.NET remoting

Clémentine Nebut & Abdelhak-Djamel Seriai
Université de Montpellier

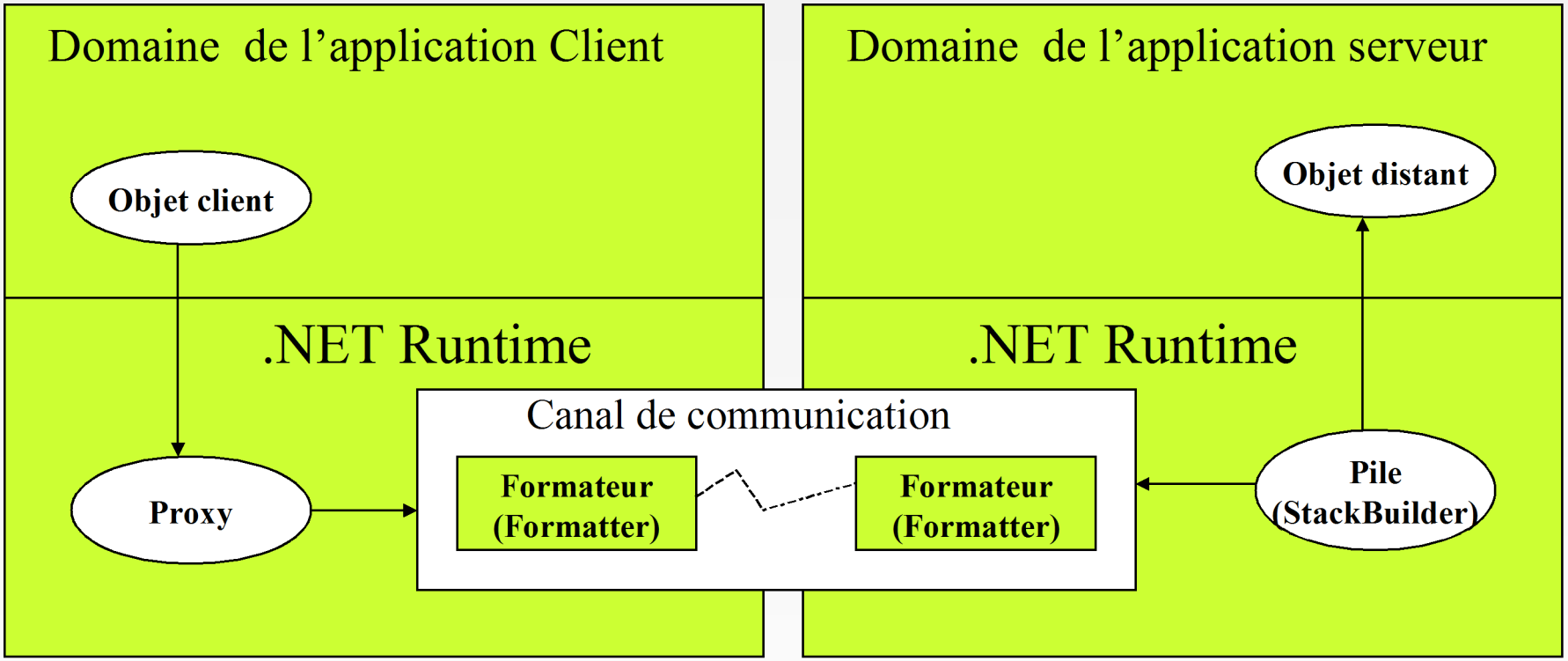
Plan

- Principes de .NET Remoting
  - côté serveur
  - côté client
- .NET Remoting en pratique
  - Les canaux de communication
  - L'activation
  - L'invocation
  - Les paramètres
  - Pour aller plus loin
- Conclusion

Introduction

- Communication client/serveur entre objets .NET distants
- Proche de Java RMI
- Successeur de DCOM

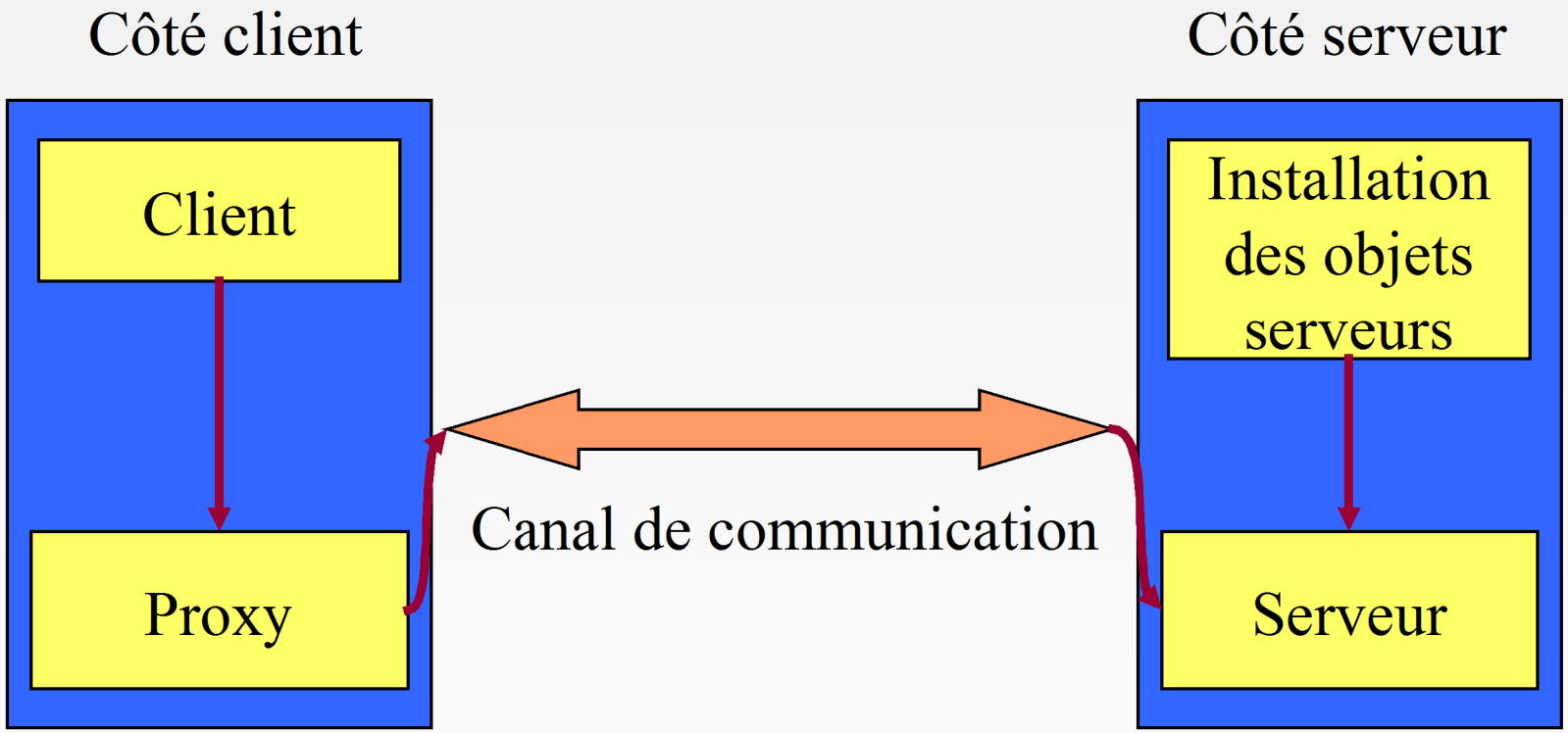
Vue d'ensemble de l'architecture .Net Remoting



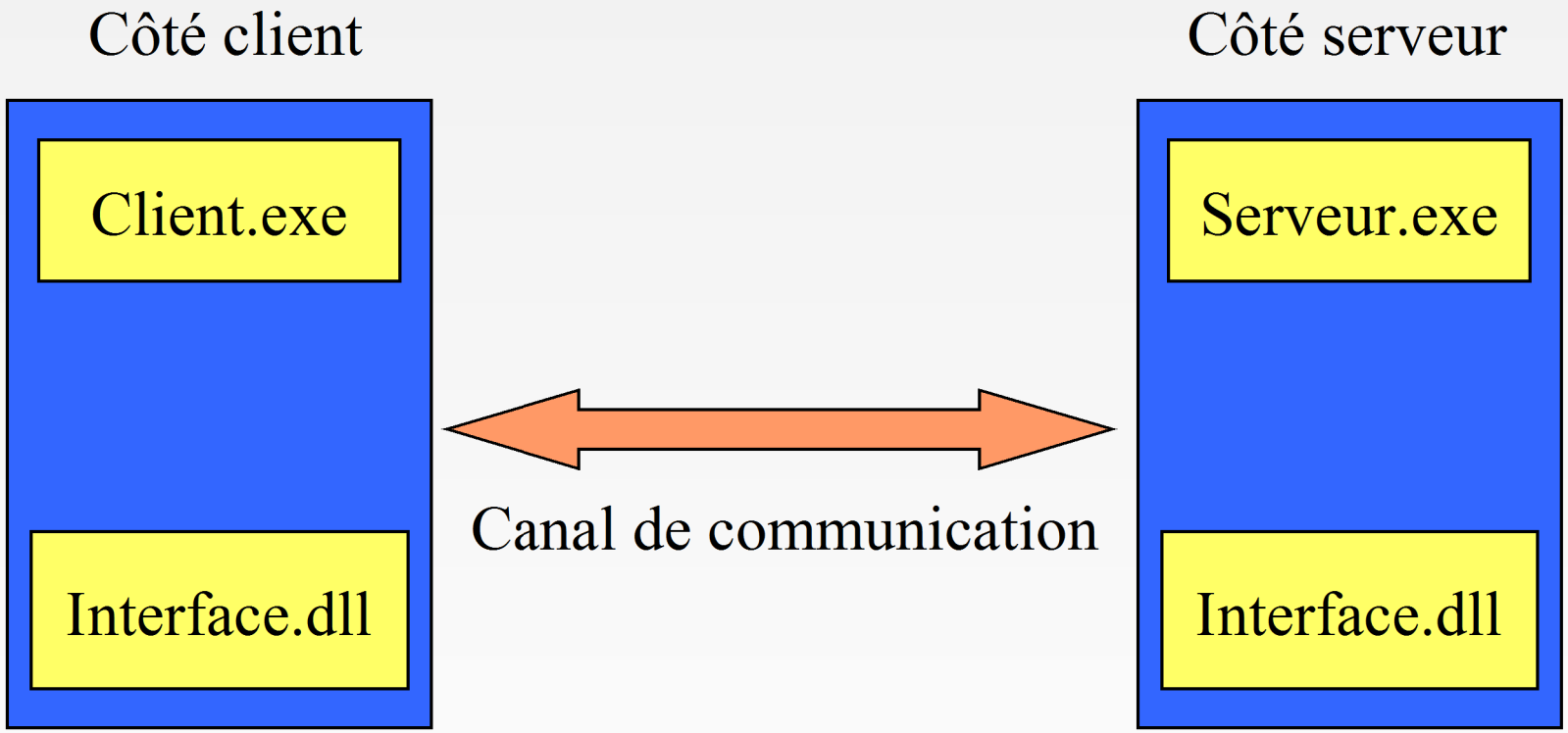
Principe de .net remoting

- Entités qui interagissent :
  - Client
  - Serveur
  - Proxy (généré dynamiquement)
  - (L'interface de l'objet serveur partagée par le client et le serveur)
- 3 programmes :
  - client.exe
  - serveur.exe
  - (interface.dll)

Communication entre les entités



Communication entre les programmes



Côté serveur

- (Définition de l'interface de l'objet distribué)
  - Facultatif mais recommandé
- Définition de l'objet distribué
  - hérite de **MarshalByRefObject** (ou est sérialisable)
  - (implémente l'interface)
- Définition du serveur
  - création du canal de communication
  - enregistrement du canal
  - enregistrement du service
    - Et pas d'un objet

Côté client

- Doit connaître l'interface de l'objet distribué
  - Ou à défaut sa classe
- Récupération d'une référence d'objet distant
- Puis utilisation de la référence.

.NET remoting en pratique

- Les canaux de communication
- Activation d'un objet distribué
- Types d'invocations
- Le passage de paramètres
- Pour aller plus loin …
  - Durée de vie des objets publiés
  - Ramasse-miettes
  - Utilisation d'un fichier de configuration

Espaces de noms à utiliser

- System.Runtime.Remoting
- System.Runtime.Remoting.Channels
- System.Runtime.Remoting.Channels.Tcp
- System.Runtime.Remoting.Channels.Http
- ...

Les canaux de communication

- 2 types de canaux « de base »
  - TCP - TcpChannel
    - format de message binaire
    - transport par socket
  - HTTP - HttpChannel
    - format de message SOAP (Simple Object Access Protocol)
    - transport par http
- On peut créer de nouveaux canaux et/ou formats
  - IIOP (ex. IIOP.NET - http://iiop-net.sourceforge.net)

Les canaux de communication

- A mettre en place côté client et côté serveur
- En TCP, 3 classes :
  - TcpClientChannel
  - TcpServerChannel
  - TcpChannel
    - Combine les fonctionnalités serveur et client

Les canaux de communication, coté serveur

- Définition du numéro de port
- Enregistrement du canal de communication

Si paramètre, création d'un canal serveur

```
//numéro de port
int numport=8085;
// on définit le canal de communication (ici : TCP)
TcpChannel canal=new TcpChannel(numport);
// on l'enregistre
ChannelServices.RegisterChannel(canal, true);
```

indique si le canal est sécurisé ou pas, true valable pour http et TCP

Les canaux de communication, coté client

- Enregistrement du canal de communication

Si pas de paramètre, création d'un canal client

```
// on définit le canal de communication (ici : TCP)
TcpChannel canal=new TcpChannel();
// on l'enregistre
ChannelServices.RegisterChannel(canal, true);
```

indique si le canal est sécurisé ou pas, true valable pour http et TCP

Activation d'un objet distribué (côté serveur)

- Quand un objet est publié côté serveur :
  - son proxy est mis en place quand un client fait un new() ou un Activator.getObject(…)
  - MAIS il n'est instancié que lors du premier appel de méthode
- Activation par le serveur
  - mode singleton = une seule instance de serveur partagée par tous les clients
  - mode singlecall = une instance par appel de méthode
- Activation par le client = une instance par client

Activation mode singleton

- Publication du service (côté serveur) :

```
RemotingConfiguration.RegisterWellKnownServiceType(
typeof(MaClasse), // le type de l'objet distribué
"nomService", // le nom que l 'on donne au service
WellKnownObjectMode.Singleton);//le mode, ici : 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) :

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

Activation mode singlecall

- Publication du service (côté serveur) :

```
RemotingConfiguration.RegisterWellKnownServiceType(
typeof(MaClasse), // le type de l'objet distribué
"nomService", // le nom que l 'on donne au service
WellKnownObjectMode.Singlecall); //mode 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) :

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

Activation côté client

- L'instance est créée dès le new du client
- Publication du service (côté serveur) :

```
RemotingConfiguration.RegisterActivatedServiceType(
typeof(MaClasse)); // le type de l'objet distribué
```

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

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

  - Si on a utilisé une interface :

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

Portée de la publication

- .NET Remoting expose les objets à d'autres domaines d'app. comme s'ils étaient locaux sauf :
  - Membres statiques et méthodes privées.
  - Champs d'instance et accesseurs. Vérif au moment de l'exécution : si objet pas un proxy, accès direct au champ ; sinon, le proxy fournit les accesseurs à l'appelant.
  - Délégués. Marshalés par valeur. L'objet au sein du délégué peut correspondre à n'importe quel type d'objet accessible à distance. Exception : un délégué de méthode d'interface ne peut pas être distant.

Types d'invocations

- Synchrone
  - cas par défaut
- Semi-synchrone (ou asynchrone avec retour)

Côté client
  - une méthode pour déclencher l'appel : BeginInvoke
  - une méthode pour récupérer le résultat : EndInvoke
  - utilisation de délégués

- Asynchrone (ou oneway)
  - côté serveur : attribut [oneway] sur la méthode, pas de type de retour
  - côté client : appel classique, exécution se poursuit même si le serveur n'a pas fini de traiter l'appel

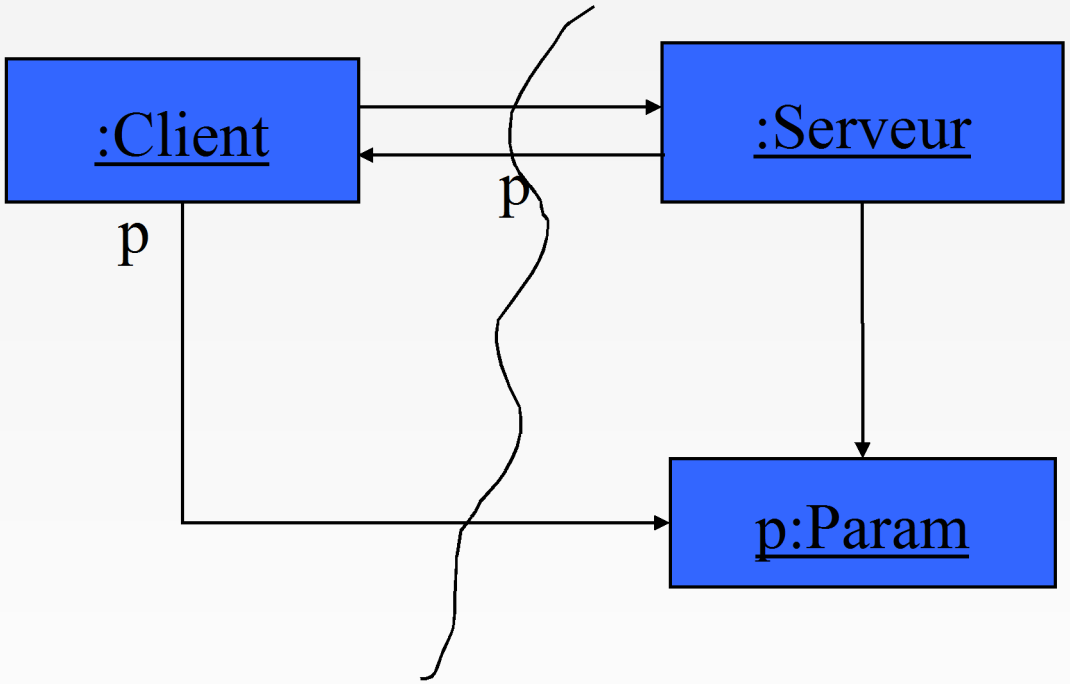
Le passage de paramètres

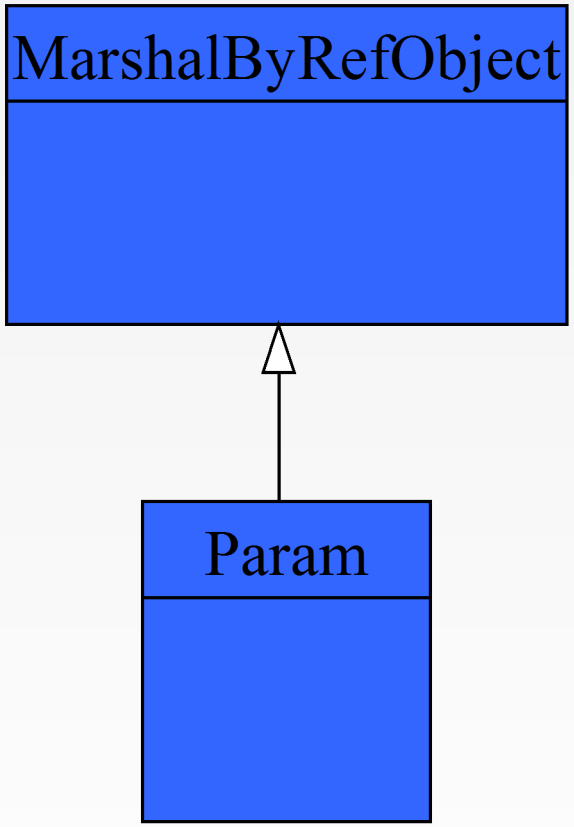
- Par valeur
- Par référence

- Types de base : par valeur
- Objets : par valeur ou par référence

Passage de paramètre par référence

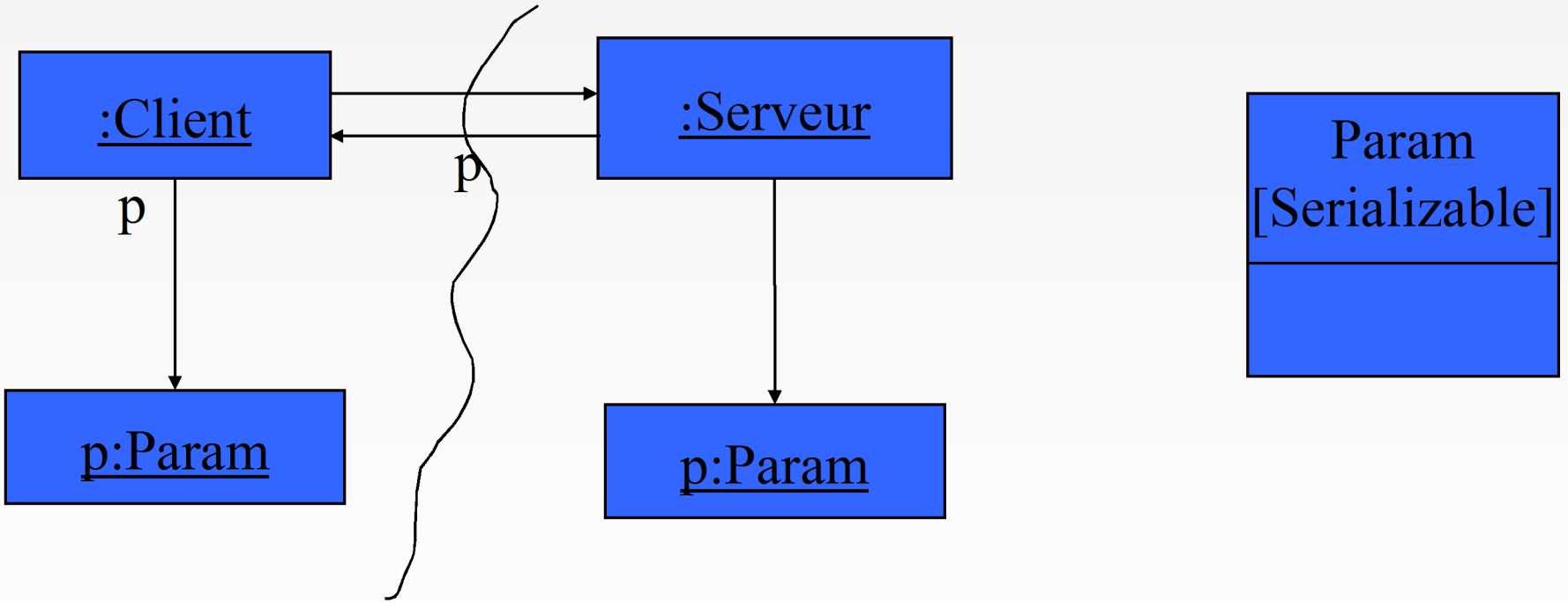
- L'objet transmis doit être d'un type héritant de MarshalByRefObject





Passage de paramètre par valeur

- L'objet transmis doit être sérialisable
  - soit implémenter ISerializable
  - soit attribut [Serializable]



Pour aller plus loin ...

- Durée de vie des objets publiés
- Ramasse-miettes
- Utilisation d'un fichier de configuration

Durée de vie des objets publiés

- Si objet activé côté serveur en singlecall
  - durée de vie pas spécifiable car 1 nouvelle instance à chaque appel
- Les objets activés côté client et côté serveur en mode Singleton possèdent un bail (lease)
  - Par défaut : bail de 5 minutes, renouvelé de 2 minutes à chaque appel de méthode
  - Possibilité de personnaliser le bail

Le bail

- Propriétés :
  - InitLeaseTime
    - durée de vie entre activation et 1er appel de méthode
    - accès en lecture et en écriture
  - RenewOnCallTime
    - durée du renouvellement de bail
    - accès en lecture et en écriture
  - CurrentLeaseTime
    - durée restante du bail
    - accès en lecture seule

Personnalisation des baux

- Possible AVANT l'activation de l 'objet (i.e. quand lease.CurrentState==LeaseState.Initial)
- En surchargeant la méthode InitializeLifetimeService de l'objet publié
- Si on veut une durée de vie illimitée, InitializeLifetimeService doit retourner null

Le ramasse-miettes

- Quand libérer la mémoire d'un objet distribué, et donc potentiellement utilisé par des clients ?
- Utilisation des baux …
  - quand le bail d'un objet expire :
    - le gestionnaire de bail interroge les sponsors (clients)
    - un sponsor qui ne répond pas au bout d'un certain temps (SponsorshipTimeout) est supprimé
    - s'il n 'y a plus de sponsors, le bail expire effectivement, et l'objet distribué est détruit

Utilisation d'un fichier de configuration

- Configuration de la connexion dans un fichier XML plutôt que dans le code …

```
RemotingConfiguration.Configure("Config.xml");
```

```
<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>
```

.NET remoting et RMI

- Points communs
  - distribution d'objets
  - passage par valeur et par référence
- Différences
  - publication des objets
  - Interface requise vs facultative
  - ...

La gestion des proxies

- Rappel : proxy = représentation locale d'un objet distant
- En java
  - Proxies générés avec l'outil rmic ou dynamiquement
- En .NET
  - Proxies générés dynamiquement avec un new
  - Nécessité d'avoir fourni des infos de localisation de l'objet distant
  - Nécessité d'avoir déployé chez le client la totalité de l'implémentation de l'objet distribué
  - Donc si on modifie le serveur, il faut modifier tous les clients
  - **Sauf si on a utilisé une interface**

Le service de nommage

- En java : service de nommage présent
  - Serveur de nom, enregistrement des associations noms logiques / localisation
- En .net : rien !
  - Si le serveur change de nom ou d'adresse, il faut changer tous les clients !
  - Pourquoi ?
    - Raisons historiques : COM s'appuyait sur d'autres systèmes pour le nommage
    - Microsoft refuse d'imposer au client un service uniquement dédié au nommage …

La sécurité

- En java
  - Intégrée au middleware de manière native
- En .NET
  - Zones de confiance (intégrité du code)
  - Encryption des messages et authentification possible avec channel TCP
  - Possible avec les serviced components, mais se base sur COM+ (cf cours sur les serviced components) – System.EnterpriseServices.SecurityCallContext
  - Possible si utilisation d'IIS
    - IIS = Internet Information Server = serveur web proposé par Microsoft, concurrent d'Apache

Les performances

- http://www.dotnetguru.org/articles/Comparatifs/ber
  - « .NET en moyenne 2 à 3 fois + rapide que J2EE ! »
  - Mais pas pour rmi vs remoting
- Guerre sur la performance entre J2EE et .NET

Conclusion

- Pas de serveur de noms comme avec RMI
- Génération dynamique de proxy
- Invocation asynchrone ou semi-synchrone
- Activation par le client

Références

- http://www.labo-dotnet.com/desktopmodules/LaboDotnet.Articles/contentfiles/pdf/remoting.pdf
- Cours Lionel Seinturier
- Cours Didier Donsez