

La technologie .NET

Le Service 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

Qu'attend t-on d'un objet distribué ?/1

➤ 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( );
```

Qu'attend t-on d'un objet distribué ?/2

- 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);
```

Quelques notions de base de .NET

Architecture de .NET

CLS (Common Language Specification)
C#, VB.NET, JScript, etc.

Bibliothèques de classe de base
ADO.NET, Forms, XML, ASP.NET, etc.

Implémentation du CLI
(Common Language Infrastructure)
CLR (Common Language Runtime)

outils

Le CLR : moteur d'exécution de .NET

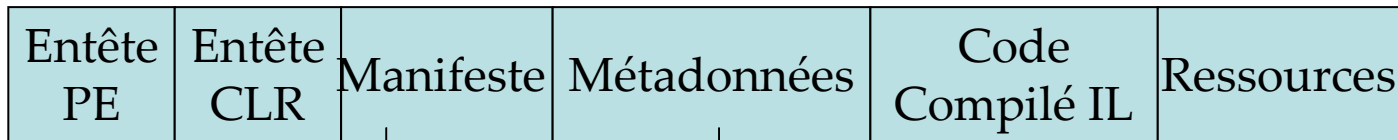
- **CLR : Common Runtime Language**
 - ✓ Élément central de l'architecture .NET
 - ✓ Gère l'exécution du code des applications
- **Les actions du CLR :**
 - ✓ Hébergement de plusieurs applications dans un même processus Windows
 - ✓ Compilation du code IL en code machine
 - ✓ Gestion des exceptions
 - ✓ 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.

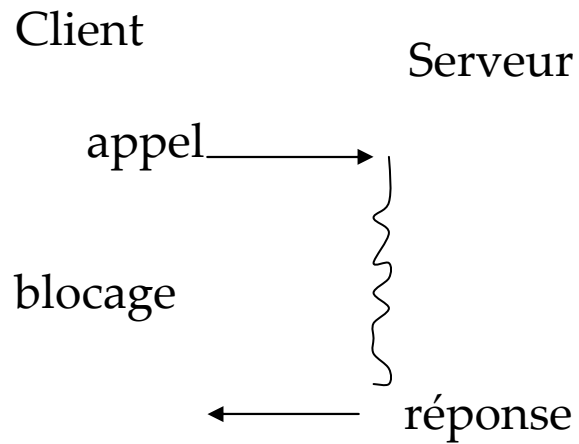
.NET Remoting

- **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.**

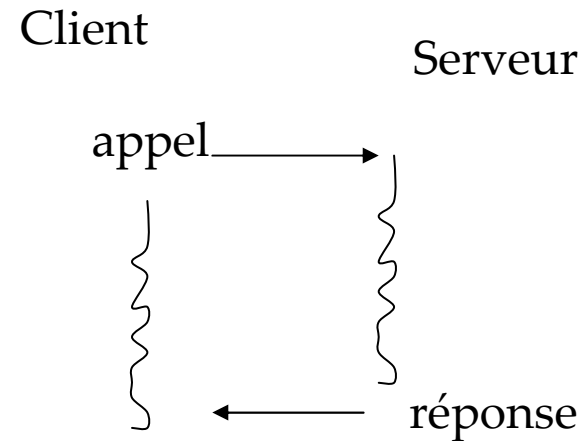
Appel synchrone/asynchrone/1

- **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

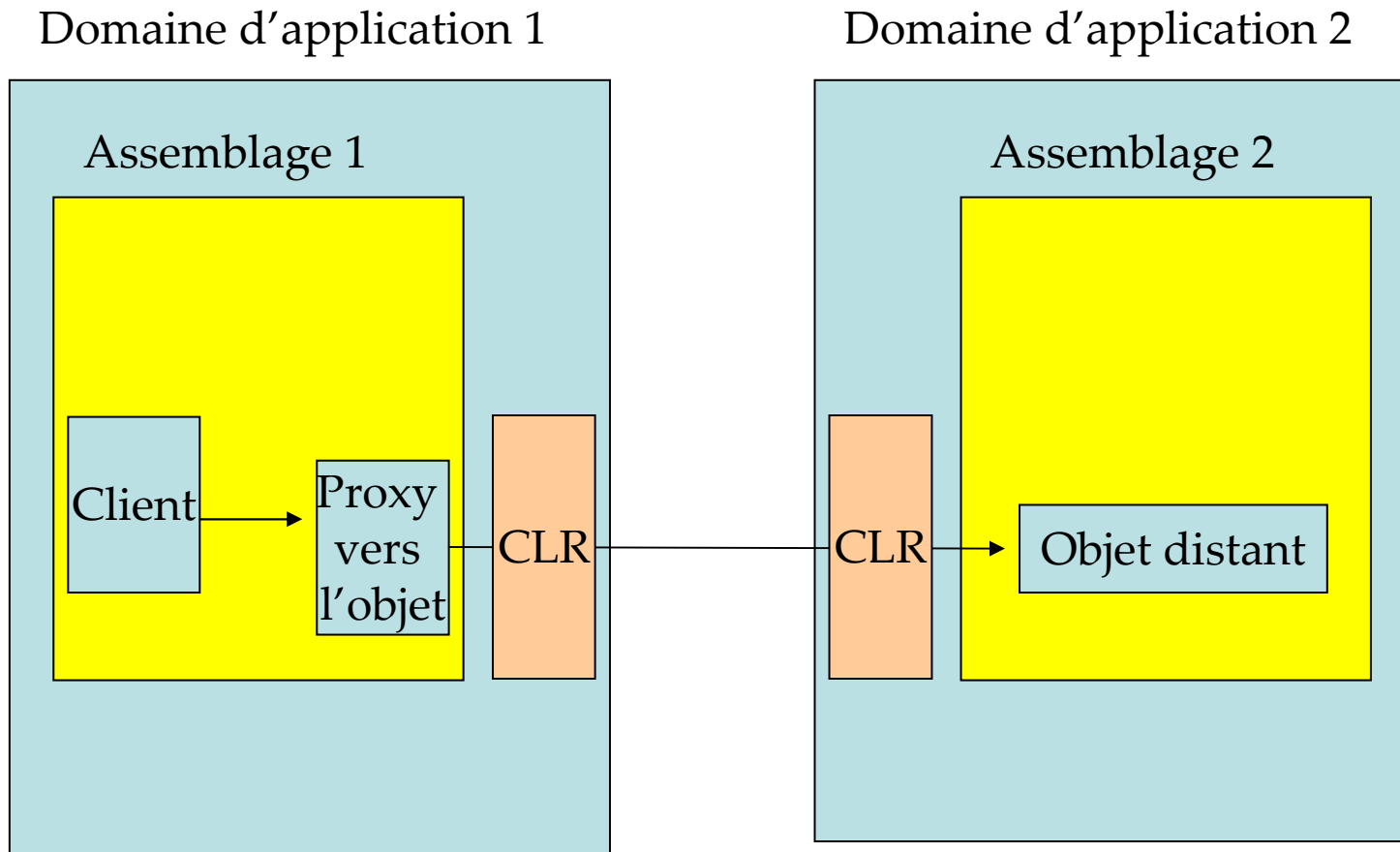
L'appel d'une méthode

- 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);
    }
}
```

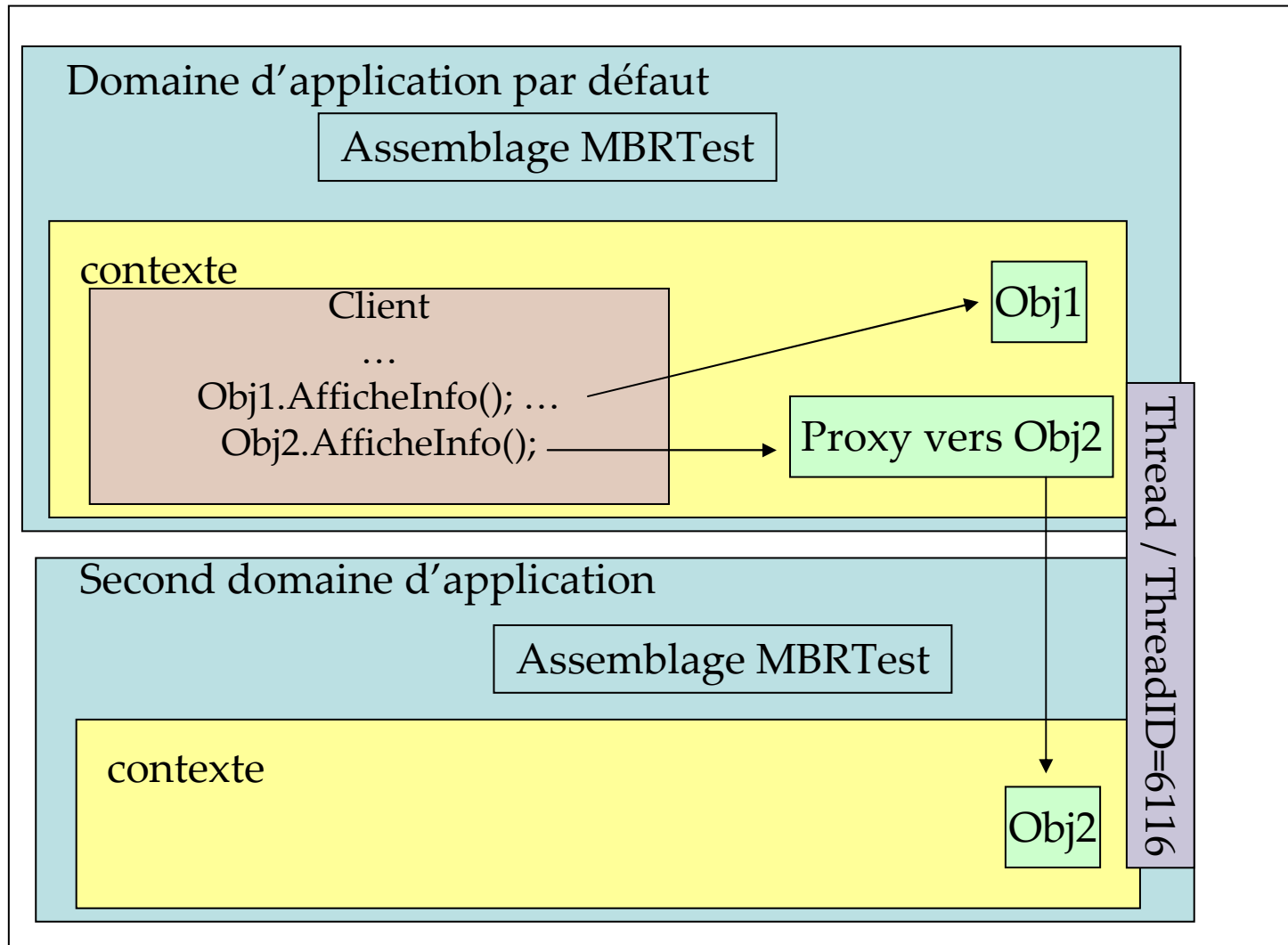
Exemple de MBR/2 en local

```
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 de l'asm contenant le type.
            "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 :

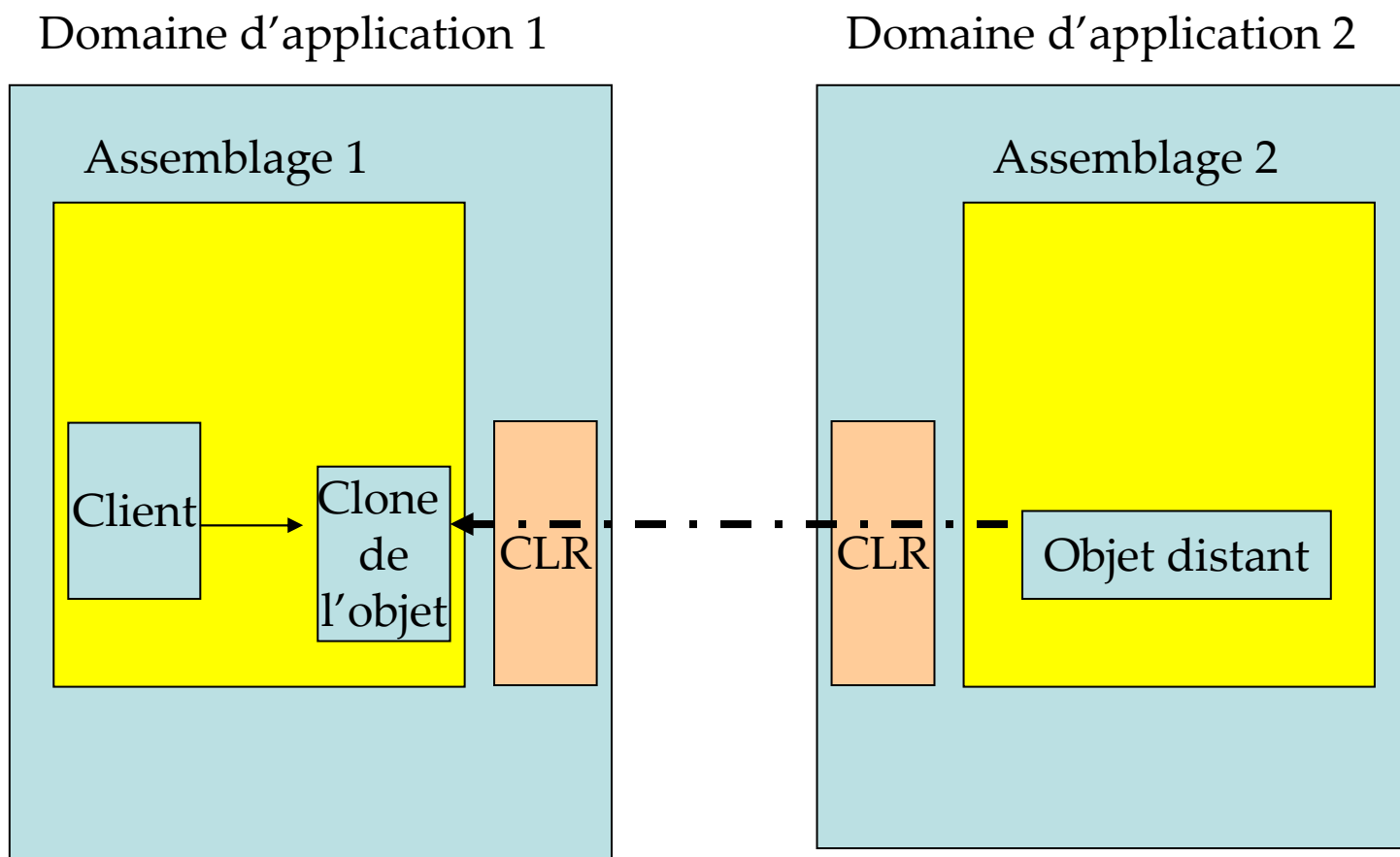
✓ Sa classe possède l'attribut `System.Serializable` signalé au CLR ou

✓ 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 désérialise le stream binaire et reconstitue l'état dans le clone**

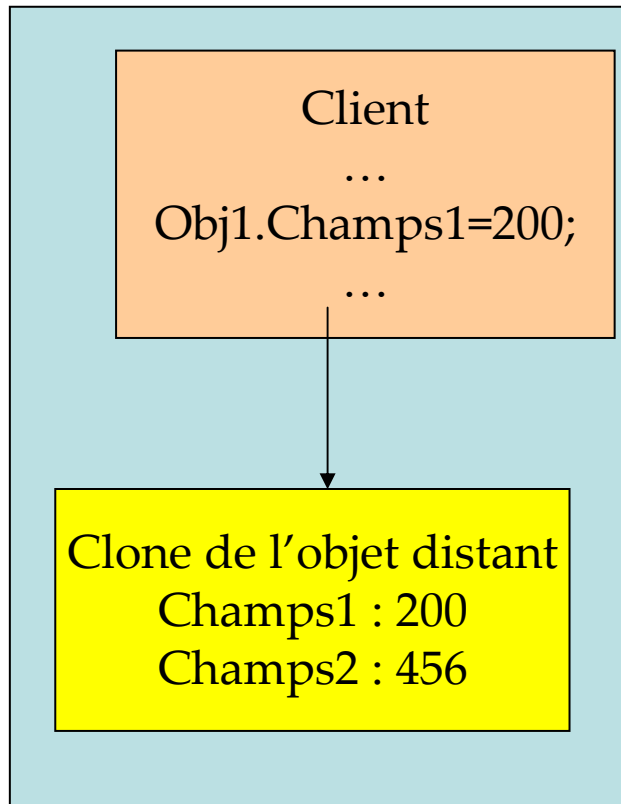
MBV (Marshalling By Value)/2



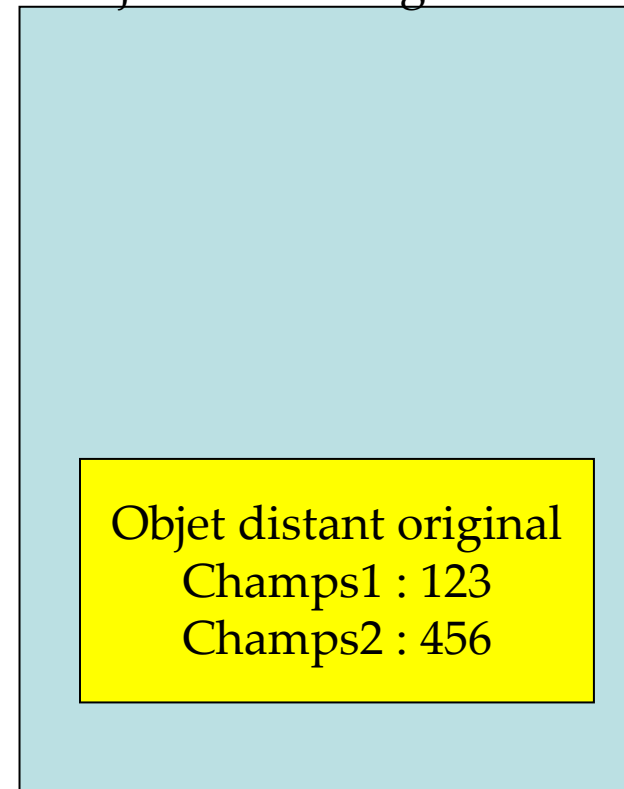
Après clonage dans le domaine du client, les états du clone et de l'objet distant sont indépendants

MBV (Marshalling By Value)/3

Domaine d'application du client

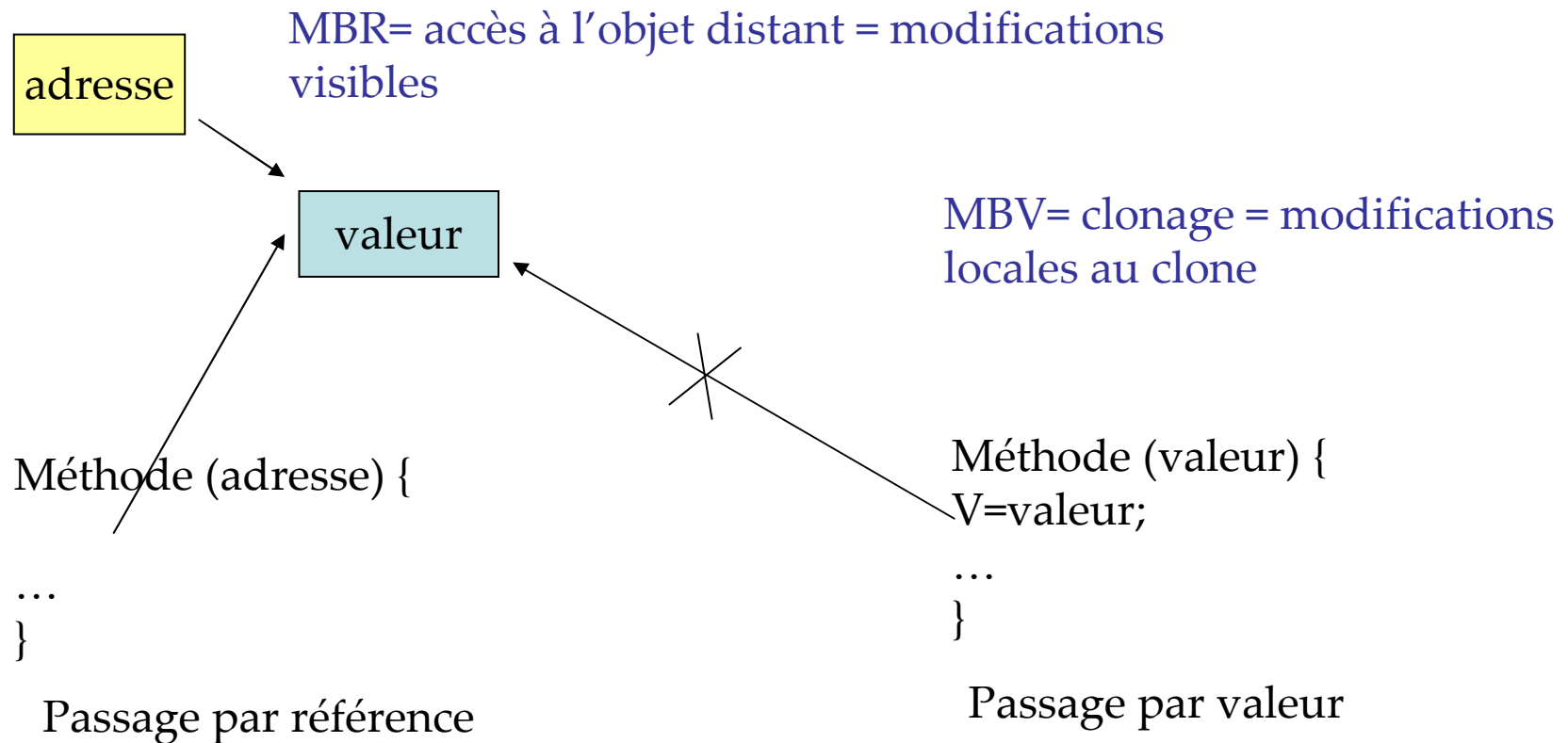


Domaine d'application contenant l'objet distant original



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

IsObjectOutOutOfAppDomain(obj1)=False

IsTransparentProxy(obj1)=False

Obj2:

Nom du domaine: MBVTest.exe

ThreadID : 3620

IsObjectOutOutOfAppDomain(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

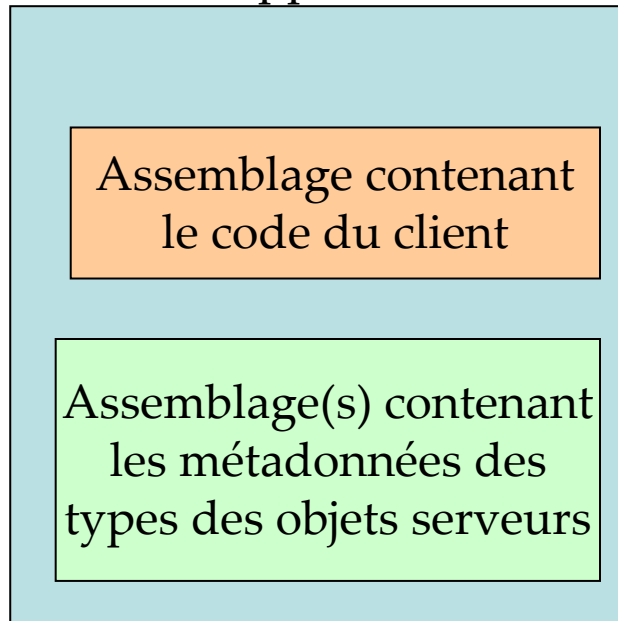


Architecture distribuée avec objets serveurs MBR

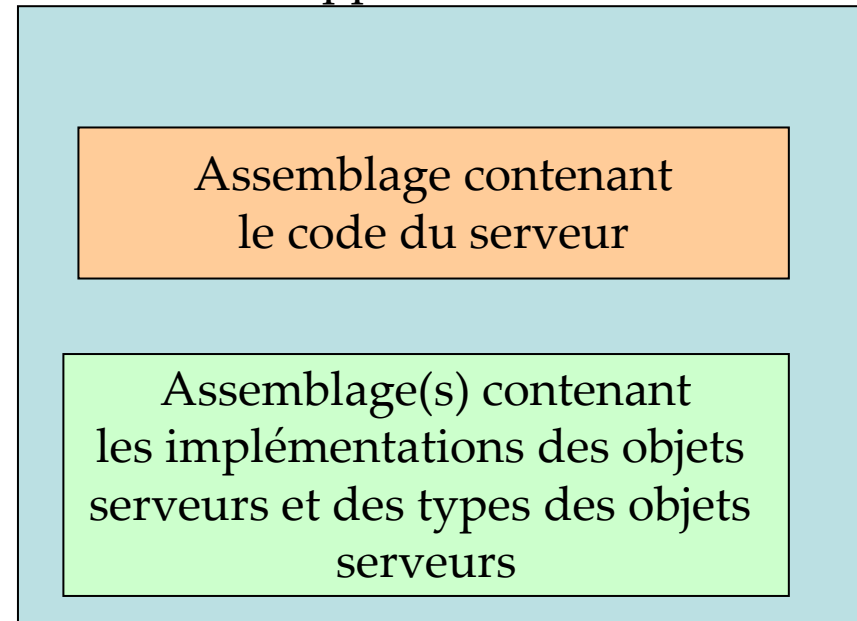
- **Il existe 4 entités :**
 - ✓ Les clients qui appellent les objets serveurs
 - ✓ Les hôtes qui hébergent les objets serveurs
 - ✓ Les métadonnées des types des objets serveurs
 - ✓ Les implémentations des objets serveurs
- **Recommandation : isoler chaque entité dans un assemblage**

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**

- **Une classe est exposée si le client doit connaître la classe à instancier**

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, HTTP), 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 Interface.cs
```

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


Exemple (Implémentation)

Compilation:

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

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
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 http sur le port 65100
        //enregistre ce canal dans le domaine d'application courant.
        HttpChannel canal = new HttpChannel(65100);
        ChannelServices.RegisterChannel(canal, false);

        // 2) Ce domaine d'application presente un objet de type
        // IAdditionneur associe au point terminal 'ServiceAjout'.
        RemotingConfiguration.RegisterWellKnownServiceType(
            typeof(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: *http://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 Client.cs /r:Interface.dll
```

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

using NommageInterface;

namespace NommageClient {
    class Program {
        static void Main() {
            // Crée un canal HTTP 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

```
HttpChannel canal = new HttpChannel(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(IAdditionneur),
        "http://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 Interface.cs  
csc.exe Serveur.cs /r:library  
csc.exe Client.cs /r:library
```

Affichage du Serveur.exe

Pressez une touche pour stopper le serveur.

```
CAdditionneur ctor  
CAdditionneur Add( 3 + 4)
```

Affichage du Client.exe

Valeur retournee:7

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)
```


Activation d'objets par le client (CAO)

- Dans le cas WKO, le client connaît uniquement l'interface de l'objet
- Dans le cas CAO (*Client Activated Object*), le client connaît la classe de l'objet
- Le serveur utilise la méthode statique `RegisterActivatedServiceType()` : classe activable à partir d'un autre domaine
- Le client utilise la méthode statique `Activator.CreateInstance()` pour récupérer le proxy sur l'objet activé.

Exemple avec CAO (Objet)

Compilation:

```
csc.exe /out:ObjServeur.dll /target:library ObjServeur.cs
```

```
using System;
```

```
namespace NommageObjServeur {  
    public interface IAdditionneur {  
        double Add(double d1, double d2);  
    }  
    // Implementation de la classe des objets serveurs.  
    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 avec CAO (Serveur)

Compilation:

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

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using NommageObjServeur;

namespace NommageServer {
    class Program {
        static void Main() {
            HttpChannel canal = new HttpChannel(65100);
            ChannelServices.RegisterChannel(canal, false);
            // Fait en sorte que ce domaine d'application presente
            // la classe CAdditionneur accessible a distance
            RemotingConfiguration.RegisterActivatedServiceType(
                typeof(CAdditionneur));
            Console.WriteLine("Pressez une touche pour stopper le serveur.");
            Console.Read();
        } } }

```

Exemple avec CAO (Client)

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() {  
            HttpChannel canal = new HttpChannel(0);  
            ChannelServices.RegisterChannel(canal, false);  
            // Activation et obtention de proxy  
            IAdditionneur obj = Activator.CreateInstance(  
                typeof(CAdditionneur),  
                null,  
                new Object[] { new UriAttribute("http://localhost:65100")})  
                as IAdditionneur;  
            double d = obj.Add(3.0, 4.0);  
            Console.WriteLine("Valeur retournee:" + d);  
        }  
    }  
}
```

Activation de l'objet par le client avec new

- Avec la syntaxe précédente, s'il y a plusieurs constructeurs, on ne peut pas spécifier quel est le constructeur qui sera lancé par le serveur
- Contrairement à COM/DCOM, le client n'est pas responsable de la durée de vie d'un objet qu'il active
 - ✓ Il peut recevoir une exception du type `System.Runtime.Remoting.RemotingException` parce que l'objet qu'il a activé n'existe plus.

Exemple CAO avec new

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() {  
            HttpChannel canal = new HttpChannel(0);  
            ChannelServices.RegisterChannel(canal, false);  
            // Activation et obtention de proxy  
            RemotingConfiguration.RegisterActivatedClientType(  
                typeof(CAdditionneur),  
                "http://localhost:65100");  
            IAdditionneur obj = (IAdditionneur)new CAdditionneur();  
            double d = obj.Add(3.0, 4.0);  
            Console.WriteLine("Valeur retournee:" + d);  
        }  
    }  
}
```

Le design pattern factory

- L'inconvénient du mode CAO est que l'assemblage du client doit référencer l'assemblage des objets serveurs.
- Ce problème ne se pose pas en WKO car le client partage juste l'interface.
- Pour remédier à ce problème, le mécanisme de design pattern factory est utilisé
 - ✓ Design pattern factory= fabrique qui a pour rôle de construire l'objet CAO par un objet WKO.
 - ✓ Dans ce cas, l'hôte n'expose que le service WKO.
 - ✓ Le client se contentera de l'interface.

Présenter une nouvelle interface

Compilation:

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

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


Implémentation et Exposition de l'objet

```
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;
    }
}
public class CFabrique : MarshalByRefObject, IFabrique {
    public IAdditionneur FabriquerNouvelAdditionneur() {
        return (IAdditionneur)new CAdditionneur();
    }
}
class Program {
    static void Main() {
        HttpChannel canal = new HttpChannel(65100);
        ChannelServices.RegisterChannel(canal, false);
        RemotingConfiguration.RegisterWellKnownServiceType(
            typeof(CFabrique),
            "ServiceFabrique",
            WellKnownObjectMode.Singleton);
        Console.WriteLine(
            "Pressez une touche pour stopper le serveur.");
        Console.Read();
    }
}
```

Utilisation de la fabrique côté client

```
...  
  
class Program {  
    static void Main() {  
  
        HttpChannel canal = new HttpChannel(0);  
        ChannelServices.RegisterChannel(canal, false);  
  
        MarshalByRefObject tmpObj = (MarshalByRefObject)  
        RemotingServices.Connect(  
            typeof(IFabrique),  
            "http://localhost:65100/ServiceFabrique");  
        IFabrique fabrique = tmpObj as IFabrique;  
        IAdditionneur obj = fabrique.FabriqueNouvelAdditionneur();  
  
        double d = obj.Add(3.0, 4.0);  
  
        ...  
    }  
}
```

Durée de vie

- Un objet WKO en mode simple est détruit immédiatement après le 1^{er} appel.
- En mode Singleton, les objets CAO et WKO ne sont pas détruits après l'appel
- Solution :
 - ✓ Mécanisme de bail (*lease* en anglais) appliqué aux objets de type **MarshalByRefObject** (objets accédés par une entité hors domaines d'appli des objets)

Administrateur de baux

- **Un Objet accédé de l'extérieur peut être récupéré par un ramasse-miettes.**
 - ✓ Pour éviter cela, chaque domaine contient un administrateur de baux
- **Un administrateur de baux :**
 - ✓ Alloue une durée de bail au moment de l'activation de chaque objet MBR accessible à distance
 - ✓ La vérification du bail par l'administrateur est faite périodiquement
 - ✓ La durée du bail peut être prolongée
 - À chaque appel de l'objet
 - Directement par un appel à la méthode **renew()**.
 - ✓ Si un bail est invalide, l'administrateur consulte des objets distants (sponsors) capables de prolonger le bail

Paramètres d'un bail d'un objet

- **Durée initiale du bail (5mn par défaut)**
- **Durée de prolongement du bail (2mn par défaut)**
- **Durée maximale d'attente pour la consultation d'un sponsor (2mn par défaut)**

Exemple avec gestionnaire de bail

```
...  
using System.Runtime.Remoting.Lifetime;  
  
namespace NommageServeur {  
    public class CAdditionneur : MarshalByRefObject, IAdditionneur {  
        public override object InitializeLifetimeService() {  
            ILease bail = (ILease)base.InitializeLifetimeService();  
            if (bail.CurrentState == LeaseState.Initial) {  
                bail.InitialLeaseTime = TimeSpan.FromSeconds(50);  
                bail.RenewOnCallTime = TimeSpan.FromSeconds(20);  
                bail.SponsorshipTimeout = TimeSpan.FromSeconds(20);  
            }  
            return bail;  
        }  
    }  
}  
...
```

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
- Publier CDiviseur en mode activable par des clients distants

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" />
        <activated type="NommageObjServeur.CDiviseur,ObjServeur" />
      </service>
      <channels>
        <channel port="65100" ref="http" />
      </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="http://localhost:65100/Service1.rem" />
        <wellknown
          type="NommageObjServeur.CMultiplicateur,ObjServeur"
          url="http://localhost:65100/Service2.rem" />
      </client>
      <client url="http://localhost:65100/">
        <activated type = "NommageObjServeur.CDiviseur,ObjServeur"/>
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
```

Configuration du client/2

Fichier Client.cs

Compilation:

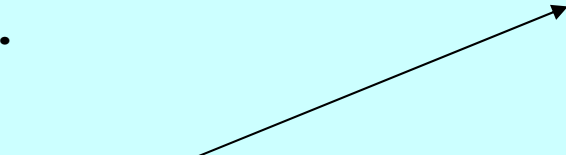
```
csc.exe /out:Client.exe /target:exe Client.cs /r:ObjServeur.dll
```

```
...
```

```
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 (O'Reilly 2005)»*