Dotnet France
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

Implémentation de la structure .NET Remoting

Sommaire

1 Introduction

Apres avoir étudié les concepts du .NET Remoting dans le chapitre précédent, nous allons voir dans celui-ci comment l'appliquer dans nos programmes. Pour ce faire ce chapitre sera consacré à l'implémentation de la structure .NET Remoting dans un programme simple ; puis nous terminerons par un long exemple de programme Messagerie instantanée.

2 Rappel Architecture distribuée

Avant de nous lancer dans le code, je tiens à faire un bref rappel sur l'architecture distribuée. Celle-ci est donc composée de quatre entités : les métadonnées, les implémentations, les serveurs et les clients. Les implémentations sont effectuées la plupart du temps directement dans les serveurs et dans les clients. C'est pour cette raison que vous ne verrez que 3 entités dans les exemples qui suivent. Dans la pratique, il est en effet commun de créer son programme distribué, en 3 projets :

- Le serveur
- Le client
- L'interface

2.1 Le serveur

Dans tous les projets de mon exemple, le serveur sera un programme console. J'ai retenu cette solution car elle permet d'avoir un programme exécutable simple (vs Winforms, …) qui donne accès à un affichage des fonctions utilisées. De plus le programme peut simplement rester ouvert grâce à la commande :

```
//C#
Console.ReadLine();
```

```
'VB.NET
Console.ReadLine()
```

Notre serveur contiendra donc l'objet distribué bien sur, mais aussi la création du canal, voire l'activation de l'objet si celui-ci est activé côté serveur (voir suite du chapitre).

2.2 Le client

Pour nos clients, nous utiliserons des programmes consoles ou Winforms. Ils nous permettront de bien voir la demande de service ainsi que les possibilités du .NET Remoting.

Le client contient l'appel aux services ainsi que la création du canal vers le serveur et comme pour le serveur, il peut contenir l'activation de l'objet distribué si celui-ci est activé côté client. Enfin, on peut aussi noter que dans les programmes ci-dessous, les clients pourront être lancés plusieurs fois afin de tester l'architecture client objet avec plusieurs clients.

2.3 L'interface

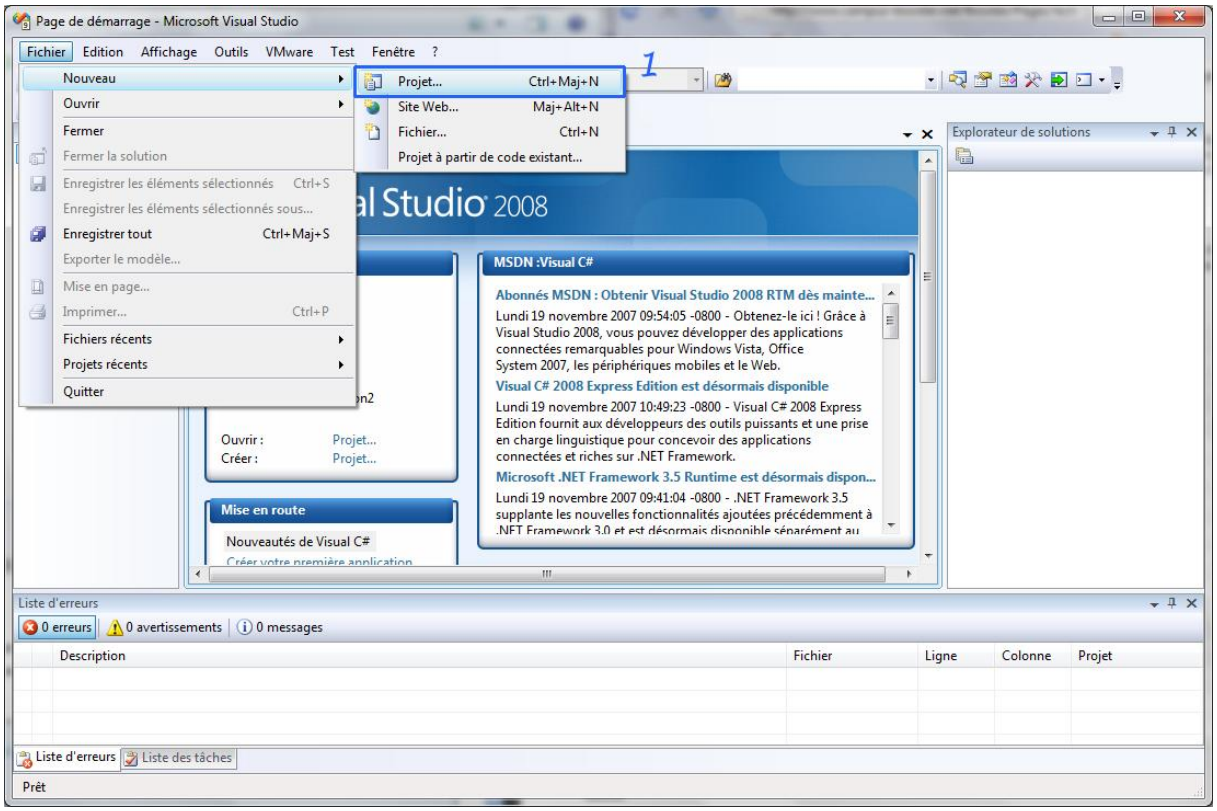
Pas vraiment encore abordée, l'interface permet de faire communiquer le serveur et le client. L'interface contient les prototypes des méthodes distribuées. L'interface doit être référencée dans le projet serveur ainsi que dans le projet client, afin de permettre la communication entre le serveur et les clients.

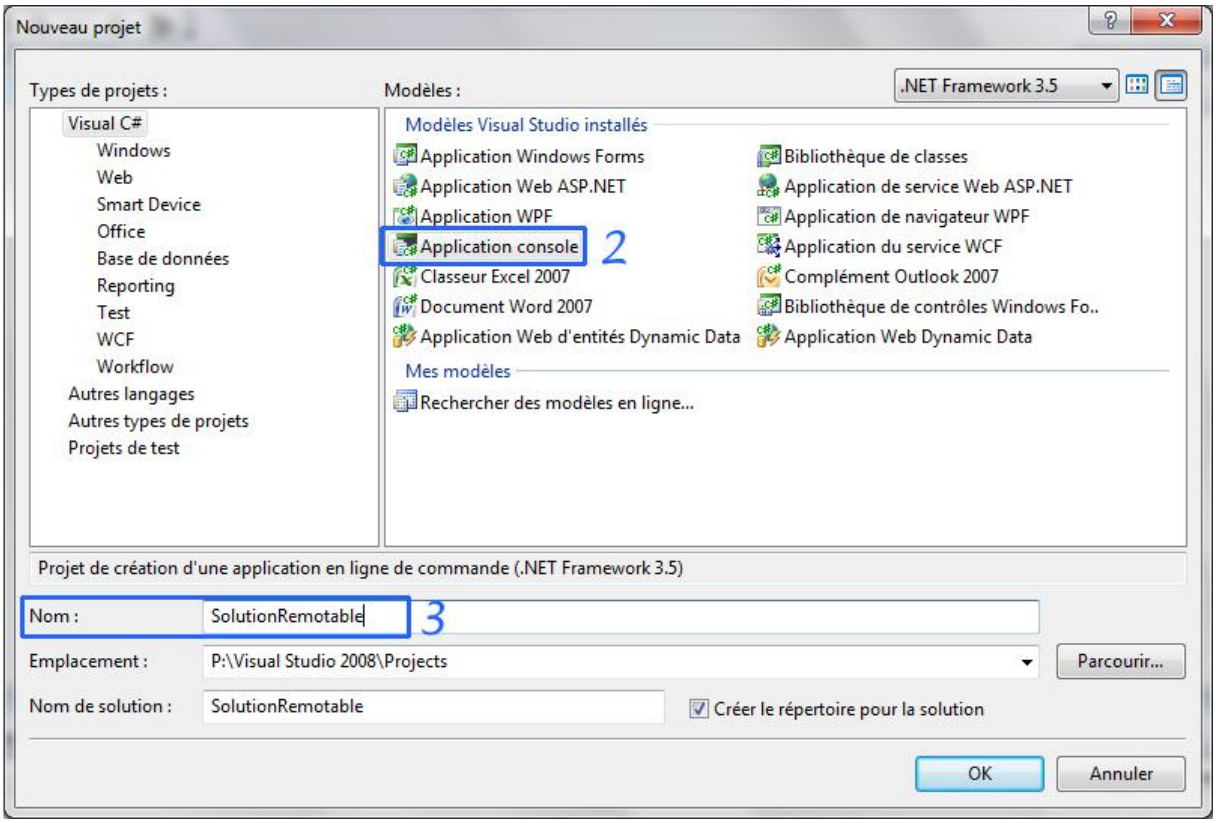
3 Préparation d'un projet incluant l'infrastructure .NET Remoting

Dans cette partie nous allons voir comment créer un programme incluant l'infrastructure .NET Remoting.

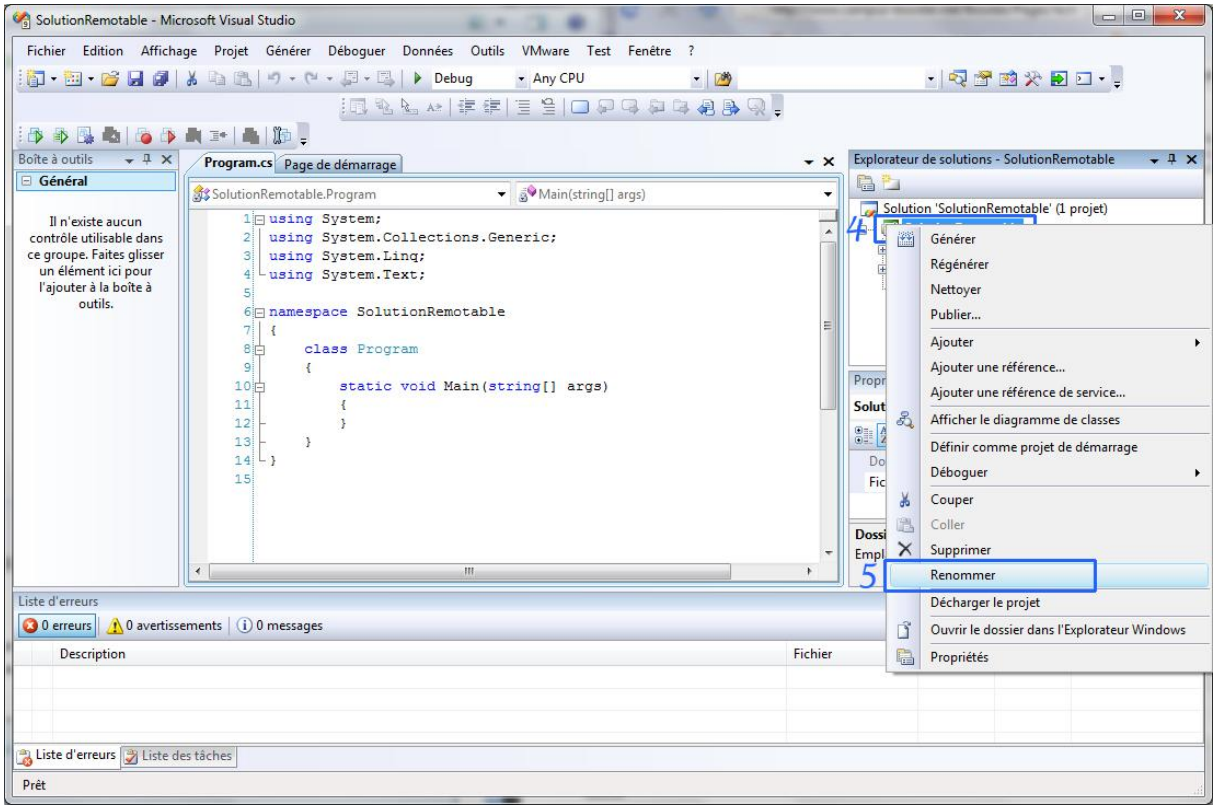
3.1 Création du projet.

Pour le moment, pas de grande différence avec un projet classique, nous allons ouvrir un programme console : Fichier/Nouveau/Projet/Application console.

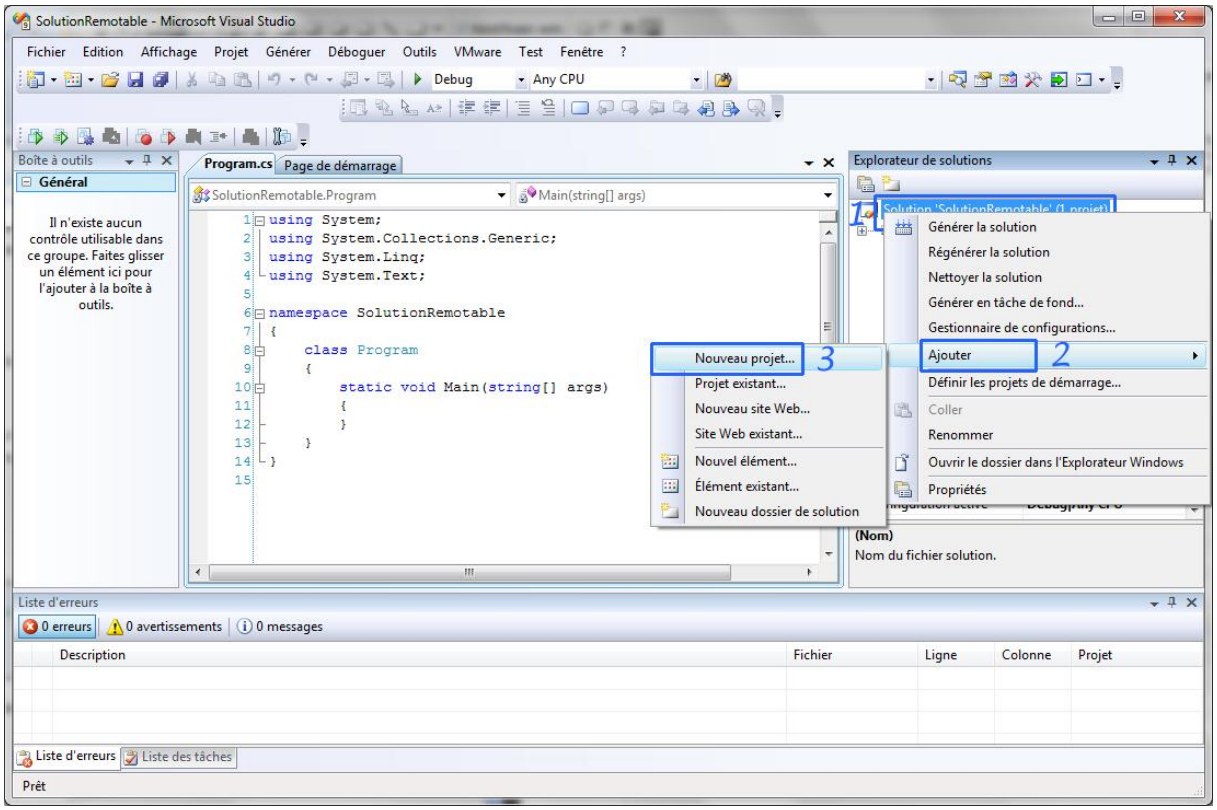


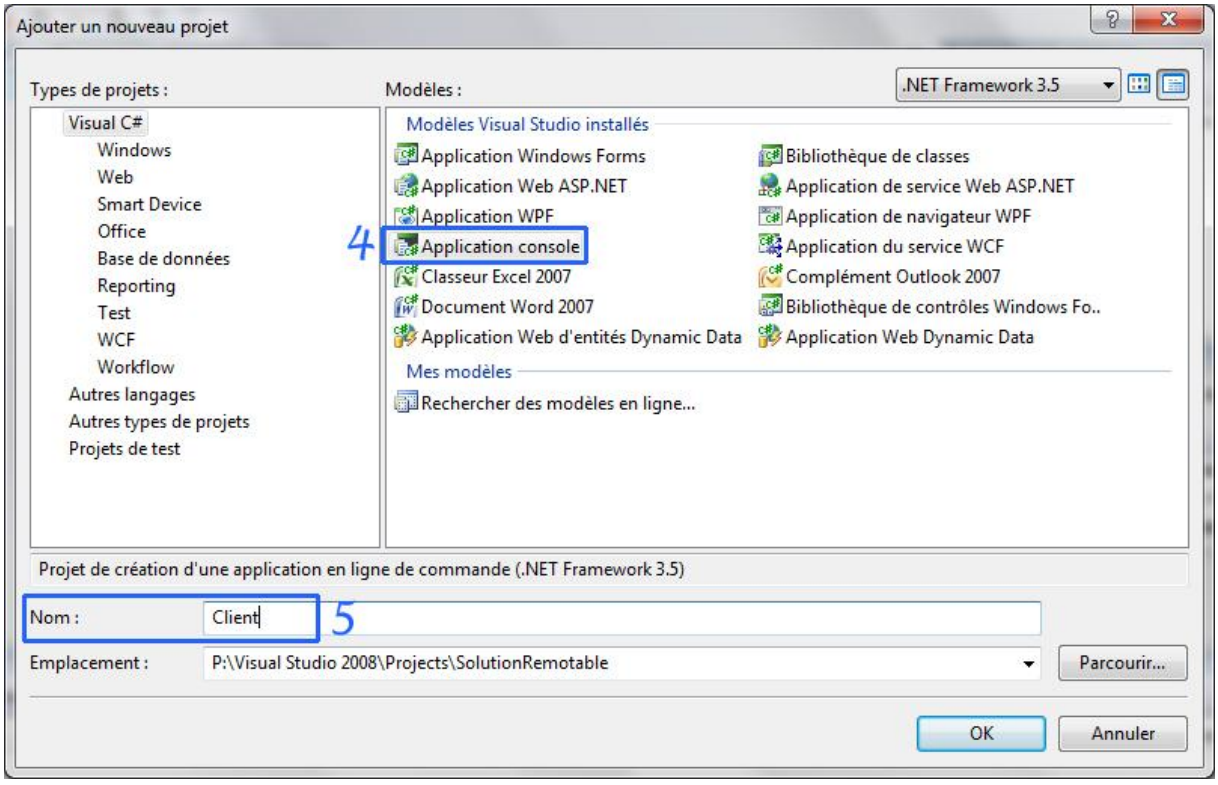


Renommer le projet sous le nom de « Serveur » (ou le nom que vous souhaitez lui donner).

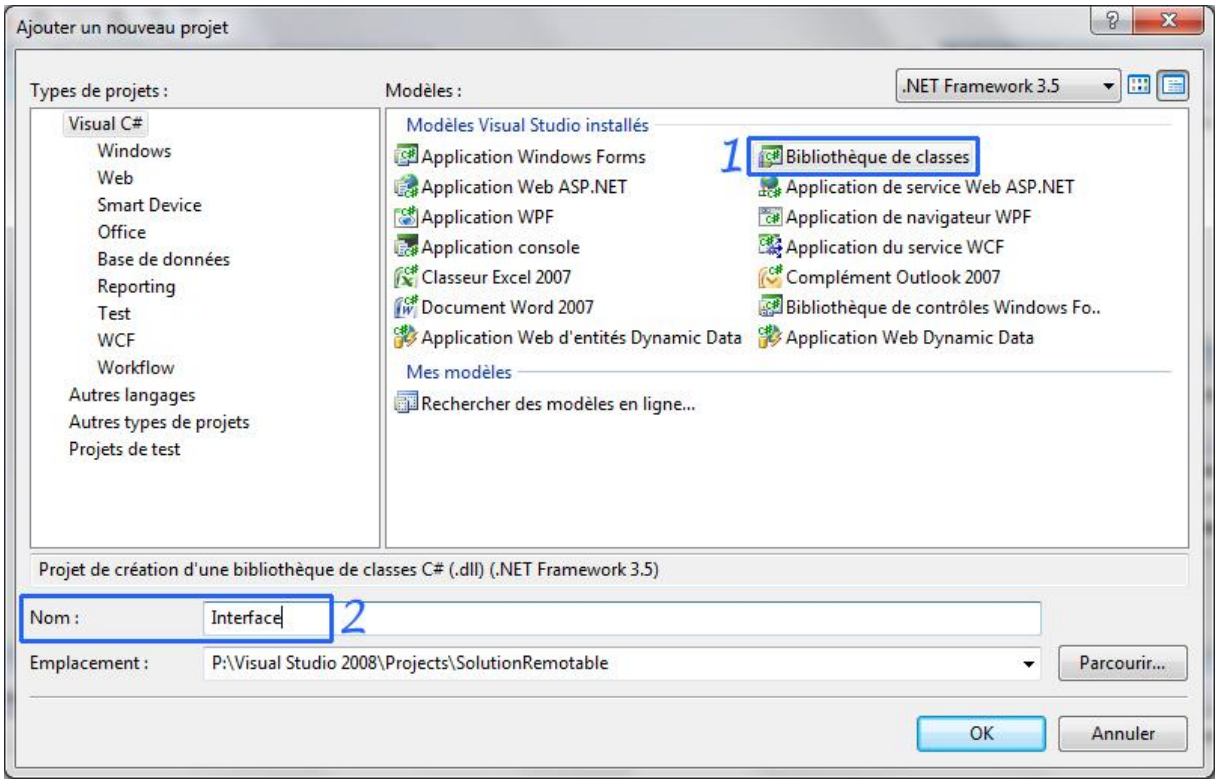


Ensuite nous allons rajouter les clients (ici encore une application console). Clique droit sur la solution puis Ajouter/Nouveau Projet. Choisir application console et appelez-la « Client ».





Enfin pour terminer, ajoutez un dernier projet à la solution. Celui-ci sera de type Bibliothèque de classe. Nommez la interface.

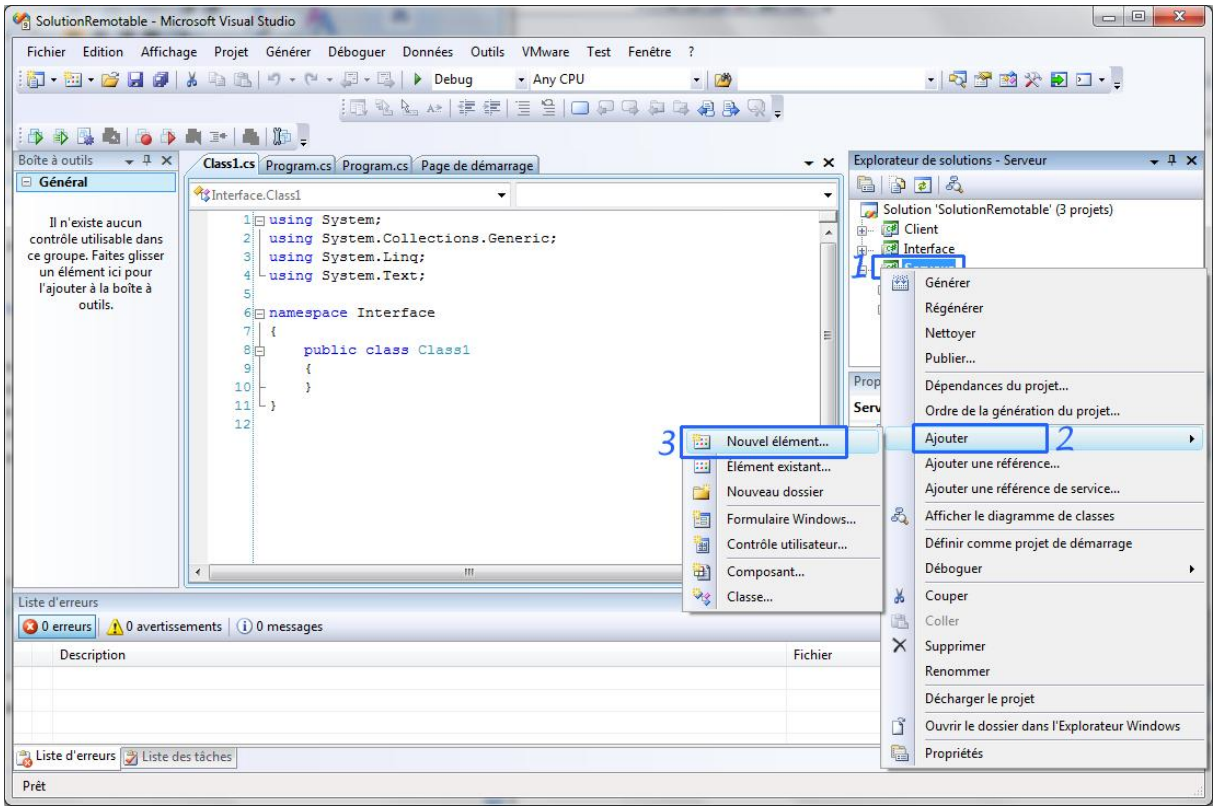


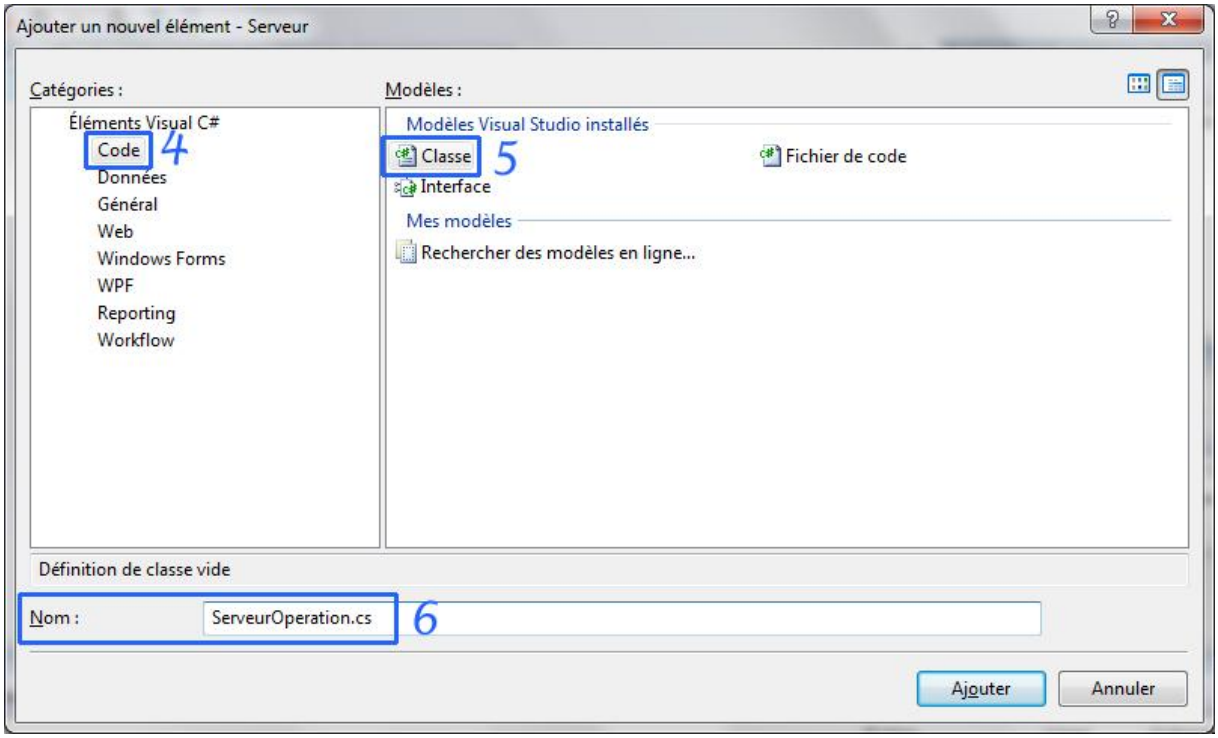
Une fois nos trois projets créés, nous allons commencer à les remplir en mettant en place l'infrastructure.

3.2 Préparation des différents projets et mise en place du canal

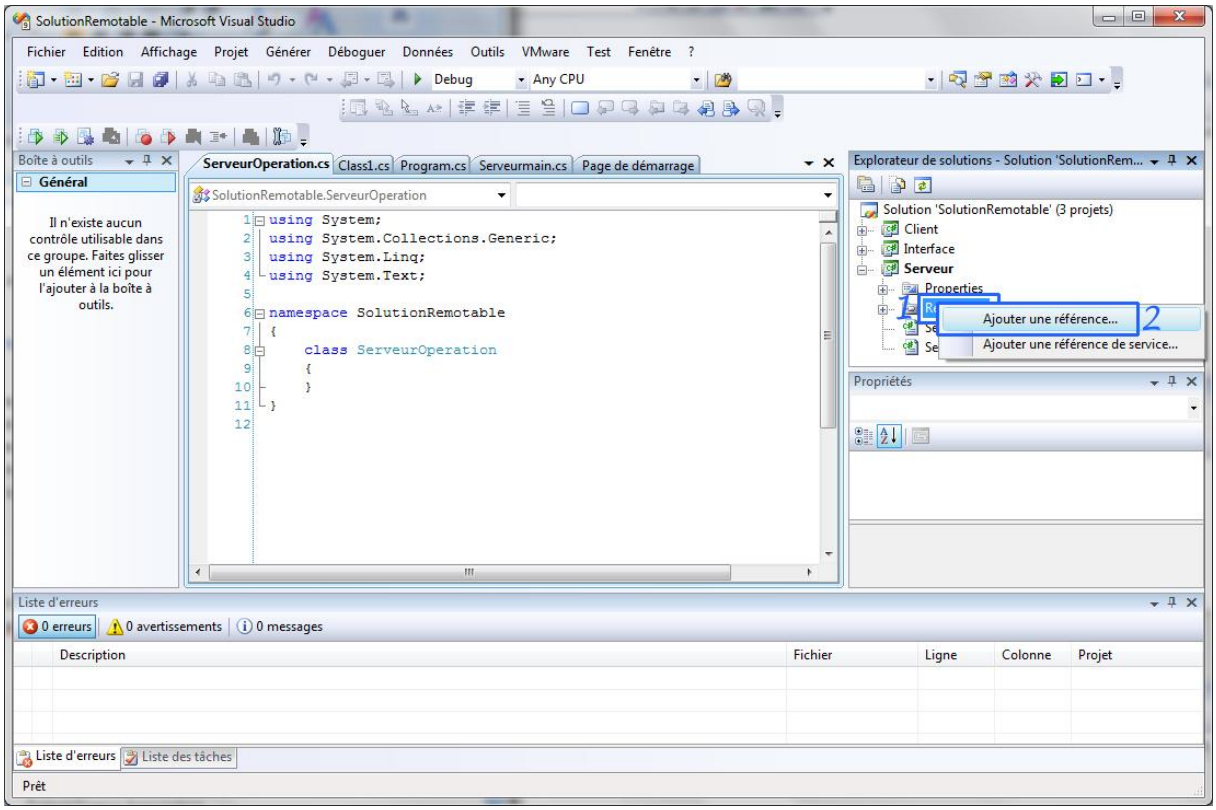
3.2.1 La partie serveur

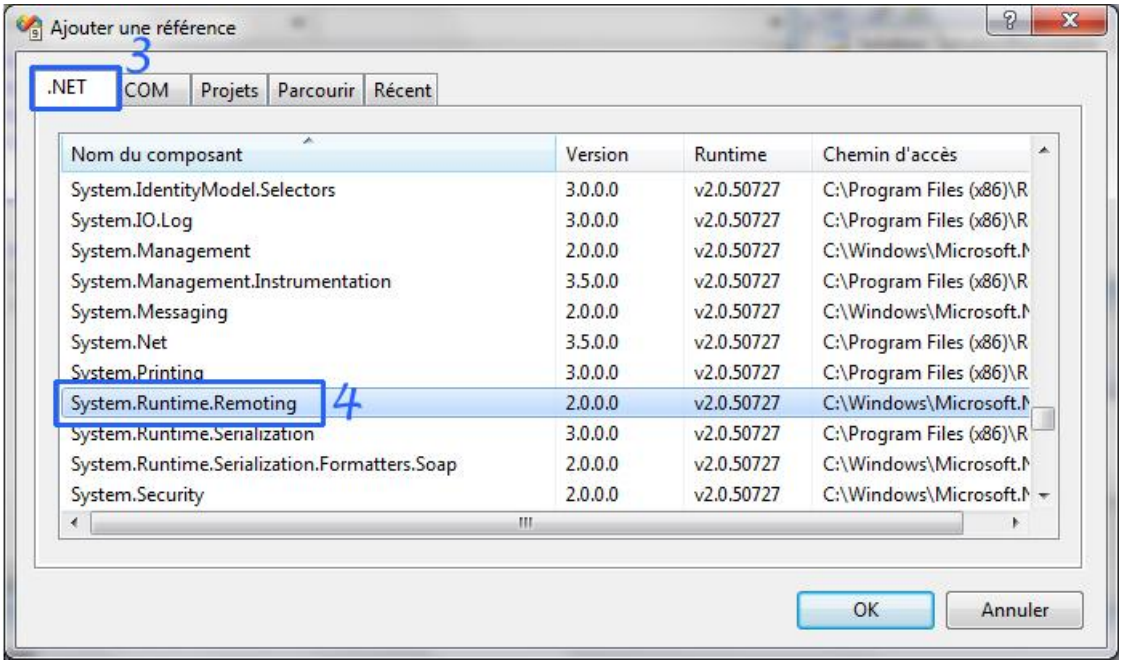
Pour commencer dans notre serveur, nous allons renommer le « program.cs » en « SeveurMain .cs » pour bien montrer que c'est le main de notre programme console et rajouter un nouvel élément au projet Serveur. On effectue un clic droit sur le projet puis Ajouter/Nouvel Elément. Ici on choisit classe (pour mieux vous repérer vous pouvez cliquer sur l'onglet code à gauche de la fenêtre. Nous allons appeler cette classe « `SeveurOperation` ».



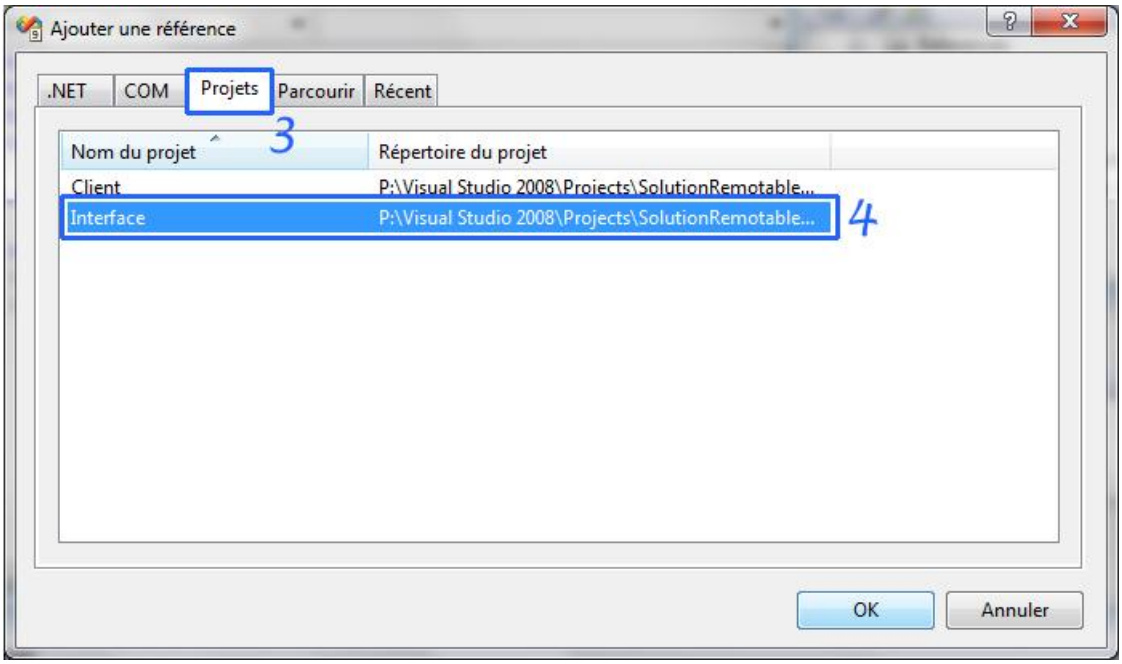


Maintenant pour finir avant de coder, nous allons rajouter les références indispensables. Pour cela faire un clic droit sur reference. Puis on ajoute la référence, `System.Runtime.Remoting` de l'onglet .NET.





Répétez l'opération pour ajouter une référence vers l'interface. Celle-ci se trouve dans l'onglet Projets.



Une fois tout cela fait, nous pouvons nous attaquer au code. Nous allons insérer les nouvelles bibliothèques de classe. Pour cela placez-vous dans le Serveurmain.cs et ajoutez les lignes suivantes à la suite des using.

```
//C#
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
//Pour une utilisation d'un canal TCP
```

```
'VB.NET
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels
Imports System.Runtime.Remoting.Channels.Tcp
'Pour une utilisation d'un canal TCP
```

Pour un canal HTTP les lignes seront les suivantes :

```
//C#
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
```

```
'VB.NET
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels
Imports System.Runtime.Remoting.Channels.Http
'Pour une utilisation d'un canal HTTP
```

Et enfin, il en est de même pour l'IPC :

```
//C#
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Ipc;
//Pour une utilisation d'un canal IPC
```

```
'VB.NET
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels
Imports System.Runtime.Remoting.Channels.Ipc
'Pour une utilisation d'un canal IPC
```

De plus nous allons rajouter dans la classe, la ligne suivante :

```
//C#
namespace SolutionRemotable
{
    class Serveurmain
    {
        [STAThread]//ligne à rajouter
```

```
'VB.NET
Module Serveurmain

    <STAThread()>'ligne à rajouter

    Sub Main()
```

Remarque : cette ligne, située juste avant le main, concerne les processus légers ou Thread, reportez-vous la page MSDN : http://msdn.microsoft.com/fr-fr/library/system.stathreadattribute.aspx .

Nous allons maintenant configurer le canal avec le numéro du port et l'enregistrer.

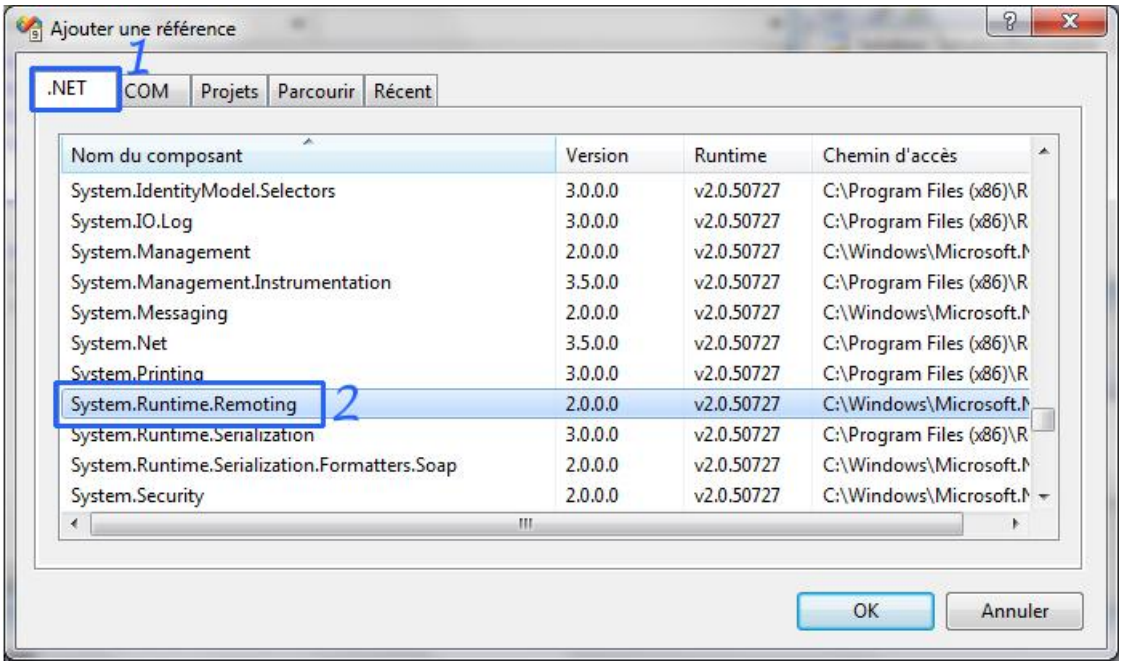
```
//C#
        static void Main(string[] args)
        {
            //configuration du canal TCP
            TcpChannel canal = new TcpChannel(8000);
            ChannelServices.RegisterChannel(canal);
```

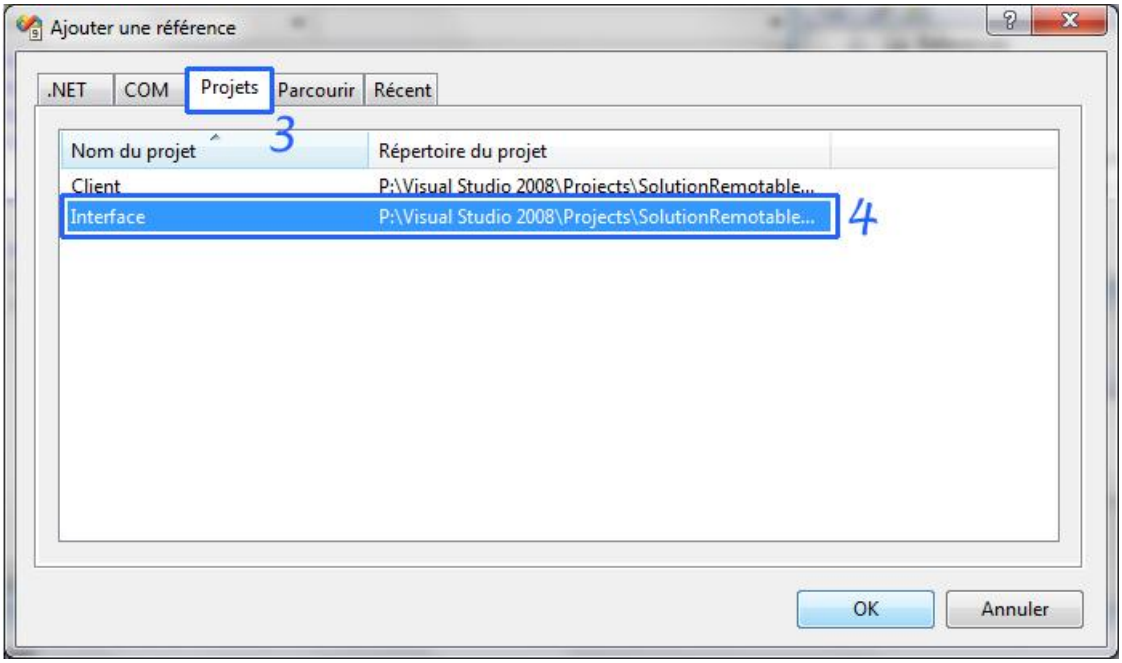
```
'VB.NET
    Sub Main()
        'configuration du canal TCP
        Dim canal As New TcpChannel(8000)
        ChannelServices.RegisterChannel(canal)
```

Ici on déclare un nouveau canal sur le port 8000 et l'enregistre juste après. Pour les autres protocoles les lignes similaires ont été examinées dans le chapitre précédant.

3.2.2 Le client

Le paramétrage de la partie client est assez similaire à celui de la partie serveur. Il consiste là aussi à ajouter les références `System.Runtime.Remoting` et Interface.





Là aussi il faut rajouter les using propres au remoting en fonction de son protocole (ici TCP).

```
//C#
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
//Pour une utilisation d'un canal TCP
```

```
'VB.NET
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels
Imports System.Runtime.Remoting.Channels.Tcp
'Pour une utilisation d'un canal TCP
```

Il faudra là encore configurer le canal TCP :

```
//C#
        static void Main(string[] args)
        {
            //configuration du canal TCP
            TcpChannel canal = new TcpChannel();
            ChannelServices.RegisterChannel(canal);
```

```
'VB.NET
    Sub Main()
        'configuration du canal TCP
        Dim canal As New TcpChannel()
        ChannelServices.RegisterChannel(canal)
```

Remarque : ici j'ai nommé le canal côté client de la même façon que côté serveur. Il est tout à fait possible de changer le nom, le seul paramètre qui doit être indispensablement le même est le protocole.

Le numéro du port ne doit pas être indiqué comme sur le serveur sinon le programme essayera de créer deux canaux différents sur le même port. Bien que votre programme compile, il cessera de fonctionner à la tentative de création du canal côté-client.

3.2.3 Interface

L'interface contiendra donc les prototypes des méthodes distribuées. Ici il faudra procéder à quelques changements. A commencer par renommer Class1.cs par « IServeurOperation.cs » (il est par convention d'usage d'appeler la classe de la même façon que la classe contenant les méthodes distribuée mais en rajoutant un I majuscule devant).

Ensuite dans le cadre il faudra remplacer la ligne

```
//C#
public class IServeurOperation
```

```
'VB.NET
Public Class IServeurOperation
```

par la ligne suivante correspondant à une interface

```
//C#
public interface IServeurOperation
```

```
'VB.NET
Public Interface IServeurOperation
```

Ici s'achève la première partie sur la configuration des canaux. Dans la seconde partie nous verrons le Marshaling.

3.3 Le marshaling

Pour rappel il existe deux versions du marshaling (comme vu dans le cours précédent).

3.3.1 Le marshaling by value

Le marshaling by value est similaire à de la sérialisation. Il faut donc que le type soit sérialisable. La plupart des types primitifs sont sérialisables tels que le int ou le string. Il est tout à fait possible de personnaliser le processus de sérialisation et désérialisation avec l'implémentation de `System.Runtime.Serialization.ISerializable` dans la classe (se reporter au cours d'ADO.NET).

```
//C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SolutionRemotable
{
    [Serializable]

    public class ServeurOperation
    {
        private int nombre1;
        private string chaine1;

        public string rechaine1
        {
            get { return this.chaine1;}
        }

        public string affichage(string rechaine1)
        {
            string chaine2 = "Hello " + rechaine1;
            return chaine2;
        }
    }
}
```

```
'VB.NET
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text

Namespace SolutionRemotable

    <Serializable()> _
    Public Class ServeurOperation
        Private nombre1 As Integer
        Private chaine1 As String

        Public ReadOnly Property rechaine1() As String
            Get
                .Return(Me.chaine1)
            End Get
        End Property

        Public Function affichage(ByVal rechaine1 As String) As String
            Dim chaine2 As String = "Hello " & rechaine1
            .Return(chaine2)
        End Function
    End Class
End Namespace
```

Ici tout notre code sera copié dans le domaine d'application et pourra être utilisé à volonté par le client. Pour notre exemple j'utiliserai plutôt la seconde méthode beaucoup plus souvent utilisée en pratique.

3.3.2 Le marshaling by Reference

Pour rappel, le marshaling by reference crée un proxy entre les deux domaines d'application afin de permettre au client d'accéder à l'objet distribué du serveur. Ce modèle est plus utilisé pour la communication entre plusieurs clients par exemple. Pour mettre en place ce modèle, il faudra tout d'abord définir la classe `ServeurOperation` public et lui ajouter le système de marshaling utilisé ainsi que l'interface. Voici celui de notre projet.

```
//C#
public class ServeurOperation : MarshalByRefObject,
Interface.IServeurOperation
```

```
'VB.NET
Public Class ServeurOperation
    Inherits MarshalByRefObject
    Implements Interface.IServeurOperation
End Class
```

Une fois la classe prête, il nous suffit de rajouter les méthodes de notre objet distribué ; dans mon exemple ce sera la méthode suivante qui permet d'effectuer une simple addition.

```
//C#
        public int addition(int a, int b)
        {
            return a + b;
        }
```

```
'VB.NET
Public Function addition(ByVal a As Integer, ByVal b As Integer) As
Integer
    Return a + b
End Function
```

Une fois terminé votre fichier ServeurOperation.cs devra ressembler à cela :

```
//C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SolutionRemotable
{
    public class ServeurOperation : MarshalByRefObject,
Interface.IServeurOperation
    {
        public int addition(int a, int b)
        {
            Console.WriteLine("Méthode addition effectuée");
            return a + b;
        }
    }
}
```

```
'VB.NET
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text

Namespace SolutionRemotable
    Public Class ServeurOperation
        Inherits MarshalByRefObject
         Implements Interface.IServeurOperation
        Public Function addition(ByVal a As Integer, ByVal b As Integer)
As Integer
            Console.WriteLine("Méthode addition effectuée")
            Return a + b
        End Function
    End Class
End Namespace
```

Remarque : on a rajouté un Console.WriteLine qui nous permettra de voir les actions côté serveur.

Pour permettre à notre programme de créer un proxy il faudra ajouter le prototype des méthodes dans l'interface (représentant le contrat de service) comme suivant (IServeurOperation.cs) :

```
//C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Interface
{
    public interface IServeurOperation
    {
        int addition(int a, int b);
    }
}
```

```vb
'VB.NET
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text

Namespace [Interface]
    Public Interface IServeurOperation
        Function addition(ByVal a As Integer, ByVal b As Integer) As
Integer
    End Interface
End Namespace
```

Ici s'achève la partie configuration du marshaling ; il reste donc à configurer l'activation de l'objet, la durée de vie et enfin nous verrons comment faire appel à l'objet.

3.4 Activation

Pour rappel il existe trois types d'activation, deux côtés serveur et un côté client.

3.4.1 Activation côté-client (CAO)

L'activation côté-client nécessite 3 étapes essentielles sur le code serveur:

- Création du Canal de communication (déjà effectué en amont)
- Enregistrement du canal dans le moteur de Remoting (déjà effectué en amont)
- Enregistrement du type `Remotable` dans le moteur de Remoting.

Pour effectuer ce dernier enregistrement, il suffit de rajouter les lignes suivantes à notre code :

```csharp
//C#
//Enregistrement du type remotable
RemotingConfiguration.ApplicationName = "Operation";
RemotingConfiguration.RegisterActivatedServiceType(typeof(ServeurOperatio
n));
```

```vb
'VB.NET
'Enregistrement du type remotable
     RemotingConfiguration.ApplicationName = "Operation"
     RemotingConfiguration.RegisterActivatedServiceType(GetType(ServeurOp
     eration))
```

La première ligne enregistre un nouveau service par un nom « `Operation` » dans notre cas. Il est tout à fait possible de l'enregistrer par un id aussi.

La seconde ligne indique le type qui sera activé par le client. Ici notre type est la classe contenant notre méthode addition.

Côté client, les trois mêmes étapes sont nécessaires pour configurer le client consommant le service CAO.

```
//C#
//Enregistrement du type remotable
RemotingConfiguration.RegisterActivatedClientType(typeof(ServeurOperation
), "tcp://localhost:8000/ServeurOperation");
```

```
'VB.NET
'Enregistrement du type remotable

RemotingConfiguration.RegisterActivatedClientType(TypeOf(ServeurOperation
),"tcp://localhost:8000/ServeurOperation")
```

3.4.2 Activation côté-serveur

Il existe deux types d'activation côté serveur, Singlecall et Singleton, mais leur programmation est assez similaire.

Côté serveur il faut enregistrer le service avec la ligne suivante :

```
//C#
RemotingConfiguration.RegisterWellKnownServiceType(typeof(ServeurOperatio
n), "ServeurOperation", WellKnownObjectMode.SingleCall);
```

```
'VB.NET
RemotingConfiguration.RegisterWellKnownServiceType(TypeOf(ServeurOperatio
n),"ServeurOperation", WellKnownObjectMode.SingleCall)
```

Alors ici les différents paramètres correspondent au type de services, nom du service et type d'activation. Il est tout à fait possible de remplacer SingleCall par Singleton pour utiliser l'autre type d'activation.

```
//C#
RemotingConfiguration.RegisterWellKnownServiceType(typeof(ServeurOperatio
n), "ServeurOperation", WellKnownObjectMode.Singleton);
```

```
'VB.NET
RemotingConfiguration.RegisterWellKnownServiceType(TypeOf(ServeurOperatio
n),"ServeurOperation", WellKnownObjectMode.Singleton)
```

Pour notre exemple je resterai en mode SingleCall qui est le plus approprié pour une simple addition.

Côté Client, il faut enregistrer les informations relatives au service et créer le proxy transparent. Pour cela il suffit d'ajouter la ligne suivante :

```
//C#
Interface.IServeurOperation Service =
(Interface.IServeurOperation)Activator.GetObject(
                typeof(Interface.IServeurOperation),
                "tcp://localhost:8000/ServeurOperation");
```

```
'VB.NET
Dim Service As Interface.IServeurOperation =
DirectCast(Activator.GetObject(GetType(Interface.IServeurOperation),
"tcp://localhost:8000/ServeurOperation"), Interface.IServeurOperation)
```

Ici on crée d'abord un service de type `Interface.IServeurOperation` et nommé `Service`.

Ensuite on l'active (création du proxy transparent) avec le type en paramètre ainsi que l'adresse du serveur (ici localhost car le serveur et le client sont situés sur la même machine).

3.5 La durée de vie

Pour la gestion de la durée de vie je rappelle que le .NET Remoting utilise un gestionnaire de crédit-bail (utilisé principalement pour les objets activés par le client), il peut cependant être ajouté pour un objet activé par le serveur.

La gestion des baux se fait dans la class `MarshalByRefObjet` (`ServeurOperation`) dans un nouvel objet `InitializeLifetimeService`.

```
//C#
        public override object InitializeLifetimeService()
        {
            return base.InitializeLifetimeService();
        }
```

```
'VB.NET
Public Overloads Overrides Function InitializeLifetimeService() As Object
        Return (MyBase.InitializeLifetimeService())
End Function
```

Ici, l'objet retourne le bail initial ; aucun changement n'y est apporté.

Pour modifier le bail on utilise l'interface `ILease`, pour cela il faudra commencer par rajouter les using suivants :

```
//C#
using System.Runtime.Remoting;
using System.Runtime.Remoting.Lifetime;
```

```
'VB.NET
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Lifetime
```

Ensuite il faudra créer un `ILease` et l'initialiser :

```
//C#
public override object InitializeLifetimeService()
        {
            ILease ddv = (ILease)base.InitializeLifetimeService();

            return ddv;
        }
```

```
'VB.NET
Public Overloads Overrides Function InitializeLifetimeService() As Object
        Dim ddv As ILease =
        DirectCast(MyBase.InitializeLifetimeService(), ILease)

        Return (ddv)
End Function
```

Remarque : ici nous avons créé le `ILease` ddv, qui prend comme valeur la durée de vie par défaut. Enfin on retourne ddv dans la méthode.

Pour personnaliser le `ILease` ddv on utilise les propriétés suivantes :

Nom	Description
ILease.CurrentLeaseTime	Obtient la période résiduelle du bail.
ILease.CurrentState	Obtient le `LeaseState` (état possible du bail) actuel du bail.
ILease.InitialLeaseTime	Obtient ou définit la durée initiale du bail.
ILease.RenewOnCallTime	Obtient ou définit la durée au terme de laquelle un appel de l'objet distant renouvelle `CurrentLeaseTime`.
ILease.SponsorshipTimeout	Obtient ou définit la période d'attente du retour d'un sponsor avec une durée de renouvellement.

```
//C#
public override object InitializeLifetimeService()
        {
            ILease ddv = (ILease)base.InitializeLifetimeService();
            //Définition de la durée initial du bail.
            ddv.InitialLeaseTime = TimeSpan.FromSeconds(5);
            //Définition de la période d'attente du retour d'un sponsor
            avec un durée de renouvellement
            ddv.SponsorshipTimeout = TimeSpan.FromSeconds(10);
            //Définition de la durée de la durée durant laquel un service
            renouvelle CurentLeaseTime
            ddv.RenewOnCallTime = TimeSpan.FromSeconds(3);

            return ddv;
        }
```

```
'VB.NET
    Public Overloads Overrides Function InitializeLifetimeService() As
Object
        Dim ddv As ILease =
DirectCast(MyBase.InitializeLifetimeService(), ILease)
        'Définition de la durée initial du bail.
        ddv.InitialLeaseTime = TimeSpan.FromSeconds(5)
        'Définition de la période d'attente du retour d'un sponsor avec
un durée de renouvellement
        ddv.SponsorshipTimeout = TimeSpan.FromSeconds(10)
        'Définition de la durée de la durée durant laquel un service
renouvelle CurentLeaseTime
        ddv.RenewOnCallTime = TimeSpan.FromSeconds(3)

        Return ddv
    End Function
```

Pour mon exemple j'ai décidé d'avoir un objet avec durée de vie par défaut, je retourne la valeur de base.

```
//C#
        public override object InitializeLifetimeService()
        {
            ILease ddv = (ILease)base.InitializeLifetimeService();
            return ddv;
        }
```

```
'VB.NET
    Public Overloads Overrides Function InitializeLifetimeService() As
Object
        Dim ddv As ILease =
DirectCast(MyBase.InitializeLifetimeService(), ILease)
        Return (ddv)
    End Function
```

3.6 L'appel du service

Après avoir fait toutes les étapes précédentes, on va pouvoir enfin créer notre code client et appeler la méthode addition à partir de notre code client.

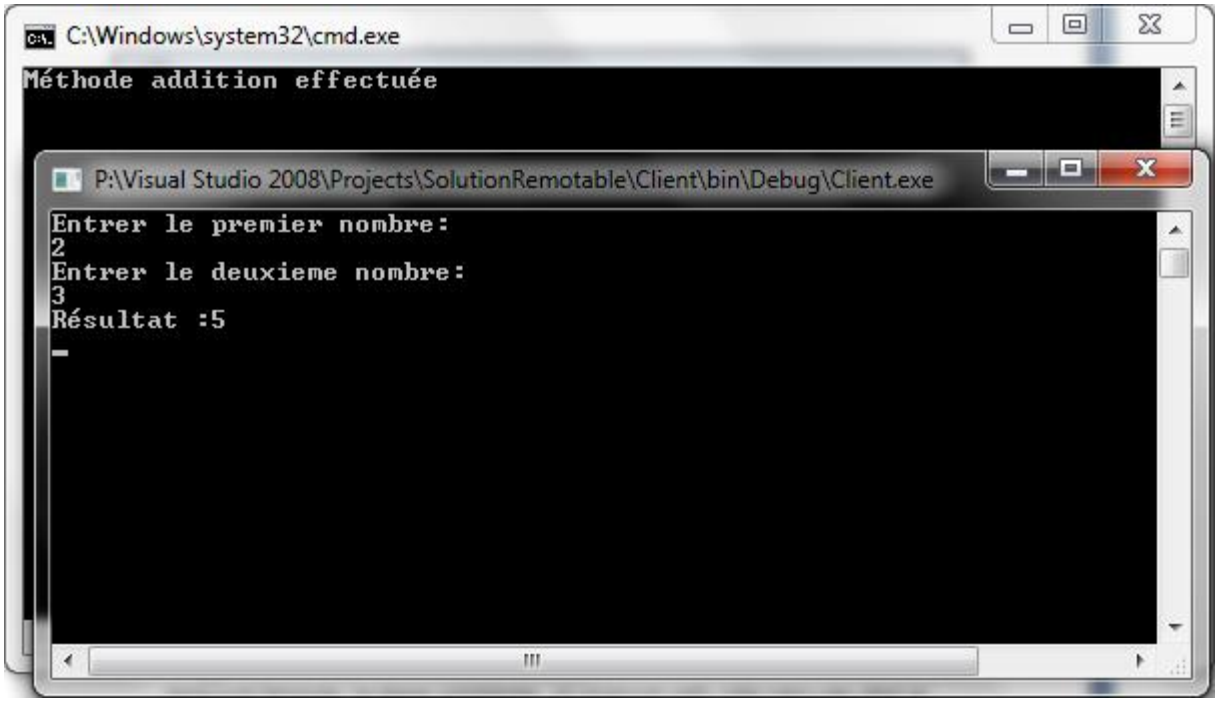
Il suffit simplement d'appeler l'interface créée précédemment (« service » dans la partie 3.4.2) puis de préciser la méthode addition en ajoutant les paramètres requis.

```
//C#
            Console.WriteLine("Entrer le premier nombre:");
            nombre1 = Int32.Parse(Console.ReadLine());
            Console.WriteLine("Entrer le deuxieme nombre:");
            nombre2 = Int32.Parse(Console.ReadLine());
            somme = Service.addition(nombre1, nombre2);
            Console.WriteLine(string.Format("Résultat :{0}",somme));
            Console.ReadLine();
```

```
'VB.NET
        Console.WriteLine("Entrer le premier nombre:")
        nombre1 = Int32.Parse(Console.ReadLine())
        Console.WriteLine("Entrer le deuxieme nombre:")
        nombre2 = Int32.Parse(Console.ReadLine())
        somme = Service.addition(nombre1, nombre2)
        Console.WriteLine(String.Format("Résultat :{0}", somme))
        Console.ReadLine()
```

Remarque : on effectue les deux premiers `ReadLine` pour récupérer les valeurs pour effectuer l'addition. Le dernier sert seulement à garder la console ouverte.

Une fois le programme compilé, il suffit de lancer le serveur puis le client pour effectuer le remoting. Il est tout à fait possible de lancer plusieurs clients avec un même serveur.



On remarque avec ce simple exemple la communication entre deux processus différents.

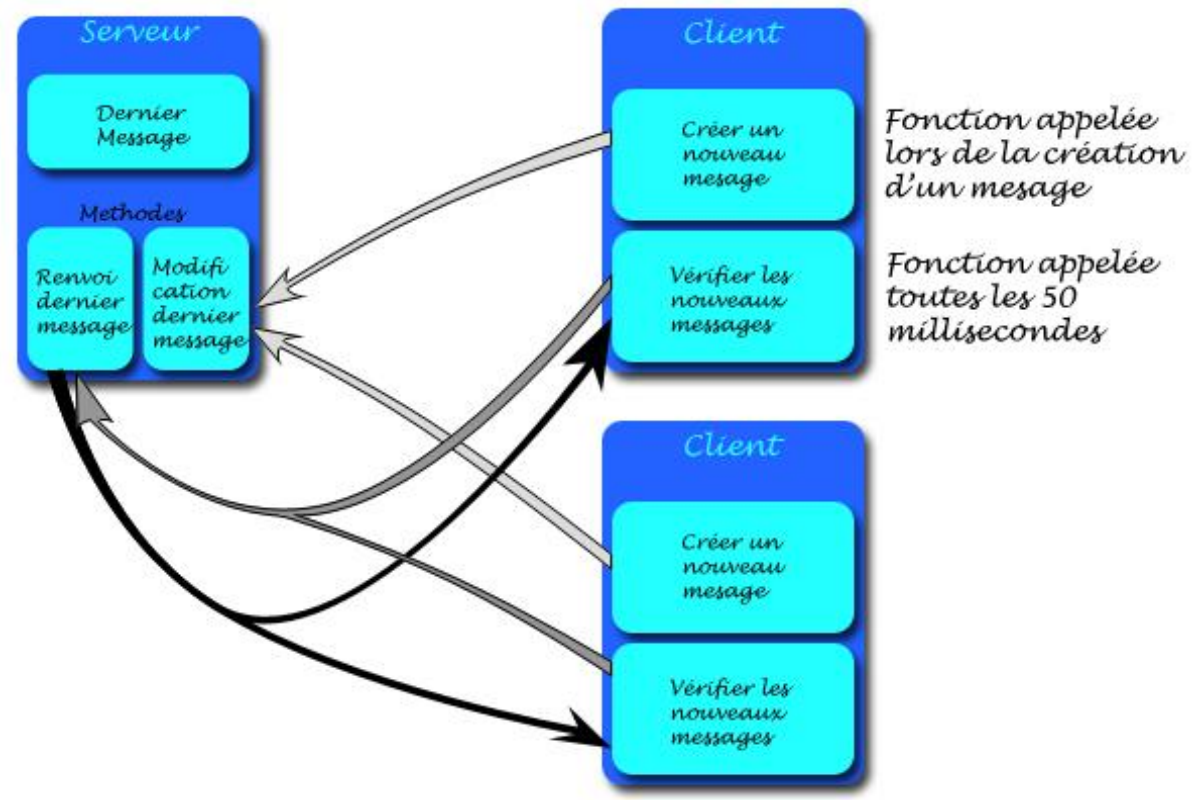
4 Exemple

Dans cette partie du cours je vais mettre un exemple complet d'un programme utilisant le .NET Remoting. Le programme se découpe en 3 parties :

- ServeurChat
  - RemoteMain.cs
  - RemoteMethode.cs
- ClientChat
  - Form1.cs
  - Form2.cs
  - Form3.cs
- ServeurInterface
  - IRemoteMethode.cs

Il utilise le Marshaling By Reference et l'activation coté serveur.

Le système utilisé pour faire la messagerie est simple. Le serveur comporte deux méthodes, une pour renvoyer le dernier message et une pour ajouter un message en tant que dernier message. Le client quant à lui appelle régulièrement la première pour vérifier les nouveaux messages et n'utilise la seconde que lorsque l'utilisateur crée un message.



Nous rappelons que pour compiler ce programme nous avons rajouté les références citées précédemment dans le cours.

4.1 Le serveur

Notre serveur est donc découpé en deux fichier.cs : l'un permet la connexion et l'autre regroupe les différentes opérations nécessaires au fonctionnement du programme de chat.

RemoteMain.cs

```
//C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

namespace ServeurChat
{
    class RemoteMain
    {
        [STAThread]
        static void Main(string[] args)
        {
            try // test de création d'un canal et d'activation des objets
            {
                TcpChannel channel = new TcpChannel(8085); // création
                 d'un canal sur le port 8085
                ChannelServices.RegisterChannel(channel);
                RemotingConfiguration.RegisterWellKnownServiceType(
                    typeof(RemoteMethode),
                    "RemoteMethode",
                    WellKnownObjectMode.Singleton);// activation coté
                    serveur en mode Singleton

                Console.WriteLine("Le serveur a démarré avec succés");
                Console.ReadLine();
            }
            catch
            {
                Console.WriteLine("Erreur serveur");
                Console.ReadLine();
            }
        }
    }
}
```

```vb
'VB.NET
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels
Imports System.Runtime.Remoting.Channels.Tcp

Namespace ServeurChat
    Class RemoteMain
        <STAThread()> _
        Private Shared Sub Main(ByVal args As String())
            Try
                ' test de création d'un canal et d'activation des objets
                Dim channel As New TcpChannel(8085)
                ' création d'un canal sur le port 8085
                ChannelServices.RegisterChannel(channel)

RemotingConfiguration.RegisterWellKnownServiceType(GetType(RemoteMethode)
, "RemoteMethode", WellKnownObjectMode.Singleton)
                ' activation coté serveur en mode Singleton
                Console.WriteLine("Le serveur a démarré avec succés")
                Console.ReadLine()
            Catch
                Console.WriteLine("Erreur serveur")
                Console.ReadLine()
            End Try
        End Sub
    End Class
End Namespace
```

RemoteMethode.cs

```
//C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ServeurChat
{
    public class RemoteMethode : MarshalByRefObject,
Serveurinterfaces.IRemoteMethode  // MBR
    {
        public override object InitializeLifetimeService()
        {
            return null; // durée de vie infini pour les objets
        }

        private string cachemsg; //variable dernier message
        private string cachecolor; //variable couleur du message


        public void ajout(string entermsg, string entercolor) //méthode
        pour modifier le dernier message
        {
            Console.WriteLine(String.Format("ajoutmsg :{0}",entermsg));
            //affichage du message coté serveur
            cachemsg = entermsg; //dernier message mis en mémoire
            cachecolor = entercolor; //couleur mis en mémoire

        }

        public string returnmsg() // retourne le dernier message
        {
            return cachemsg;
        }

        public string returncolor() // retourne la derniere couleur
        utilisée

        {
            return cachecolor;
        }

    }
}
```

```vb
'VB.NET
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text

Namespace ServeurChat
    Public Class RemoteMethode
        Inherits MarshalByRefObject
        Implements Serveurinterfaces.IRemoteMethode
        ' MBR
        Public Overloads Overrides Function InitializeLifetimeService()
        As Object
            ' durée de vie infini pour les objets
            Return Nothing
        End Function

        Private cachemsg As String
        'variable dernier message
        Private cachecolor As String
        'variable couleur du message

        Public Sub ajout(ByVal entermsg As String, ByVal entercolor As
        String)
            'méthode pour modifier le dernier message
            Console.WriteLine([String].Format("ajoutmsg :{0}", entermsg))
            'affichage du message coté serveur
            cachemsg = entermsg
            'dernier message mis en mémoire
            'couleur mis en mémoire
            cachecolor = entercolor
        End Sub

        Public Function returnmsg() As String
            ' retourne le dernier message
            Return cachemsg
        End Function

        Public Function returncolor() As String
            ' retourne la derniere couleur utilisée
            Return cachecolor
        End Function

    End Class
End Namespace
```

4.2 L'interface

Pas de changement par rapport aux autres interfaces vues précédemment ; on retrouve les prototypes des différentes méthodes distribuées.

IRemoteMethode.cs

```
//C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Serveurinterfaces
{
    public interface IRemoteMethode //interface entre le client et le
     serveur
    {
        void ajout(string entermsg, string entercolor);
        string returnmsg();
        string returncolor();
    }
}
```

```
'VB.NET
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text

Namespace Serveurinterfaces
    Public Interface IRemoteMethode
        'interface entre le client et le serveur
        Sub ajout(ByVal entermsg As String, ByVal entercolor As String)
        Function returnmsg() As String
        Function returncolor() As String
    End Interface
End Namespace
```

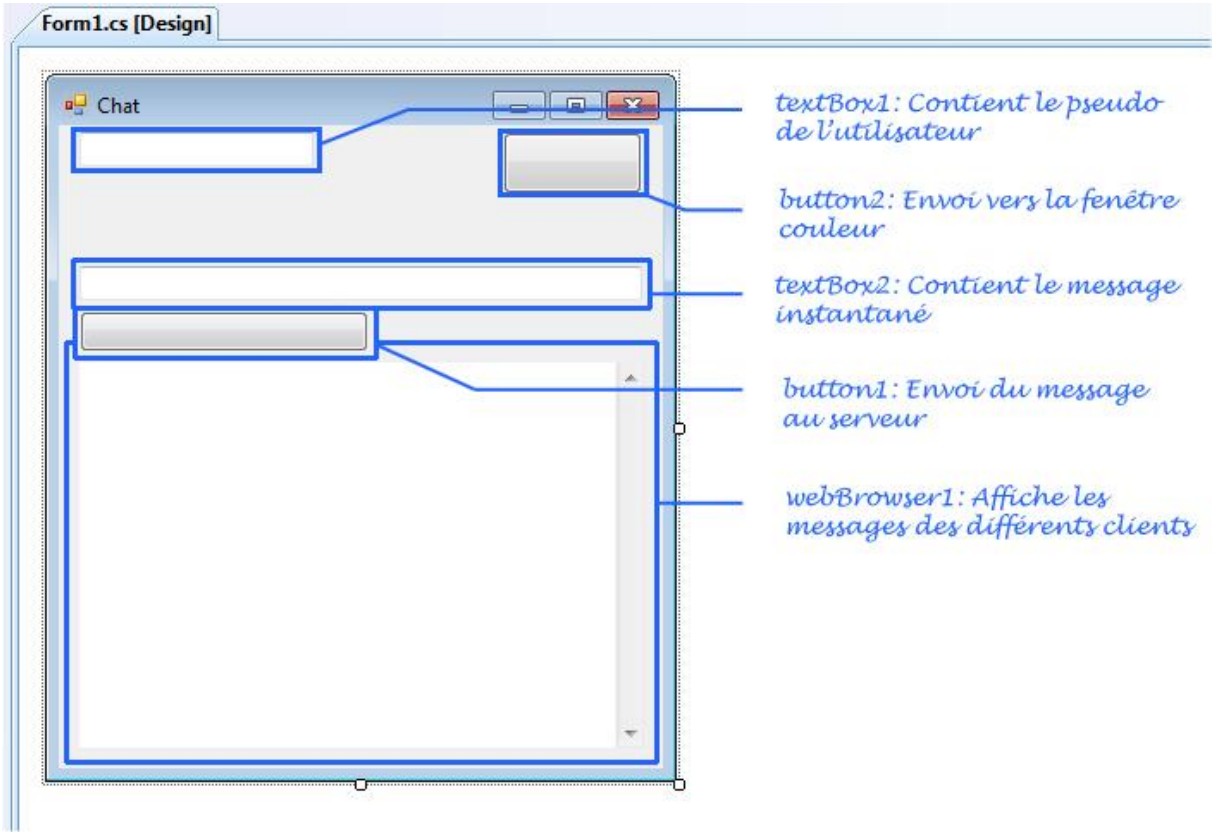
4.3 Le client

Notre client quant à lui est beaucoup plus complexe, il se sépare en 3 fenêtres :

- Fenêtre principal avec message instantané (Form1)
- Fenêtre de connexion avec demande d'ip (Form3)
- Fenêtre des couleurs pour ses propres messages (Form2)

4.3.1 Fenêtre Principale

Tout notre client est réalisé avec la technologie WinFom. Je vais vous montrer les différents contrôles ainsi que le code associé.




```
//C#
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using System.Windows.Threading;
using System.Runtime.InteropServices;
using System.Threading;


namespace ClientChat1
{
    public partial class Form1 : Form
    {
```

```
//C#
        private Serveurinterfaces.IRemoteMethode remoteMethode;
        //déclaration de l'interfaces

        private DispatcherTimer timer1; // déclaration des différentes
        variables
        private DispatcherTimer timer2;
        private string cachelastmsg;
        private string colorserveur;
        private string addresschannel;
        private string colortext;
        private bool feFocus;

        [DllImport("user32.dll")] //importation pour le clignotement de
        la fenêtre
        static extern bool FlashWindow(IntPtr hwnd, bool bInvert);

        public Form1()
        {
            InitializeComponent();
            button1.Text = "Envoyer";
            textBox2.Text = "";
            textBox1.Text = "Pseudo";
            colorserveur = "Black";
            button2.Text = "Couleur";
            cachelastmsg = null; // initialisation des différents
            contrôles

            Form3 Ipform = new Form3();
            Ipform.ShowDialog(this); // appel de la fenêtre pour entrer
            l'ip
            if (Ipform.address != null)
            {
                addresschannel = "tcp://" + Ipform.address +
                ":8085/RemoteMethode"; // création de l'url de connexion
                connection(addresschannel); // appel de la méthode de
                connexion
            }


            timer1 = new DispatcherTimer();
            timer1.Interval = TimeSpan.FromMilliseconds(50);
            timer1.Tick += new EventHandler(refresh);
            timer1.Start(); // appel de la fonction refresh toutes les 50
            millisecondes

            timer2 = new DispatcherTimer();
            timer2.Interval = TimeSpan.FromSeconds(1);
            timer2.Tick += new EventHandler(clignotement); //clignotement
            de la fenêtre toutes les secondes

            this.LostFocus += new EventHandler(LostFe); //si la fenêtre
            perd le focus
        }
```

```
//C#
private void button1_Click(object sender, EventArgs e)
        {
            try //tentavive d'ajout d'un message
            {
                if (remoteMethode != null)
                {
                    if (textBox2.Text.ToString() != "") //si le message
                    est différent de nul
                    {
                    string newmsg = textBox1.Text.ToString() + " : "; //
                    le nouveau message commence par le pseudo
                    newmsg += textBox2.Text.ToString(); //on y ajoute
                    ensuite le message

                        remoteMethode.ajout(newmsg,colorserveur);
                    }
                }
            }
            catch { MessageBox.Show("Erreur !"); }
            textBox2.Text = null; // on efface le champ message
        }

        void refresh(object sender, EventArgs e) //fonction de
        vérification des nouveaux messages
        {
            try
            {
                if (remoteMethode.returnmsg() != null &&
                remoteMethode.returnmsg() != cachelastmsg)
                    //si le message est différent de nul ou du message en
                cache (pour éviter d'afficher le même message toutes les
                50 millisecondes)
                {
                    cachelastmsg = remoteMethode.returnmsg();//
                    récupération du dernier message
                    colortext = remoteMethode.returncolor();//
                    récupération de la dernière couleur

                    string newChaine = "";

                    foreach(var myChar in cachelastmsg.ToCharArray()) //
                    modification évitant les scripts dans le webBrowser
                    d'affichage
                    {
                        if (myChar.Equals('<'))
                        {
                            newChaine += "&lt;";
                        }
                        else if (myChar.Equals('>'))
                        {
                            newChaine += "&gt;";
                        }
                        else
                        {
                            newChaine += myChar.ToString();
                        }
                    }

                    webBrowser1.DocumentText = "<div style=\"color: " +
                    colortext + "\">" + newChaine + "<br /></div>" +
                    webBrowser1.DocumentText;
                    // affichage du message dans le WebBrowser
```

```
//C#
if (feFocus == false) // si le focus n'est pas sur la fenêtre
                    {
                        timer2.Start(); // on commence le clignotement
                    }
                }
            }
            catch { MessageBox.Show("Erreur !!!!!!!"); this.Close(); }

        }


        private void textBox2_KeyPress(object sender, KeyPressEventArgs
        e)
        {
            if (e.KeyChar.Equals('\r')) // si l'utilisateur appuie sur le
            bouton entrée
            {
                button1_Click(null, new EventArgs()); //appel de la
                fonction Button1_Click
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Form2 ColorForm = new Form2();
            ColorForm.ShowDialog(this); // Appel de la fonction
            changement de couleur d'écriture

            colorserveur = ColorForm.color; // Enregistrement de la
            couleur dans les variables
        }

        private void connection(string address)
        {
            try // tentative de connexion avec l'ip fourni par
            l'utilisateur
            {
                TcpChannel channel = new TcpChannel();
                ChannelServices.RegisterChannel(channel);
                remoteMethode =
                 (Serveurinterfaces.IRemoteMethode)Activator.GetObject(
                   typeof(Serveurinterfaces.IRemoteMethode),
                   addresschannel);
            }
            catch { MessageBox.Show("Erreur connexion"); }
        }

        private void clignotement(object sender, EventArgs e)
        {
            FlashWindow(this.Handle, true); // clignotement de la fenêtre

        }

        private void Form1_Activate(object sender, EventArgs e)
        {
            timer2.Stop(); // arrêt du clignotement
            feFocus = true; // gain sur la fenêtre
        }
```

```
//C#
        private void LostFe(object sender, EventArgs e)
        {
            feFocus = false; // perte du focus
        }
    }
}
```

```
'VB.NET
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Linq
Imports System.Text
Imports System.Windows.Forms
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels
Imports System.Runtime.Remoting.Channels.Tcp
Imports System.Windows.Threading
Imports System.Runtime.InteropServices
Imports System.Threading


Namespace ClientChat1
    Partial Public Class Form1
        Inherits Form
        Private remoteMethode As Serveurinterfaces.IRemoteMethode
        'déclaration de l'interfaces
        Private timer1 As DispatcherTimer
        ' déclaration des différentes variables
        Private timer2 As DispatcherTimer
        Private cachelastmsg As String
        Private colorserveur As String
        Private addresschannel As String
        Private colortext As String
        Private feFocus As Boolean

        'importation pour le clignotement de la fenetre
        <DllImport("user32.dll")> _
        Private Shared Function FlashWindow(ByVal hwnd As IntPtr, ByVal
        bInvert As Boolean) As Boolean
        End Function

        Public Sub New()
            InitializeComponent()
            button1.Text = "Envoyer"
            textBox2.Text = ""
            textBox1.Text = "Pseudo"
            colorserveur = "Black"
            button2.Text = "Couleur"
            cachelastmsg = Nothing
            ' initialisation des différents controle
            Dim Ipform As New Form3()
            Ipform.ShowDialog(Me)
            ' appel de la fenêtre pour entrer l'ip
            If Ipform.address IsNot Nothing Then
                addresschannel = "tcp://" & Ipform.address &
                ":8085/RemoteMethode"
                ' création de l'url de connection
```

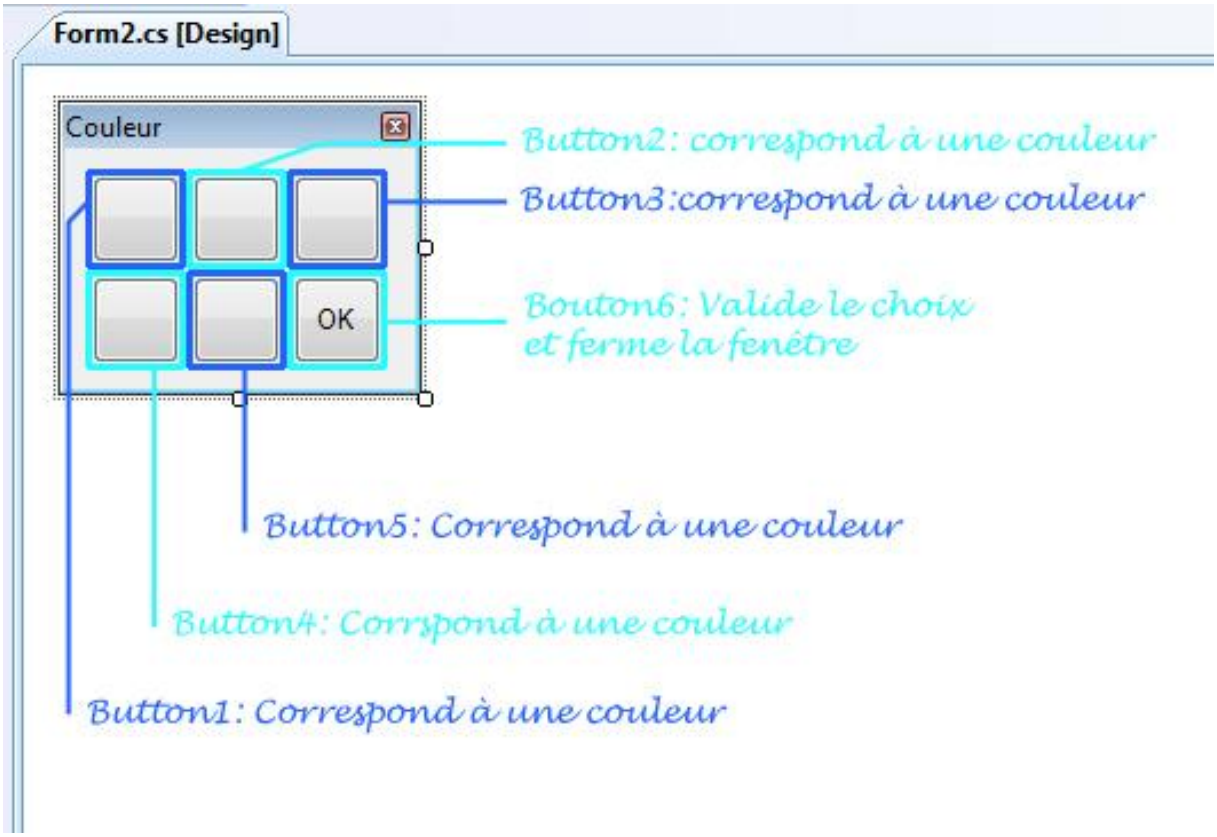
```
'VB.NET
' appel de la méthode de connection
                connection(addresschannel)
            End If


            timer1 = New DispatcherTimer()
            timer1.Interval = TimeSpan.FromMilliseconds(50)
            AddHandler timer1.Tick, AddressOf refresh
            timer1.Start()
' appel de la fonction refresh toutes les 50 milisecondes
            timer2 = New DispatcherTimer()
            timer2.Interval = TimeSpan.FromSeconds(1)
            AddHandler timer2.Tick, AddressOf clignotement
'clignotement de la fenètre toute les secondes
'si la fenèttre perd le focus
            AddHandler Me.LostFocus, AddressOf LostFe
        End Sub

Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
    Try
        'tentavive d'jout d'un message
        If remoteMethode IsNot Nothing Then
            If textBox2.Text.ToString() <> "" Then
                'si le message est différent de null
                Dim newmsg As String = textBox1.Text.ToString() & " : "
                ' le nouveau message commence par le pseudo
                newmsg += textBox2.Text.ToString()
                'on y ajoute ensuite le message
                remoteMethode.ajout(newmsg, colorserveur)
            End If
        End If
    Catch
        MessageBox.Show("Erreur !")
    End Try
    ' on efface le champs message
    textBox2.Text = Nothing
End Sub

Private Sub refresh(ByVal sender As Object, ByVal e As EventArgs)
    'fonction de verification des nouveau message
    Try
        If remoteMethode.returnmsg() IsNot Nothing AndAlso
remoteMethode.returnmsg() <> cachelastmsg Then
            'si le message est différent de null ou du message en cache
            (pour eviter d'afficher le même message toutes les
            50milisecondes
            cachelastmsg = remoteMethode.returnmsg()
            ' récupération du dernier message
            colortext = remoteMethode.returncolor()
            ' récupération de la derniere couleur
            Dim newChaine As String = ""

            For Each myChar In cachelastmsg.ToCharArray()
                ' modification évitant les script dans le webbrowser
                 d'affichage
                If myChar.Equals("<"c) Then
                    newChaine += "&lt;"
                ElseIf myChar.Equals(">"c) Then
                    newChaine += "&gt;"
                Else
                    newChaine += myChar.ToString()
                End If
            Next
```

```
'VB.NET
            webBrowser1.DocumentText = (("<div style=""color: " &
            colortext & """>") + newChaine & "<br /></div>") +
            webBrowser1.DocumentText
            ' affichage du message dans le WebBrowser
            If feFocus = False Then
                ' si le focus n'est pas sur la fenètre
                ' on commence le clignotement
                timer2.Start()
            End If
        End If
    Catch
        MessageBox.Show("Erreur !!!!!!!")
        Me.Close()

    End Try
End Sub


Private Sub textBox2_KeyPress(ByVal sender As Object, ByVal e As
KeyPressEventArgs)
    If e.KeyChar.Equals(ControlChars.Cr) Then
        ' si l'utilisateur appuie sur le bouton entrée
        'appel de la fonction Button1_Click
        button1_Click(Nothing, New EventArgs())
    End If
End Sub

Private Sub button2_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim ColorForm As New Form2()
    ColorForm.ShowDialog(Me)
    ' Appel de la fonction changement de couleur d'écriture
    ' Enregistrement de la couleur dans les variable
    colorserveur = ColorForm.color
End Sub

Private Sub connection(ByVal address As String)
    Try
        ' tentative de connection avec l'ip fourni par l'utilisateur
        Dim channel As New TcpChannel()
        ChannelServices.RegisterChannel(channel)
        remoteMethode =
        DirectCast(Activator.GetObject(GetType(Serveurinterfaces.IRemoteM
        ethode), addresschannel), Serveurinterfaces.IRemoteMethode)
    Catch
        MessageBox.Show("Erreur connexion")
    End Try
End Sub

Private Sub clignotement(ByVal sender As Object, ByVal e As EventArgs)
    ' clignotement de la fenètre
    FlashWindow(Me.Handle, True)
End Sub

Private Sub Form1_Activate(ByVal sender As Object, ByVal e As EventArgs)
    timer2.[Stop]()
    ' arret du clignotement
    ' gain sur la fenètre
    feFocus = True
End Sub
```

```
'VB.NET
Private Sub LostFe(ByVal sender As Object, ByVal e As EventArgs)
    ' perte du focus
    feFocus = False
End Sub
    End Class
End Namespace
```

4.3.2 Fenêtre couleur

Cette fenêtre permet simplement de changer la couleur d'écriture : elle n'est constituée que de boutons.




```
//C#
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ClientChat1
{
    public partial class Form2 : Form
    {
        private string TextColor;

        public Form2()
        {
```

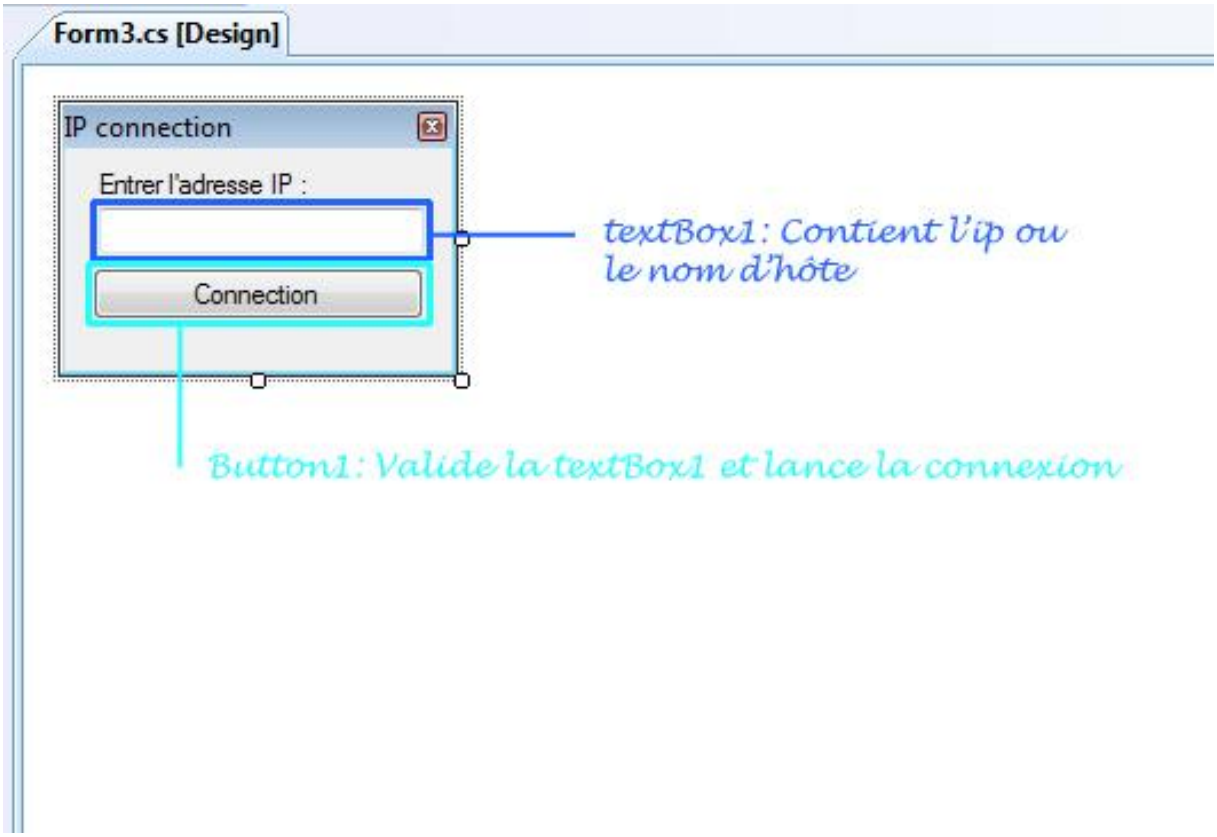
```
//C#
            InitializeComponent();
            button1.Text = ""; // initialisation des boutons (texte et
couleur)
            button1.BackColor = Color.Black;
            button2.Text = "";
            button2.BackColor = Color.Blue;
            button3.Text = "";
            button3.BackColor = Color.Red;
            button4.Text = "";
            button4.BackColor = Color.Gray;
            button5.Text = "";
            button5.BackColor = Color.Green;
            button6.Text = "OK";
        }

        public string color { get { return TextColor; } } // retourne la
        couleur demandée

        private void button1_Click(object sender, EventArgs e)
        {
            TextColor = "Black";
        }

        private void button2_Click(object sender, EventArgs e)
        {
            TextColor = "Blue";
        }

        private void button3_Click(object sender, EventArgs e)
        {
            TextColor = "Red";
        }

        private void button4_Click(object sender, EventArgs e)
        {
            TextColor = "Gray";
        }

        private void button5_Click(object sender, EventArgs e)
        {
            TextColor = "Green";
        }

        private void Form2_Click(object sender, EventArgs e)
        {
            TextColor = "Black";
        }

        private void button6_Click(object sender, EventArgs e)
        {
            this.Close(); // ferme la fenètre
        }



    }
}
```

```
'VB.NET
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Linq
Imports System.Text
Imports System.Windows.Forms

Namespace ClientChat1
    Partial Public Class Form2
        Inherits Form
        Private TextColor As String

        Public Sub New()
            InitializeComponent()
            button1.Text = ""
            ' initialisation des boutons (texte et couleur)
            button1.BackColor = Color.Black
            button2.Text = ""
            button2.BackColor = Color.Blue
            button3.Text = ""
            button3.BackColor = Color.Red
            button4.Text = ""
            button4.BackColor = Color.Gray
            button5.Text = ""
            button5.BackColor = Color.Green
            button6.Text = "OK"
        End Sub

        Public ReadOnly Property color() As String
            Get
                Return TextColor
            End Get
        End Property
        ' return la couleur demandé
        Private Sub button1_Click(ByVal sender As Object, ByVal e As
        EventArgs)
            TextColor = "Black"
        End Sub

        Private Sub button2_Click(ByVal sender As Object, ByVal e As
        EventArgs)
            TextColor = "Blue"
        End Sub

        Private Sub button3_Click(ByVal sender As Object, ByVal e As
        EventArgs)
            TextColor = "Red"
        End Sub

        Private Sub button4_Click(ByVal sender As Object, ByVal e As
        EventArgs)
            TextColor = "Gray"
        End Sub

        Private Sub button5_Click(ByVal sender As Object, ByVal e As
        EventArgs)
            TextColor = "Green"
        End Sub
```

```
'VB.NET
Private Sub Form2_Click(ByVal sender As Object, ByVal e As EventArgs)
    TextColor = "Black"
End Sub

Private Sub button6_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' ferme la fenètre
    Me.Close()
End Sub



    End Class
End Namespace
```

4.3.3 Fenêtre de connexion

Première fenêtre ouverte à l'ouverture du client, elle demande simplement une ip (ou le nom d'hôte.



```
//C#
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ClientChat1
{
    public partial class Form3 : Form
    {
        private string ip;
        public Form3()
        {
            InitializeComponent();
            textBox1.Text = "localhost"; // ip par défaut et localhost
        }

        public string address { get { return ip; } }

        private void button1_Click(object sender, EventArgs e)
        {
            ip = textBox1.Text; // récupération de l'ip
            this.Close();
        }

        private void textBox1_KeyPress(object sender, KeyPressEventArgs
        e)
        {
            if (e.KeyChar.Equals('\r')) // possibilité d'appuyer sur
            entrée pour confirmer
            {
                button1_Click(null, new EventArgs());
            }
        }

    }
}
```

```vb
'VB.NET
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Linq
Imports System.Text
Imports System.Windows.Forms

Namespace ClientChat1
    Partial Public Class Form3
        Inherits Form
        Private ip As String
        Public Sub New()
            InitializeComponent()
            ' ip par défaut et localhost
            textBox1.Text = "localhost"
        End Sub

        Public ReadOnly Property address() As String
            Get
                Return ip
            End Get
        End Property

        Private Sub button1_Click(ByVal sender As Object, ByVal e As
        EventArgs)
            ip = textBox1.Text
            ' récupération de l'ip
            Me.Close()
        End Sub

        Private Sub textBox1_KeyPress(ByVal sender As Object, ByVal e As
        KeyPressEventArgs)
            If e.KeyChar.Equals(ControlChars.Cr) Then
                ' possibilité d'appuyer sur entrée pour confirmer
                button1_Click(Nothing, New EventArgs())
            End If
        End Sub

    End Class
End Namespace
```

5 Conclusion

Dans ce chapitre, nous avons mis en pratique les différentes notions du Chapitre 1. On a abordé l'implantation des différents modèles de distribution d'objets (MBR, MBV, CAO, WKO,…). Enfin un long exemple permet de mettre en pratique le .NET Remoting ; en effet le programme de chat est un exemple parfait d'application distribuée Clients-Serveur.