



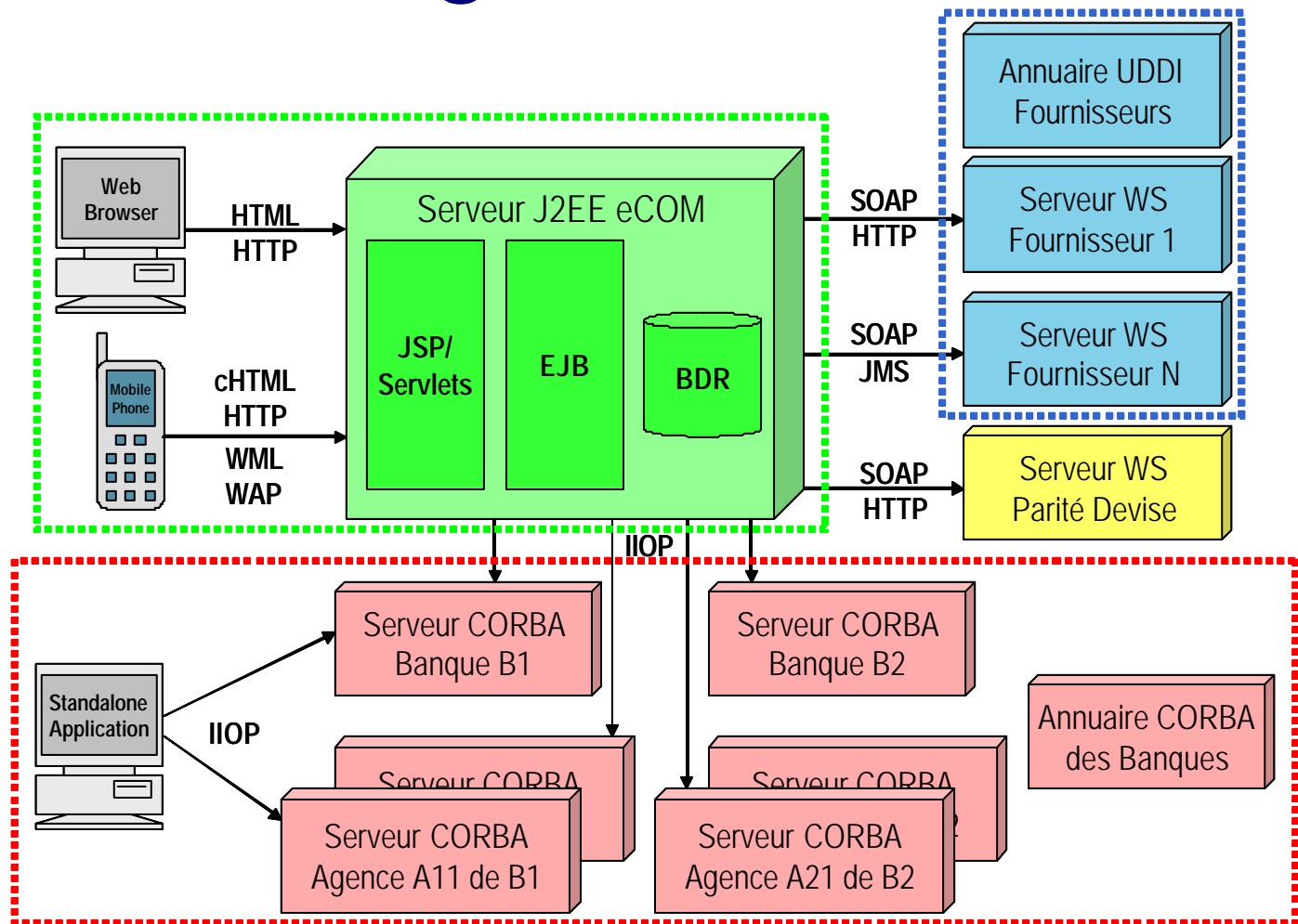
GICOM – M2GI SRR RICOM – M2RICM

Etape 2 : Sous-système bancaire

Sara Bouchenak, Didier Donsez
Pierre-Yves Gibello, Jean-François Méhaut

{Sara.Bouchenak, Didier.Donsez}@imag.fr
{Pierre-Yves.Gibello, Jean-François.Mehaut}@imag.fr

Architecture globale de GICOM



Etapes du projet

1. Extension de eCOM
2. *Sous-système bancaire*
3. Sous-système bancaire persistant
4. Sous-système bancaire persistant et fiable
5. Sécurisation des communications
6. Sous-système fournisseur

Motivations du projet GICOM / eCOM

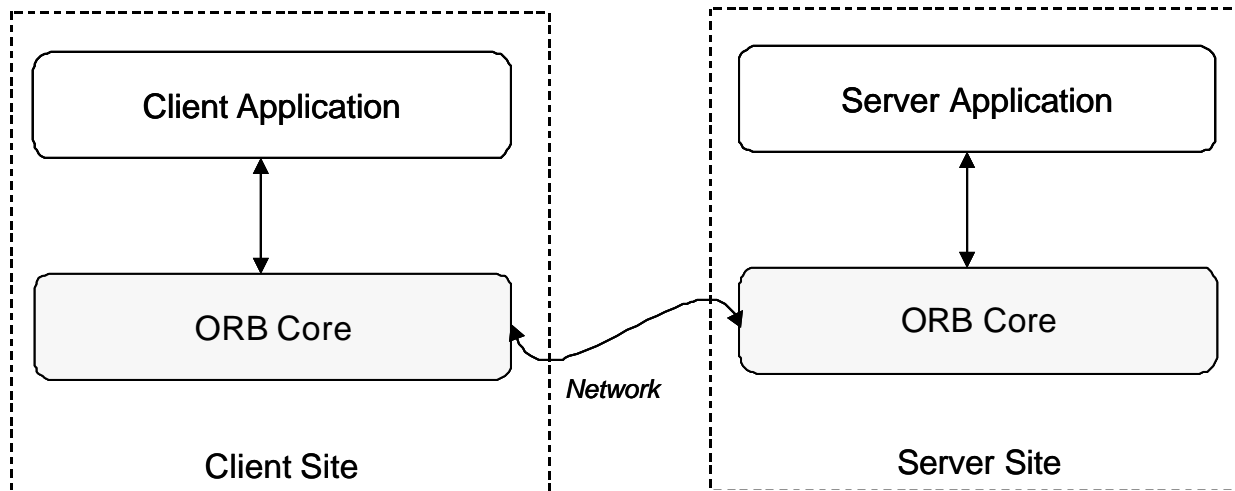
- Après avoir utilisé des fonctionnalités proposées par un middleware (persistance dans serveurs EJB, eCOM)
- Réaliser ces fonctionnalités dans le middleware (persistance dans CORBA, GICOM)

Plan

1. Présentation de CORBA
2. Présentation du sous-système bancaire
3. Organisation et méthodologie de travail

Qu'est-ce que CORBA ?

- CORBA (Common Object Request Broker Architecture)
- Un middleware (une plate-forme) pour l'exécution d'applications réparties



Motivations de CORBA

- **Répartition.** Abstractions nécessaires à une gestion transparente de la répartition
- **Hétérogénéité.** Cacher l'hétérogénéité des systèmes sous-jacents :
 - Réseau : Ethernet, ATM, Token ring, etc.
 - Architecture : x86, Sparc, etc.
 - Système d'exploitation : Unix, Windows, Solaris, etc.
 - Langage de programmation des applications : C, COBOL, Ada, Smalltalk, C++, Java, C#

Terminologie CORBA

- **Objet CORBA.** Une entité qui peut être localisée par un ORB et sur laquelle des requêtes peuvent être invoquées.
- **Requête.** Une opération qui peut être invoquée sur un objet CORBA.
- **Client.** Une entité qui invoque une requête sur un objet CORBA, localisé ou non sur le même site que le client.
- **Serveur.** Une application dans laquelle des objets CORBA existent.
- **Servant.** Implantation / représentation concrète d'un objet CORBA dans un serveur. On dit qu'un servant *incarne* un objet CORBA.
- **Référence d'objet.** Information qui permet d'identifier, de localiser et d'accéder à un objet CORBA (IOR: Interoperable Object Reference).

IDL (Interface Definition Language)

- Client invoque une requête sur un objet CORBA réparti

↳ Connaître l'interface fournie par l'objet

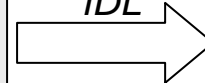
- IDL :

- ☐ Définition de l'interface d'objets CORBA
- ☐ Indépendamment de tout langage de programmation
- ☐ Spécification de l'IDL : <http://www.omg.org>

Exemple d'IDL pour *Hello*

```
module HelloApp
{
  interface Hello
  {
    string sayHello(in string name);
    oneway void shutdown();
  };
};
```

Compilation
IDL

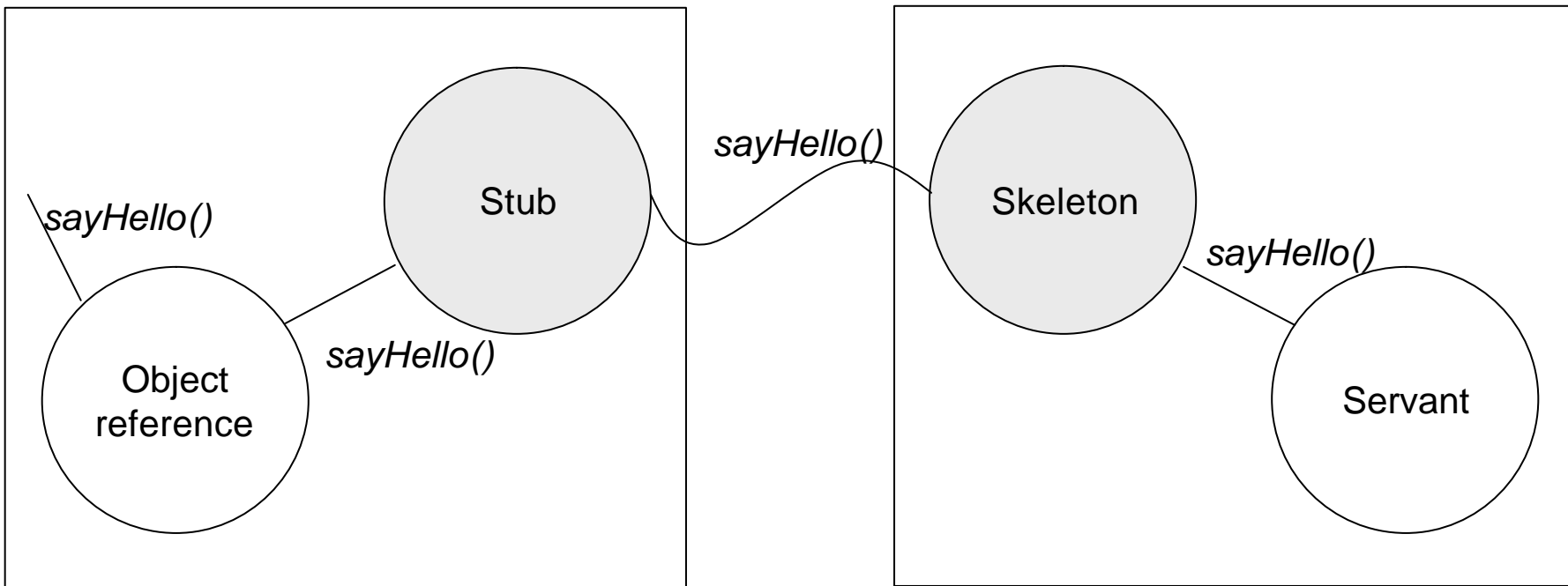


- Code du talon client
- Code du squelette serveur

Talon (*stub*) / Squelette (*skeleton*) : Gestion de la répartition

Client Site

Server Site



Etapes de développement d'une application CORBA

1. Déterminer les objets répartis (objets CORBA) de l'application et définir leurs interfaces en IDL
2. Compiler l'IDL vers un langage de programmation pour produire des talons clients / squelettes serveur
3. Ecrire les classes des servants qui incarnent les objets CORBA définis
4. Ecrire et compiler le programme principal du serveur
5. Ecrire et compiler le programme client

Etapes de développement de l'application Hello (1 / 5)

1. Déterminer les objets répartis : objets CORBA *Hello*
2. Définir leurs interfaces en IDL : fichier *Hello.idl*

```
module HelloApp
{
    interface Hello
    {
        string sayHello(in string name);
        oneway void shutdown();
    };
};
```

Etapes de développement de l'application Hello (2 / 5)

1. Compiler l'IDL vers un langage de programmation

- Utilisation du compilateur IDL/Java fourni par le Java CORBA du J2SDK 1.4 : commande *idlj Hello.idl*

2. Classes produites par le compilateur IDL

- **Hello** : interface Java correspondant à l'interface IDL *Hello*
- **HelloOperations** : interface d'un objet distribué CORBA ayant pour interface IDL *Hello*
- **HelloPOA** : classe abstraite définissant le squelette d'accès à un objet distribué CORBA d'interface *Hello*
- **HelloHelper** : classe gérant la création de talons pour les objets distribués CORBA d'interface *Hello*
- **_HelloStub** : classe définissant le talon associé à un objet distribué d'interface *Hello*

Etapes de développement de l'application Hello (3 / 5)

■ Ecrire la classe du servant qui incarne l'objet CORBA

```
public class HelloServant
    extends HelloPOA {
    private ORB orb;
    private String language;

    // constructor
    public HelloServant(String language,
                        ORB orb){
        this.language = language;
        this.orb      = orb;
    }

    public void shutdown() {
        if (orb != null) {
            orb.shutdown(false);
        }
    }
}
```

```
public String sayHello(String name) {
    if (language.equals("fr")) {
        return "\nSalut " + name + " !!\n";
    } else if (language.equals("it")){
        return "\nCiao " + name + " !!\n";
    } else if (language.equals("es")){
        return "\nHola " + name + " !!\n";
    }else if(language.equals("en-us")){
        return "\nHi " + name + " !!\n";
    } else if (language.equals("de")){
        return "\nHallo " + name + " !!\n";
    } else {
        return "\nHello " + name + " !!\n";
    }
}
```

Etapes de développement de l'application Hello (4 / 5)

■ Ecrire et compiler le programme principal du serveur

```
public class HelloServer {  
  
    public static void main(String args[]) {  
        // Create and initialize the ORB  
        ORB orb = ORB.init(args, null);  
  
        // Reference to rootpoa, activate POAManager  
        CORBA.Object poa_ref = orb.resolve_  
            initial_references("RootPOA");  
        POA rootpoa = POAHelper.narrow(poa_ref);  
        rootpoa.the_POAManager().activate();  
  
        // Create servant and register it in ORB  
        HelloServant helloServant =  
            new HelloServant("en-us", orb);  
  
        // Get object reference from the servant  
        CORBA.Object ref = rootpoa.servant_  
            to_reference(helloServant);  
        Hello href = HelloHelper.narrow(ref);  
    }  
}
```

```
String bindingname = args[1];  
// Get the root naming context  
CORBA.Object nsRef = orb.resolve_  
    initial_references("NameService");  
// Use NamingContextExt (INS spec.)  
NamingContextExt ncRef =  
    NamingContextExtHelper.narrow(nsRef);  
  
// Bind the Object Reference in Naming  
NameComponent path[] = ncRef.  
    to_name(bindingname);  
ncRef.rebind(path, href);  
  
// Wait for invocations from clients  
System.out.println("HelloServer ready  
    and waiting ...");  
orb.run();  
  
}
```

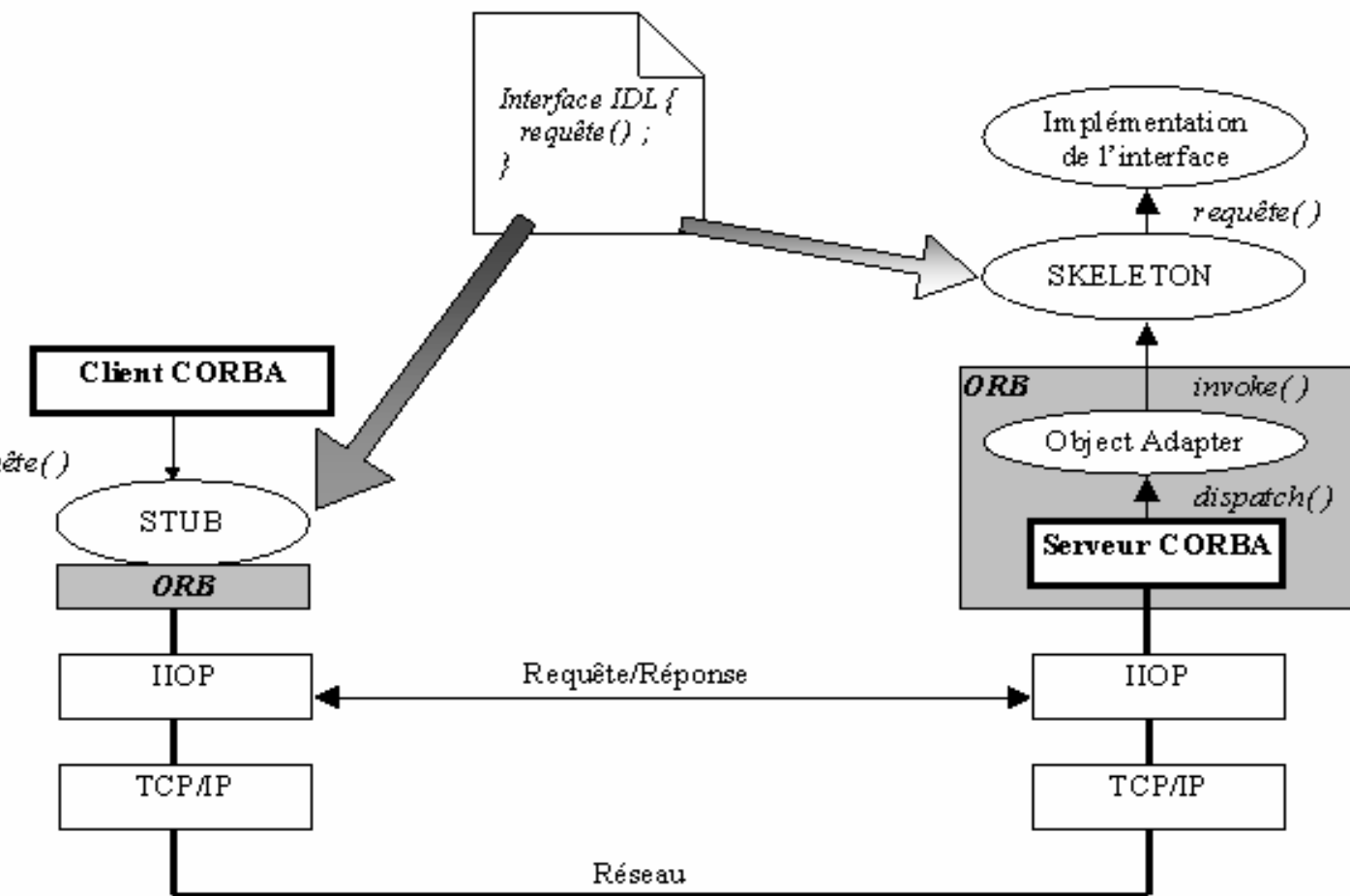

Etapes de développement de l'application Hello (5 / 5)

■ Ecrire et compiler le programme principal du client

```
public class HelloClient {  
    public static void main(String args[]) {  
        // Create and initialize the ORB  
        ORB orb = ORB.init(args, null);  
  
        // Get the root naming context  
        CORBA.Object ncoRef = orb.resolve_  
            initial_references("NameService");  
        // Use NamingContextExt (INS spec.)  
        NamingContextExt ncRef =  
            NamingContextExtHelper.narrow(ncoRef);
```

```
        // Resolve the Object Reference in Naming  
        String bindingname = args[1];  
        CORBA.Object objRef = ncRef.  
            resolve_str(bindingname);  
        Hello hello = HelloHelper.narrow(objRef)  
  
        // Request methods on the CORBA object  
        System.out.println("Obtained a handle on  
            server object: " + hello);  
        System.out.println(  
            hello.sayHello("World"));  
        hello.shutdown();  
    }  
}
```

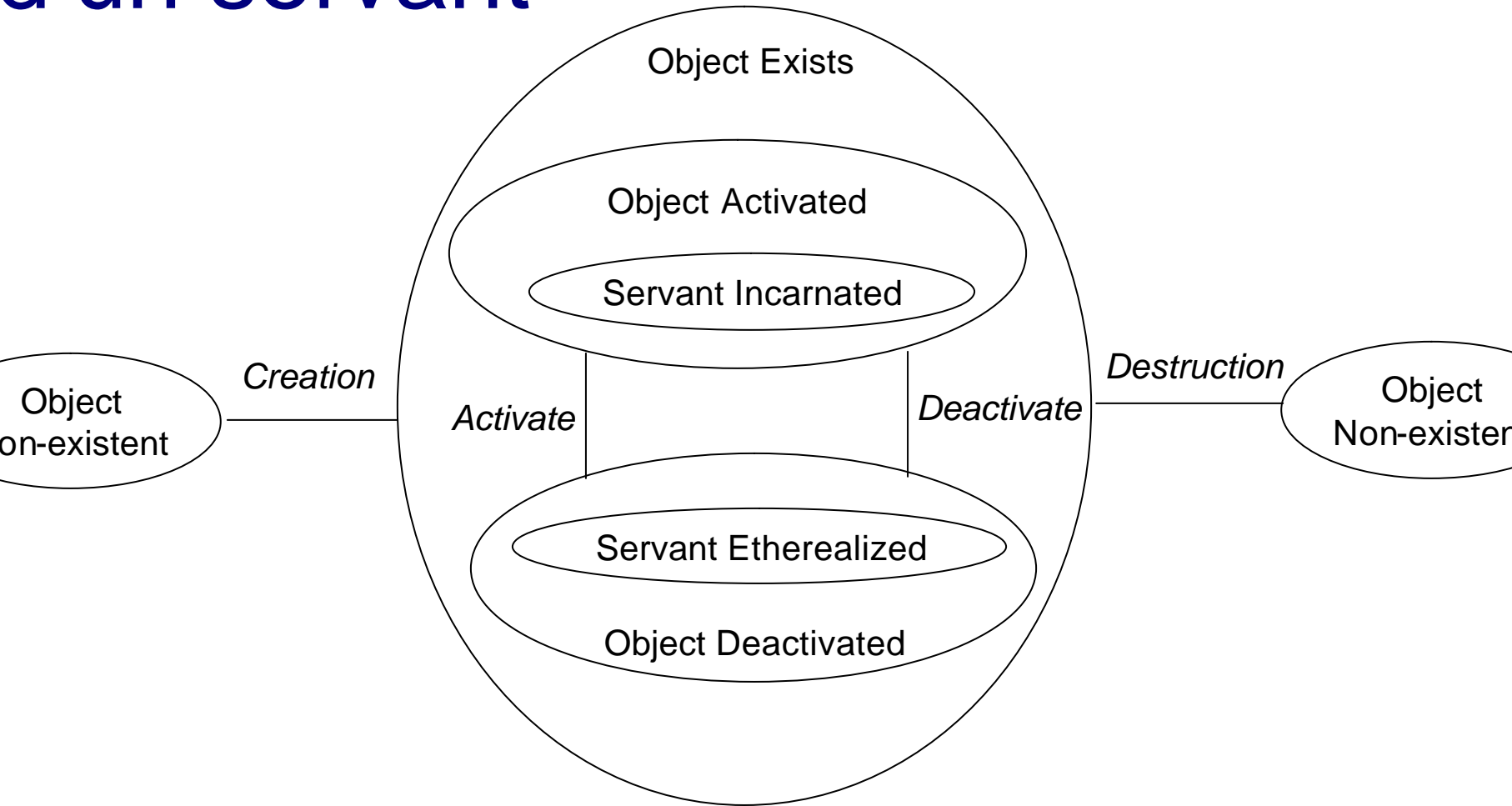
POA (Portable Object Adapter)



⇒ Rôle du POA :

- Créer les références d'objets
- Activer les objets
- Transmettre les requêtes aux servants

Cycles de vie d'un objet CORBA et d'un servant



Modes d'activation d'objets CORBA

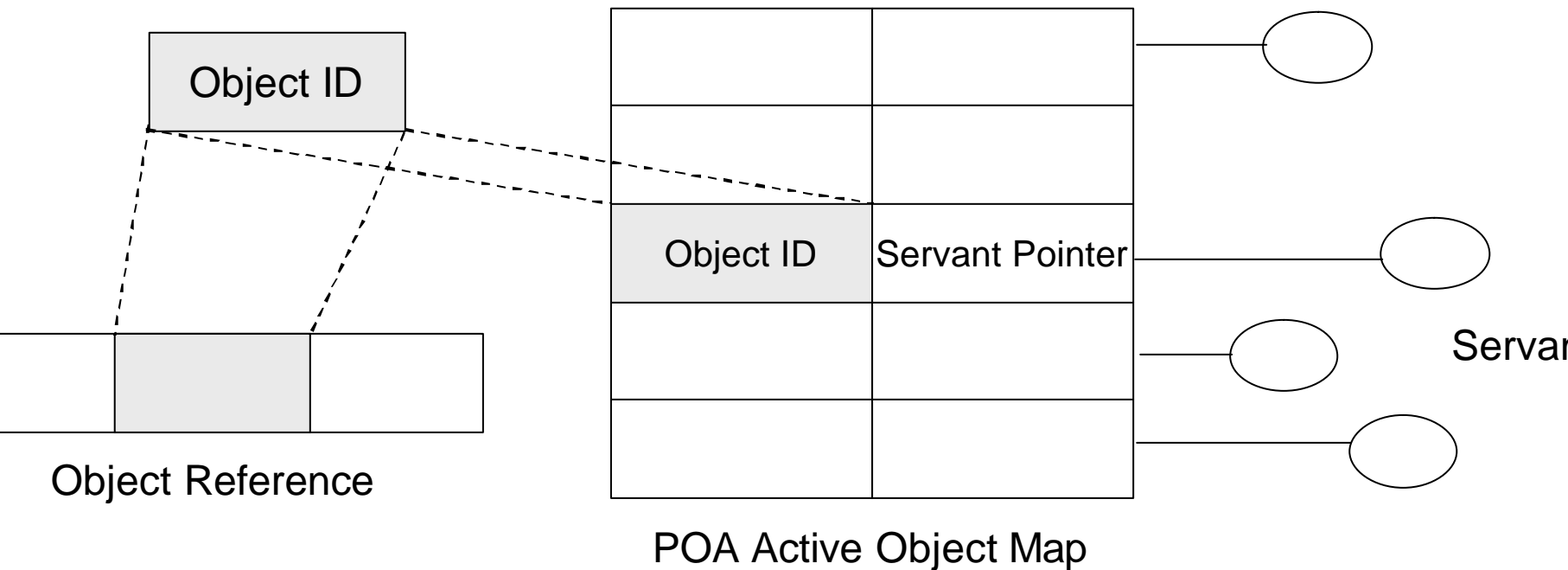
■ Mode implicite :

- La création d'une référence d'objet CORBA auprès d'un POA entraîne automatiquement l'activation de l'objet

■ Mode explicite :

- Le programmeur doit explicitement activer un objet CORBA auprès d'un POA
- *PortableServer.POA.activate_object*
- *PortableServer.POA.activate_object_with_id*

Identification d'un objet CORBA



Modèles d'implantation d'un objet distribué CORBA

■ Modèle par héritage :

- L'implantation de l'objet CORBA (le servant) hérite squelette serveur

■ Modèle par délégation :

- L'implantation de l'objet CORBA (le servant) est un délégué associé au squelette serveur

Application Hello selon le modèle par délégation

1. Compiler l'IDL vers un langage de programmation

- Utilisation du compilateur IDL/Java : commande
idlj - tie Hello.idl

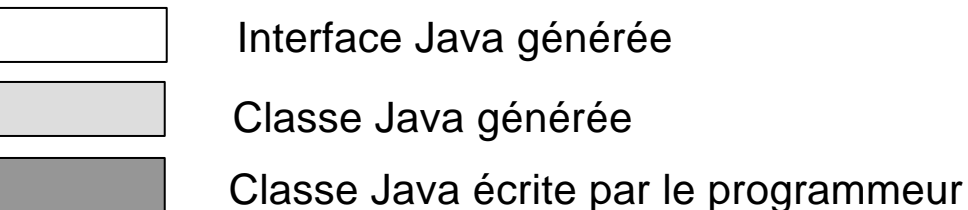
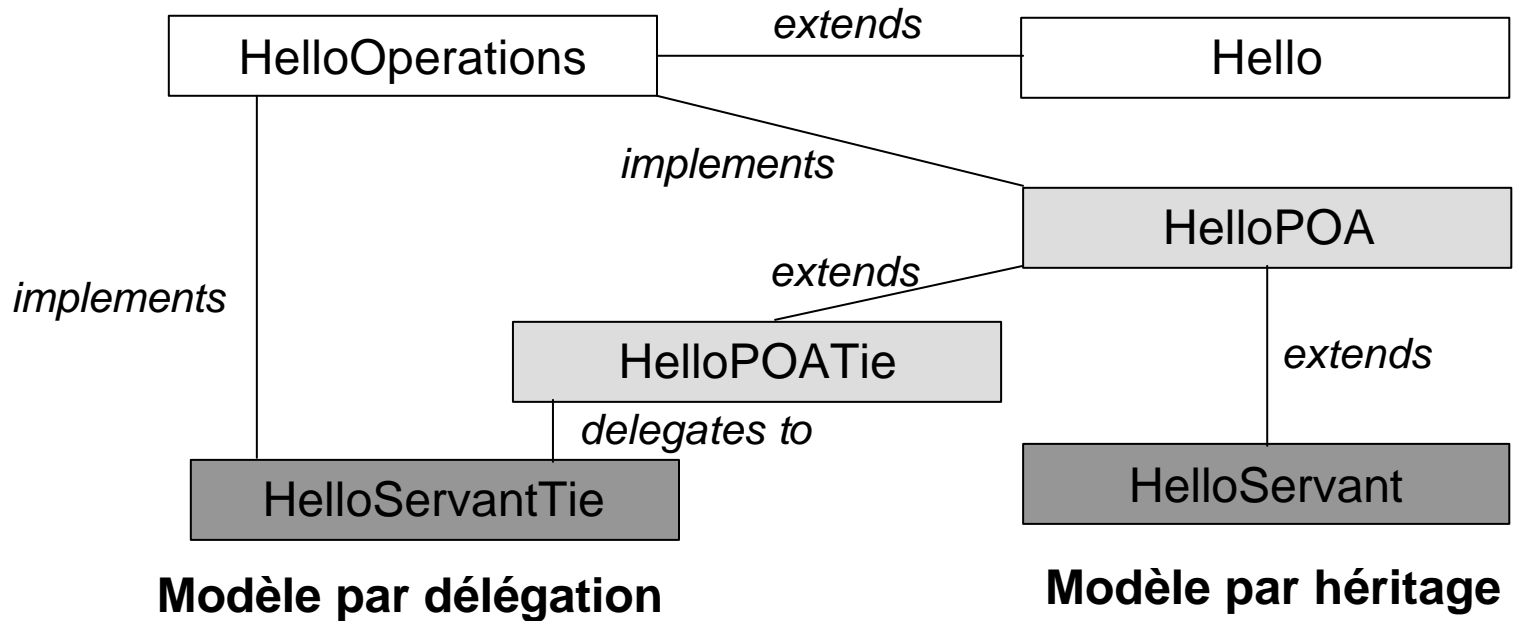
2. Classes générées par le compilateur IDL : une classe supplémentaire par rapport au modèle par héritage

- **HelloPOATie** : classe implantant le squelette associé aux objets d'interface *Hello*.
Ce squelette utilise un objet *HelloTieServant* comme délégué.

```
public class HelloServantTie  
extends HelloOperations { ... }
```

```
public class HelloServer {  
    ...  
    // Create servant  
    HelloServantTie hello = new HelloServantTie("en-us")  
    HelloPOATie helloServant = new HelloPOATie(hello);  
}
```

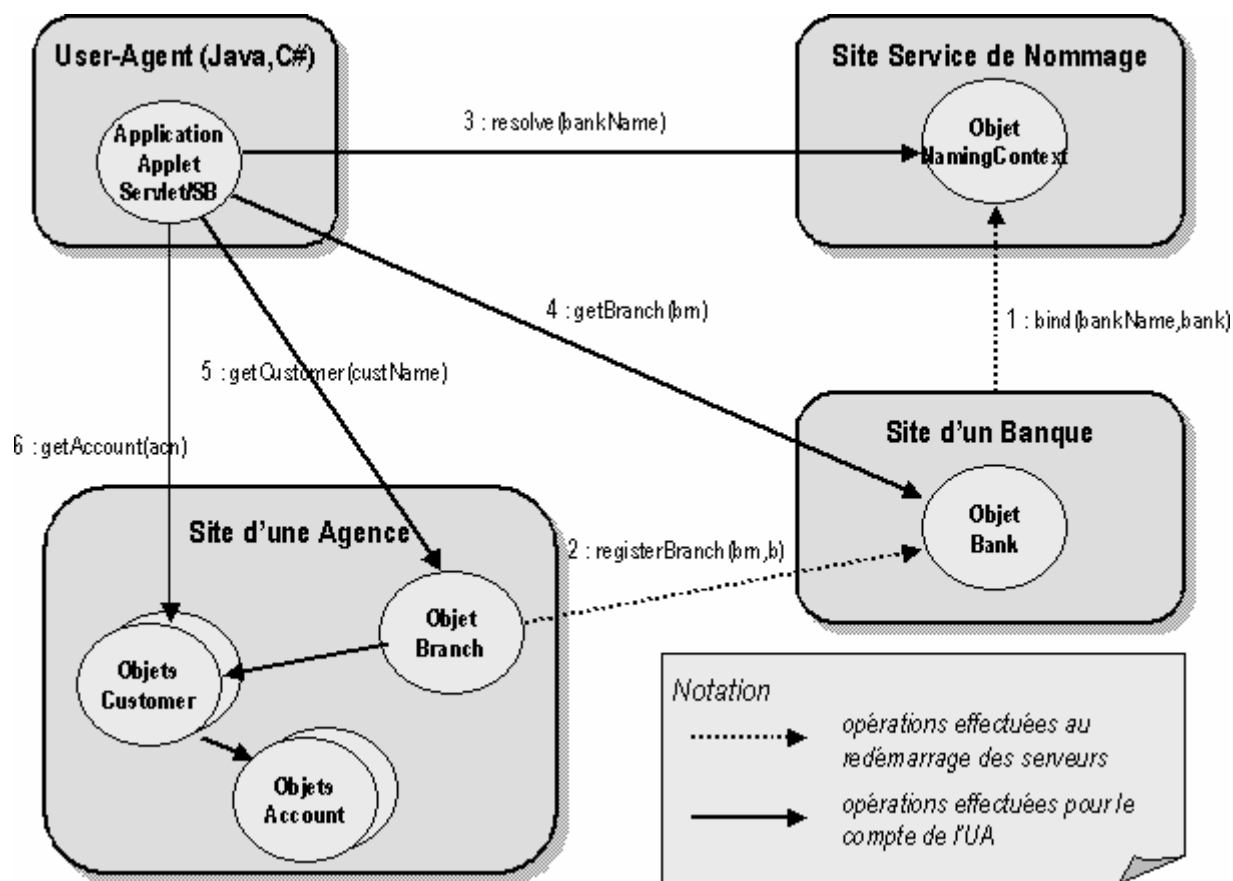
Application Hello : héritage vs. délégation



Plan

1. Présentation de CORBA
 - Motivations et terminologie CORBA
 - IDL CORBA
 - Etapes de développement d'une application CORBA
 - POA
 - Modèles d'implantation d'un objet CORBA
2. *Présentation du sous-système bancaire*
3. Organisation et méthodologie de travail

Architecture proposée



Interfaces IDL proposées

```
module bankServices {  
  
  interface Bank {  
  
    readonly attribute string number;  
    readonly attribute string name;  
    readonly attribute Address addr;  
  
    void registerBranch(in string number,  
                        in Branch b);  
    void unregisterBranch(in string number);  
    Branch findBranch(in string number);  
    Branches findAllBranches();  
  
  };  
  
  interface Branch {  
  
    readonly attribute string number;  
    readonly attribute string name;  
    readonly attribute Address addr;  
    readonly attribute Bank refBank;
```

```
    Customer createCustomer(in string number,  
                            in string name, in Gender gend,  
                            in Address addr);  
    void removeCustomer(in string number);  
    Customer findCustomer(in string number);  
    Customers findAllCustomers();  
  
  };  
  
  interface Customer {  
  
    readonly attribute string number;  
    readonly attribute string name ;  
    readonly attribute Gender gend ;  
    attribute Address addr ;  
    readonly attribute Branch refBranch ;  
  
    Account createAccount(in Currency  
                          balance);  
    void removeAccount(in string  
                       accountnumber);  
    Account findAccountByPrimaryKey(in string  
                                     accountnumber);  
    Accounts findAllAccounts();  
  
  }; };
```

Suivre les étapes de développement proposées

1. Déterminer les objets répartis (objets CORBA) de l'application et définir leurs interfaces en IDL
2. Compiler l'IDL vers un langage de programmation pour produire des talons clients / squelettes serveur
3. Ecrire les classes des servants qui incarnent les objets CORBA définis
4. Ecrire et compiler le programme principal du serveur
5. Ecrire et compiler le programme client

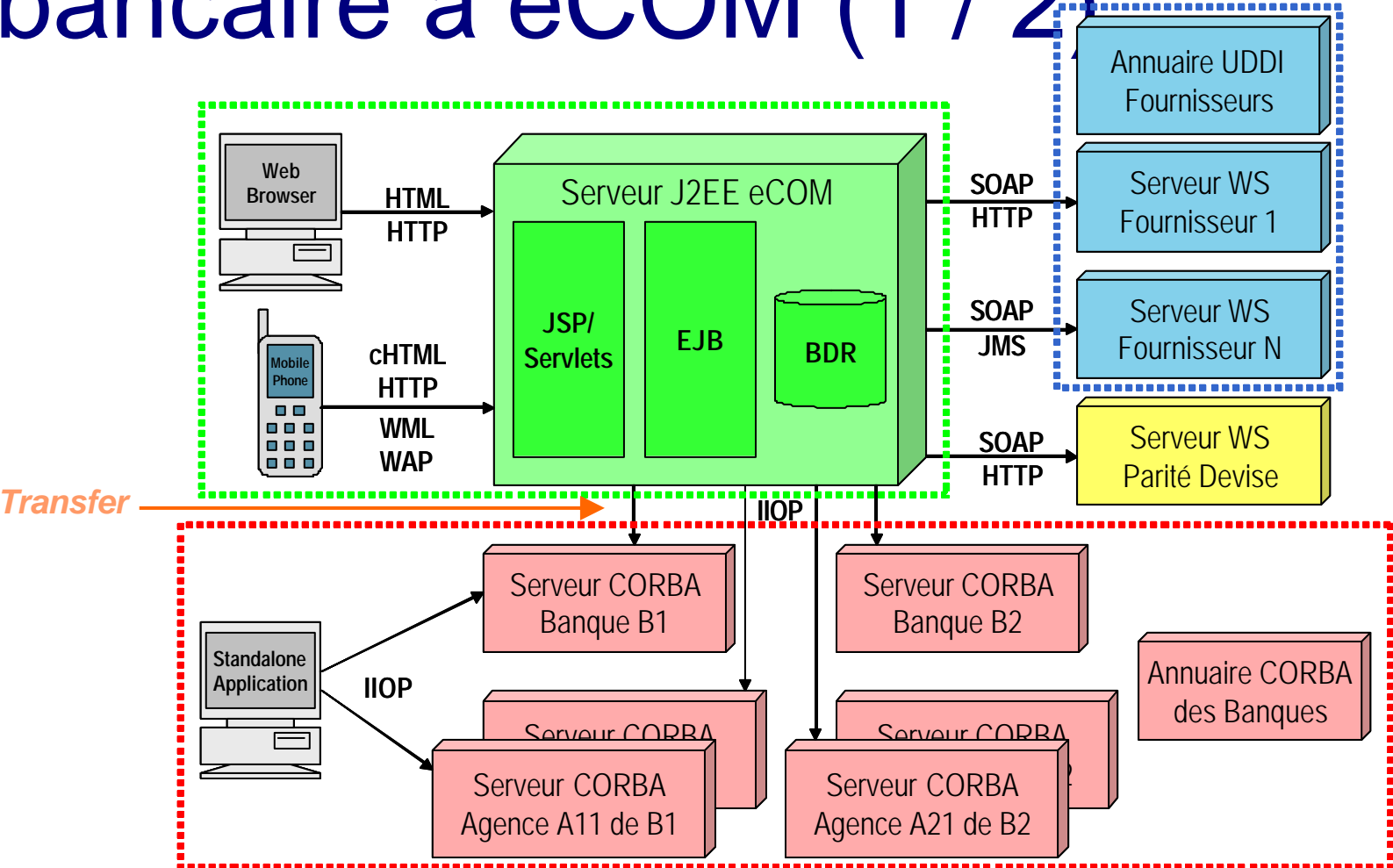
Programme principal du serveur : Serveur générique

- Ce qui est fourni : ([exemples/genericserver/src/GenericServer.java](#))
 - Programme d'un serveur CORBA générique Suivant le modèle par héritage
 - Avec un RootPOA
 - Sans paramètres
- A faire : Etendre le serveur générique pour prendre en compte :
 - Le modèle par délégation
 - L'ajout d'autres POA
 - Des paramètres utilisés lors de l'instanciation des services *client* et *compte*
- Important
 - Entre un serveur de *banque* et un serveur d'*agence*, la seule chose qui change le fichier de configuration (config.properties) et pas l'implantation du serveur CORBA

Programme principal du client : User-Agent

- Opérations effectuées par un client User-Agent :
 - Création, consultation, modification, suppression de clients et de comp
- A faire : deux User-Agents à programmer
 - Langage de programmation
 - Un écrit en Java
 - L'autre écrit en C#
 - ⇒ exemple de client C# dans [exemples/hello/src/HelloClient.cs](#)
 - Mode de commande
 - Un en mode ligne de commande (à récupérer de eCOM)
 - L'autre en mode graphique (optionnel)
 - ⇒ ne pas passer trop de temps sur le mode graphique

Intégration du sous-système bancaire à eCOM (1 / 2)



Intégration du sous-système bancaire à eCOM (1 / 2)

■ Etape 1 : Extension de eCOM

- Un nouveau bean *FundsTransfer* (remplaçant du bean *Account*)
- Opération de transfert de fonds multi-comptes entre les serveurs bancaires

■ Etape 2 : Intégration

- eCOM et sous-système bancaire
- Via le bean *FundsTransfer*

Plan

1. Présentation de CORBA
2. Présentation du sous-système bancaire
 - Architecture et interfaces IDL proposées
 - Serveur générique et User-Agents
 - Intégration à eCOM
3. *Organisation et méthodologie de travail*

Développement (1 / 2)

■ Automatisation

- Ant

■ Environnement

- Variables à définir JAVA_HOME, ANT_HOME, PATH

■ Organisation logicielle

- *idl* : fichiers IDL
- *src* : fichiers de programmes sources
- *generated* : fichiers des classes générées par la compilation IDL
- *classes* : fichiers des classes générées par la compilation Java
- *build* : fichiers jar et .exe pour exécuter les serveurs et UA

■ Nomres de programmation

- Java, C#
- Outils de vérification de respect des normes, ex. JavaStyle

Développement (2 / 2)

■ Documentation

- ☐ Documentation globale du projet
- ☐ JavaDoc, JavaSrc

■ Traces

- ☐ Pas de traces avec *System.out.println(...)*
- ☐ Utiliser des *loggers* (*java.util.logging.Logger*)

■ Métriques sur les sources

- ☐ Nombre de packages
- ☐ Nombre de classes
- ☐ Nombre de lignes de code
- ☐ Nombre de commentaires
- ☐ Visibilité des attributs, méthodes, classes
- ☐ etc.

Déploiement

■ Automatisation

- Déploiement des serveurs sur une dizaine de machines
- Soutenance : moins de 15 minutes, automatisation

■ Configurabilité

- Pas de chemins absolus dans vos fichiers : portabilité Unix/Windows
- Ne pas fixer les adresses/noms de serveurs dans vos programmes

■ Sécurité

- Chaque serveur a son propre fichier *java.policy*
- Limiter l'accès du serveur aux ressources de la machines

■ Organisation hiérarchique

- Un répertoire par serveur à déployer
- Fichier de propriétés, *java.policy*, répertoire lib pour les Jar

■ Performances

- Mesurer les performances du serveur Web, Apache JMeter,
- 10 produits achetés dans 2 magasins, 3 modifications, 5 visualisations du cadd
achat

Organisation

■ Se partager les tâches

- Dans un trinôme, spécialités identifiées pour chaque membre
- Spécialiste outils (Ant, Logger, ...)
- Architecte
- Chef de projet
- etc.

■ Paralléliser les tâches

- Des étapes de projets peuvent être conduites en parallèle
- Dépendances entre les étapes
 - Etape 1 : dépend de eCOM
 - Etape 2 : ne dépend de rien, intégration avec eCOM
 - Etape 3 : dépend de l'étape 2
 - Etape 4 : dépend de l'étape 3
 - Etape 5 : dépend des étapes 1 et 2
 - Etapes 6a et 6b : dépendent de l'étape 1

Documentation

■ CORBA

- <http://www.omg.org>
- **Advanced CORBA Programming with C++**. J. Henning, S. Vinoski, Addison-Wesley 2002.
- **Java Programming with CORBA**. G. Brose, A. Vogel, K. Duddy, John Wiley & Sons, 2001.

■ Java / CORBA

- <http://java.sun.com/j2se/1.4.1/docs/guide/idl/>
- API du J2SDK 1.4 : *org.omg.**

Plan

1. Présentation de CORBA
2. Présentation du sous-système bancaire
3. Organisation et méthodologie de travail
 - Développement
 - Déploiement
 - Organisation
 - Documentation