

Berthet Laurent
Julien Bugnard

Compte rendu du TP de DotNet Chat avec .Net Remoting

Décembre 2007

Sommaire

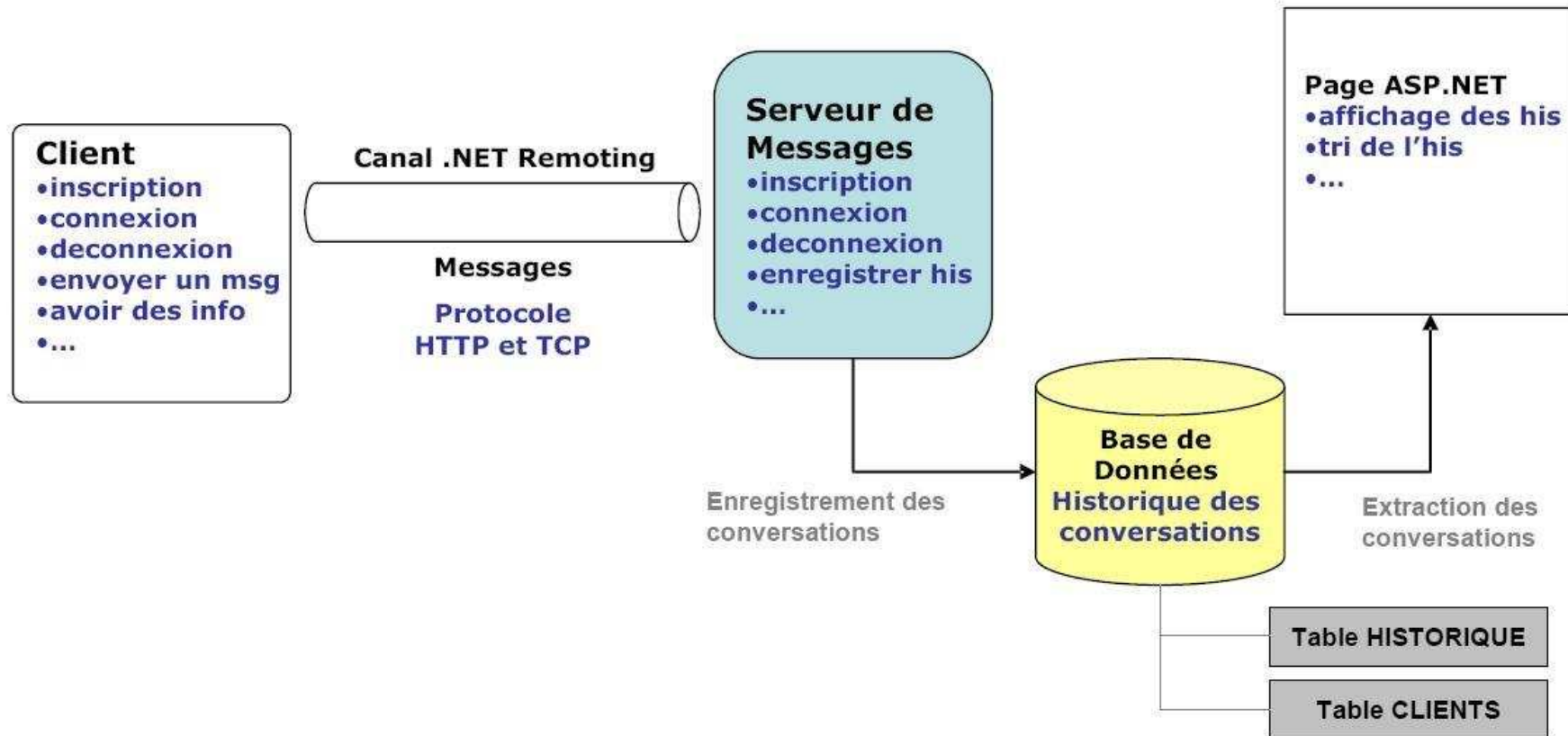
Sommaire.....	2
Introduction	3
Architecture de l'application.....	4
Fonctionnement du système	5
• L'interface	5
• Le serveur.....	6
• Le client	6
Utilisation de la Base de données	8
Utilisation de l'application.....	9
• Inscription d'un client dans la base de données.....	9
• Connection d'un client sur le serveur	9
• Le serveur.....	10
• L'interface web	10
• Chat avec un autre client	11
Conclusion.....	13

Introduction

Dans ce TP il est demandé de réaliser un chat permettant à des clients distants de s'échanger des messages grâce à un serveur de messages. Chaque client doit se connecter d'abord au serveur, pour cela il doit dans un premier temps s'inscrire avec un login et un mot de passe. Pour réaliser ce TP nous devions utiliser dotNet Remoting.

Architecture de l'application

Voici l'architecture de l'application qui nous était demandée :



Fonctionnement du système

- *L'interface*

Pour que le client ait connaissance des méthodes distantes (qu'il va pouvoir appeler), une interface contenant leur déclaration doit être distribuée côté client et côté serveur. L'objet exposé sur le serveur devra implémenter cette interface.

Méthodes :

```
bool inscription(string login, string pass)
int nombreConnecterserveur()
string dateServeur()
void ecrire(string message, string dest)
bool seConnecter(string login, string pass)
bool deconnecter(string login)
ArrayList getClients()
string getChatSession()
```

- `bool inscription(string login, string pass)` : cette méthode est appelée quand un client veut s'inscrire pour pouvoir par la suite communiquer avec les autres clients. Le booléen qui est retourné indique si l'opération s'est effectuée correctement. Pour pouvoir s'inscrire, le client aura besoin d'un login et d'un mot de passe.
- `bool seConnecter(string login, string pass)` : Cette méthode est appelée une fois que le client est inscrit, il faut qu'il se connecte au serveur pour envoyer des messages aux clients déjà connectés. Le booléen indique si l'opération s'est bien passée.
- `bool deconnecter(string login)` : La méthode est utilisée lorsque le client veut quitter le chat, pour ce faire nous avons besoin que du login du client car il est impossible que deux clients aient le même login.
- `void ecrire(string message, string dest)` : Cette méthode est utile pour le client qui veut écrire un message à un autre client qui est connecté sur le chat.

- ***Le serveur***

Le serveur implémente donc les méthodes définies dans l'interface. C'est le serveur qui enregistre les conversations des clients dans la base de données.

Le serveur a deux attributs :

- un ArrayList contenant le login des clients connectés : Dès qu'un client se connecte, son login est ajouté dans la liste. Cette liste est retransmise aux clients.
- une chaîne de caractères contenant une conversation : cette chaîne va servir à transmettre aux clients un message aux autres clients.

Implémentations des méthodes de l'interface :

Le serveur possède aussi l'accès à la base de données, c'est à dire qu'il a les méthodes pour ajouter des éléments dans la base.

Pour que le serveur soit accessible par tous les clients, le serveur est déclaré comme un "singleton" avec "Remoting". Nous avons préféré utiliser le singleton plutôt que le singleCall car avec notre implémentation, le singleCall ne marchait pas.

- ***Le client***

Le client a deux attributs :

- Un login
- Un password

Toutes les actions de serveur (envoi d'un message, etc) sont faites directement dans les actions des boutons des fenêtres.

Pour qu'un client puisse voir les messages des autres clients ainsi que la liste des clients connectés. Nous avons donc créé un thread qui est lancé dès un client se connecte et qui est stoppé quand le client se déconnecte.

Code du thread :

```
public void threadServer()
{
    // boucle infinie
    while (true)
    {
        try
        {
            //thread inactif pendant 1000
            Thread.Sleep(1000);

            //On déclare un ArrayList local clients qui est une copie de celle sur le serveur
            ArrayList clients = remoteOperation.getClients();

            // on efface la liste
            listeClients.Items.Clear();

            //On affiche tous les clients présents dans listClients (liste déroulante)
            foreach (string clName in clients)
            {
                if(!clName.Equals(client.getLogin())){
                    listeClients.Items.Add(clName);
                }
            }

            //On déclare un string local qui est une copie du chatSession du serveur
            string sessionText = remoteOperation.getChatSession();

            //On met à jour la zone d'affichage du chat
            affMessage.Clear();
            affMessage.Text = sessionText;
        }
        catch (Exception ex)
        {
            MessageBox.Show("Probleme de connection avec le serveur : " + ex.Message);
        }
    }
}
```

Ce thread ne sert qu'à afficher les messages provenant des autres clients. Par souci de simplicité nous n'avons pas géré quand il n'y a pas de changements, c'est à dire si aucun client n'envoie de message, le thread réactualise quand même la conversation. Pour la liste des clients, on vérifie quand même de ne pas afficher son nom dans la liste car un client ne peut pas se parler à lui-même.

- *Utilisation de la Base de données*

Pour enregistrer les conversations et les clients, le serveur a besoin d'avoir accès à une base de données. Nous avons créés une classe « Bd » qui contient toutes les méthodes pour gérer la base de données. Cette classe peut ouvrir une base, faire des opérations (Select, Insert, Delete, ...) et fermer la base après utilisation.

On ne peut pas ouvrir n'importe quel type de base de données. Dans notre cas, on utilise des bases de données Access. Chaque base de données a son mode d'ouverture qui est spécifiée dans la méthode d'ouverture de la base. Cette classe nous permet après avoir fait un « Select » de consulter les résultats obtenus qui sont stockés dans un « ArrayList ».

Utilisation de l'application

- *Inscription d'un client dans la base de données*

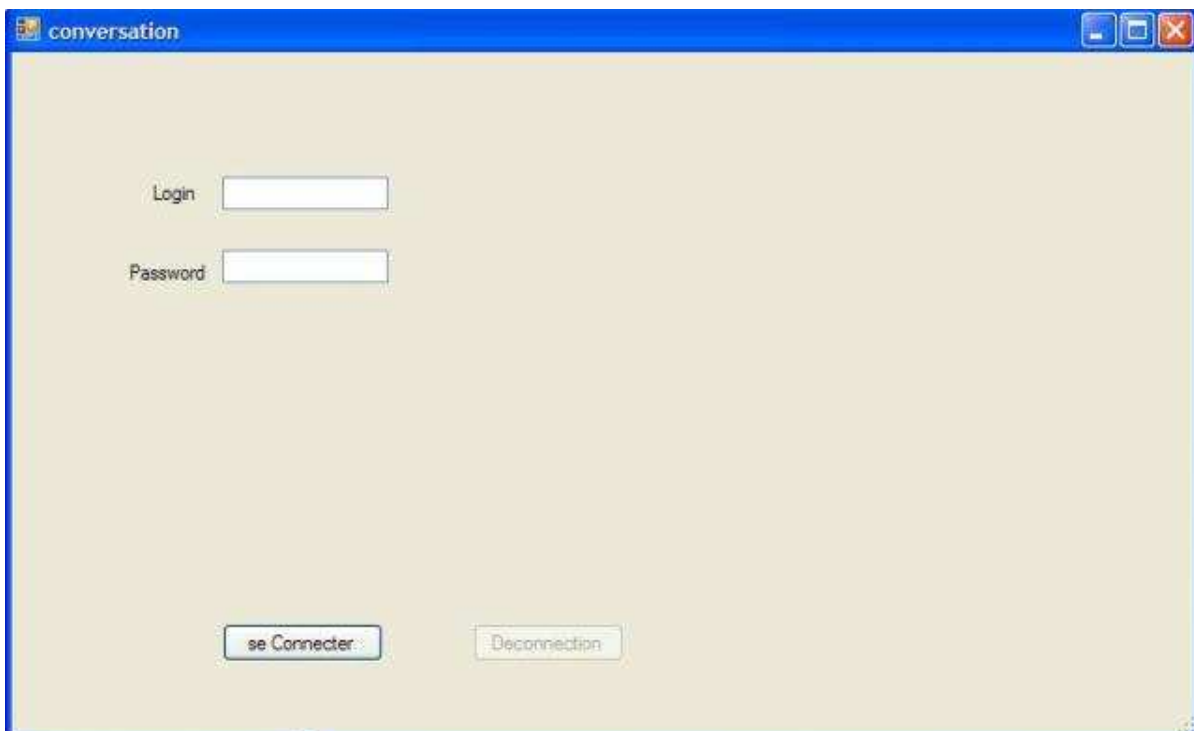
La fenêtre ci dessous est celle qui s'ouvre au lancement de l'application. A partir de celle ci, le client peut soit s'inscrire, soit passer à l'étape suivante.



The screenshot shows a window titled "Inscription" with a blue title bar. It contains two text input fields: "Login" and "Password". Below the fields are two buttons: "Inscription" and "deja inscrit".

- *Connection d'un client sur le serveur*

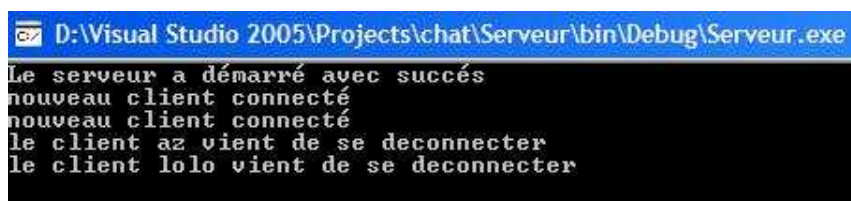
Une fois que le client s'est inscrit dans la base de données, il va pouvoir se connecter pour communiquer avec les autres clients. Pour ce faire, il faut qu'il entre son login et son mot de passe.



The screenshot shows a window titled "conversation" with a blue title bar. It contains two text input fields: "Login" and "Password". Below the fields are two buttons: "se Connecter" and "Déconnection".

- *Le serveur*

Une fois le serveur lancé, aucune action n'est possible. Dans la console, on peut voir des informations sur les clients qui se connectent et se déconnectent. Le serveur ne fait que passer un message provenant d'un client à un autre client.



- *L'interface web*

A partir de cette interface, un client peut consulter tous les messages qu'il a écrit. Pour qu'il puisse les voir, le client aura besoin de son login. Un client ne pourra pas voir tous les messages qui ont été écrit.

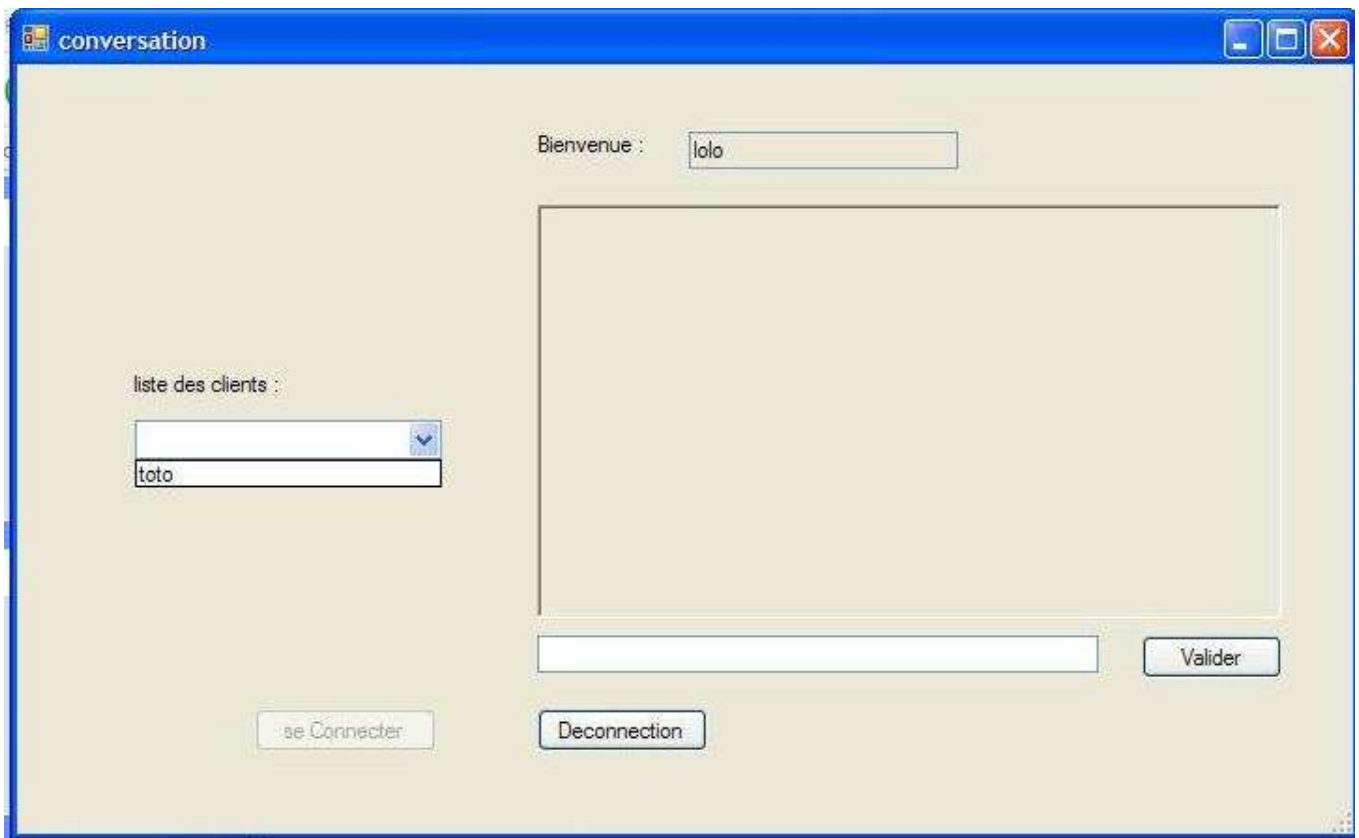


- *Chat avec un autre client*

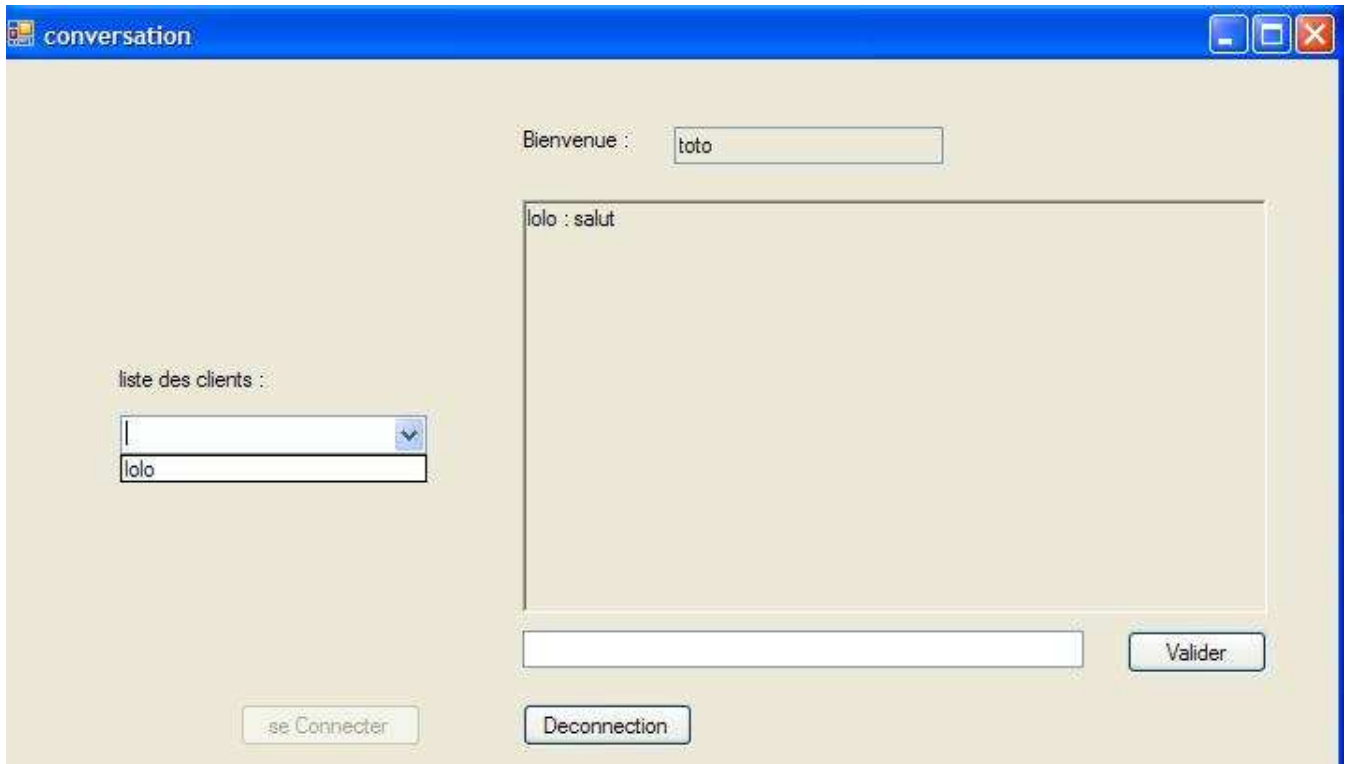
Une fois connecté, le client peut envoyer des messages aux autres clients connectés. Pour voir les clients qui sont connectés, il faut utiliser la liste déroulante.

Dans la grande zone de texte non éditable, les messages des clients et du serveur apparaîtront.

Dans cette fenêtre, le client "lolo" ne se voit pas dans la liste des clients connectés.



Dans cette fenêtre, on peut voir que le client "toto" peut écrire à "lolo" et que "lolo" a écrit "salut".



Après déconnection d'un client, son nom disparaît de la liste déroulante et le serveur envoie un message aux clients que le client vient de se déconnecter. Le serveur informe les clients de la même façon de la connexion d'un nouveau client.



Conclusion

En conclusion, ce TP nous a permis de voir la différence entre les modes de connections des serveurs. En utilisant le mode « singleCall » on ne pouvait pas réaliser le TP c'est pour cela que nous avons utilisé le mode « singleton ».

Pour faire une application robuste, il aurait fallu mettre un verrou sur la variable contenant les messages sur le serveur car si plusieurs clients envoient un message en même temps, il y a un risque que certains messages ne parviennent pas aux autres clients.