs et d'opérations (sans implantation)
- héritage (multiple) entre interfaces
- opération : méthode avec des paramètres (in, out, inout)

# Compilation de la souche serveur

exemple : compilateur IDL en C++

compilation de l'interface : souches serveur (squelette)

squelette contient :

- types définis en IDL
- code pour faire correspondre les requêtes aux méthodes
- corps vide des méthodes

compilation classes complètes : implantation des objets



# Programme serveur

instanciation de l'objet serveur

enregistrement de cet objet par l'adaptateur d'objet  
l'objet est alors dans le référentiel d'implantation  
et accessible pour rendre les services

# Compilation de la souche client

compilateur IDL en Java

souche client : classe Java

classe est complète

le corps des méthodes contient les informations permettant de relayer l'appel sur l'objet serveur.

# Applet cliente

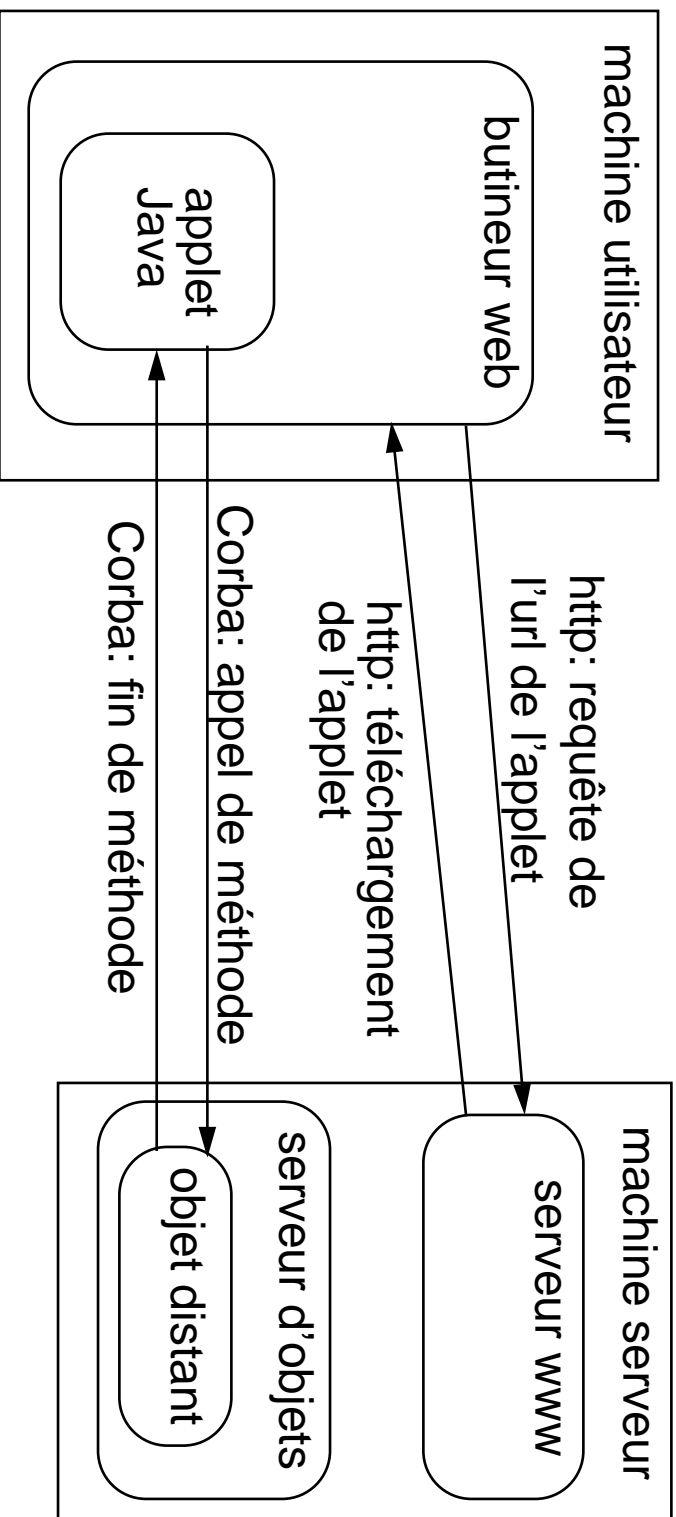
obtention d'une référence sur l'objet distant

— on utilise le nom d'enregistrement de l'objet serveur

appels des méthodes de l'objet de manière transparente

les souches prennent en charge l'arrangement des paramètres et leur circulation à travers le courtier

# Utilisation : Applet Java/Corba



# Avantages de Corba

**client et serveurs : langages quelconques**

indépendance des langages

— interfaces définies en IDL

réutilisation du code existant

neutralité par rapport aux fournisseurs

— saine émulation

services de Corba

— sécurité,

— transaction, ...

# Java/Corba versus Corba

**clients Java**

**serveur langage quelconque**

déploiement

- installation automatique
- indépendance de la plate-forme :

Avec Corba seulement, la version du logiciel doit être gérée pour chaque plate-forme

maintenance

# DCOM versus Corba

contrôlé par un seul fournisseur

dépendance du système d'exploitation (Win95 / NT)

réutilisation du code existant limité aux plateformes et Win95 / NT

ajout de services en cours

applets Java/DCOM ne fonctionnent que sur Win95 / NT

# Java /RMI

même principe que CORBA :

- interfaces
- élaboration de souches

Exemple : gestion de compte en banque à distance

Applet Java:

l'utilisateur effectue des dépôts et des retraits d'argent sur  
un compte géré à distance

Objet distant de gestion du compte :

maintient le solde et effectue les opérations



## Interface Java

```
public interface comptedistant extends java.rmi.Remote {  
    // depot d'argent  
    float depot(float amount)  
    throws java.rmi.RemoteException;  
  
    // retrait d'argent  
    float retrait(float amount)  
    throws java.rmi.RemoteException;  
  
    // lecture du solde  
    float litSolde() throws java.rmi.RemoteException;  
}
```

# Développement de l'objet serveur

définition de l'interface

définition de l'implantation

spécification du code des méthodes de l'interface

définition du constructeur pour l'objet distant

lancement du serveur

création et installation d'un gestionnaire de sécurité

création d'une instance de l'objet distant

enregistrement de l'objet sur "le référentiel d'implantation"

# Application serveur

```
public class compteServeur extends UnicastRemoteObject
    implements compteDistant {
    float solde = 0;
    ... // constructeur

public static void main(String args[]) {
    System.setSecurityManager(new RMISecurityManager() );
try {
        compteServeur UnCompte = new compteServeur();
        Naming.rebind("gestionCompteDistant", UnCompte);
    }
catch (Exception e) { ... }
} // implantation des méthodes de l'interface ...
}
```

# Développement de l'applet cliente

Déclaration d'une référence sur un objet implantant l'interface

Affectation de cette référence à l'objet serveur déclaré dans le référentiel d'implantation

appels des méthodes distantes (comme en local)

## Applet cliente

```
public class AppletClient extends Applet
    implements ActionListener {

    private compteDistant compte;

    public AppletClient(){
        try {
            compte=(compteDistant) Naming.lookup
                ("//site/gestionCompteDistant");
        }
        catch (Exception e) {... }
    }
    ...
    compte.depot(montant); // appel de méthode
    ...
```

# Compilation et déploiement

## compilation

compilation des sources serveur et client(`javac`)  
génération des souches (`fmic`)

## déploiement

démarrer le référentiel d'implantation(`mi registry`)  
lancer le programme serveur  
télécharger et exécuter l'applet

# Avantages de Java/RMI

**clients Java**

**serveur Java**

technologie identique à Corba

pas d'IDL : interface Java

ramasse-miettes distribué

téléchargement automatique des classes des arguments des méthodes

# Java/RMI versus Java/Corba

## programmation versus intégration

pas d'achat de licences de développement et d'utilisation

dépendance du langage Java

indépendance des plates-formes

services moindres (pour le moment)

problème de réutilisation du code ancien

performances



## Quelques pistes...

### intégration de l'existant

SGBD --> JDBC (pilote sur le serveur ou sur le client)  
programmes --> Corba

### développement de nouvelles applications distribuées

critères de distribution des objets

RMI Corba

### coté client

applets Java