

Développement en c#.NET

Ing. Meryem OUARRACHI

Plan du module

Langage C#

- ☐ L'environnement .Net
- ☐ Initiation à la programmation C#
- ☐ Programmation Orienté Objet C#

Programmation avancée en .Net ,C#

- ☐ **Programmation distribuée**
- ☐ Gestion de base de donnée
- ☐ Application WPF

CHAPITRE 5:

.Net Remoting

Introduction

- Remoting est un mécanisme de communication entre les objets qui ne sont pas dans le même Process.
- Ca pourra être sur la même machine ou des machines distantes.
- Microsoft *.NET Remoting* est une Interface de programmation (API) pour la communication interprocessus.
- On parle d'un Objet distribué Client/Serveur.

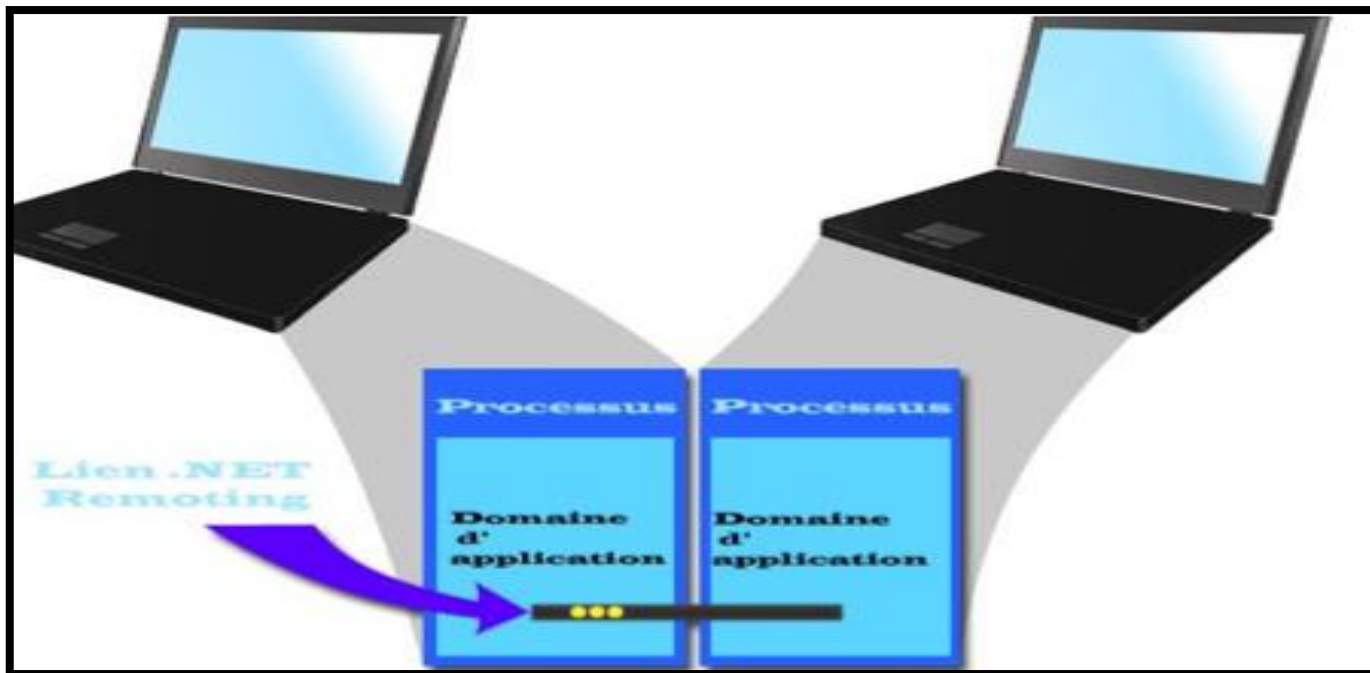
Le .NET Remoting et les sockets

- Les sockets autorisent d'envoyer des informations par le réseau, **sous forme d'octets**.
- Par contre, Le .NET Remoting permet d'envoyer **des objets** par le réseau.
- la communication avec les objets permet d'exécuter les propriétés et méthodes bien plus facilement que la mise en place d'un système à sockets.

Domaine d'application

- Quand une application est chargée en mémoire, un processus est créé, et dans ce processus, un domaine d'application est créé, et dans ce cas l'application sera chargée dans le domaine.
- Cette isolation assure la fiabilité, la sécurité et la disponibilité de l'application.
- Ce domaine d'application est le plus souvent créé par le *Common Language Runtime* (CLR) avant l'exécution de l'application.

Domaine d'application



Marshaling

- Le *marshaling* est le fait d'encoder une information sous la forme d'une suite d'information plus petite.
- l'objet est converti en une suite d'octets dans le but d'être transmis ou stocké. Le principe inverse est l'Unmarshaling.



- Il existe deux types de Marshaling :

1- By Reference (MBR) 2. By Value (MBV)

Marshaling By Value

- Le Marshaling By Value crée une copie de l'objet distant dans le domaine d'application du client.
- Le client utilise et modifie une copie qui **ne modifiera pas l'original** situé dans le domaine d'application **serveur**.
- le client peut utiliser les propriétés et méthodes de l'objet directement dans le domaine d'application client Sans avoir faire d'autres appel du serveur.

Marshaling By Value

Domaine courant

assembly "exo-cours.exe"

objet O1

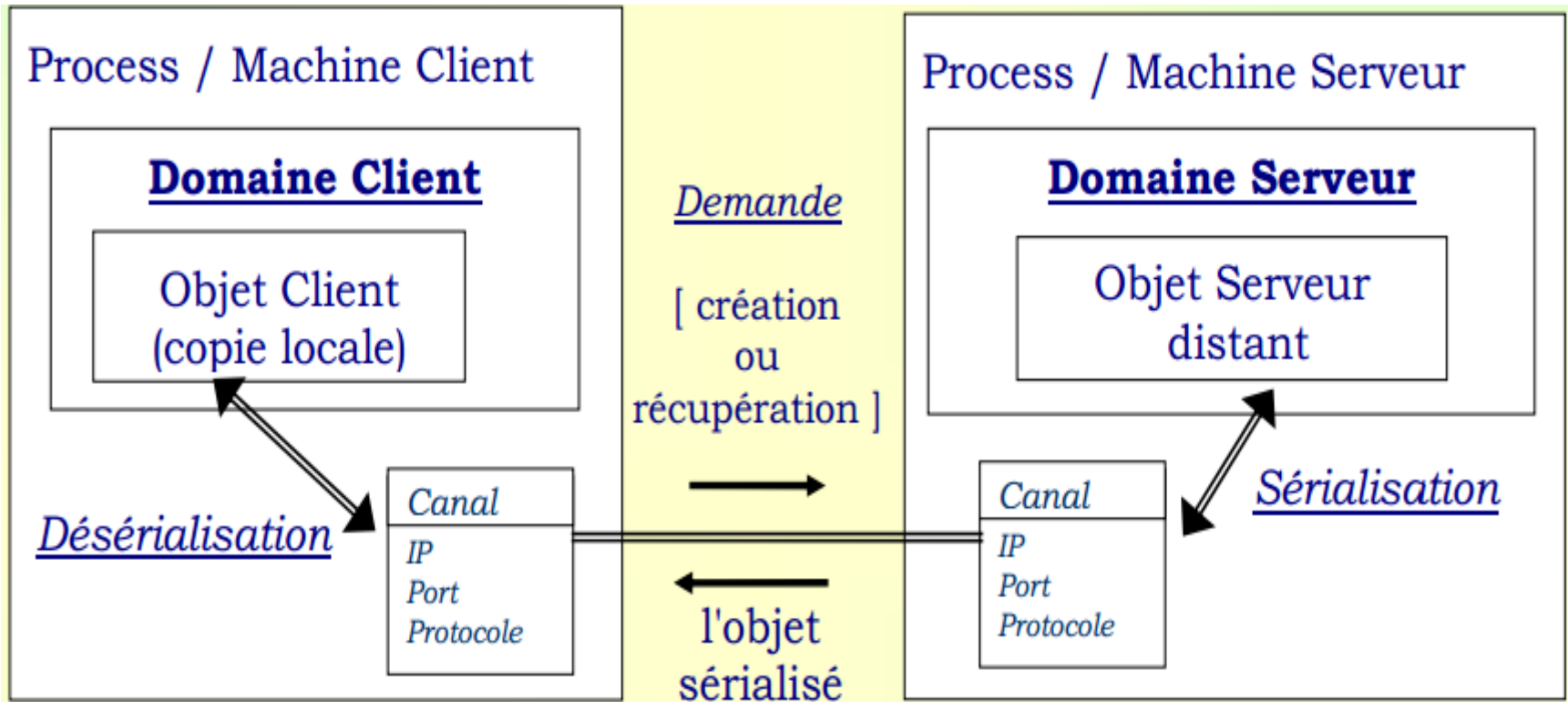
objet O2
(Copie)

Domaine RemotingServeur

assembly "exo-cours.exe"

objet O2
(original)

Marshaling By Value



Marshaling By Value

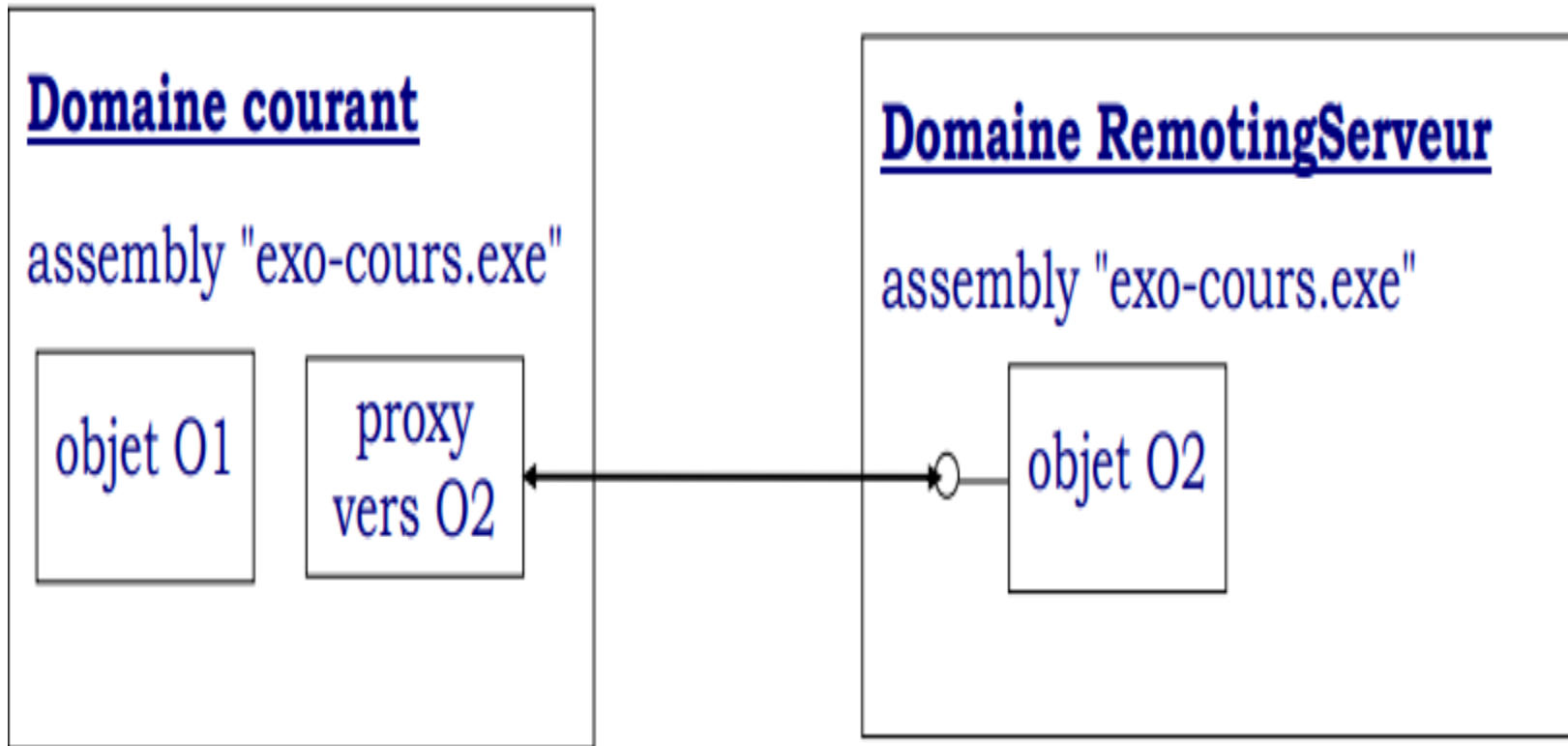
- Limites de MBV :

- Tout ou partie de l'objet distant peut avoir aucune pertinence en dehors de son contexte local .
- Les parties de l'objet à distance peuvent ne pas être serialiazable, par exemple la connexion de base de données

Marshaling By Reference

- Lorsqu'un client appelle un objet serveur par *le Marshaling By Reference*, le Framework crée un objet dans le domaine client.
- Cet objet, appelé *Proxy*, est utilisé pour accéder à l'objet original dans le domaine d'application serveur.
- Le client utilise directement les propriétés et méthodes de l'objet .
- La classe doit hériter **MarshalByRefObject**.

Marshaling By Reference



MBV ou MBR

-MBV: on l'utilise si on a des données lourdes: car le MBV fait une copie et arrête toute transmission ensuite.

-MBR: rapidité

Channels

-Ce sont des objets qui permettent la communication entre le client et le serveur.

-System.runtime.remoting.Channels propose deux types de canaux :
TcpChannel et HttpChannel

*Remarque:*ajouter Espace de noms Remoting: Référence→ ADD →
assembly→system.Runtime.Remoting

```
// Création d'un nouveau canal d'écoute sur le port 100  
TcpChannel channel = new TcpChannel(100);
```

```
// Enregistrement du canal  
ChannelServices.RegisterChannel(channel);
```


Activation

- L'activation : la création des objets qui circulent entre les deux domaines.
- L'activation d'un objet dans des applications distantes peut se faire selon deux choix possibles :
 - Activation par le Serveur (SAO ou WKO:Well known Object)
 - Activation par le client (CAO : Client Activated Object)

Activation par le Serveur (WKO)

- Un objet activé coté serveur est instancié lors du premier appel de méthode
- C à d que lorsque le client fait un new,le proxy est mis en place mais l'objet n'existe pas encore physiquement.
- Objets activés par le serveur ne supporte que des constructeurs par défaut.

Activation par le Serveur (WKO)

Deux Modèles possibles d'activation par le serveur :

- **mode singleton** : une seule instance de l'objet partagée avec tous les clients.
- **mode singlecall** : à chaque appel de méthode une nouvelle instance de l'objet publié est créée sur le serveur → **une instance par appel de méthode.**

Activation mode singleton

- Publication du service (côté serveur)

```
RemotingConfiguration.RegisterWellKnownServiceType(typeof(MaClasse),  
"nomService", WellKnownObjectMode.Singleton);
```

- Récupération de l'objet distribué (côté client) :

```
RemotingConfiguration.RegisterWellKnownClientType(typeof(MaClasse),  
"tcp://localhost:8085/nomService");  
  
MaClasse proxy = new MaClasse();
```

- Si on a utilisé une interface (si MaClasse est une interface) :

```
MonInterface proxy =  
(MonInterface)Activator.GetObject(typeof(MonInterface)," tcp://  
localhost:8085/nomService");
```

Activation mode singlecall

- Publication du service (côté serveur)

```
RemotingConfiguration.RegisterWellKnownServiceType(typeof(MaClasse),  
"nomService", WellKnownObjectMode.Singlecall);
```

- Récupération de l'objet distribué (côté client) :

```
RemotingConfiguration.RegisterWellKnownClientType(typeof(MaClasse),  
"tcp://localhost:8085/nomService");  
  
MaClasse proxy = new MaClasse();
```

- Si on a utilisé une interface (si MaClasse est une interface) :

```
MonInterface proxy =  
(MonInterface)Activator.GetObject(typeof(MonInterface), " tcp://  
localhost:8085/ nomService");
```

Activation par le client (CAO)

Cette fois l'instance est créée dès l'appel du new chez le client. Pas besoin d'attendre le premier appel de méthode, et il est donc possible de passer des paramètres au constructeur.

- Une instance pour chaque client.

Activation par le client (CAO)

- Publication du service (côté serveur)

```
RemotingConfiguration.RegisterActivatedServiceType(typeof(MaClasse));
```

- Récupération de l'objet distribué (côté client) :

```
RemotingConfiguration.RegisterActivatedClientType(  
typeof(MaClasse),"tcp://localhost:8085");  
MaClasse proxy = new MaClasse();
```

- Si on a utilisé une interface (si MaClasse est une interface) :

```
object[] url = new object[] {  
new UriAttribute("tcp://localhost:8085")};  
Interface proxy =  
(Interface)Activator.CreateInstance(typeof(MaClasse),null,url);
```

Utilisation d'un fichier de configuration

```
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode="Singleton"
          type="TypeObj, AssemblyName"
          objectUri= "URI" />
      </service>
      <channels> <channel ref="tcp" port="8085" />
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```


Utilisation d'un fichier de configuration

Exemple:

- Publier CAdditionneur en WKO, mode singleton
- Publier CMultiplicateur en mode simple appel
- Publier CDiviseur en mode activable par des clients distants

Utilisation d'un fichier de configuration

Configuration de 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="tcp" />
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

Nom de la classe

Nom du fichier Serveur

Utilisation d'un fichier de configuration

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();  
        }  
    }  
...  
...
```

Utilisation d'un fichier de configuration

Configuration du client

Fichier Client.config

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


Utilisation d'un fichier de configuration

Dans le corps du client

Fichier Client.cs

```
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