

# **Intergiciels à Objets Répartis : Exemple de Java RMI et de .NET Remoting**

*Samia Bouzefrane*

Maître de Conférences

CEDRIC –CNAM

samia.bouzefrane@cnam.fr  
<http://cedric.cnam.fr/~bouzefra>

## Introduction

## Notion d'Intergiciel (Middleware)

- **Intergiciel** : est une classe de logiciels systèmes qui permet d'implanter une approche répartie: exemple CORBA
  - ✓ Fournit une API d'interaction et de communication: interactions de haut niveau pour des applications réparties: exemple invocation distante de méthode pour des codes objets.
  - ✓ Fournit un ensemble de services utiles pour des applications s'exécutant en environnement réparti: désignation, cycle de vie, sécurité, transactionnel, etc.
  - ✓ Fonctionne en univers ouvert (supporte des applications tournant sur des plate-formes matérielles et logicielles différentes).

## Principales Catégories d'Intergiciels

- Intergiciels à **messages**
  - ✓ **MOM: Message Oriented Middleware** : JMS (Java Message Service de J2EE de Sun), Microsoft Message Queues Server, etc.
- Intergiciels de **bases de données** (ODBC)
- Intergiciels à **appel de procédure distante** (DCE)
- Intergiciels à **objets répartis** (CORBA, JAVA RMI, .NET Remoting)
- Intergiciels à **composants** (EJB, CCM, Web services)

## **Communication dans un contexte réparti: exemple de l'invocation de méthodes distantes**

## L'appel de procédure distante

### ➤ Mode de communication entre deux programmes distants

- ✓ Un programme jouera le rôle de client
- ✓ L'autre programme jouera le rôle de serveur

### ➤ Le but de l'interaction du client avec le serveur

- ✓ le client peut faire exécuter à distance une **procédure** par le serveur.

### ➤ Service basique (API d'appel de procédure distante, bibliothèque système)

*/\* Côté client : invoque l'appel distant et récupère le résultat\*/*

**invoque (id\_client, id\_serveur, nom\_procedure, paramètres);**

*/\* Côté serveur : reçoit, traite un appel et répond \*/*

**traite (id\_client, id\_serveur, nom\_procedure, paramètres);**

### ➤ Service niveau langage (API définie au niveau du langage de programmation)

*/\* Côté client : on invoque une procédure localisée à distance\*/*

**ref\_objet\_serveur.nom\_procedure (paramètres);**

*/\* Côté serveur : on déploie l'objet qui implante la procédure\*/*

**method nom\_procedure (paramètres);**

## **Avantages de l'appel de procédure distante**

- **S'affranchir du côté basique des communications en mode message**  
Ne pas avoir à programmer des échanges au niveau réseau en mode message.
- **Utiliser une structure familière: l'appel de procédure**  
Problème: ne pas ignorer les différences centralisé/réparti.
- **Disposer de mécanismes modernes de programmation**
  - ✓ Vision modulaire des applications réparties (en approche objets répartis ou par composants sur étagères).
  - ✓ Réutilisation en univers réparti.

# Les implantations de l'appel de procédure distante

## ➤ Les approches à RPC traditionnelles

SUN RPC

## ➤ Approches à RPC intégrées dans les systèmes d'objets répartis

- OMG CORBA (Object Management Group - Common Object Request Broker Architecture )
- SUN Java RMI
- Microsoft .NET

## ➤ Approches à RPC intégrées dans les systèmes de composants

- SUN J2EE EJB: Java 2 (Platform) Enterprise Edition – Enterprise Java Beans
- OMG CCM: Object Management Group - Corba Component Model
- WS-SOAP: Web Services - Simple Object Access Protocol



# ***Java Remote Method Invocation (RMI) ou les invocations de méthodes Java distantes***

## Rappel : Appel local (interface et objet)

```
public interface ReverseInterface {  
    String reverseString(String chaine);  
}
```

```
public class Reverse implements ReverseInterface  
{  
    public String reverseString (String ChaineOrigine){  
  
        int longueur=ChaineOrigine.length();  
        StringBuffer temp=new StringBuffer(longueur);  
        for (int i=longueur; i>0; i--) {  
            temp.append(ChaineOrigine.substring(i-1, i));  
        }  
        return temp.toString();  
    }  
}
```

## Rappel : Appel local (programme appelant )

```
import ReverseInterface;  
  
public class ReverseClient  
{  
    public static void main (String [] args)  
    {  
        Reverse rev = new Reverse();  
        String result = rev.reverseString (args [0]);  
        System.out.println ("L'inverse de "+args[0]+" est  
"+result);  
    }  
}  
  
$javac *.java  
$java ReverseClient Alice  
L'inverse de Alice est ecilA  
$
```

## Qu'attend t-on d'un objet distribué ?

Un objet distribué doit pouvoir être vu comme un objet « normal ».

Soit la déclaration suivante :

**ObjetDistribue monObjetDistribue;**

- On doit pouvoir invoquer une méthode de cet objet situé sur une autre machine de la même façon qu'un objet local :

**monObjetDisribue.uneMethodeDeLOD( ) ;**

- On doit pouvoir utiliser cet objet distribué sans connaître sa localisation. On utilise pour cela un service sorte d'annuaire, qui doit nous renvoyer son adresse.

```
monObjetDistribue=  
ServiceDeNoms.recherche( 'myDistributedObject' );
```

- On doit pouvoir utiliser un objet distribué comme paramètre d'une méthode locale ou distante.

```
x=monObjetLocal.uneMethodeDeLOL(monObjetDistribue);  
  
x= monObjetDistribue.uneMethodeDeLOD(autreObjetDistribue);
```

- On doit pouvoir récupérer le résultat d'un appel de méthode sous la forme d'un objet distribué.

```
monObjetDistribue=autreObjetDistribue.uneMethodeDeLOD( );
```

```
monObjetDistribue=monObjetLocal.uneMethodeDeLOL( );
```

# Java RMI

## *Remote Method Invocation*

permet la communication entre machines virtuelles Java (JVM) qui peuvent se trouver physiquement sur la même machine ou sur deux machines distinctes.

## Présentation

RMI est un système d'objets distribués constitué uniquement d'objets java ;

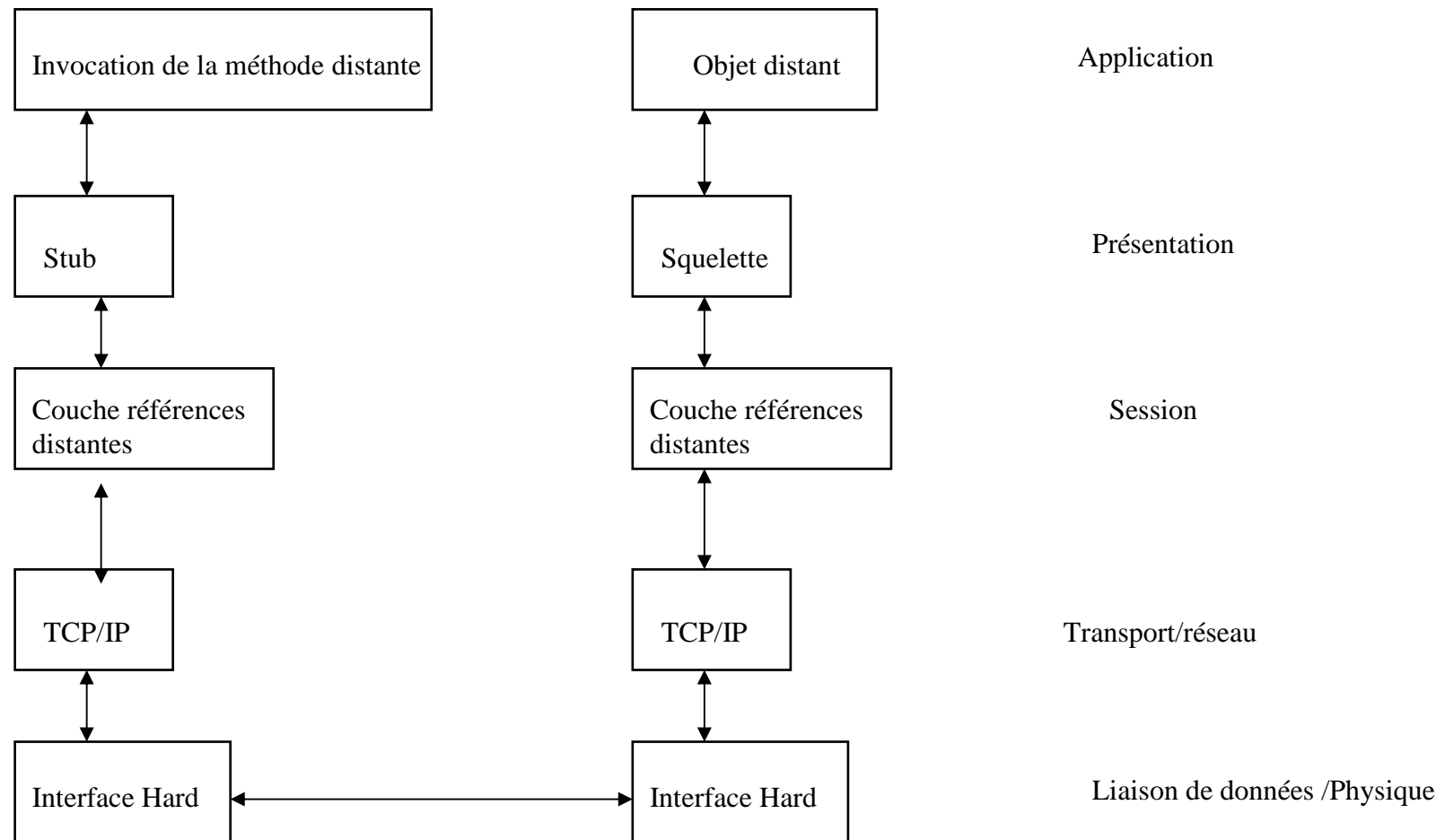
- RMI est une **A**pplication **P**rogramming **I**nterface (intégrée au JDK 1.1 et plus) ;
- Développé par Sun ;



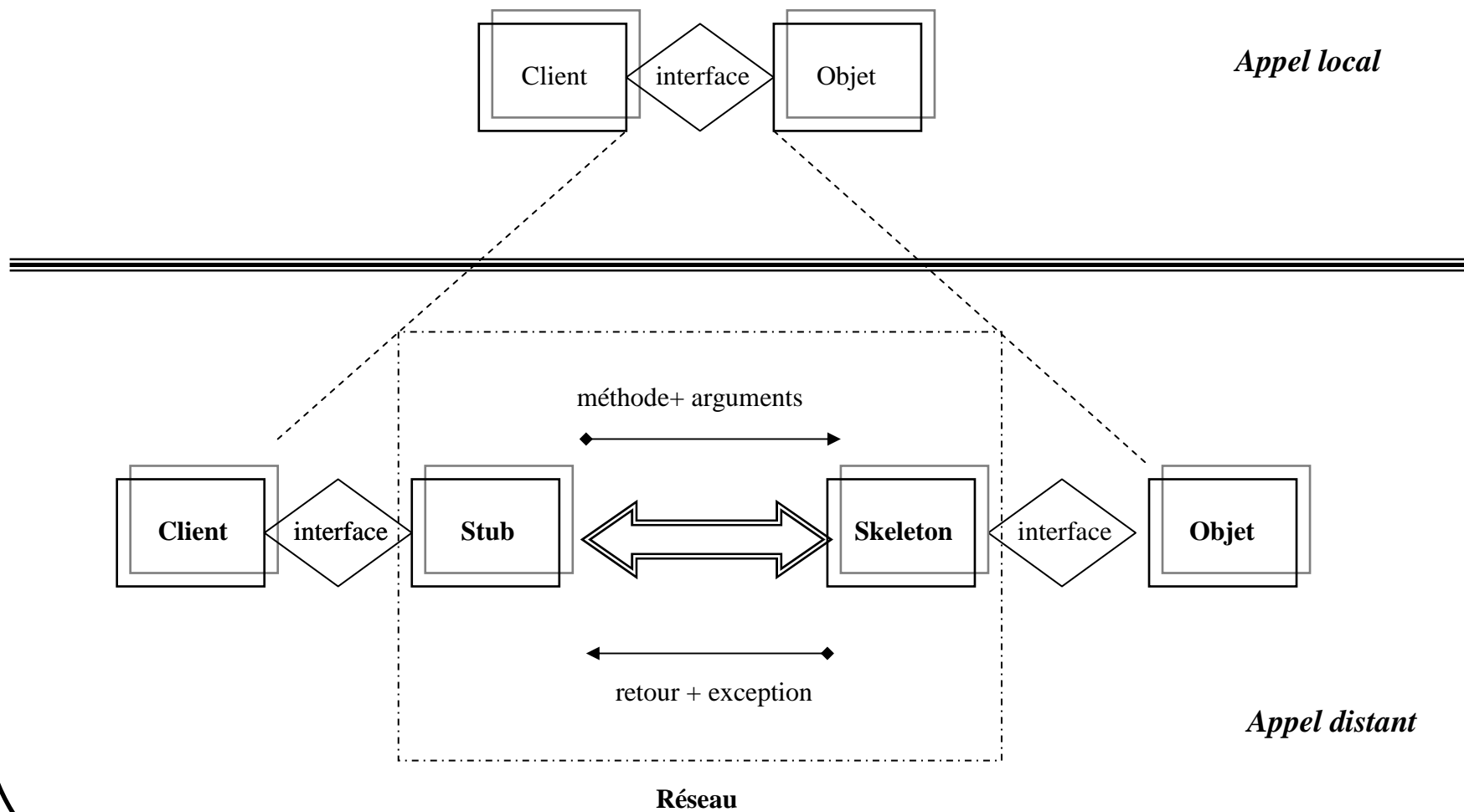
# RMI

- Mécanisme qui permet l'appel de méthodes entre objets Java qui s'exécutent éventuellement sur des JVM distinctes ;
- L'appel peut se faire sur la même machine ou bien sur des machines connectées sur un réseau ;
- Utilise les sockets ;
- Les échanges respectent un protocole propriétaire : **Remote Method Protocol** ;
- RMI repose sur les classes de sérialisation.

# Architecture



# Appel local versus Appel à distance



## Les amorces (Stub/Skeleton)

- Elles assurent le rôle d'adaptateurs pour le transport des appels distants
- Elles réalisent les appels sur la couche réseau
- Elles réalisent l'assemblage et le désassemblage des paramètres (*marshalling, unmarshalling*)
- Une référence d'objets distribué correspond à une référence d'amorce
- Les amorces sont créées par le générateur **rmic**.

## Les Stubs (Souches)

- Représentants locaux de l'objet distribué ;
- Initient une connexion avec la JVM distante en transmettant l'invocation distante à la couche des références d'objets ;
- Assemblent les paramètres pour leur transfert à la JVM distante ;
- Attendent les résultats de l'invocation distante ;
- Désassemblent la valeur ou l'exception renvoyée ;
- Renvoient la valeur à l'appelant ;
- S'appuient sur la sérialisation.

## Les Skeleton (squelettes)

- Désassemblent les paramètres pour la méthode distante ;
- Font appel à la méthode demandée ;
- Assemblage du résultat (valeur renvoyée ou exception) à destination de l'appelant.

## La couche des références d'objets

### Remote Reference Layer

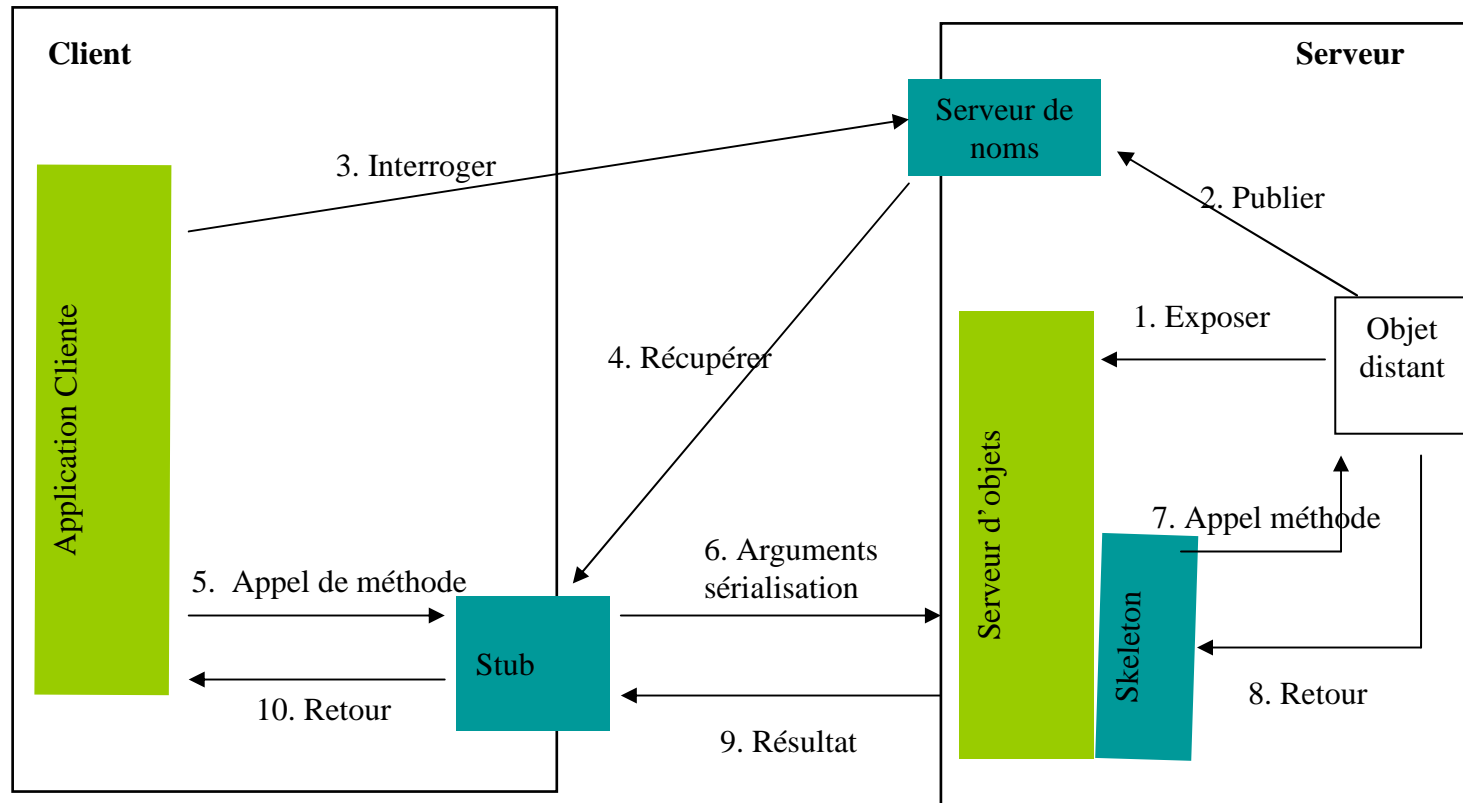
- Permet d'obtenir une référence d'objet distribué à partir de la référence locale au stub ;
- Cette fonction est assurée grâce à un service de noms **rmiregister** (qui possède une table de hachage dont les clés sont des noms et les valeurs sont des objets distants) ;
- Un unique **rmiregister** par JVM ;
- **rmiregister** s'exécute sur chaque machine hébergeant des objets distants ;
- **rmiregister** accepte des demandes de service sur le port 1099;

## La couche transport

- réalise les connexions réseau basées sur les flux entre les JVM
- emploie un protocole de communication propriétaire (**JRMP**: Java Remote Method Invocation) basé sur TCP/IP
- Le protocole JRMP a été modifié afin de supprimer la nécessité des squelettes car depuis la version 1.2 de Java, une même classe skeleton générique est partagée par tous les objets distants.



## Etapes d'un appel de méthode distante



## Développer une application avec RMI : Mise en œuvre

1. Définir une interface distante (**Xyy.java**) ;
2. Créer une classe implémentant cette interface (**XyyImpl.java**) ;
3. Compiler cette classe (**javac XyyImpl.java**) ;
4. Créer une application serveur (**XyyServer.java**) ;
5. Compiler l'application serveur ;
6. Créer les classes stub et skeleton à l'aide de **rmic XyyImpl\_Stub.java** et **XyyImpl\_Skel.java** (**Skel** n'existe pas pour les versions >1.2) ;
7. Démarrage du registre avec **rmiregistry** ;
8. Lancer le serveur pour la création d'objets et leur enregistrement dans **rmiregistry** ;
9. Créer une classe cliente qui appelle des méthodes distantes de l'objet distribué (**XyyClient.java**) ;
10. Compiler cette classe et la lancer.

## Inversion d'une chaîne de caractères à l'aide d'un objet distribué

Invocation distante de la méthode **reverseString()** d'un objet distribué qui inverse une chaîne de caractères fournie par l'appelant.

On définit :

- **ReverseInterface.java** : interface qui décrit l'objet distribué
- **Reverse.java** : qui implémente l'objet distribué
- **ReverseServer.java** : le serveur RMI
- **ReverseClient.java** : le client qui utilise l'objet distribué

## Fichiers nécessaires

### *Côté Client*

- l'interface : `ReverseInterface`
- le client : `ReverseClient`

### *Côté Serveur*

- l'interface : `ReverseInterface`
- l'objet : `Reverse`
- le serveur d'objets : `ReverseServer`

## Interface de l'objet distribué

- Elle est partagée par le client et le serveur ;
- Elle décrit les caractéristiques de l'objet ;
- Elle étend l'interface **Remote** définie dans **java.rmi**.

Toutes les méthodes de cette interface peuvent déclencher une exception du type **RemoteException**.

Cette exception est levée :

- si connexion refusée à l'hôte distant
- ou bien si l'objet n'existe plus,
- ou encore s'il y a un problème lors de l'assemblage ou le désassemblage.

## Interface de la classe distante

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface ReverseInterface extends Remote {  
    String reverseString(String chaine) throws RemoteException;  
}
```

## Implémentation de l'objet distribué

- L'implémentation doit étendre la classe **RemoteServer** de **java.rmi.server**
- **RemoteServer** est une classe abstraite
- **UnicastRemoteObject** étend **RemoteServer**
  - c'est une classe concrète
  - une instance de cette classe réside sur un serveur et est disponible via le protocole TCP/IP

## Implémentation de l'objet distribué

```
import java.rmi.*;
import java.rmi.server.*;
public class Reverse extends UnicastRemoteObject implements
ReverseInterface
{
public Reverse() throws RemoteException {
    super();
}
public String reverseString (String ChaineOrigine) throws
RemoteException {

    int longueur=ChaineOrigine.length();
    StringBuffer temp=new StringBuffer(longueur);
    for (int i=longueur; i>0; i--)
    {
        temp.append(ChaineOrigine.substring(i-1, i));
    }
    return temp.toString();
} }
```



## Le serveur

- Programme à l'écoute des clients ;
- Enregistre l'objet distribué dans `rmiregistry`  
`Naming.rebind("rmi://hote.cnam.fr:1099/MyReverse", rev);`
- On installe un gestionnaire de sécurité si le serveur est amené à charger des classes (inutile si les classes ne sont pas chargées dynamiquement)  
`System.setSecurityManager(new RMISecurityManager());`

## Le serveur

```
import java.rmi.*;
import java.rmi.server.*;

public class ReverseServer {
    public static void main(String[] args)
    {
        try {
            System.out.println( "Serveur : Construction de l'implémentation ");
            Reverse rev= new Reverse();
            System.out.println("Objet Reverse lié dans le RMIregistry");
            Naming.rebind("rmi://sinus.cnam.fr:1099/MyReverse", rev);
            System.out.println("Attente des invocations des clients ...");
        }
        catch (Exception e) {
            System.out.println("Erreur de liaison de l'objet Reverse");
            System.out.println(e.toString());
        }
    }
} // fin du main
} // fin de la classe
```

## Le Client

- Le client obtient un stub pour accéder à l'objet par une URL RMI

```
ReverseInterface ri = (ReverseInterface) Naming.lookup  
("rmi://sinus.cnam.fr:1099/MyReverse");
```

- Une URL RMI commence par **rmi://**, le nom de machine, un numéro de port optionnel et le nom de l'objet distant.

**rmi://hote:2110/nomObjet**

Par défaut, le numéro de port est 1099 défini (ou à définir) dans  
**/etc/services :**

**rmi 1099/tcp**

## Le Client

- Installe un gestionnaire de sécurité pour contrôler les stubs chargés dynamiquement :

```
System.setSecurityManager(new RMISecurityManager());
```

- Obtient une référence d'objet distribué :

```
ReverseInterface ri = (ReverseInterface) Naming.lookup  
("rmi://sinus.cnam.fr:1099/MyReverse");
```

- Exécute une méthode de l'objet :

```
String result = ri.reverseString ("Terre");
```

# Le Client

```
import java.rmi.*;
import ReverseInterface;

public class ReverseClient
{
    public static void main (String [] args)
    {
        System.setSecurityManager(new RMISecurityManager());
        try{
            ReverseInterface rev = (ReverseInterface) Naming.lookup
                ("rmi://sinus.cnam.fr:1099/MyReverse");
            String result = rev.reverseString (args [0]);
            System.out.println ("L'inverse de "+args[0]+" est "+result);
        }
        catch (Exception e)
        {
            System.out.println ("Erreur d'accès à l'objet distant.");
            System.out.println (e.toString());
        }
    }
}
```

## Le Client

Pour que le client puisse se connecter à rmiregistry, il faut lui fournir un fichier de règles de sécurité **client.policy**.

```
$more client.policy
grant
{
permission java.net.SocketPermission
    ":1024-65535", "connect" ;
permission java.net.SocketPermission
    ":80", "connect";
};
```

```
$more client1.policy
grant
{
permission java.security.AllPermission;
};
```

## Compilation et Exécution

- Compiler les sources (interface, implémentation de l'objet, le serveur et le client ) :

```
sinus> javac *.java
```

- Lancer `rmic` sur la classe d'implémentation :

```
sinus>rmic -v1.2 Reverse
```

```
sinus>transférer *Stub.class et ReverseInterface.class  
vers la machine cosinus
```

- Démarrer `rmiregistry` :

```
sinus>rmiregistry -J-Djava.security.policy=client1.policy&
```

- Lancer le serveur :

```
sinus>java ReverseServer &  
Serveur :Construction de l'implémentation  
Objet Reverse lié dans le RMIregistry  
Attente des invocations des clients ...
```

- Exécuter le client :

```
cosinus>java -Djava.security.policy=client1.policy ReverseClient  
Alice
```

L'inverse de Alice est ecilA



## Résumé

### *Côté Client*

- l'interface : `ReverseInterface`
- le stub : `Reverse_Stub`
- le client : `ReverseClient`

### *Côté Serveur*

- l'interface : `ReverseInterface`
- l'objet : `Reverse`
- le serveur d'objets : `ReverseServer`

## Références bibliographiques (RMI)

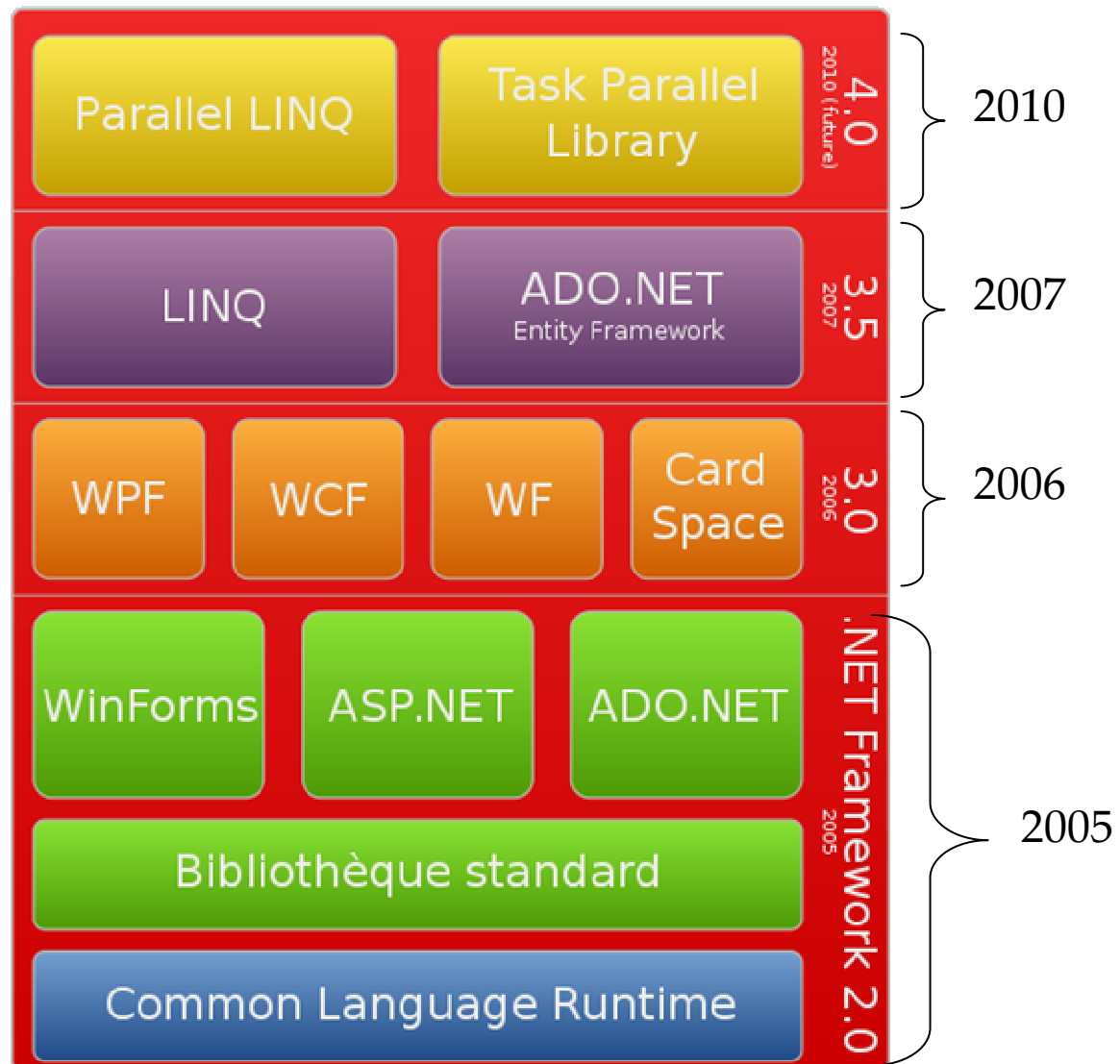
- *Java & Internet* de Gilles Roussel et Etienne Duris, Ed Vuibert, 2000.
- *Mastering RMI*, Rickard Öberg, Ed. Willey, 2001.
- *J2EE*, Ed. Wrox, 2001.
- *Cours RMI*, D. Olivier & S. Bouzefrane, DESS SOR, université du Havre, 2001/2002
- Tutorial de Java RMI de Sun :  
<http://java.sun.com/docs/books/tutorial/rmi/index.html>

# ***La technologie .NET et le service .NET Remoting***

## Quelques notions de base de .NET

# Les Frameworks .NET

Directement intégré  
dans Windows Vista  
et Windows Server 2008



Pile de composants de .NET Framework

[http://fr.wikipedia.org/wiki/Fichier:DotNet\\_French.svg](http://fr.wikipedia.org/wiki/Fichier:DotNet_French.svg)

## Nouvelles fonctionnalités depuis .NET 3.0

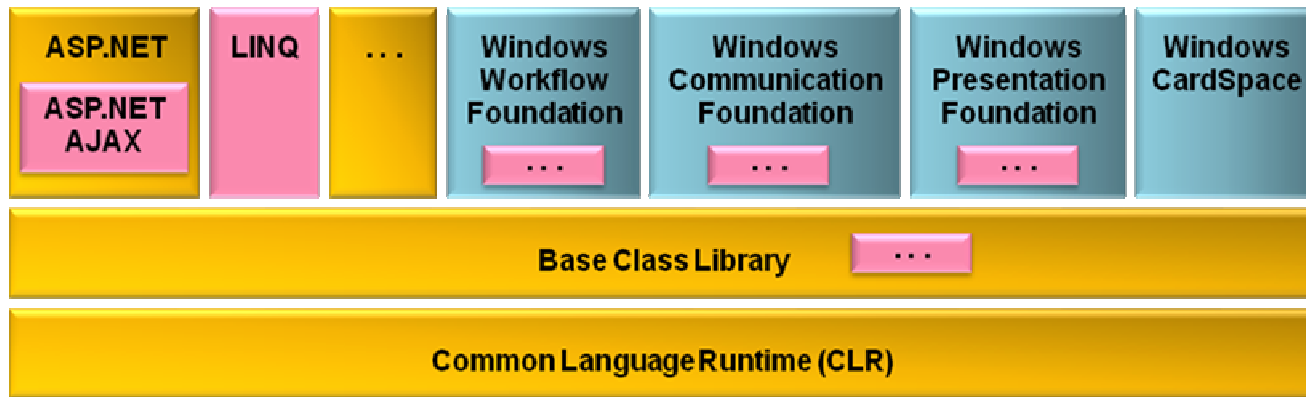
### ➤ .NET 3.0

- Windows Presentation Foundation (**WPF**) : un nouveau sous-système d'interface utilisateur basé sur XML et le dessin vectoriel. WPF utilise les cartes graphiques 3D et Direct3D.
- Windows Communication Foundation (**WCF**) : système de messagerie orienté services qui permet aux programmes de communiquer localement ou à distance (d'une façon analogue aux services web).
- Windows Workflow Foundation (**WF**) permet la construction de transactions ou tâches automatisées (gestion des processus métier)
- Windows **CardSpace**, anciennement InfoCard est un composant logiciel destiné à mémoriser de façon sécurisée les informations numériques relatives à une personne et fournit une interface unifiée pour le choix des identités pour une transaction particulière.

### ➤ .NET 3.5

- **Language Integrated Query (LINQ)** permettant des requêtes objet aussi bien sur des Data, des Collections, du XML ou des DataSets.
- **Ajax.Net** avec de nouveaux protocoles (AJAX, JSON, REST, RSS, Atom) et d'autres standards WS-.\*.

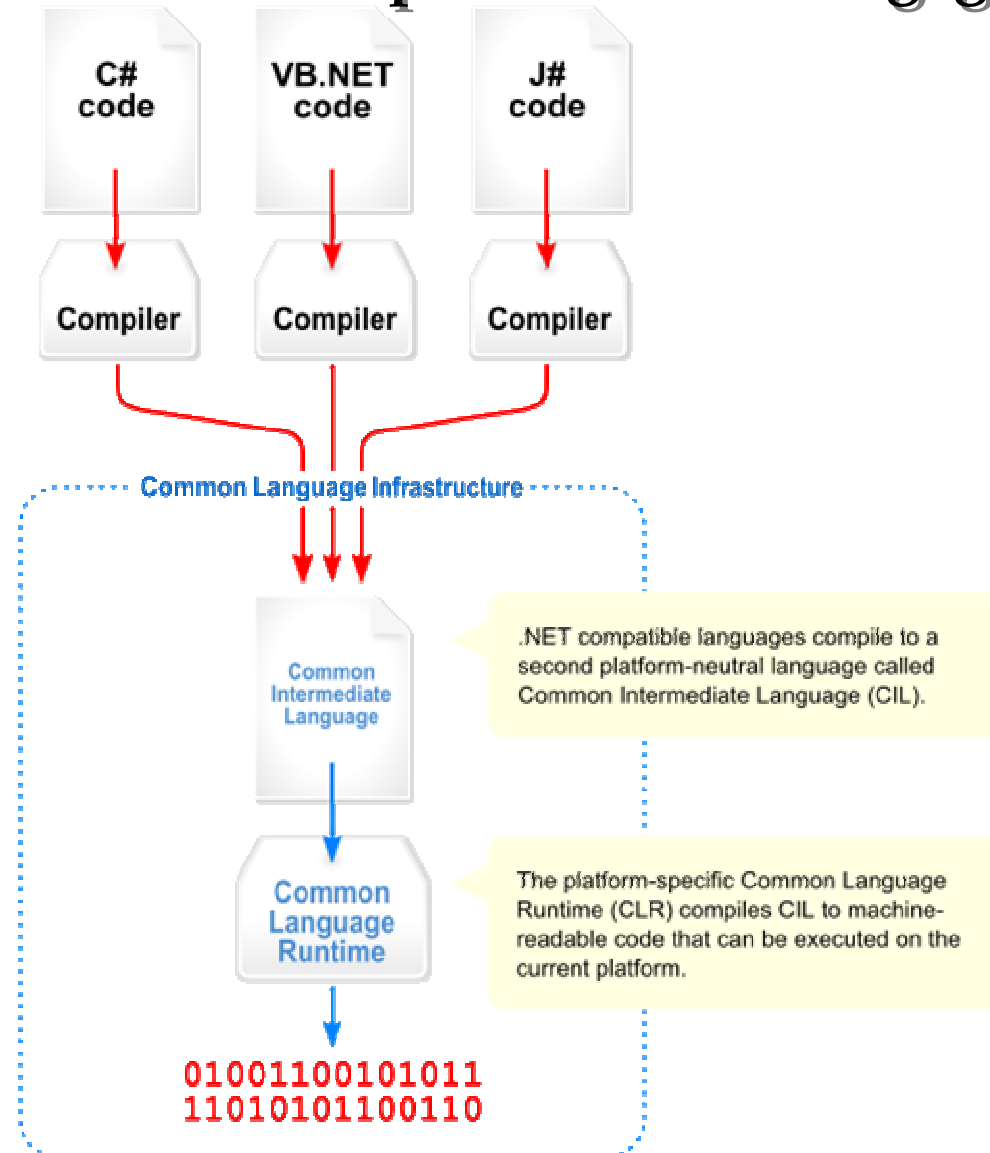
# Les Frameworks .NET



- .NET Framework 2.0**
- .NET Framework 3.0 Additions**
- .NET Framework 3.5 Additions**

[http://download.microsoft.com/download/f/3/2/f32ff4c6-174f-4a2f-a58f-ed28437d7b1e/Introducing NET Framework 35 v1.doc](http://download.microsoft.com/download/f/3/2/f32ff4c6-174f-4a2f-a58f-ed28437d7b1e/Introducing_NET_Framework_35_v1.doc)

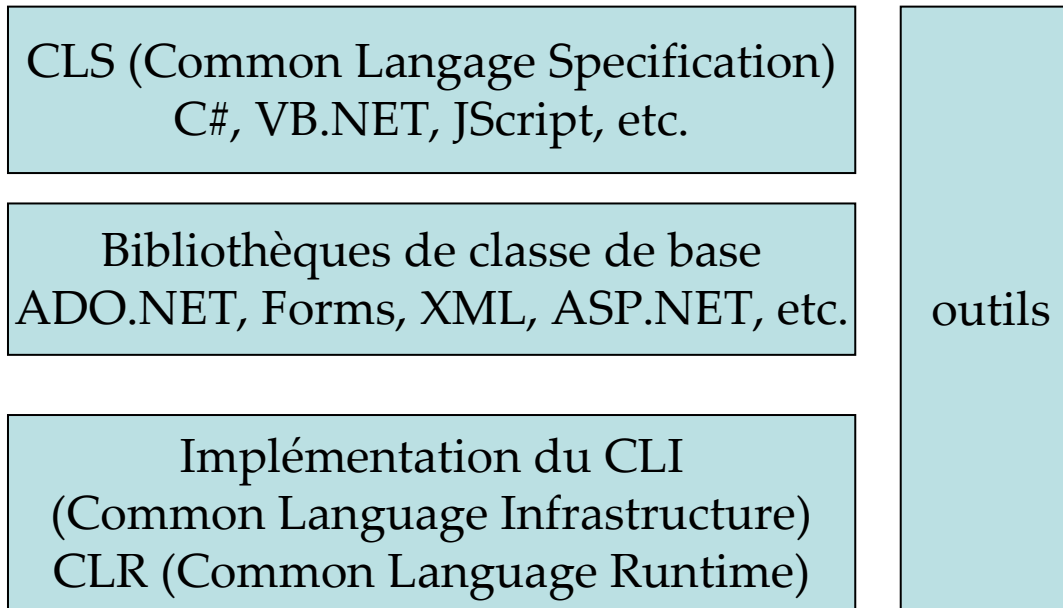
## Plateforme indépendante des langages



[http://fr.wikipedia.org/wiki/Fichier:Overview\\_of\\_the\\_Common\\_Language\\_Infrastructure.svg](http://fr.wikipedia.org/wiki/Fichier:Overview_of_the_Common_Language_Infrastructure.svg)



## Architecture de .NET



## Le CLR : moteur d'exécution de .NET

### ➤ CLR : Common Runtime Language

- ✓ Élément central de l'architecture .NET
- ✓ Gère l'exécution du code des applications

### ➤ Les actions du CLR :

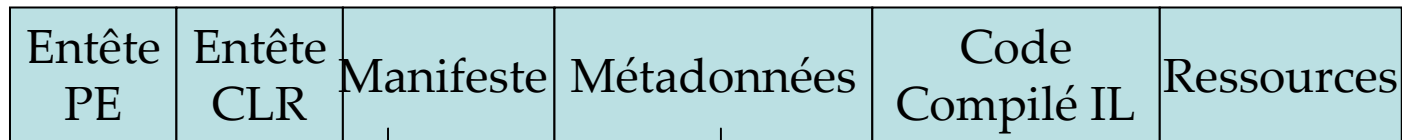
- ✓ Hébergement de plusieurs applications dans un même processus Windows
- ✓ Compilation du code IL en code machine
- ✓ Gestion des exceptions
- ✓ Destruction des objets devenus inutiles
- ✓ Chargement des assemblages
- ✓ Résolution des types

## Notion d'assemblage

- **Assemblage = composant de la plate-forme .NET (équivalent au .exe ou .dll sous Windows)**
- **Un assemblage = plusieurs fichiers appelés Modules**
- **les fichiers d'un même assemblage doivent appartenir au même répertoire**
- **Un assemblage porte le nom d'un module principal qu'il contient**
- **Le module principal joue un rôle particulier car :**
  - ✓ Tout assemblage en comporte un et un seul
  - ✓ Si un assemblage comporte plusieurs modules alors le module principal est chargé en premier
  - ✓ Un module principal (.exe ou .dll), module non principal (.netmodule)

## Notion de module

Module principal

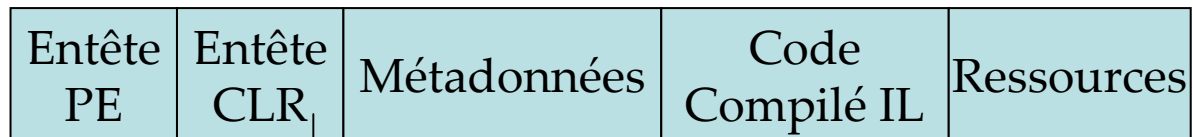


PE: Portable  
Executable

↓  
Références  
vers les autres  
modules et ressources

↓  
Description  
des types

Autre module



↓  
Version du CLR

## Domaine d'application

- **AppDomain** : est l'équivalent d'un processus léger
- **On associe un domaine par application**
- **Un processus Windows** : est un ensemble de domaines d'application
  - ✓ La création d'un domaine consomme moins de ressources qu'avec un processus Windows
  - ✓ Les domaines d'application hébergés dans un même processus Windows partagent les ressources du processus (ex. CLR, les types de base de .NET, l'espace d'adressage, les threads, etc.)
  - ✓ Ne pas confondre threads (unités d'exécution) avec domaines d'application (unités d'isolation d'exécution)

## **Assemblage/Domaine d'application**

- **Lorsqu'un assemblage exécutable est démarré, le CLR crée automatiquement un domaine par défaut**
- **Un domaine d'application a un nom (son nom par défaut est celui du module principal de l'assemblage lancé)**
- **Un même assemblage peut être chargé par plusieurs domaines**
- **Les domaines sont isolés les uns des autres par le CLR**
- **L'assemblage mscorlib (types de base de .NET) est chargé dans un processus hors domaine d'application**
- **L'isolation se fait au niveau des types, de la sécurité et de la gestion d'exceptions (pas d'isolation au niveau des threads)**

## .NET Remoting

## **.NET Remoting**

permet la communication entre programmes (domaines d'application) qui peuvent se trouver physiquement sur la même machine ou sur deux machines distinctes.



## Présentation/1

- .NET Remoting est un système d'objets distribués
- .NET Remoting est une **A**pplication **P**rogramming **I**nterface

## Présentation/2

- Mécanisme qui permet l'appel de méthodes entre objets qui s'exécutent éventuellement sur des machines distinctes ;
- L'appel peut se faire sur la même machine ou bien sur des machines connectées sur un réseau ;
- Utilise les sockets ;
- .NET Remoting repose sur les classes de sérialisation.

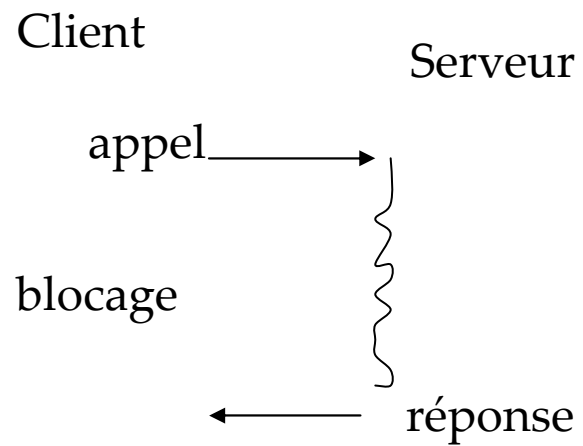
## **.NET Remoting**

- **Peut être vu comme le successeur de DCOM**
- **Est l'infrastructure .NET qui permet à des objets (appartenant à des domaines d'applications différents) de communiquer entre eux**
- **L'objet appelant : client**
- **L'objet appelé : serveur**
- **Les domaines d'application différents appartiennent :**
  - ✓ au même processus Windows
  - ✓ à des processus différents sur la même machine
  - ✓ à des processus différents sur deux machines différentes
- **Les données échangées sont emballées par des objets appelés formateurs.**

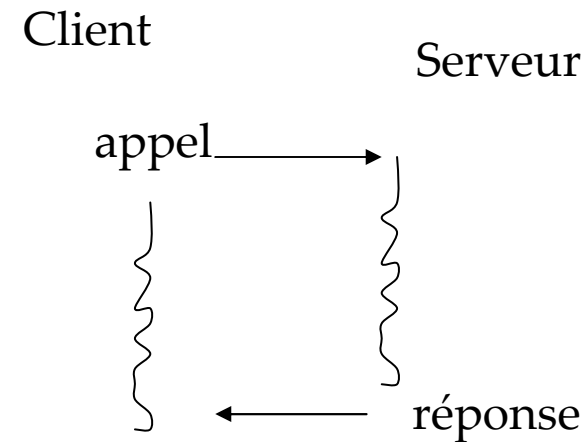
## **Appel synchrone/asynchrone/1**

- Par défaut, l'appel entre le client et le serveur est synchrone (blocage du client jusqu'au retour du résultat de la méthode)
- L'appel asynchrone existe aussi : le client poursuit après l'appel mais le serveur peut avertir le client de l'exécution de l'appel

## Appel synchrone/asynchrone/2



Appel synchrone



Appel asynchrone

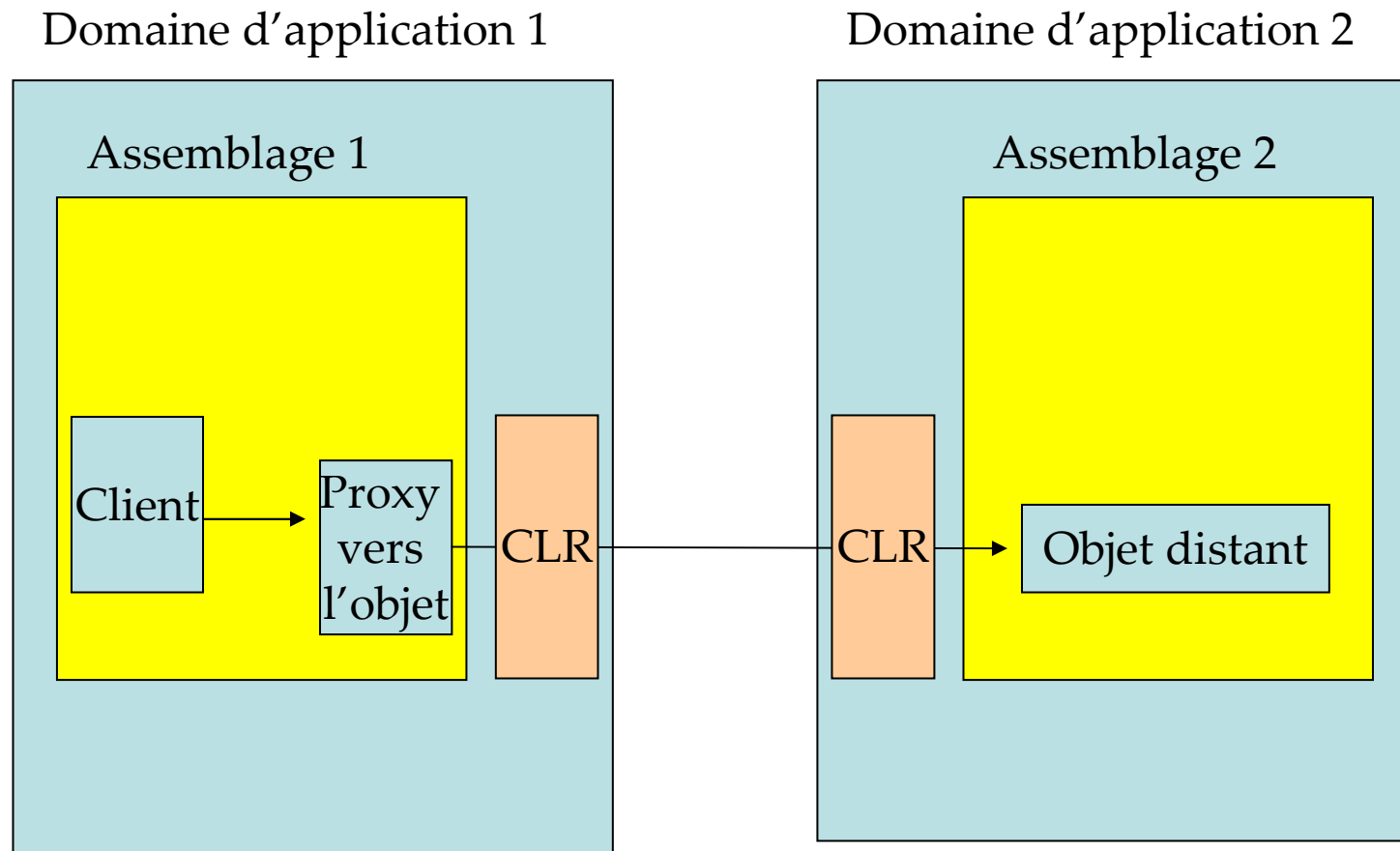
## **L'appel d'une méthode**

- **L'appel d'une méthode à distance se fait selon deux solutions :**
  - ✓ **Marshalling By Value (MBV)**
  - ✓ **Marshalling By Reference (MBR)**

## **MBR (Marshalling By Reference)**

- **Consiste à obtenir un nouvel objet appelé proxy transparent dans le domaine d'application du client**
- **Questions :**
  - ✓ **Qui est responsable de la création d'objets distants**
  - ✓ **Comment récupère-t-on un proxy transparent ?**
  - ✓ **Comment la classe proxy fait-elle pour faire transiter les données sur le réseau**

## MBR (Marshalling By Reference)



Faire dériver une classe à partir de *MarshalByRefObject()* indique au compilateur JIT qu'une instance de cette classe peut être utilisée d'une manière distante grâce à un proxy transparent.



## Exemple de MBR/1 en local

```
using System;
using System.Runtime.Remoting.Contexts;
using System.Runtime.Remoting;
using System.Threading;

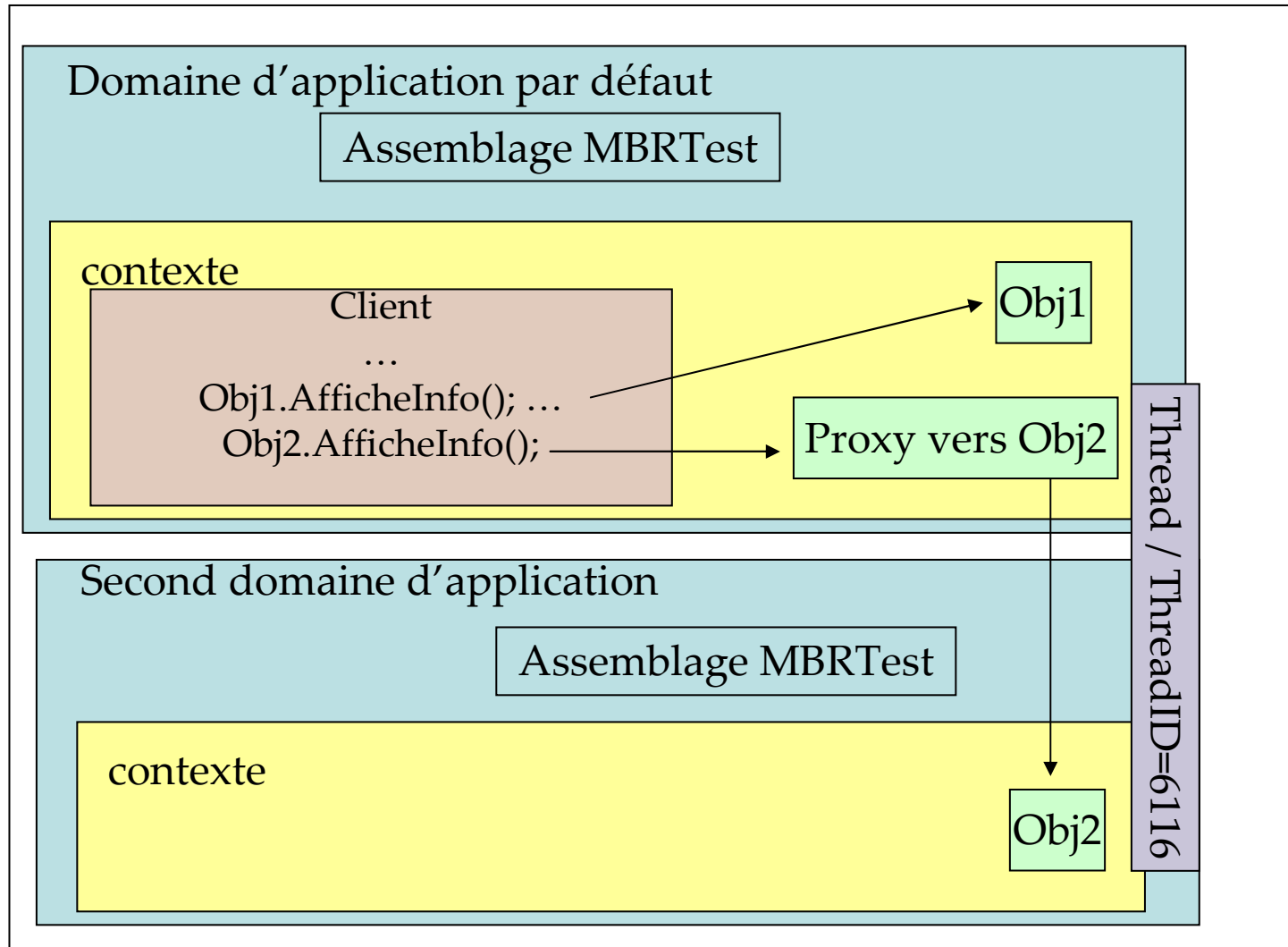
public class Foo : MarshalByRefObject {
    public void AfficheInfo(string s) {
        Console.WriteLine(s);
        Console.WriteLine("  Nom du domaine: " +
                          AppDomain.CurrentDomain.FriendlyName);
        Console.WriteLine("  ThreadID      : " +
                          Thread.CurrentThread.ManagedThreadId);
    }
}
```

## Exemple de MBR/2 en local

```
public class Program {  
    static void Main() {  
        // obj1  
        Foo obj1 = new Foo();  
        obj1.AfficheInfo("obj1:");  
        Console.WriteLine("  IsObjectOutOfAppDomain(obj1)=" +  
            RemotingServices.IsObjectOutOfAppDomain(obj1));  
        Console.WriteLine("  IsTransparentProxy(obj1)=" +  
            RemotingServices.IsTransparentProxy(obj1));  
  
        // obj2  
        AppDomain appDomain = AppDomain.CreateDomain("Autre domaine.");  
        Foo obj2 = (Foo)appDomain.CreateInstanceAndUnwrap(  
            "MBRTest", // Nom du projet sous Visual C# (nom asm)  
            "Foo");    // Nom du type.  
        obj2.AfficheInfo("obj2:"); // Ici, le code client ne sait pas  
            // qu'il manipule un proxy transparent.  
        Console.WriteLine("  IsObjectOutOfAppDomain(obj2)=" +  
            RemotingServices.IsObjectOutOfAppDomain(obj2));  
        Console.WriteLine("  IsTransparentProxy(obj2)=" +  
            RemotingServices.IsTransparentProxy(obj2));  
    }  
}
```

## Exemple de MBR/3 en local

### Processus



## Exécution

Compilation:

```
csc.exe /out:MBRTest.exe /target:exe MBRTest.cs
```

Obj1:

Nom du domaine: MBRTest.exe

ThreadID : 6116

IsObjectOutOfAppDomain(obj1)=False

IsTransparentProxy(obj1)=False

Obj2:

Nom du domaine: MBRTest.exe

ThreadID : 6116

IsObjectOutOfAppDomain(obj2)=True

IsTransparentProxy(obj2)=True

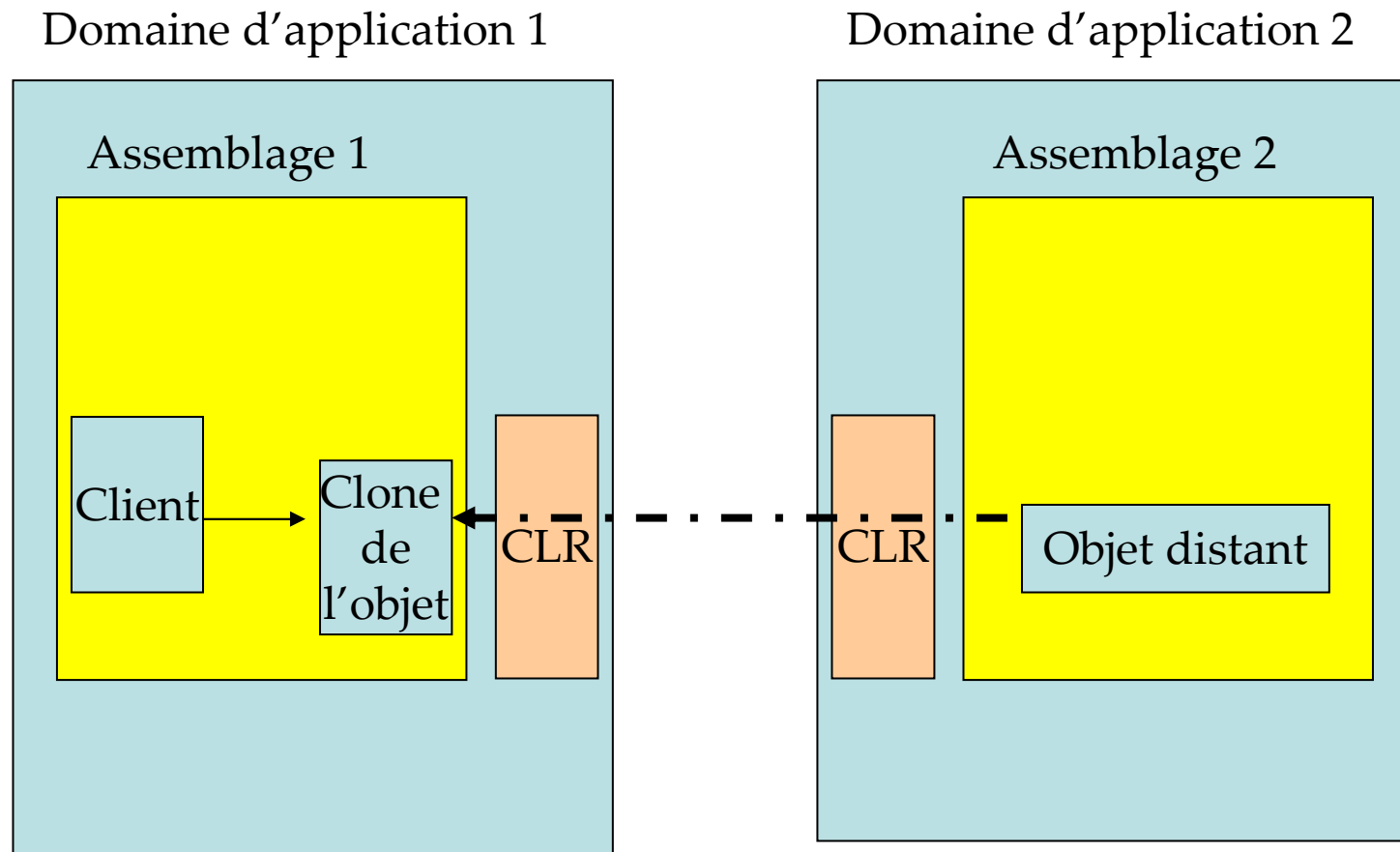
## Sérialisation/ Désérialisation

- Un objet est sérialisable si :
  - ✓ Sa classe possède l'attribut `System.Serializable` signalé au CLR ou
  - ✓ Sa classe implémente l'interface `System.Runtime.Serialization.ISerializable` pour implémenter son propre mécanisme de sérialisation

## **MBV ( Marshalling By Value)/1**

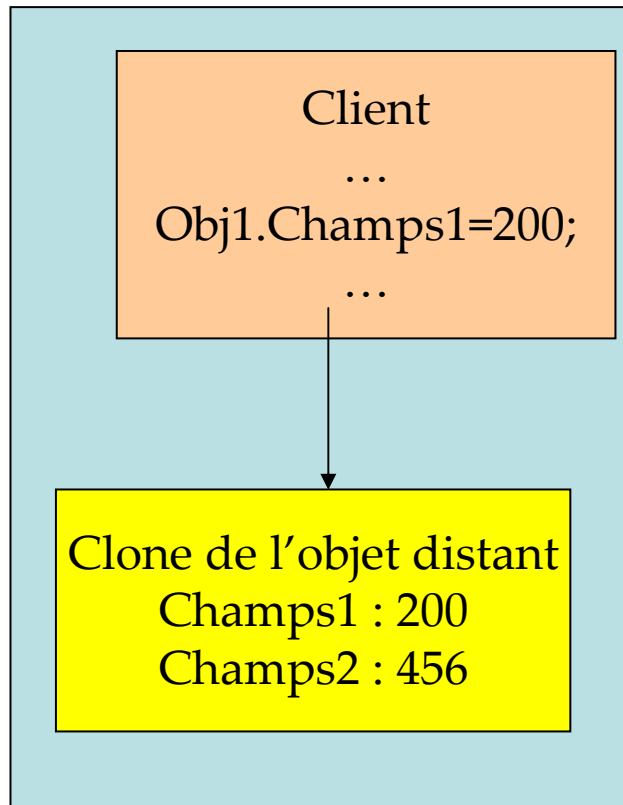
- **Consiste à fabriquer un clone de l'objet distant dans le même domaine que le client**
- **Le CLR fait en sorte que l'objet cloné ait le même état (même valeurs des champs) que l'objet distant**
- **Le client n'a pas besoin de proxy transparent pour accéder à l'objet**
- **Le domaine du client doit pouvoir charger l'assemblage qui contient la classe de l'objet distant original**
- **L'objet original ne doit pas contenir des références à des objets non clonables**
- **C'est le CLR qui envoie l'état de l'objet distant dans un stream binaire au domaine d'application du client : on parle de sérialisation**
- **Le CLR déséréalise le stream binaire et reconstitue l'état dans le clone**

## MBV ( Marshalling By Value)/2

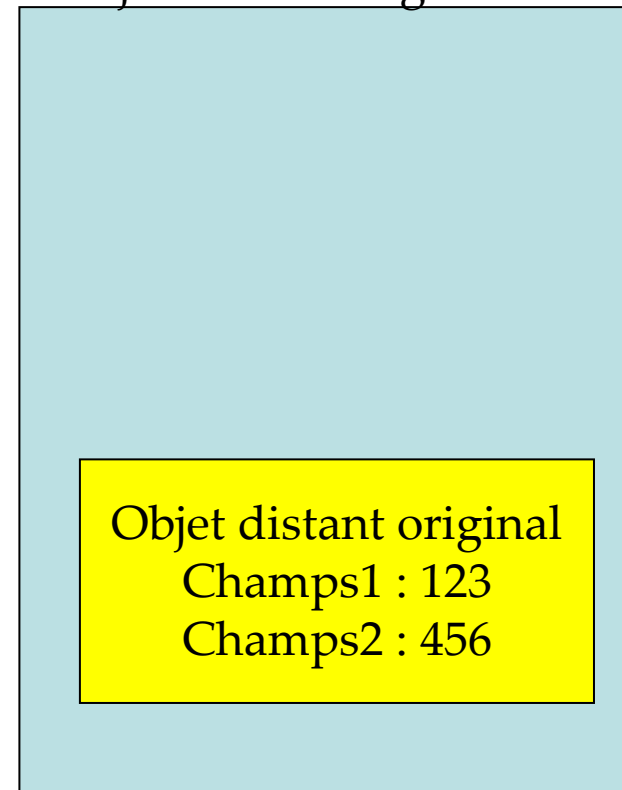


## MBV ( Marshalling By Value)/3

Domaine d'application du client



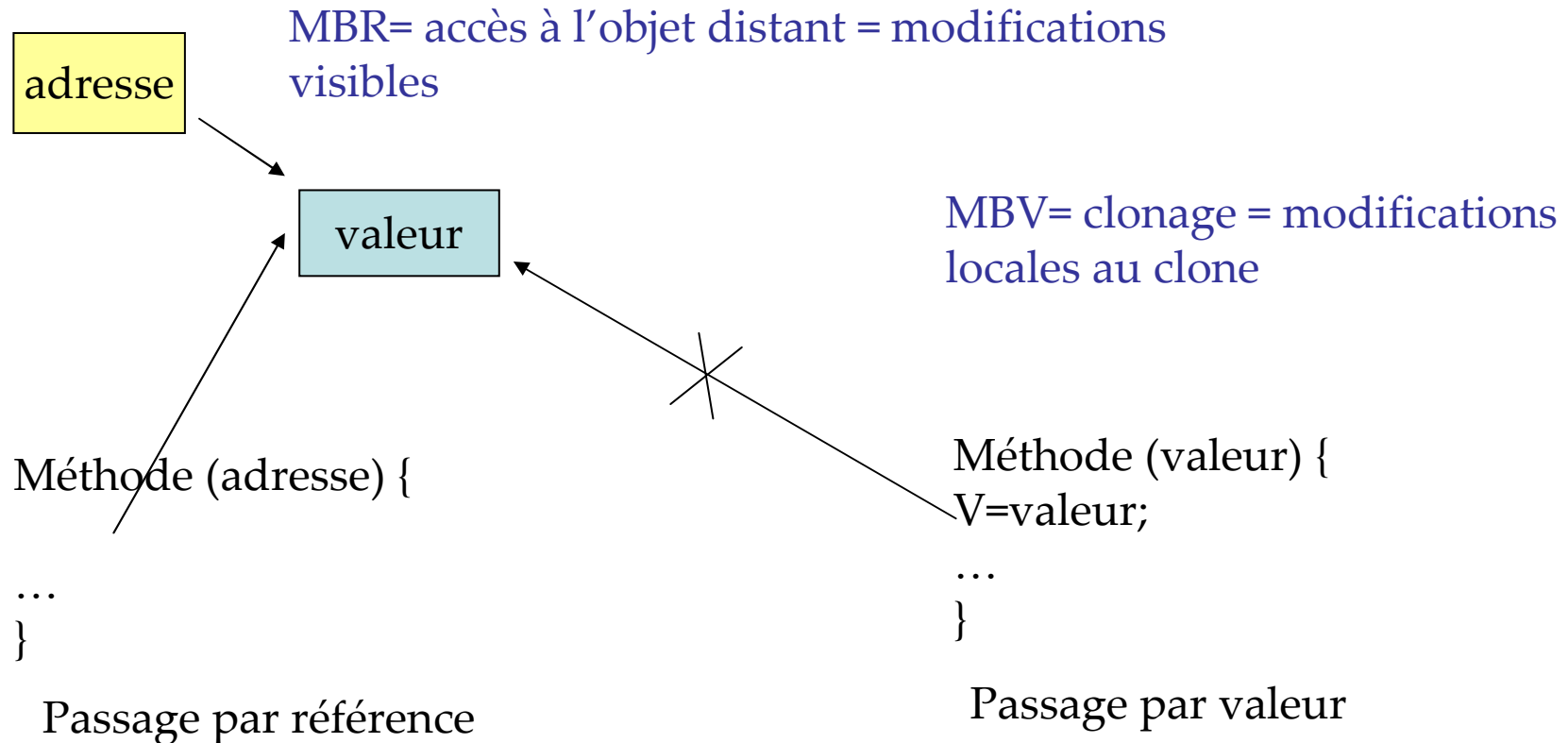
Domaine d'application contenant  
l'objet distant original





## MBV / MBR et le passage d'arguments

- MBV : similaire au passage de paramètres par valeur
- MBR : similaire au passage de paramètres par référence



## Exemple avec MBV/1

...

`[Serializable]`

```
public class Foo { // : MarshalByRefObject  <- en commentaire
    public void AfficheInfo(string s) {
        Console.WriteLine(s);
        Console.WriteLine("  Nom du domaine: " +
                           AppDomain.CurrentDomain.FriendlyName);
        Console.WriteLine("  ThreadID      : " +
                           AppDomain.GetCurrentThreadId());
    }
}
```

...

## Exécution

Compilation:

```
csc.exe /out:MBVTest.exe /target:exe MBVTest.cs
```

Obj1:

Nom du domaine: MBVTest.exe

ThreadID : 3620

IsObjectOutOfAppDomain(obj1)=False

IsTransparentProxy(obj1)=False

Obj2:

Nom du domaine: MBVTest.exe

ThreadID : 3620

IsObjectOutOfAppDomain(obj2)=False

IsTransparentProxy(obj2)=False

## La classe *ObjectHandle* : autre manière de créer un objet

### Deux manières de créer un objet distant :

- ✓ À l'aide de **CreateInstanceAndUnwrap()** (voir exemple MBR)

- ✓ À l'aide de deux opérations :

  - **ObjectHandle()** représente l'objet distant créé

  - **Unwrap()** qui permet de créer un proxy transparent si l'objet distant est MBR ou bien de créer un clone de l'objet si celui-ci est MBV.

```
ObjectHandle hObj = appDomain.CreateInstance("WrapTest","Foo");  
Foo obj = (Foo) hObj.Unwrap();
```

Nom du domaine



## Architecture distribuée avec objets serveurs MBR

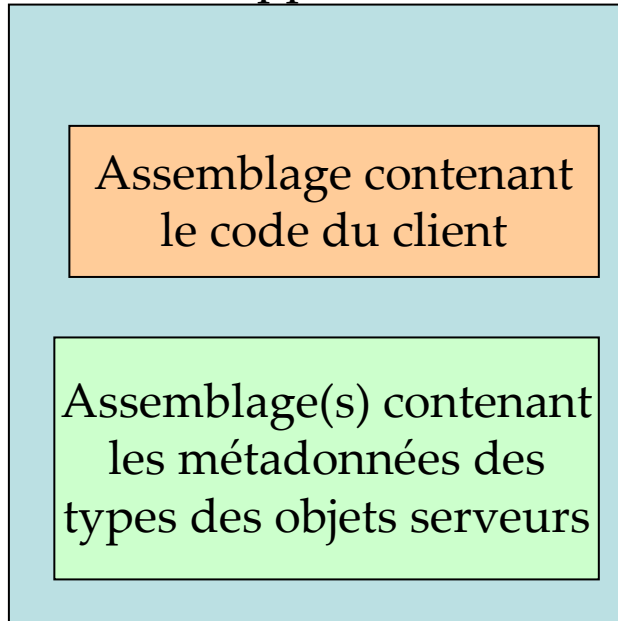
➤ Il existe 4 entités :

- ✓ Les clients qui appellent les objets serveurs
- ✓ Les hôtes qui hébergent les objets serveurs
- ✓ Les métadonnées des types des objets serveurs
- ✓ Les implémentations des objets serveurs

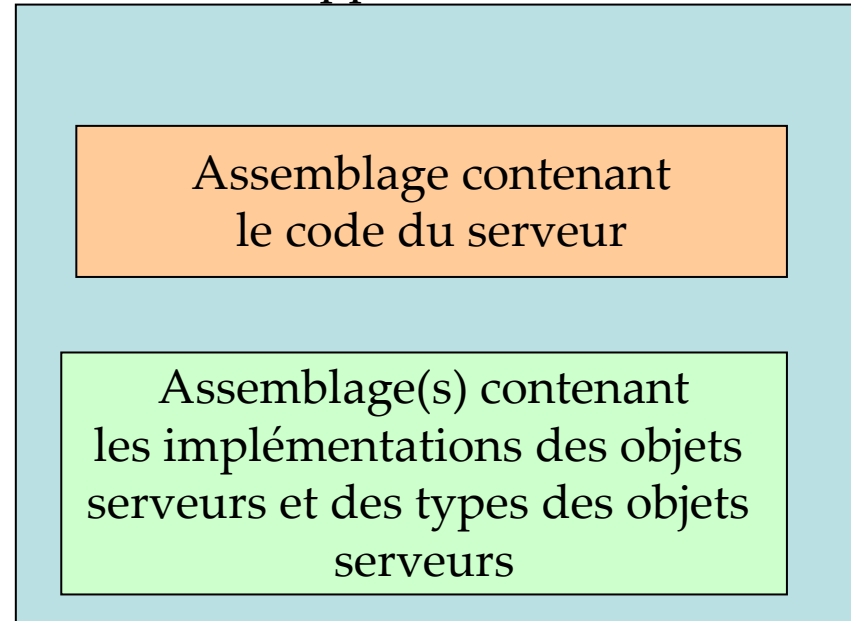
➤ **Recommandation : isoler chaque entité dans un assemblage**

## Éléments de l'architecture distribuée

### Domaine d'application Client



### Domaine d'application Serveur



## Responsabilité d'un hôte

- **Un hôte doit :**
  - ✓ créer un ou plusieurs canaux (channels)
  - ✓ exposer des classes ou des objets serveurs accessibles par des clients à travers d'URIs
  - ✓ maintenir le processus qui contient les objets serveurs.
  
- **Un objet est exposé si le client doit connaître l'objet à utiliser**

## Les canaux

- Un canal est un objet qui permet de communiquer via le réseau ou bien entre processus locaux (IPC)
- Un canal est caractérisé par trois paramètres :
  - ✓ le port réseau de la machine qui l'utilise
  - ✓ le protocole de communication utilisé (TCP, HTTP, IPC)
  - ✓ le type de formatage de données (format binaire, SOAP, XML propriétaire)
- Par défaut, le formatage binaire est associé au protocole TCP et IPC
- Par défaut, le formatage SOAP est associé au protocole HTTP
- .NET Remoting utilise deux canaux utilisant le même formatage et le même protocole (un canal côté client et un canal côté serveur).



## Activation d'objets

- En .NET Remoting, on parle plutôt d'activation d'objets que de création d'objets car la création génère l'activation de plusieurs actions avant la disponibilité effective de l'objet
- A la spécification de l'architecture, il est important de définir qui (client ou hôte) va activer chaque objet distant

## **Service d'activation par le serveur (WKO)**

- **L'objet WKO (Well known Object) est accessible à distance**
- **Si l'objet est activé par le serveur alors :**
  - ✓ Le client n'appelle pas le constructeur de la classe
  - ✓ Le client ne connaît pas la classe de l'objet
  - ✓ Le client connaît les interfaces supportées par la classe de l'objet
  - ✓ Ces interfaces figurent dans l'assemblage des métadonnées des types des objets serveurs
- **L'assemblage qui contient les interfaces est présent dans le domaine du client et dans celui du serveur**

## Étapes de développement de l'application

- Écriture de la classe d'interface
- Écriture du serveur
- Écriture du client

## Étapes de développement du serveur

- Implémentation de l'interface (classe de l'objet serveur)
- Création d'un canal (n° de port, TCP), formatage binaire par défaut
- Enregistrement du canal auprès du domaine d'application courant
- Publication de l'objet

## Exemple (Interface)

Compilation:

```
csc.exe /out:Interface.dll /target:library NommageInterface.cs
```

```
namespace NommageInterface {  
    public interface IAdditionneur {  
        double Add(double d1, double d2);  
    }  
}
```

## Exemple (Implémentation)

Compilation:

```
csc.exe /out:Serveur.exe /target:exe NommageServer.cs /r:Interface.dll
```

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using NommageInterface;

namespace NommageServer {
    public class CAdditionneur : MarshalByRefObject, IAdditionneur {
        public CAdditionneur() {
            Console.WriteLine("CAdditionneur ctor");
        }
        public double Add(double d1, double d2) {
            Console.WriteLine("CAdditionneur Add( {0} + {1} )", d1, d2);
            return d1 + d2;
        }
    }
}
```

## Exemple (Hôte)

```
class Program {
    static void Main() {
        // 1) Creation d'un canal Tcp sur le port 65100
        // enregistre ce canal dans le domaine d'application courant.
        TcpChannel canal = new TcpChannel(65100);
        ChannelServices.RegisterChannel(canal, false);

        // 2) Ce domaine d'application presente un objet de type
        // IAdditionneur associe au point terminal 'ServiceAjout'.
        RemotingConfiguration.RegisterWellKnownServiceType(
            typeof(NommageServer.CAdditionneur),
            "ServiceAjout",
            WellKnownObjectMode.SingleCall);

        // 3) Maintien du processus courant.
        Console.WriteLine(
            "Pressez une touche pour stopper le serveur.");
        Console.Read();
    }
}
```

## Étapes de développement du client

- Création du canal
- Construire l'URI (ex: *tcp://localhost:65100/ServiceAjout*)
- Récupération du proxy transparent de l'objet distant
- Appel de la méthode distante
  - ✓ Appel *Single Call* : associe un appel à un objet, d'où n appels correspondent à n objets serveurs
  - ✓ Appel *Singleton* : associe plusieurs appels au même objet (gestion d'accès aux ressources communes)



## Exemple de client/1

Compilation:

```
csc.exe /out:Client.exe /target:exe NommageClient.cs /r:Interface.dll
```

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

using NommageInterface;

namespace NommageClient {
    class Program {
        static void Main() {
            // Crée un canal TCP puis enregistre
            // ce canal dans le domaine d'application courant.
            // (la valeur 0 pour le numéro de port côté client signifie
            // que ce numéro de port est choisi automatiquement
            // par le CLR).
        }
    }
}
```

## Exemple de client/2

```
TcpChannel canal = new TcpChannel(0);
ChannelServices.RegisterChannel(canal, false);

// Obtient un proxy transparent sur l'objet distant à partir de
// son URI, puis transtype le proxy distant en IAdditionneur.
MarshalByRefObject objRef = (MarshalByRefObject)
    RemotingServices.Connect(
        typeof(NommageInterface.IAdditionneur),
        "tcp://localhost:65100/ServiceAjout");
IAdditionneur obj = objRef as IAdditionneur;

// Appel d'une methode sur l'objet distant.
double d = obj.Add(3.0, 4.0);
Console.WriteLine("Valeur retournee:" + d);
    }
}
}
```

## Exécution

Chaque ligne correspond à un assemblage

```
csc.exe /target:library NommageInterface.cs  
csc.exe NommageServer.cs /r:library  
csc.exe NommageClient.cs /r:library
```

Affichage du Serveur

Pressez une touche pour stopper le serveur.

CAdditionneur ctor

CAdditionneur Add( 3 + 4 )

Affichage du Client.exe

Valeur retournee:7

>>> Sous Visual Studio Express 2008, il faut rajouter une référence vers le namespace System.Runtime.Remoting qui contient les classes des canaux.

## Exécution en mode Single Call

Hôte :

```
RemotingConfiguration.RegisterWellKnownServiceType(  
    typeof(CAdditionneur),  
    "ServiceAjout",  
    WellKnownObjectMode.SingleCall);
```

Client:

```
Obj1.Add(3.0 , 4.0);  
Obj1.Add(5.0 , 6.0);
```

Serveur :

```
Pressez une touche pour stopper le serveur.  
CAdditionneur ctor  
CAdditionneur Add( 3 + 4)  
CAdditionneur ctor  
CAdditionneur Add( 5 + 6)
```

## Exécution en mode Singleton

Hôte :

```
RemotingConfiguration.RegisterWellKnownServiceType(  
    typeof(CAdditionneur),  
    "ServiceAjout",  
    WellKnownObjectMode.Singleton);
```

Client:

```
Obj1.Add(3.0 , 4.0);  
Obj1.Add(5.0 , 6.0);
```

Serveur :

```
Pressez une touche pour stopper le serveur.  
CAdditionneur ctor  
CAdditionneur Add( 3 + 4)  
CAdditionneur Add( 5 + 6)
```

## **Autre implémentation : Configuration d'une application**

- **Ne pas figer les URIs dans le code**
- **Paramètres modifiables côté hôte et côté client**
- **Paramètres stockés dans un fichier XML**

## Configuration d'un hôte

- Publier CAdditionneur en WKO, mode singleton
- Publier CMultiplicateur en mode simple appel

# Configuration du serveur

## Fichier Hote.config

```
<configuration>
  <system.runtime.remoting>
    <application name = "Serveur">
      <service>
        <wellknown type="NommageObjServeur.CAdditionneur,ObjServeur"
                    mode ="Singleton" objectUri="Service1.rem" />
        <wellknown type="NommageObjServeur.CMultiplicateur,ObjServeur"
                    mode ="SingleCall" objectUri="Service2.rem" />
      </service>
      <channels>
        <channel port="65100" ref="tcp" />
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

Nom de la classe

Nom du fichier Serveur



## Dans le corps du serveur

Fichier Serveur.cs

```
...  
    class Program {  
        static void Main() {  
            RemotingConfiguration.Configure("Hote.config", false);  
            Console.WriteLine(  
                "Pressez une touche pour stopper le serveur.");  
            Console.Read();  
        }  
    }  
...  
...
```

## Configuration du client/1

### Fichier Client.config

```
<configuration>
  <system.runtime.remoting>
    <application name = "Client">
      <client>
        <wellknown
          type="NommageObjServeur.CAdditionneur,ObjServeur"
          url="tcp://localhost:65100/Service1.rem" />
        <wellknown
          type="NommageObjServeur.CMultiplieur,ObjServeur"
          url="http://localhost:65100/Service2.rem" />
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
```

## Configuration du client/2

Fichier Client.cs

Compilation:

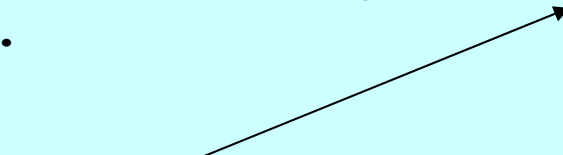
```
csc.exe /out:Client.exe /target:exe Client.cs /r:ObjServeur.dll
```

```
...
```

```
using System.Runtime.Remoting.Activation;
```

```
using NommageObjServeur;
```

```
namespace NommageClient {  
    class Program {  
        static void Main() {  
            RemotingConfiguration.Configure("Client.config", false);  
            CAdditionneur objA = new CAdditionneur();  
            ...  
        }  
    }  
}
```



L'appel au constructeur permet de récupérer une référence vers le proxy transparent

## Bibliographie

- Les exemples de programmes présentés dans ce cours ont été repris du livre « Pratique de .NET2 et C#2, Patrick Smacchia, Ed. Oreilly, 2005. »
- [//ditch.developpez.com/](http://ditch.developpez.com/) par Didier Danse
- [msdn.microsoft.com](http://msdn.microsoft.com)
- Apprentissage du Langage C#, par Serge Tahé, ISTIA, Université d'Angers
- « Créer et Consommer un service web avec .NET », Par Stéphane Eyskens.
- Visual Studio 2008 (versions Express gratuites)  
<http://www.microsoft.com/express/>