

Environnement Réparti objet

*Omniprésence des SR → une place importante occupée
par le développement des AR*

Pr. Dalila Chiadmi

2018-2019

Organisation



- Eléments de modules de 28 heures réparties en
 - 14 h de cours + 2h de contrôle
 - 12 h de TP
- Evaluation final est une compilation de
 - Note examen,
 - Note TP
 - Bonus : questions en classes, présence, participation

Bibliographie

- ✧ Advanced CORBA Programming with C++, M. Henning & S. Vinoski, Addison-Wesley Professional; 1st edition 1999
- ✧ Client/Server Programming with Java and CORBA, R. Orfali and D. Harkey, Wiley, 2nd edition 1998
- ✧ Pure Corba, Fintan Bolton, Sams Publishing; 1st edition, 2001, ISBN-10: 0672318121
- ✧ <http://rangiroa.essi.fr/cours/>
- ✧ http://www.omg.org/technology/documents/corba_spec_catalog.htm

Constat

Over the past several years, computing has undergone a series of changes from platform and environmental aspects. Due to the invention of the Internet, centralized computing became obsolete and new computing paradigm, in the form of distributed was introduced.

Distributed computing

- is a field ... that solves computational pbs of distributed systems
- divides pb in many tasks, each one is solved by one or more computational entities
- involves several things:
 - (a) Several autonomous computational entities with local memory
 - (b) Communication among computational entities through message passing
 - (c) Fault tolerance in each computing entity
 - (d) Each computing element has limited view of the entire system.
- is essential in applications where data produced in one physical location and required by another location.
- is the process of aggregating the power of several computing entities, which are logically distributed and may even be geologically distributed, to collaboratively run a single computational task in a transparent and coherent way, so that they appear as a single, centralized system. (Cao, 2005)

Constat

C/S is ... model of distributed computing in the early days. The computing is distributed among the presentation layer and application layer. This kind of architecture has been widely used in the several applications such as ERP, billing and inventory applications. The main limitations of the C/S model are:

- need of robust client systems as complex business processing done on client.
- It is very vulnerable to security breaches as logic depends on the client.
- Scalability restrictions
- Difficulty in the maintenance and upgradation of client applications.

JAVA RMI is ... popular distributed architecture developed by Sun. It uses the lightweight object persistent techniques that makes the transaction among different components very efficiently. it uses the object oriented distributed computing. It suffers from several drawbacks:

- It is specific to JAVA code
- Hard to implement callback methods over the Internet
- Security sometimes compromised, particularly when using dynamic class loading

Microsoft DCOM distributed architecture designed by the Microsoft specifically for the windows operating systems based software components. It is successful in providing distributed supports on windows platform. It has the following limitations:

- It is specifically for Microsoft languages
- Scalability issues
- Issues in session management

Vers CORBA ...



Object Management Group has developed ... category of distributed computing architecture known as “**Common Object Request Broker Architecture (CORBA)**”. It provides an object oriented solution. With the help of this architecture, the **applications can run on any hardware platform anywhere on the network**. Interface Definition Language (**IDL**) is designed to deal with **methods of a remote object**. It suffers from the following limitations:

- There is no CORBA standard available to bind the Request Broker and its clients. Need to rely on vendor specific options.
- The interfaces of CORBA are very complex in comparison of traditional C/S architecture.
- Lacking in the implementation of CORBA services

Plan

Sem.	Date	contenu
S22	Jeu 25 avril Ven 26 avril	Introduction général programmer avec Corba
S23	Jeu 2 mai	
S24	Jeu 9 mai	HPC
S25	Jeu 16 mai	Archi. CORBA & modèle OMA Mise en œuvre des objets CORBA
S26	Jeu 23 mai	Aperçu sur le langage IDL et sa projection IDL/Java Service de Nommage
S27	Jeu 30 mai	Intro au Cloud Computing
S28	Jeu 6 juin	Férié : Aid moubarak

Objectifs du cours



1- Les principes de construction et le fonctionnement des applications réparties

1.a- Partant des caractéristiques et des besoins des AR, on fera un focus sur les middlewares permettant la construction et le déploiement d'applications réparties.

1.b- Partant des caractéristiques communes des Middleware, on fera un focus sur le service commun offert

2- Un volet du cours sera consacré au cloud computing

Rappel

Applications réparties

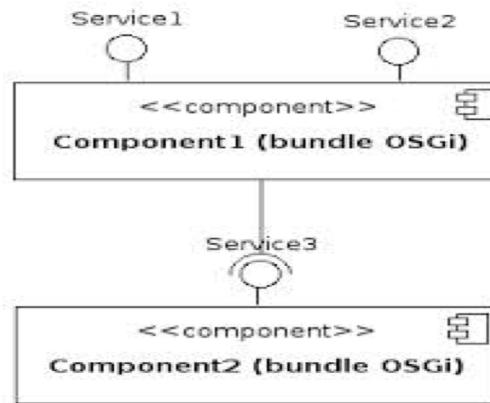
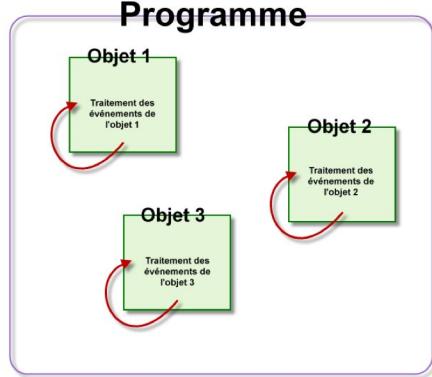
- composée de plusieurs entités résidant sur des machines distantes (environnement spécifique)
- Besoin de (paradigme de) communication

Rappel : entités de communication

Du Point de vue problème (programmation)

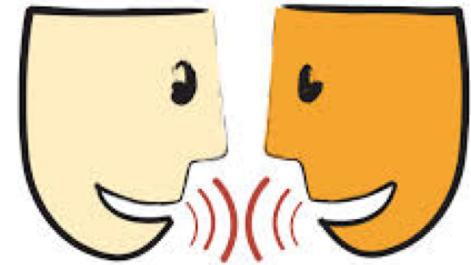
Différentes abstractions possibles

- **Objet (approche programmation OO)**
- **Composant (approche programmation par composants)**
- **Web services**



Rappel : Paradigme de Communication

- 3 Paradigmes de communication
 - Communication Inter-Processus
 - Invocation distante
 - **Communication Indirecte**
 - Communication par Flux streaming



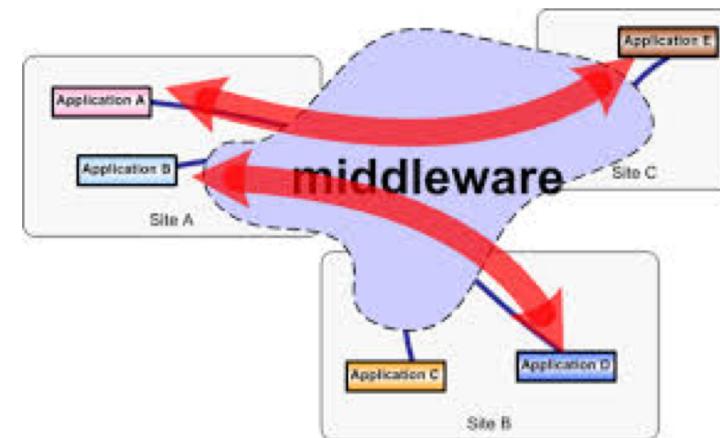
Rappel : Communication indirecte

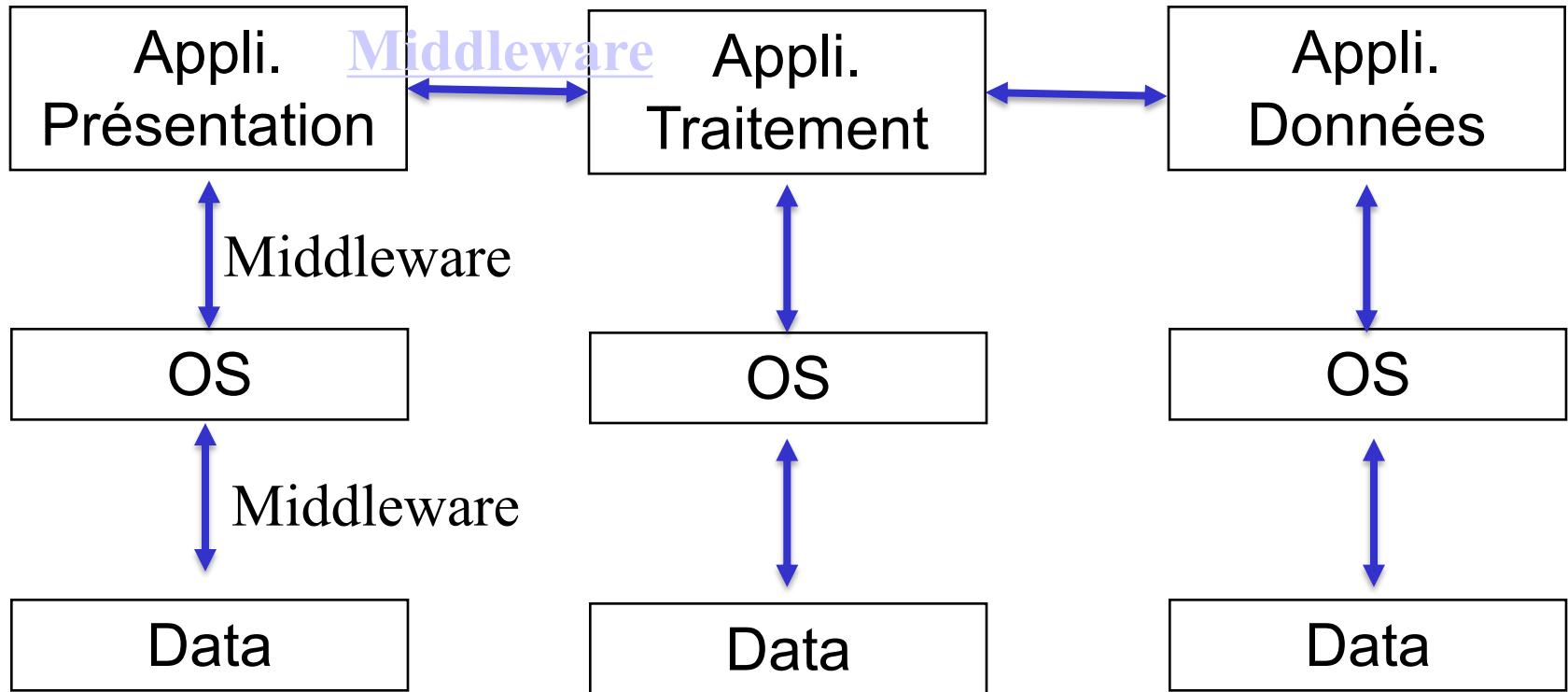
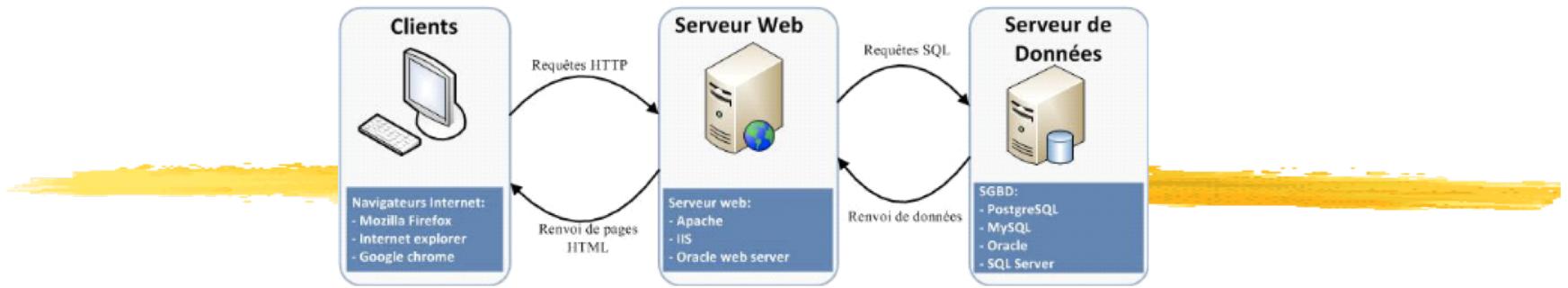
Principe : Communication entre des entités à travers un **intermédiaire** sans couplage de l'émetteur et du destinataire

Intergiciel, Middleware, Bus logiciel, Broker, etc.

Recours au middleware pour

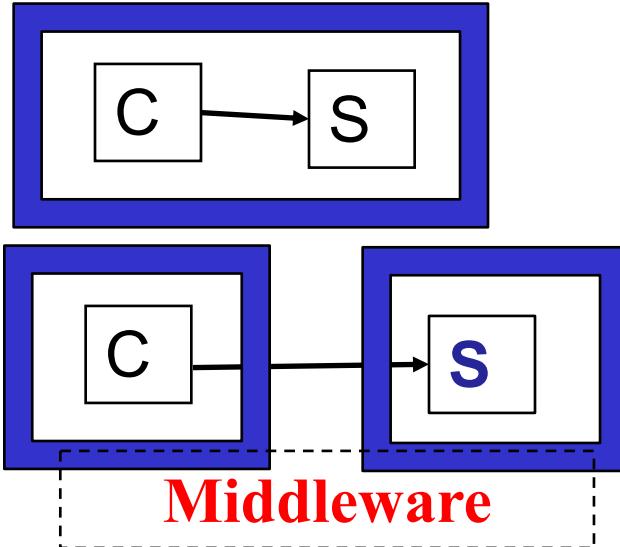
- Une communication un-à-plusieurs // one-to-many
- Une élimination du couplage spatial et temporel





Middleware : Explicite

- CS : client et serveur sont développés en collaboration
- C et S distants :
 - Client et serveur liés par une interface
 - Couche réseau masqué au client et au serveur
- Partie de code applicatif : code métier
- Partie de code non applicatif : Accès réseau, Accès BD, ...



```
Class Client {  
    public void test ()  
    { res = s.calcul (); }  
}
```

AR avec Serveur d'applications ?

- Ex de serveur d'appli : Weblogic BEA, WebSphere IBM, etc.
- Code applicatif :
 - Développement «classique»
 - Appel à des services externes explicite (ou implicite)
- Packaging du code :
 - Le code est regroupé dans une archive (jar, tar, rpm)
 - Du code d'exécution
 - Des indications de dépendances
 - Des indications d'interaction avec les services
- Déploiement du code
 - Le code est déployé sur une machine d'exploitation

Cours J2EE



- J2EE (Java 2 Platform, Enterprise Edition) définit un modèle pour développer des applications reparties. Cette architecture comprend généralement plusieurs tiers.
- Définir un service spécifique (interface/composant)
- Déployer le service (nommage)
- Piloter l'exécution du service (cycle de vie)
- Piloter les échanges entre services (communication)

Quizz

- Quels sont les éléments nécessaires pour assurer une communication entre 2 entités distantes ?
- Qu'offre un middleware ?
- Quels sont les composants d'une appli J2EE ?

JNDI Properties

//

org.omg.CORBA.ORBInitialHost=localhost

Lié à l'utilisation du serveur GlassFish qui utilise l'ORB de CORBA et le nom de service Interoperable

??

```
package ma.ac.emi.bean;
import javax.ejb.Stateless;
@Stateless (mappedName="NumberServiceJNDIName")
public class NumberServiceImplementation
    implements NumberService
{
    public NumberServiceImplementation() {
    }

    public double getNumber(double range {
        return(Math.random() * range);
    }
}
```

Nom "commercial" de JNDI



??

```
package ma.ac.emi.bean;
import javax.ejb.Remote;
@Remote
public interface NumberService
{
    public double getNumber(double range);
}
```

Interface métier : Interface Remote

??

```
package ma.ac.emi.bean;  
import javax.naming.InitialContext; // Interface  
import javax.naming.NamingException;  
public class NumberServiceDesktopApp {  
    public static void main(String[] args)  
        throws NamingException  
{  
    InitialContext context = new InitialContext();  
    NumberService service=(NumberService)context.lookup  
        ("NumberServiceJNDIName");  
    System.out.printf  
        ("Small..nb: %6.2f.%n",service.getNumber(10));  
    System.out.printf  
        ("Medium..nb: %6.2f.%n", service.getNumber(100));  
    System.out.printf  
        ("Big ..nb: %6.2f.%n",service.getNumber(1000));  
}
```

}



Exécution ?

Cas de CORBA

Signature du service

```
package ma.ac.emi.bean;  
import javax.ejb.Remote;  
@Remote  
public interface NumberService  
{  
    public double getNumber(double range);  
}
```

interface NbService

{float getNumber(in float range); } ;

Service

```
package ma.ac.emi.bean;
import javax.ejb.Stateless;
@Stateless (mappedName="NumberServiceJNDIName")
public class NumberServiceImplementation
    implements NumberService
{   public NumberServiceImplementation() {     }
    public double getNumber(double range {
        return(Math.random() * range); }}
```

public class **NbServiceImpl** extends NbServicePOA
{

Public float **getNumber** (float range) {
 return(Math.random() * range); }

Serveur



...

```
NbServiceImpl unNbServiceImpl = new NbServiceImpl;
```

```
NbService unNbService = unNbServiceImpl._this(orb);
```

...

```
orb.run()
```

Client

1 . Initialiser l' environnement d' exécution du client

```
public static void main (String args[])
{
    ...
    System.out.printf
        ("Small..nb: %6.2f.%n",service.getNumber(10));
    System.out.printf
        ("Medium..nb: %6.2f.%n",
         service.getNumber(100));
    System.out.printf
        ("Big ..nb: %6.2f.%n",service.getNumber(1000))
    ...
}
```

Rappel : besoin de middleware

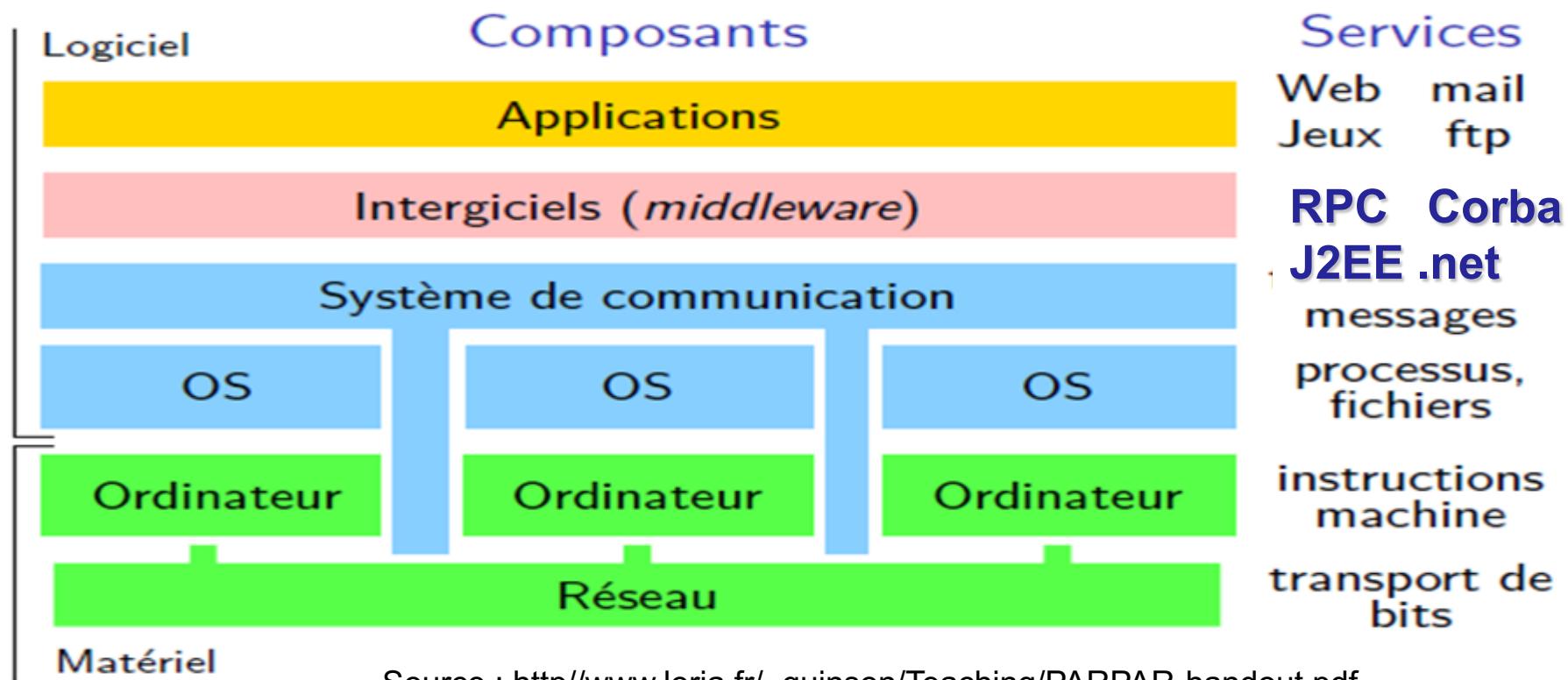


Offre de méthodes et des outils spécifiques pour gérer les besoins d'AR en termes de communications, d'hétérogénéité, d'intéropérabilité, de pannes, etc. :

- Interfaces de programmation standards (API)
- Des formats d'échanges
- Protocole d'intéropérabilité

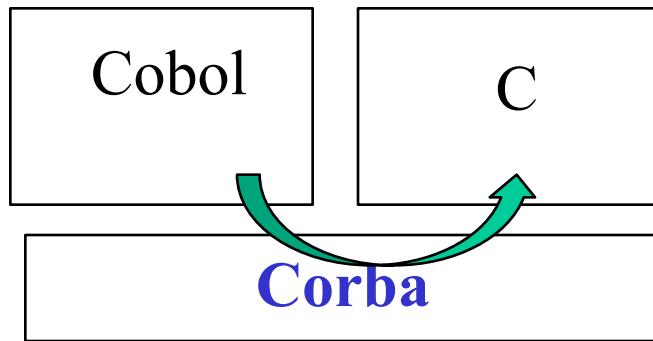
Rappel : Intergiciel

Couche logicielle située entre l'application et le système d'exploitation assure le « dialogue » entre les composants d'une AR



Types de l'hétérogénéité ?

Ex : Langage de programmation (Cobol, java, C, C++, VB, ...)



Plus général :

OS : Unix, MVS, VMS, MacOS, Windows, Android, Systèmes embarqués et temps-réel (Vrtx, qnx, lynxOs, vx ...)

Réseaux : Ethernet, IP, ATM, NFC, Bluetooth, ...

Middleware orienté objets

- Convergence de 2 technologies :
 - OO : héritage, encapsulation, polymorphisme
 - RPC : répartition, hétérogénéité, emballage et des déballage des données
- Ex : RMI

Besoins AR Vs offres intergiciel

Besoins des Applications réparties

B1 - Transparence

B2 - Abstraction.

B3 – Réutilisation de l'existant.

B4 - Capacité d'évolution.

B1 : Transparence

Df : indépendance à la localisation φ pour l'accès aux informations et aux services.

Conséquences :

- utilisation d'un service distant de la même manière qu'un service local,
- déplacer des informations ou des services d'un site à un autre sans changement visible pour les programmes ou les personnes qui les utilisent.

B2 : Abstraction¹



Df : Séparation entre interface d'accès à un service et réalisation de ce service.

Conséquence : L'abstraction permet de construire une AR par assemblage de composants, et de remplacer un composant par un autre à condition qu'ils aient la même interface.

1- Cette propriété n'est pas propre aux AR ; parfois appelée ouverture

B3 : Réutilisation

Df : Récupérer une application existante

Conséquence :

- AR construites en ajoutant des capacités d'accès à distance ou d'intercommunication à des applications existantes.
- Réduction du coût de développement
- Assurance de la continuité du service,

B4: Evolution



- Les applications doivent évoluer en permanence pour faire face à l'évolution des besoins, aux modifications de l'environnement (systèmes, outils, infrastructure de communications), et à la croissance de la demande.
- Cette évolution doit se faire de manière continue et sans interruption du service.

Schéma commun

- Modèle de base : client-serveur
- Identification et localisation des objets
- Liaison entre objets clients et serveurs
- Exécution des appels de méthode à distance
- Gestion du cycle de vie des objets (création, activation, ...)
- Services divers (sécurité, transactions, etc.)

2 : Programmer avec CORBA



- Aperçu sur l' architecture
- Adaptateurs Réseaux
- Invocation Statique C/S sous Corba
- Schéma de développement : interface, serveur, client
- Exemple : application bancaire
- Compilation Java

Rappel : Vision générale

Mise en œuvre

- Un service est accessible via une interface
- Une interface décrit l'interaction entre client et fournisseur du service. Il s'agit d'un point **d'accès et d'une vue sur les services offerts** (contrat)

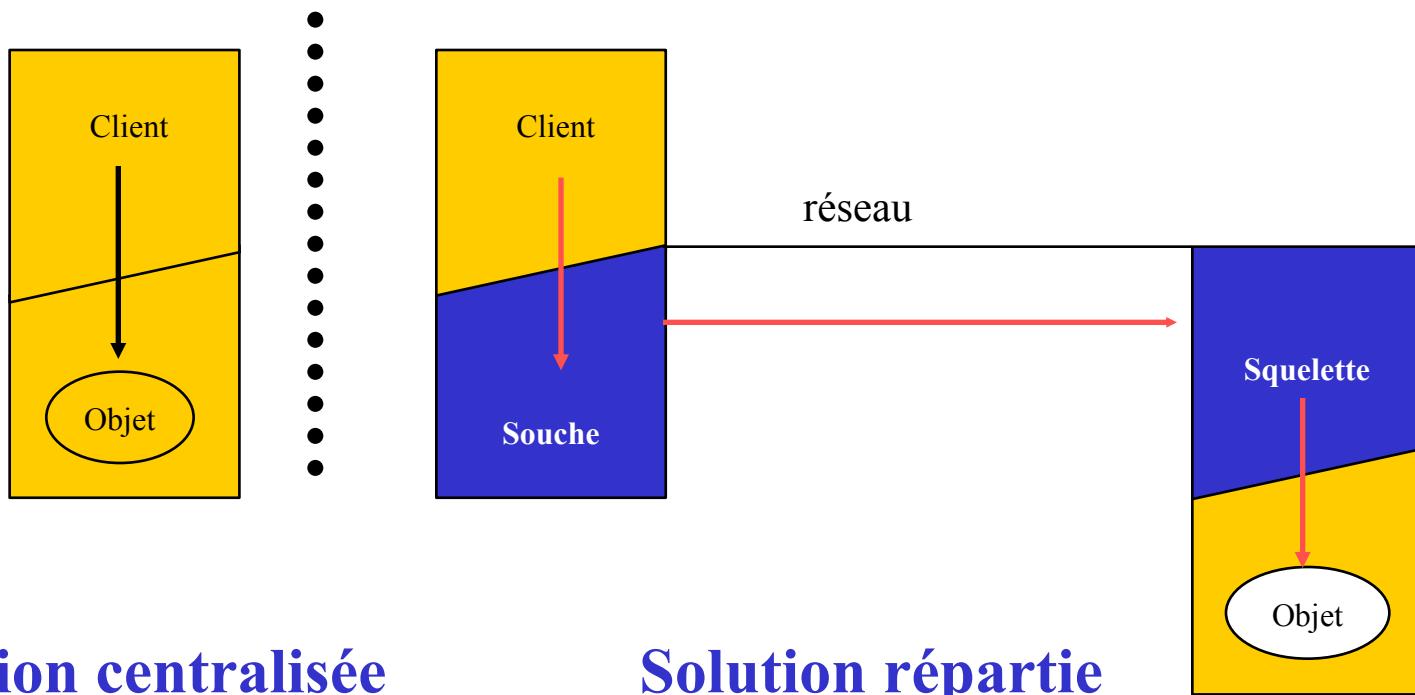
interface	implantation
Identifiant	<u>Définition_méthodes</u>
Signature_méthodes	<hr/> int somme(int a, int b) { return(a+b); }

Partie publique

Partie privée

Adaptateurs réseau

Communication C/S s'appuie sur des **adaptateurs réseau**

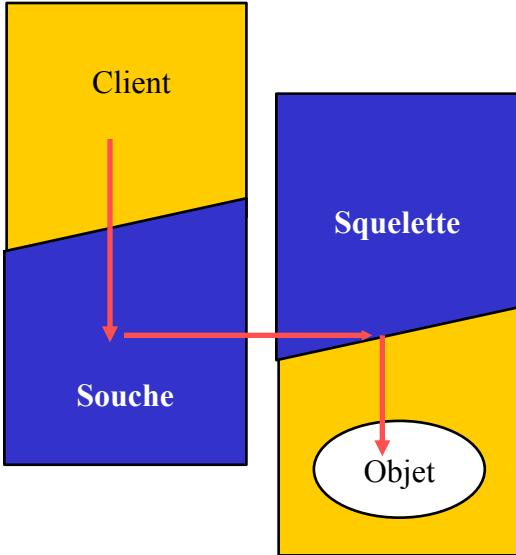


Solution centralisée

Solution répartie

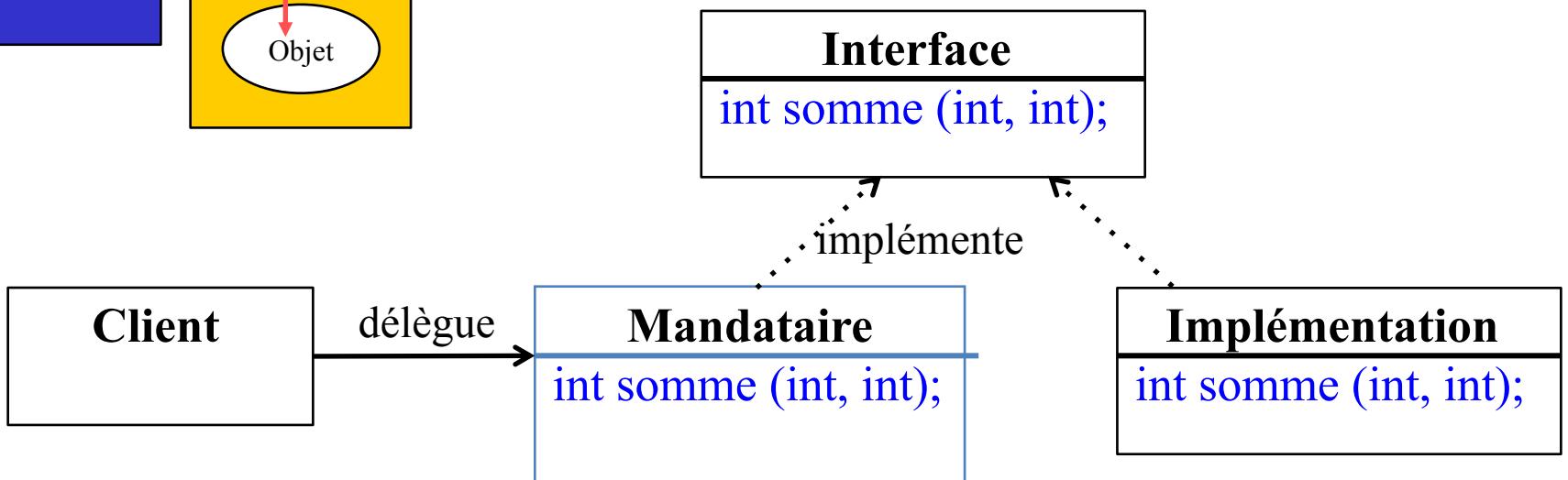
Cas de Corba : **adaptateurs réseau dérivés de l' interface**

Adaptateurs réseau



un objet **mandataire** dans la souche côté client représente l'objet d'implémentation

- le client a la référence de l'objet mandataire client
- la souche (via le mandataire) et le squelette se chargent de la présentation des données (emballage et déballage),





Client Mandataire

Implémentation

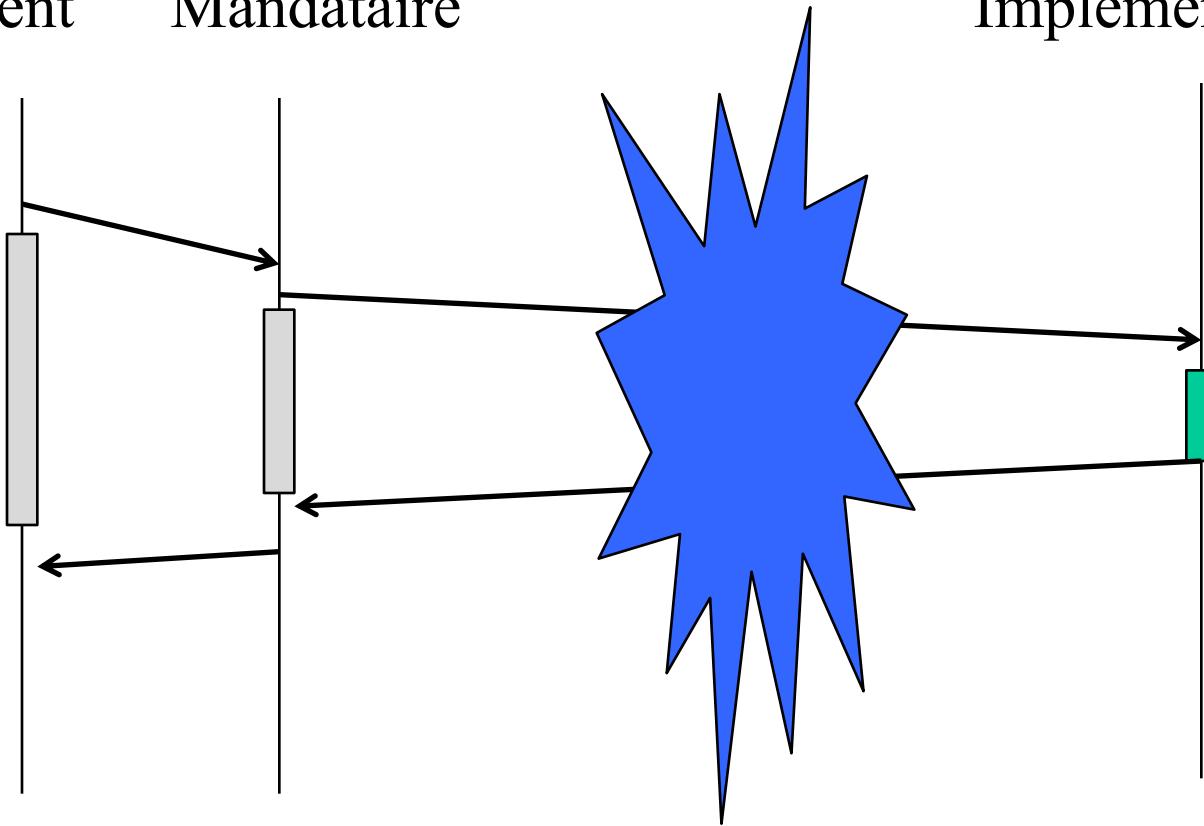


Schéma de développement

- Ecrire la description de l' **interface** (langage IDL : Interface Definition Language) et la compiler.
- Ecrire l' **objet d'implémentation** (qui met en œuvre l' interface) en utilisant un langage de pgation C, Java...) et le compiler.
- Ecrire le code du **serveur** hébergeant l'objet d'implémentation en utilisant un langage de pgation C, Java...) et le compiler.
- Ecrire le code du **client** (en utilisant un langage identique ou \neq de celui utilisé pour le serveur) et le compiler.

Interface VS Contrat

La fourniture d' un service met en jeu 2 interfaces

- Interface requise (côté client)
- Interface fournie (côté fournisseur)

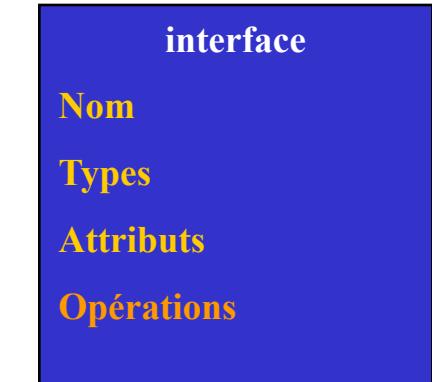


Le contrat spécifie la conformité entre ces interfaces

- Au delà de l' interface, chaque partie est une “boîte noire” pour l' autre (principe d' encapsulation)
- Conséquence : client ou fournisseur peuvent être remplacés du moment que le composant remplaçant est conforme au contrat

Décrire l' interface

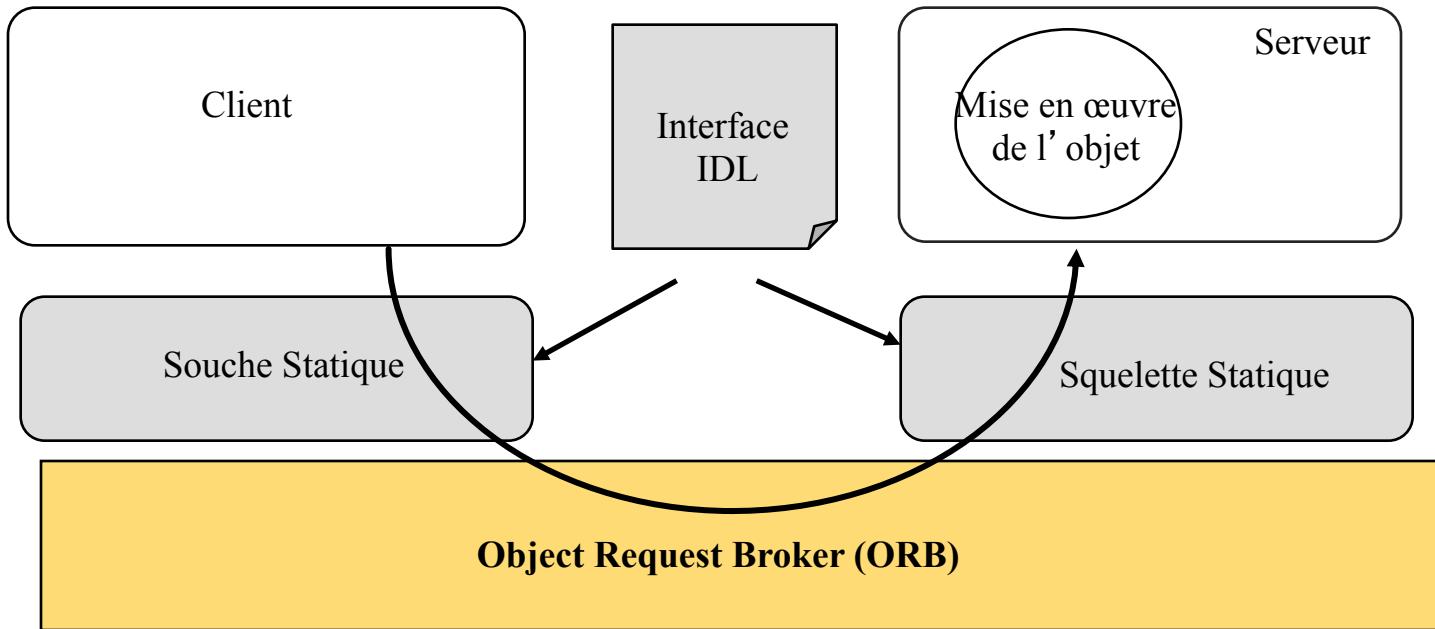
L' interface possède un **nom** et contient la df des **types** (notamment des exceptions), des **attributs** et des **opérations**.



Les opérations sont décrites avec leurs **paramètres**. Chaque paramètre est précédé par son type et son sens de passage.



1^e vue du schéma de développement



Corba repose sur un modèle C/S

- « Objet de mise en œuvre » est vu comme un S invoqué par un C
- ORB est vu comme **courtier** entre demandes et offres de services : C invoque l'ORB qui se charge de contacter le S approprié.

Object Request Broker

Les fonctions fournies par l'ORB :

- identification des objets par des **références d'objet**
- localisation d'objets
- **activation** du serveur hébergeant l'objet
- **activation** de l'objet d'implémentation
- **acheminement** des requêtes

Localisation de l'objet

Client et serveur résident dans des espaces mémoires différents

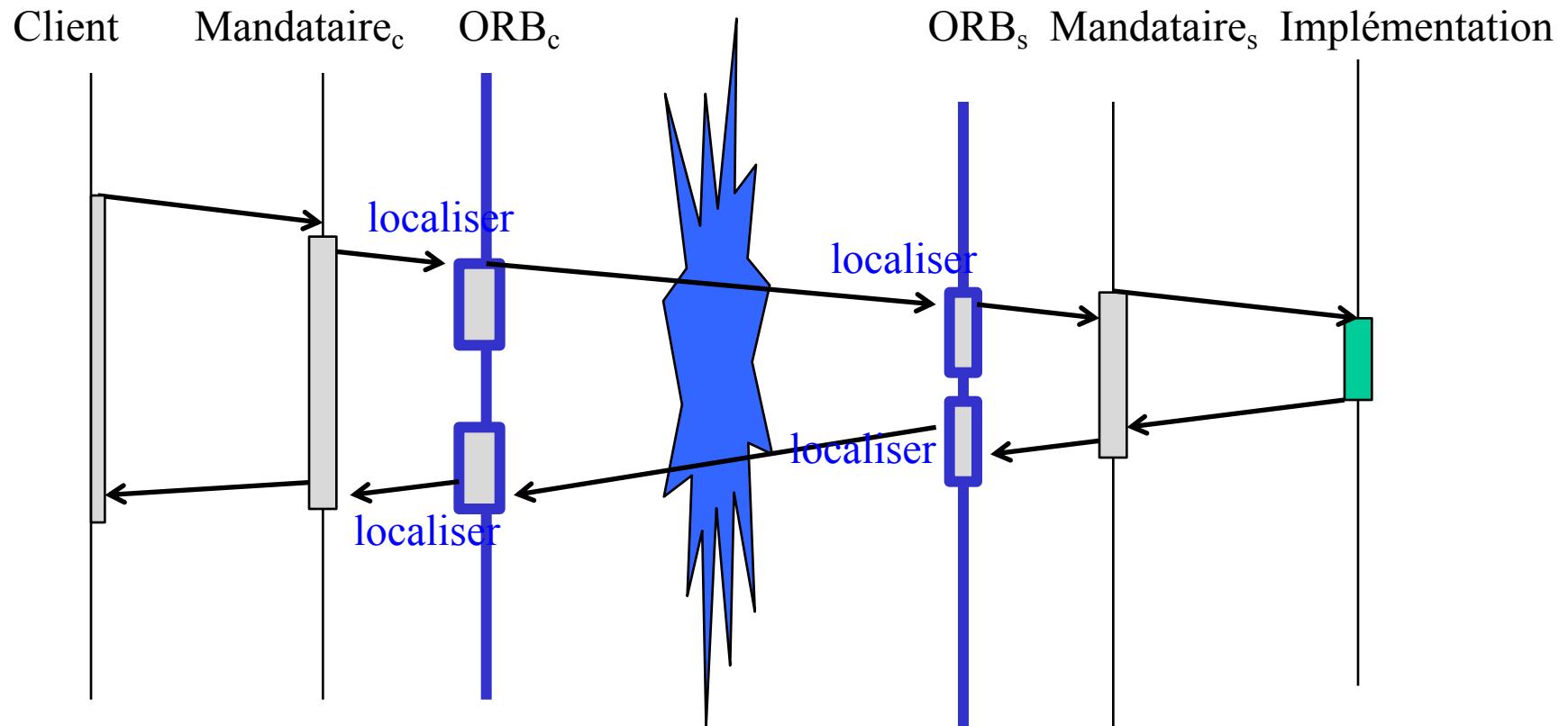
Objet d'implémentation :

- dispose d'une **référence locale** au serveur récupérée lors de l'instanciation,
- invoqué à distance grâce à la **référence globale** attribué par l'ORB

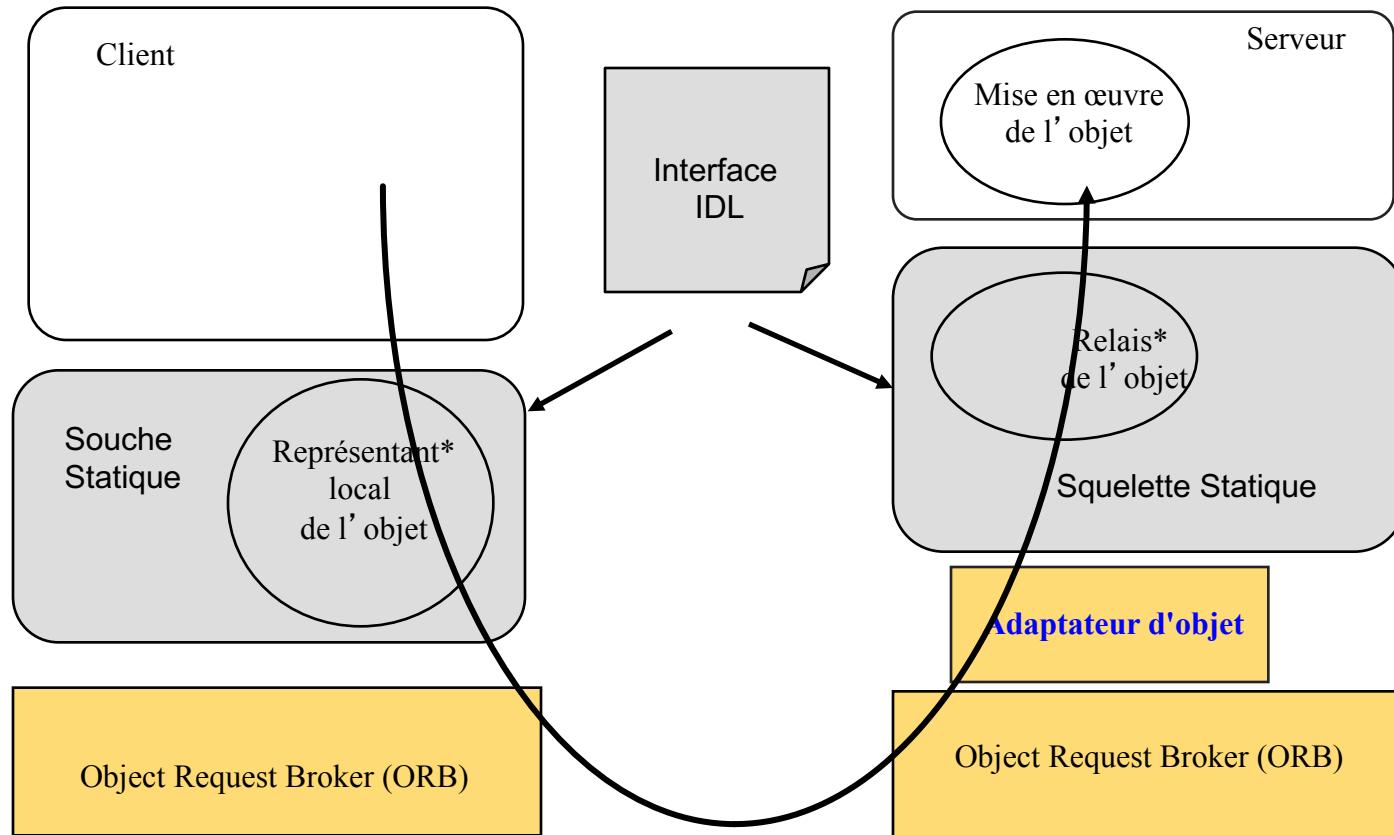
Référence globale :

- transmise au client,
- utilisée par l'ORB pour localiser le serveur lors d'une invocation

Architecture Client-Mandataire



2^e vue du schéma de développement



- assure la localisation de l'objet d'implémentation et son activation
- Traite la partie locale de la référence d'objet,
- transmet la requête au mandataire de l'objet d'implémentation

Objet d'implémentation

Réaliser l' objet d'implémentation

Réaliser le serveur

Réaliser le serveur en 4 étapes :

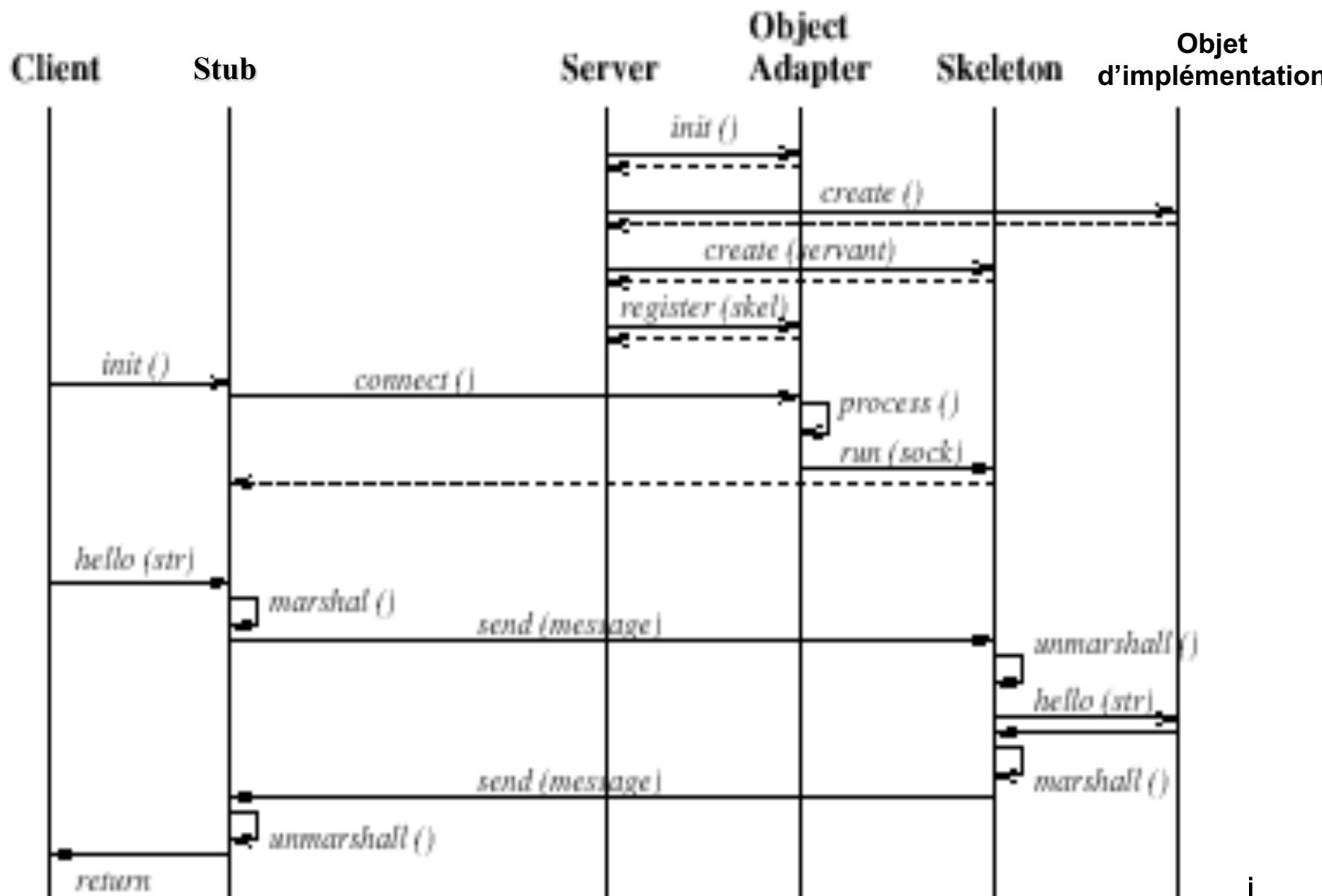
- 1- Initialiser l'environnement d'exécution du serveur : initialiser l'**ORB** et l'**adaptateur d'objets (AO)***
- 2- Instancier l'objet d'implémentation et le rendre public : exporter sa référence d'objet.
- 3- Attendre les requêtes des clients : activer **AO** et l'**ORB**.
- 4- Arrêter l'environnement d'exécution du serveur : arrêter l'**ORB** et l'**AO**.

* aiguillage des requêtes des clients vers l'objet d'implémentation

Réaliser le client

Réaliser le client en 4 étapes :

- 1- Initialiser l' environnement d' exécution : initialiser l'ORB.
- 2- Se lier à l' objet d'implémentation distant : Récupérer la référence de l' objet globale et en déduire la référence vers le mandataire (représentant de l' objet distant).
- 3- Appeler les opérations : Appel des fonctions de l' interface via des appels locaux du mandataire.
- 4- Arrêter l' environnement d' exécution du client : arrêter ORB.

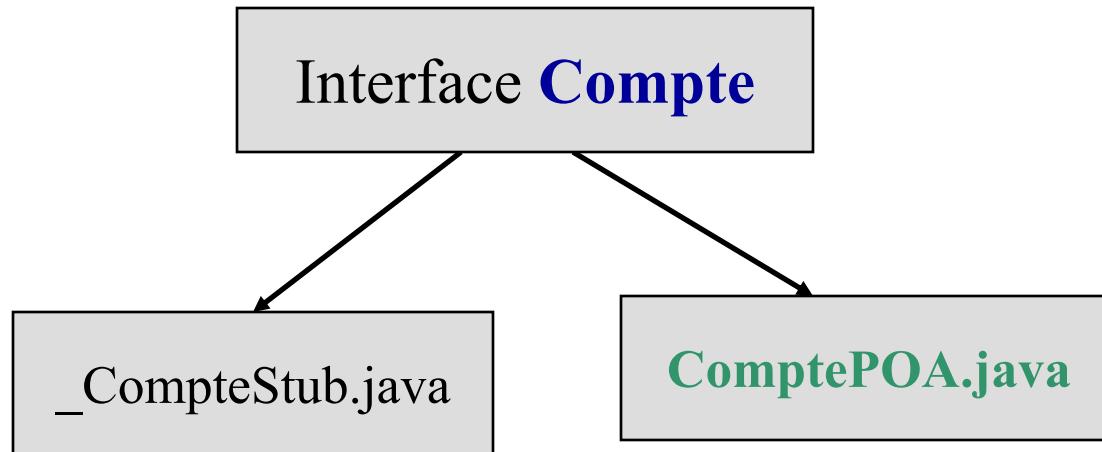


Exemple : Interface bancaire

La définition IDL pour créditer un compte

```
interface Compte  
{ void créditer ( in float somme_crédit ); } ;
```

La compilation IDL



Classe « représentant local »

Classe « relais »

Réaliser l' objet d'implémentation

```
❶ public class CompteImpl extends ComptePOA {  
  
    private float solde;  
  
    CompteImpl (float montant_init) {  
        solde = montant_init;    }  
  
    public void crediter (float montant_credit) {  
        solde =solde + montant_credit;    }  
}
```

Serveur bancaire 1/4

1 : Initialiser l' environnement d' exécution du serveur

```
public static void main(String args[]) {  
    org.omg.CORBA.ORB orb = null;  
    try {
```

// Initialisation de l'ORB effectuée par un appel standardisé

❶ orb = ORB.init(args, null);

// Récupérer la référence de l'AO « RootPOA » par un mécanisme de références initiales et l'adapter par *narrow*

❷ ... rootPOA = ... POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

// Initialisation du gérant de POA qui réalise effectivement la fonction d' aiguillage

❸ ... gerant = rootPOA.the_POAManager();
 } catch(SystemException ex) {

...

Serveur bancaire 2/4

2 : Rendre l' objet public

// création d' une instance de l' objet d'implémentation

① CompteImpl unCompteImpl = new CompteImpl(500);

// Création de la référence CORBA et déclaration implicite à l' ORB
et enregistrement dans le POA

② Compte unCompte = unCompteImpl._this(orb);

// transformer la référence CORBA de l' objet sous forme d' une
chaîne de caractères, puis la recopier dans un fichier

③ String ref = orb.object_to_string(unCompte);

String refFile = "Compte.ref";

FileOutputStream file = new FileOutputStream(refFile);

PrintWriter out = new PrintWriter(file);

out.println(ref);

file.close();

Serveur bancaire (4)

3 : Attendre les requêtes des clients

...

```
try {
```

// Activation du gérant de POA

```
① gerant.activate();
```

// Activation de l'ORB

```
② orb.run();
```

```
} catch(SystemException ex) {
```

```
    System.err.println(" -- Erreur CORBA -- \n" + ex.getMessage());
```

```
    ex.printStackTrace();
```

```
    return;
```

```
}
```

Serveur bancaire (5)

4 : Arrêter l' environnement d' exécution du serveur

...

```
try {
```

// Terminaison de l'ORB

```
① orb.shutdown(true);
```

```
} catch(SystemException ex) {
```

...

Compilation du Serveur bancaire

- compilation Java classique qui consiste en la génération de codes intermédiaires “.class”.

Client bancaire (1)

1 . Initialiser l' environnement d' exécution du client

```
public static void main (String args[]) {  
    org.omg.CORBA.ORB orb = null;  
                                // initialisation de l ' ORB  
    orb = ORB.init(args, null);  
}  
catch(SystemException ex)  
{  
    System.err.println(" -- Erreur CORBA -- \n" + ex.getMessage());  
    ex.printStackTrace();  
    return null;  
}
```

Client bancaire (2)

2 . Se lier à l' objet distant

...

```
String refFile = "Compte.ref";
FileInputStream file = new FileInputStream(refFile);
BufferedReader in = new BufferedReader(new
InputStreamReader(file));
ref = in.readLine();
file.close();
// Référence convertie en référence CORBA générique
```

① org.omg.CORBA.Object obj = orb.string_to_object(ref);

// Conversion de la référence Corba en référence locale Réf.
conformément à l' interface représentée

② Compte compte = CompteHelper.narrow(obj);

...

Client bancaire (3)

3 . Faire appel aux opérations

...

// Appel vers les opérations distantes

```
try {  
    System.out.println( " Indiquer le montant du credit : ");  
    montant_credit=lire_float();  
    compte.crediter(montant_credit);  
} catch(SystemException ex) {  
    System.err.println(" -- Erreur CORBA -- \n" + ex.getMessage());  
    ex.printStackTrace();  
    return;  
}
```

...

Client bancaire (4)

4 . Arrêter l' environnement d' exécution du client

```
...
try {
    ...
    orb.shutdown(true);
} catch(SystemException ex) {
    ...
    // Terminaison de l'ORB
}
```

Compilation Java

- Liste des fichiers .java :
 - développés : client_banque.java, serveur_banque.java, CompteImpl.java.
 - générés automatiquement :
[_CompteStub.java](#), Compte.java,
CompteHelper.java, CompteHolder.java,
CompteOperations.java, [ComptePOA.java](#).
- Génération des fichiers .class:
 - À partir des fichiers .java développés et générés automatiquement
(ex) javac *.java

Archi. CORBA & modèle OMA



- L' OMG
- L' architecture CORBA
- Le modèle OMA
- Les produits

OMG : Object Management Group



Consortium international, créé en 1989, composé de près de mille membres : constructeurs informatiques, développeurs de logiciels et utilisateurs.

OMG : Mission



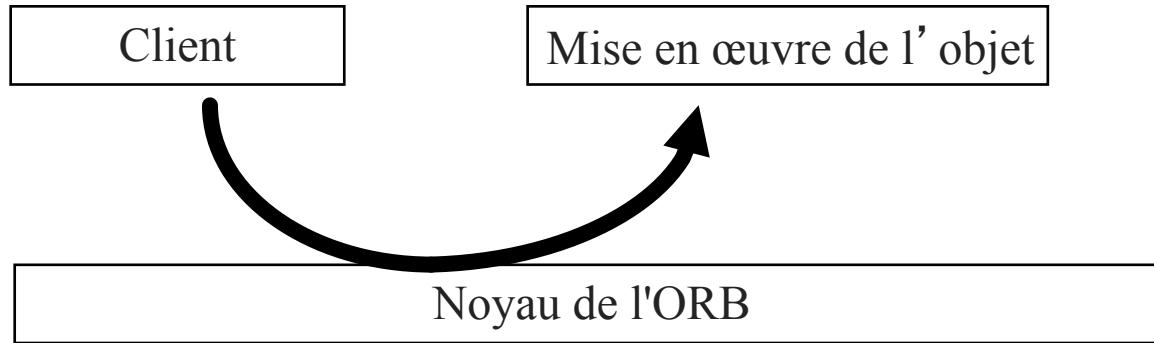
- Promouvoir la **technologie OO** dans les systèmes informatiques répartis
- Produire des **spécifications pour la conception d'AR OO**
- Fournir une **architecture de base** pour l' intégration d' AR garantissant la **réutilisabilité, l' interopérabilité et la portabilité.**

OMG : Spécifications



- **CORBA** : Common Object Request Broker Architecture
- **OMA** : Object Management Architecture (Services CORBA, Utilitaires CORBA et Objets de domaine)
- Analyse et conception

L'architecture CORBA (1)



Communication est prise en charge par CORBA : **le noyau de l'ORB** assure le transport des requêtes entre le client de l'objet et sa mise en œuvre distante de **façon transparente**.

Quelques composants de l'ORB

Les 2 pbs **portabilité** et **interopérabilité** sont gérés par les composants de l'ORB :

- IDL-OMG
- Protocole de Communication

IDL : Interface Definition Language

1. Langage de définition des interfaces entre clients et serveurs d'objets
2. Résout le problème de la portabilité

Protocole de Communication : GIOP



GIOP : (General Inter ORB Protocol) protocole générique

1. Représentation commune des données (CDR)
2. format de références d'objet interopérable (IOR)
3. ensemble de messages de transport (Request, Reply, ...)

IIOP nouveau

- Un client RMI-IIOP *attend* pendant que le serveur effectue le traitement d'une requête,
- Fiabilité
 - Lorsqu'un client RMI-IIOP parle avec un serveur, ce dernier doit être en train de fonctionner. S'il crashe, ou si le réseau crashe, le client est coincé.
- Pas de broadcasting !
 - RMI-IIOP limite les liaisons 1 client vers 1 serveur

Protocole de Communication : IIOP

Internet Inter ORB Protocol, implantation de GIOP au dessus de TCP/IP, résout le pb de l'interopérabilité,

les IOR (IIOP) doivent contenir

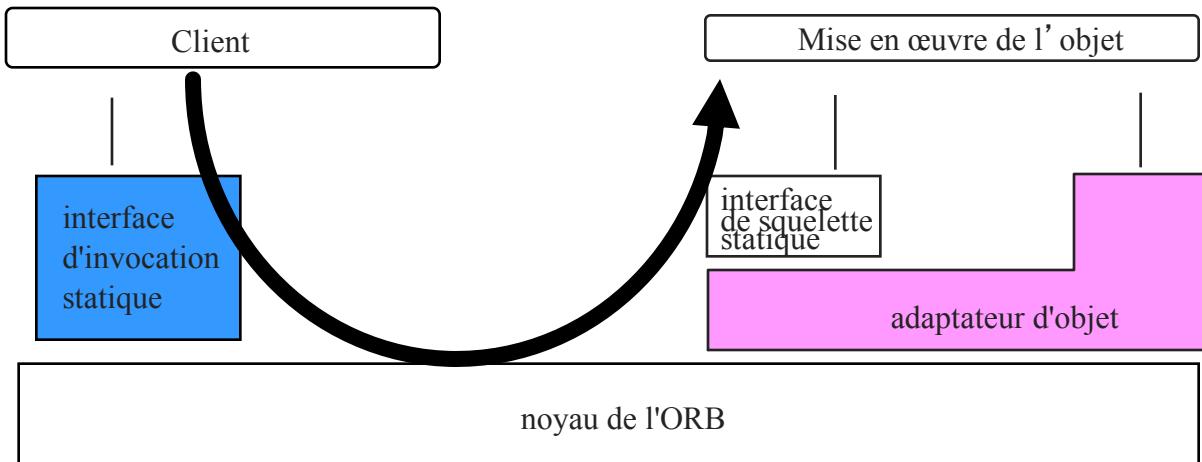
- nom complet de l'interface OMG-IDL de l'objet
- adresse IP de la machine Internet où se trouve l'objet
- n° port TCP pour se connecter au serveur de l'objet
- clé pour désigner l'objet chez le serveur

l'ORB

Le noyau de l'ORB doit être assez générique pour permettre de véhiculer n'importe quel type de requêtes vers n'importe quel type d'objet.

Il est donc nécessaire d'avoir une couche intermédiaire d'adaptation entre l'application et le noyau de l'ORB.

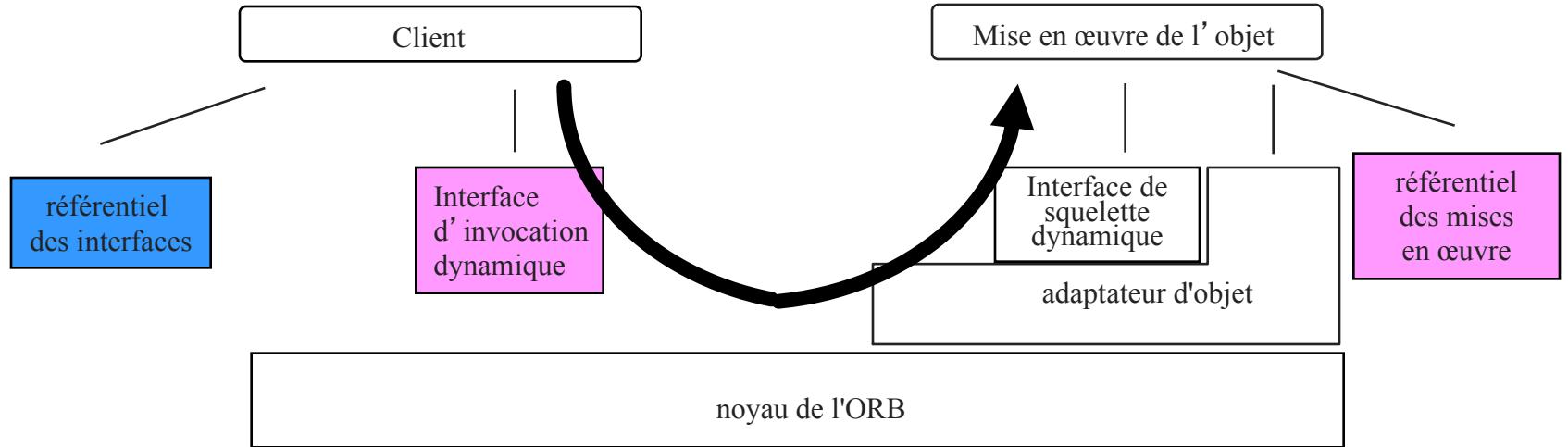
L'architecture CORBA (2)



Adaptateur d'objet : assure le lien entre références d' objet CORBA manipulée par le noyau de l' ORB et la référence d' objet locale correspondant à la mise en œuvre effective de l' objet (servant)

Invocation statique : assure les fonctions d'encodage et de décodage des requêtes et des réponses véhiculées entre objets.

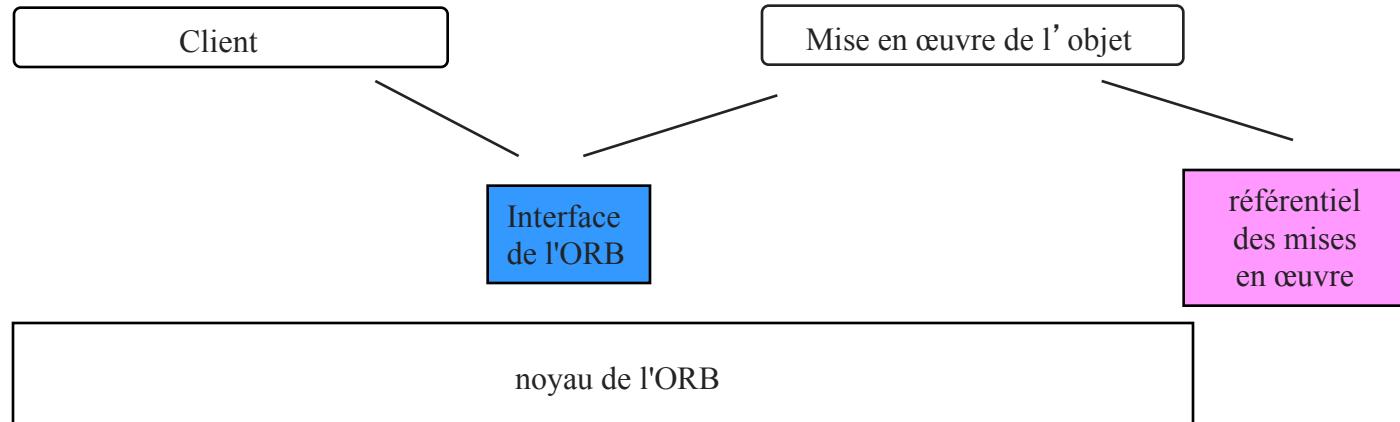
L'architecture CORBA (3)



Invocation dynamique : permet de construire et de recevoir dynamiquement des requêtes vers des objets non connus à la compilation.

Référentiel des interfaces (RI) : contient toutes les définitions IDL des interfaces.

L'architecture CORBA (4)

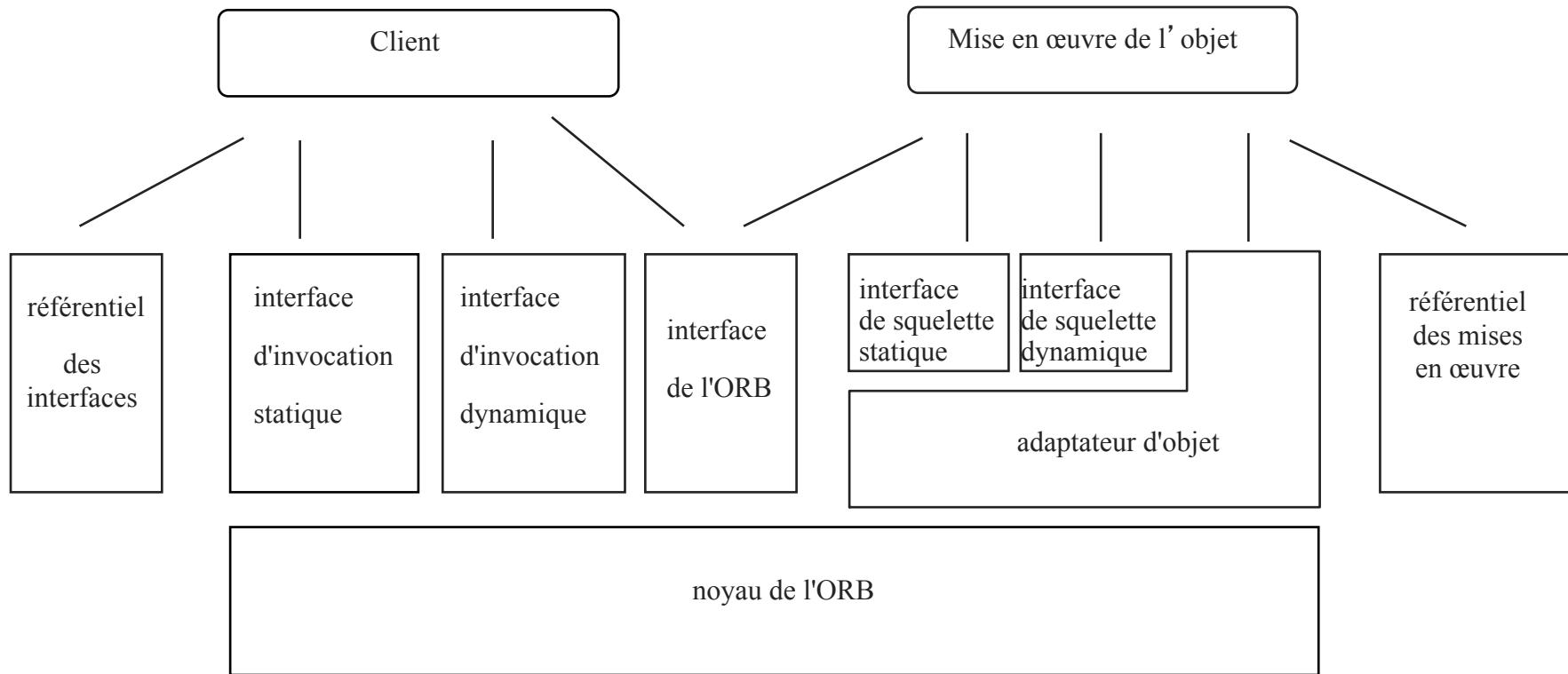


Référentiel des implantations : stockage des informations relatives aux mises en œuvre des objets. Il offre un service similaire au RI mais vis-à-vis des réalisations effectives des interfaces et non pas seulement de leur déclaration statique.

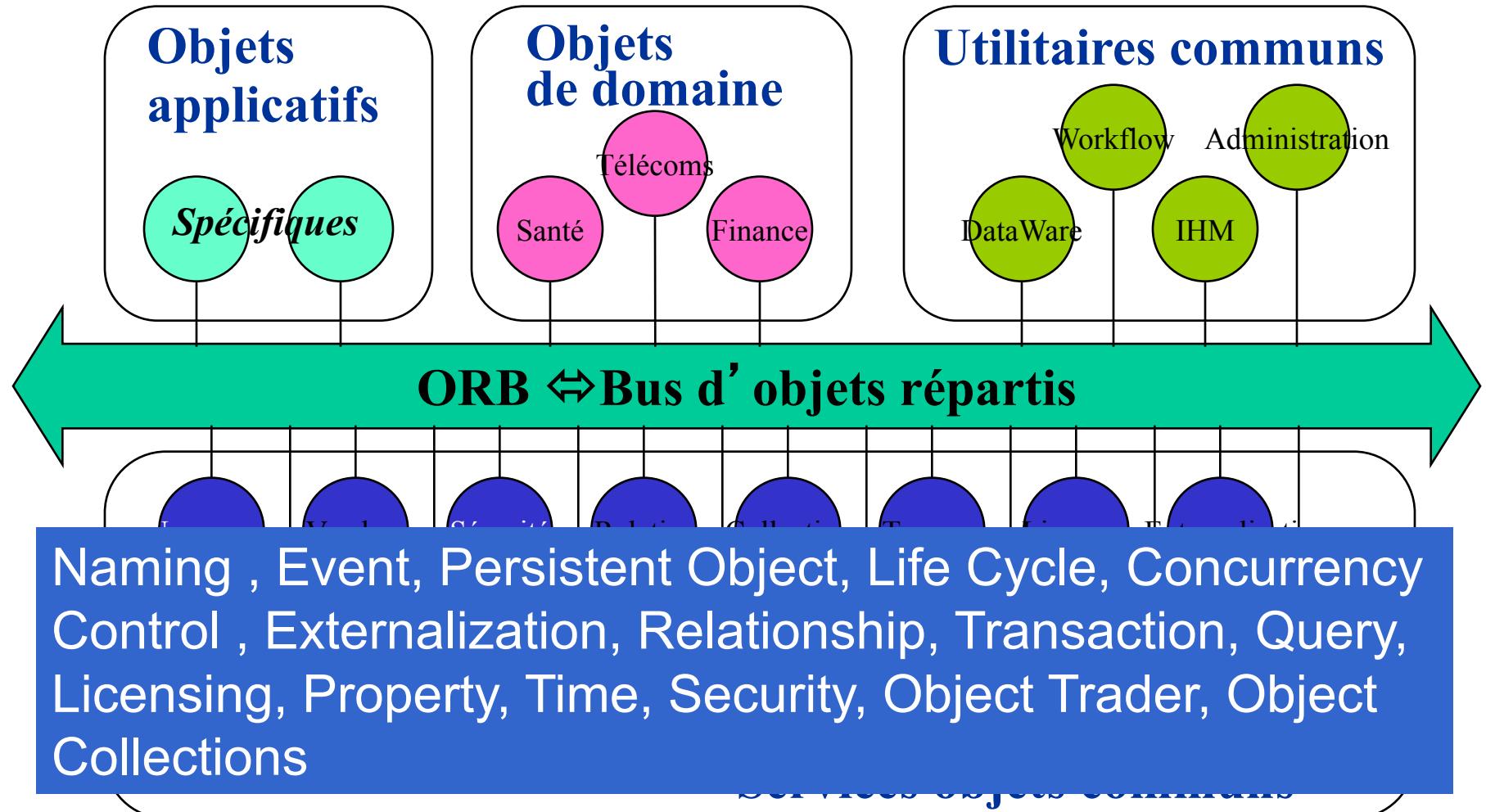
Interface de l'ORB : offre des services de manipulation des références d'objets et d'initialisation.

L'architecture CORBA (5)

Récapitulatif



OMA : Object Management Architecture



Les Objets applicatifs

- Les Objets applicatifs (Application Interfaces)
 - Spécification d'interfaces IDL
 - Définis par une application de l'utilisateur
 - Hors du champ de standardisation de l' OMG
 - Possibilité de standardisation pour des objets émergents

Les objets de domaines

- Standardisés par l' OMG, leurs fonctionnalités peuvent être étendues ou spécialisées par héritage
- Spécifiques à un domaine d' application

Telecom : AVS (Audio/Video Streams), Telecom Log Service, CORBA/TMN

Finance : Currency Specification, General Ledger, Party Management

Médical : Person ID Specification, Lexicon Query, Resource Access Decision Facility, Clinical Observation Access Service

...

Les utilitaires communs (Common Facilities)

- Spécification d' interfaces IDL
- Leurs fonctionnalités peuvent être étendues ou spécialisées par héritage
- Ensemble de services de plus haut niveau fournissant des fonctionnalités utiles dans de nombreuses applications
- Indépendants des domaines d' application
- Exemple : Internationalization and Time, Mobile Agent, Printing, workflow, impression, aide en ligne, messages d' erreurs, etc.

Services objets communs (Object Services)

- Spécification d' interfaces IDL
- Leurs fonctionnalités peuvent être étendues ou spécialisées par héritage
- Interfaces indépendantes des domaines d' application
- Objectif : étendre les fonctions de l' ORB
- Très utiles aux programmeurs
- Ne sont pas obligatoires pour :
 - avoir une mise en œuvre compatible CORBA
 - développer une application CORBA

Les produits : Offre CORBA

- ORBs d'éditeurs logiciels
 - Orbix de IONA Technologies,
 - Visibroker de Borland,
 - Object Broker de BEA Systems.
- ORBs de constructeurs
 - Component Broker de IBM,
 - ORB Plus de HP.
- ORBs du domaine public
 - .or bacus
 - jacorb..

Critères d'évaluation

- Projections vers des langages
 - C, C++, Smalltalk, Cobol, Ada, Java, Lisp
- Protocole d'interopérabilité IIOP
- Interfaces dynamiques DII et DSII
- Intégration COM
- Services et utilitaires CORBA
- Utilitaires de développement,
d'administration, ...

Conclusion

- OMG: consortium qui définit les standards CORBA, OMA, CORBAservices et CORBAfacilities.
- Spécifications consolidées pour l'ORB, encore peu stables pour les CORBAservices et en cours pour les CORBAfacilities.
- Produits disponibles chez plusieurs éditeurs de logiciels, constructeurs informatiques et dans le domaine public.

Mise en œuvre des objets CORBA



- Objet CORBA
- Classe *org.omg.CORBA.Object*
- Hiérarchie des références
- Hiérarchie de classes : côté client
- Mise en oeuvre du service : héritage et délégation

Objet CORBA



- Composant logiciel **autonome**
- Utilisation **indépendante** du langage, compilateur, OS, localisation
- Vu par son **interface**
- Identifié par une **référence** (IOR : Interoperable Object Reference)

Rôle de l' objet CORBA

- **Objet CORBA** : entité virtuelle localisée par le bus ORB, reçoit les requêtes clients qui lui sont destinées
- **Service** : entité réelle programmée en un langage, implante un objet CORBA et évolue dans le contexte d'un serveur.
- Pour qu'un objet CORBA traite des requêtes, il a besoin d'un Servant qui **concrétise** son existence et réponde aux requêtes.

La classe *org.omg.CORBA.Object*

- Pour échanger des objets de tout type sans en être dépendant, l'ORB manipule des objets **génériques** (*org.omg.CORBA.Object*) plutôt que des objets utilisateur (dt le type dépend de l'appli.)
- Tous les objets CORBA (donc les interfaces IDL) héritent, dans la projection Java, de cette classe **générique**,
 - 
 - Ttout objet utilisateur peut être utilisé comme objet générique au moyen de l'héritage.

La classe « org.omg.CORBA.Object »

L' objectif de cette classe est de :

- représenter un objet distant.
- gérer toutes les références sur un même objet distant.
- fournir des primitives pour créer de requêtes «dynamiques ».

Objet générique Vs objet utilisateur

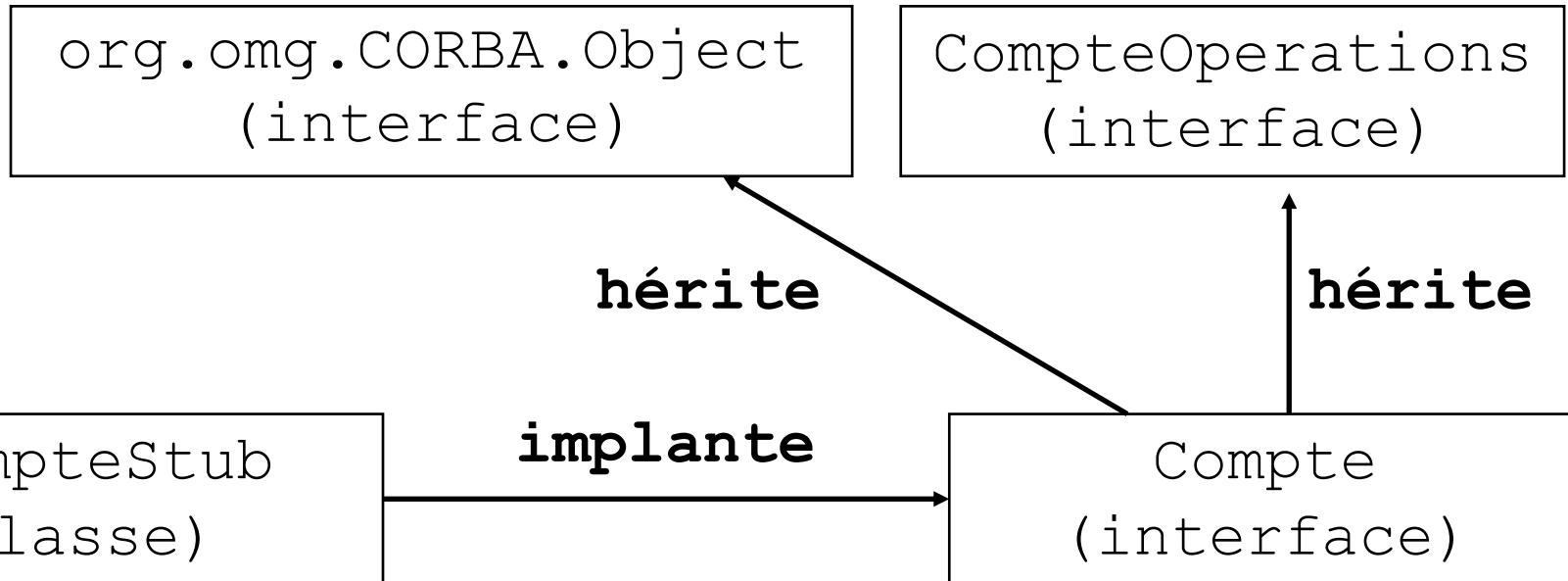
Pour convertir un objet générique vers un objet utilisateur, il faut s'assurer de la validité de la conversion : C'est le rôle de la fonction de spécialisation **narrow**.

Ex : au niveau du client

```
org.omg.CORBA.Object obj;  
Compte unCompte =  
    CompteHelper.narrow(obj);
```

Hiérarchie de classes : côté client

(Java)



L' interface Compte est générée dans la souche à partir de la description IDL. Elle hérite de l' interface générique org.omg.CORBA.Object et de l' interface spécifique CompteOperations dérivée de la description IDL.

Les instances de _CompteStub supportant l' interface Compte correspondent aux objets représentant local

Mise en œuvre du serveur



Deux stratégies :

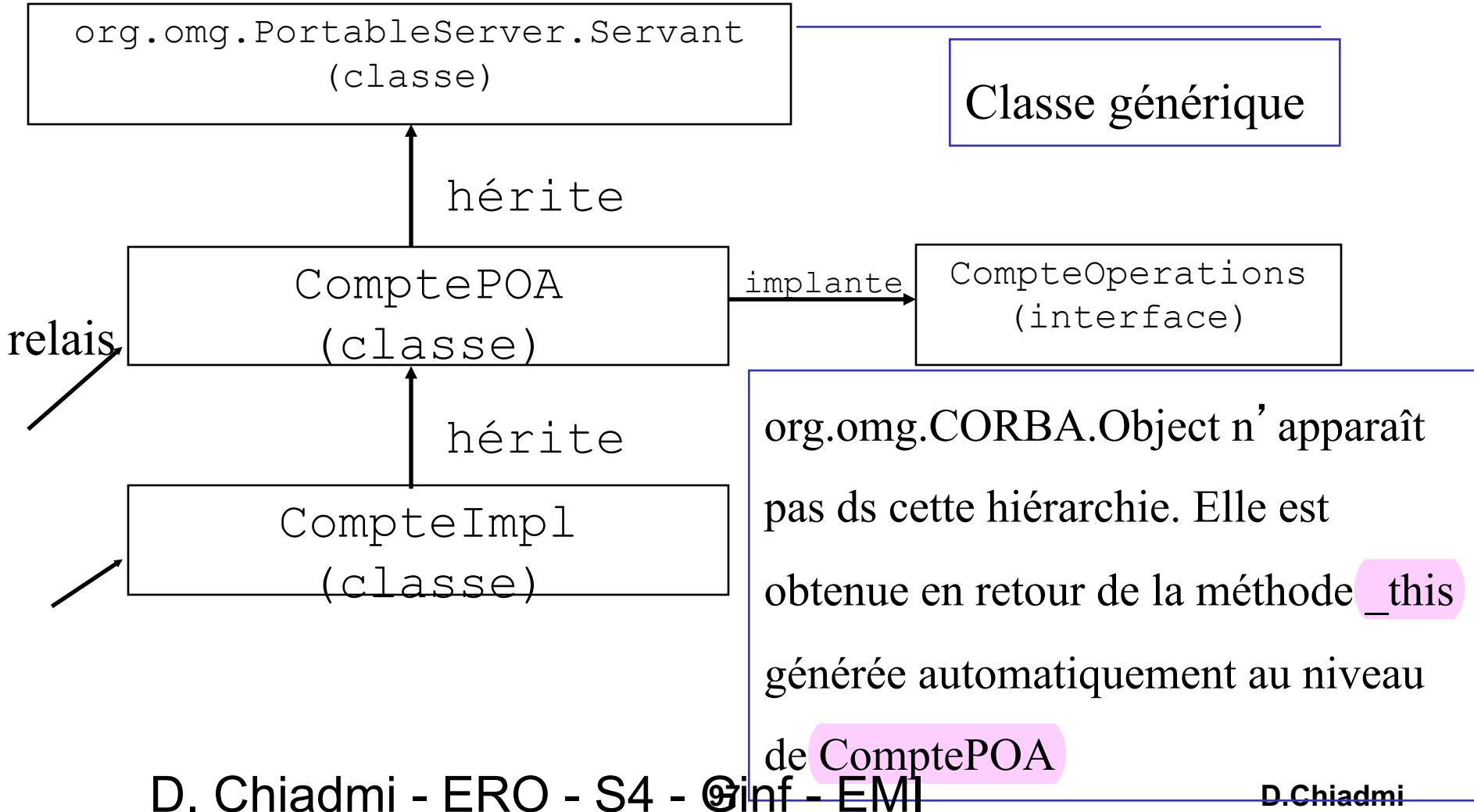
- Par Héritage (utilisée dans l' exemple)
- Par Délégation

Mise en œuvre du serveur : par héritage

La classe de l' objet de mise en œuvre hérite de la classe skeleton (relais) générée par la compilation de l' IDL.

Hiérarchie de classes côté serveur

Mise en œuvre par héritage en Java



Programmation en Java

D'où la nécessité de rajouter l'héritage de ComptePOA dans la déclaration de la classe de l'objet de mise en œuvre et les 2 instructions permettant d'initialiser un objet de mise en œuvre servant (objet CORBA).

```
// Objet de mise en œuvre
public class CompteImpl extends ComptePOA
{ ... }
```

```
// Serveur
CompteImpl unCompteImpl = new CompteImpl();
Compte unCompte = unCompteImpl._this(orb);
```

Mise en œuvre par délégation

S'il n'est pas possible ou souhaitable de changer la hiérarchie de classes de l'objet de mise en œuvre, on peut utiliser une classe de délégation (appelée tie) qui hérite ou réalise les méthodes du skeleton et qui redirige les appels de l'interface vers l'objet de mise en œuvre (classe déléguée).

L'objet de délégation

S'il hérite d'autres classes, il y a héritage multiple.

Pb : en Java, l'objet d'implantation ne peut maintenir son héritage d'interfaces et l'implantation de l'héritage devient l'unique solution.

L'objet de délégation

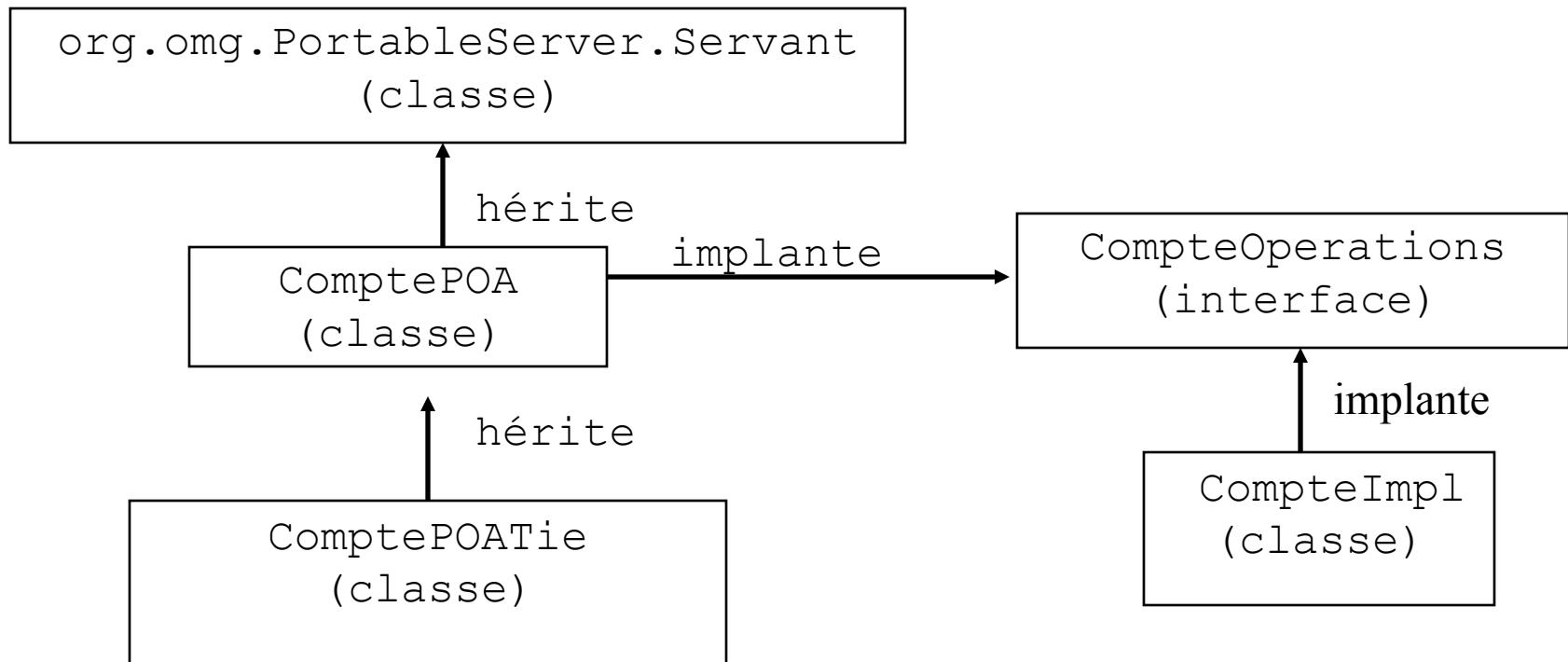
- L'objet de délégation est automatiquement généré à partir de la description IDL. (tie)
- En plus de l' objet de mise en œuvre, on doit **initialiser** l'objet de délégation et le **connecter** à l'adaptateur d'objets.
- Il faut ensuite **exporter** au client la référence de l'objet de délégation.

Délégation en Java

- En Java, la classe de délégation est une classe qui doit être initialisée avec la classe qui implante les opérations de l' interface.
- La classe de délégation comporte deux opérations importantes :
 - *son constructeur* qui prend en paramètre l'objet délégué.
 - un opération appelée « *_delegate* » qui retourne le délégué.

Hiérarchie de classes côté serveur

Mise en œuvre par délégation en Java



Programmation en Java

Objet de mise en œuvre

```
public class CompteImpl implements CompteOperations  
{  
    ...  
}
```

Serveur

```
CompteImpl unCompte_delegue = new CompteImpl();  
ComptePOATie unCompteImpl = new  
    ComptePOATie(unCompte_delegue);  
Compte unCompte = unCompteImpl._this(orb);
```