# 1. Chat Messenger

### Grunddesign

Die Übertragung der Chat Messenger-Daten basiert auf den System.Net-Klassen des .NET Frameworks. Diese beinhalten einen Listener, welcher auf eingehende Verbindungen wartet, einen TcpClient, welcher eine Verbindung zum Listener aufbaut und weitere nützliche Klassen für die Ethernet-Übertragung.
Damit die Übertragung auf einer hohen Abstraktionsebene erfolgt, wird eine Nachrichten-Klasse erstellt, welche direkt übertragen wird. Das Senden und Empfangen erfolgt mit den Serialize/Deserialize-Funktionen. Diese erzeugen aus einem Objekt einen Bytestrom, der direkt über das geöffnete Netzwerk-Stream-Interface übertragen und empfangen werden kann. Die Nachrichten-Klasse beinhaltet verschiedene Informationen, welche für die aktuelle Übertragung relevant sind.

Einträge der Nachrichten-Klasse:

- Kommando (enum: CONNECT, DISCONNECT, SHOWUSERS, CHAT)
- Nick Name (string)
- User ID (integer: 0...Server, -1...Client versendet Nachricht,
  1,2,3... ...Server leitet Client Nachricht den anderen Clients weiter)
- Nachricht (string)
- Server Zeit (DateTime: Eintreffen der Nachricht am Server)
- verbundene User (User-Liste falls Anfrage auf verbundenen Benutzer)

Dem Client stehen Befehle für das Öffnen einer Verbindung(CONNECT), das Schließen einer Verbindung (DISCONNECT), das Schreiben einer Nachricht (CHAT) und das Abfragen aller verbundenen Benutzer (SHOWUSERS) zur Verfügung.
Verbindet sich ein Client zum Server, so teilt der Server den anderen Teilnehmern den neuen Benutzer mit (CONNECT), ebenfalls beim Schließen der Verbindung. Eine Nachricht wird allen Teilnehmern, auch dem Sender, weiter geleitet (CHAT). Eine Anfrage aller Teilnehmer liefert die Liste mit den Benutzern zurück (SHOWUSER).
Server-Verbindung schließen, ist die einzige Nachricht, die der Server ohne eingehende Nachricht schickt (DISCONNECT).

Damit der Server alle Clients gleichzeitig bedienen kann, wird pro eingehender Client-Verbindung ein eigener Thread gestartet. Dieser Übernimmt das Empfangen und Versenden von Nachrichten an einen Client. Das Warten auf eine neue Verbindung erfolgt ebenfalls in einem Thread, somit bleibt die graphische Oberfläche immer bedienbar.
Im Client-Programm wird beim Verbindungsaufbau ein eigener Thread, ebenfalls für das Senden/Empfangen von Nachrichten zum/vom Server, gestartet. Wird eine Nachricht, durch drücken des Senden-Buttons, generiert, so wird sie einer Queue angehängt, welche durch den Thread abgearbeitet wird. Die Abarbeitung der empfangenen Nachrichten erfolgt bereits im Thread. Das Ändern der Form-Einträge, -Eigenschaften, erfolgt über eine Delegate-Methode, welche durch Invoke zum Form-Thread synchronisiert wird.

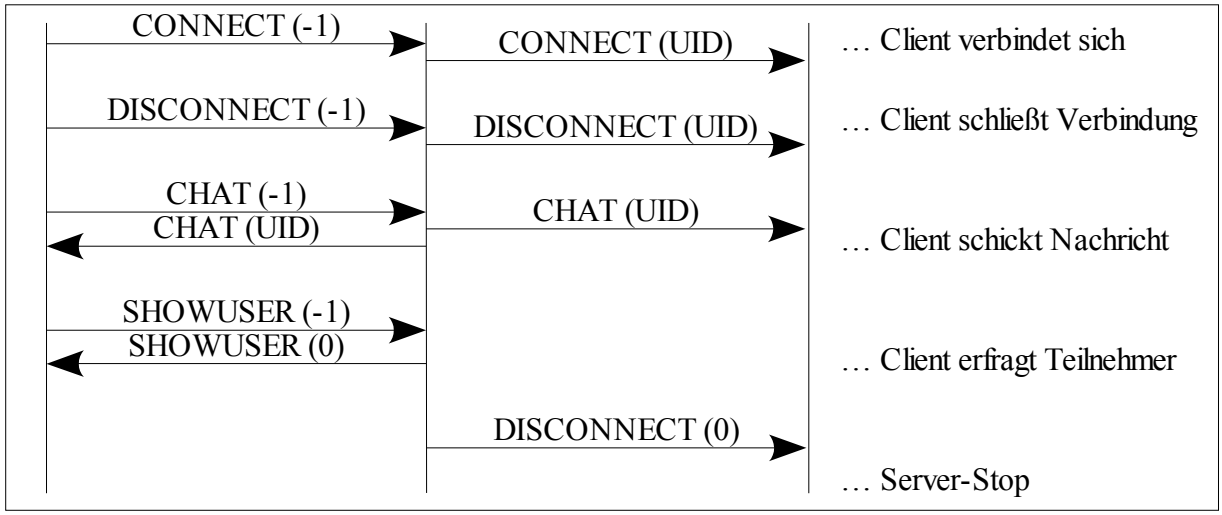


**Bild 1** – Sequenz-Diagramm

**Benutzerinterface Server**

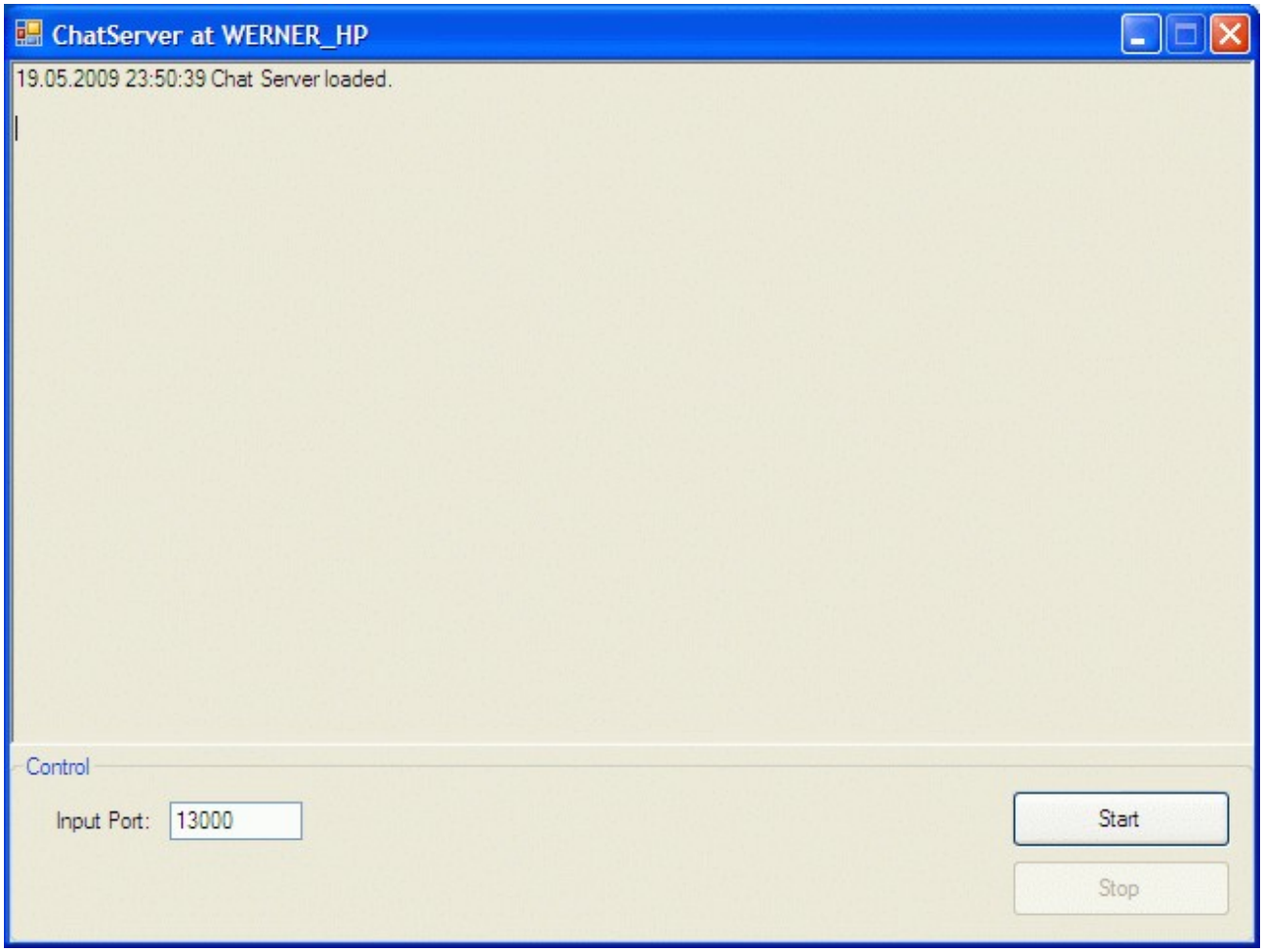


**Bild 2** – Server GUI

Die obere Hälfte der Server GUI besteht aus einem Textfeld, wo alle Nachrichten angezeigt werden. Unterhalb befinden sich Buttons zum Starten/Stoppen und ein Eingabefeld für die Port-Nummer. Das Textfeld ist als ReadOnly definiert und kann durch ein Kontextmenü-Eintrag gelöscht werden. Das Eingabefeld wird ständig auf Richtigkeit der Eingabe überprüft (Integer). Mittels Start-Button wird der Server-Thread gestartet, welcher auf eingehende Verbindungen wartet. Der Stop-Button terminiert den Server- und alle verbundenen Client-Threads. Beide Schaltflächen sind gegenseitig ausschließend und sind nur dann aktiv, wenn der Server für die Eingabe bereit ist. Damit die Verbindungen beim Schließen der Anwendung ebenfalls terminiert werden, wurde ein Event, welches vor dem Schließen der Anwendung generiert wird, implementiert.

**Benutzerinterface Client**

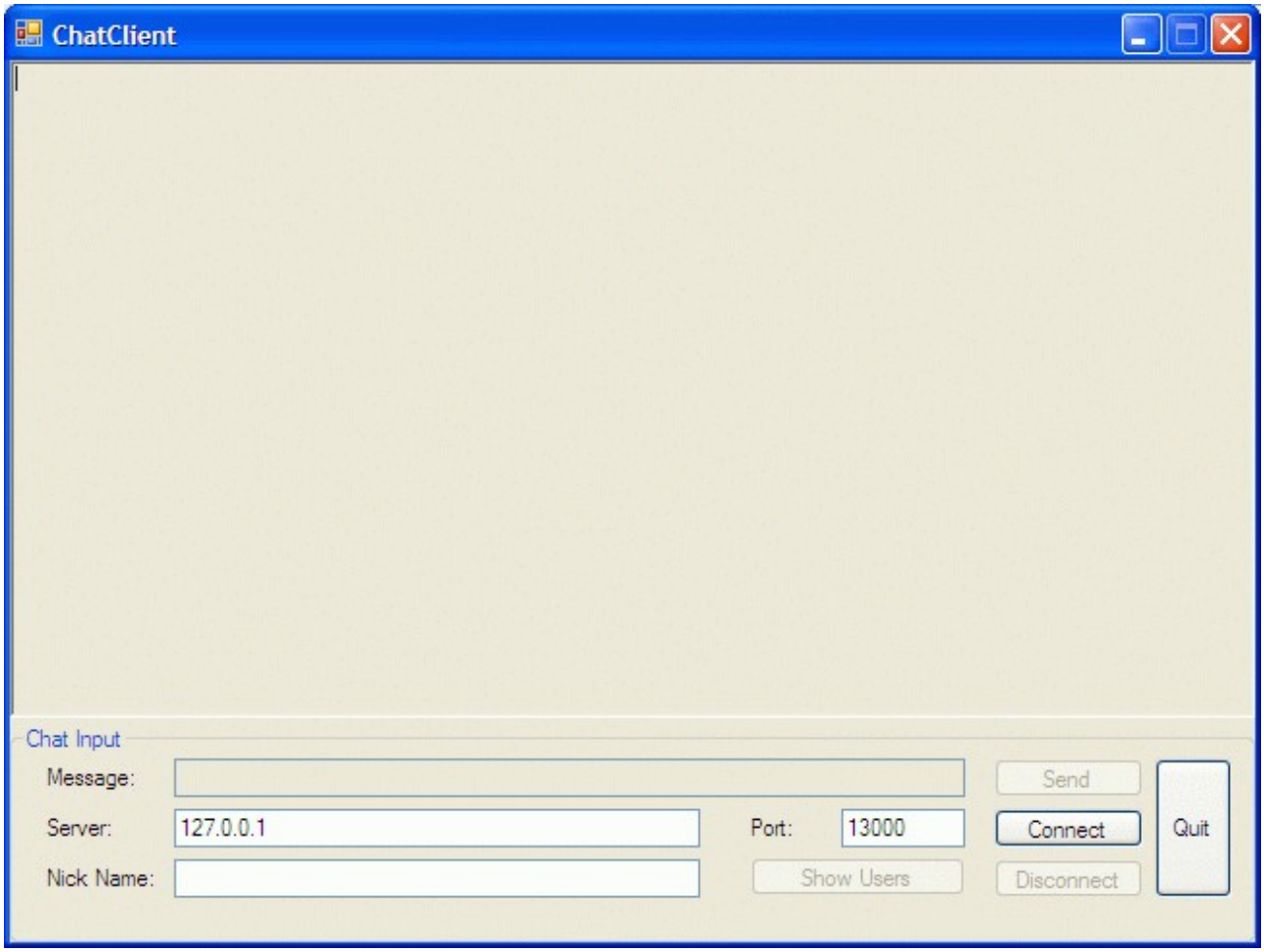


**Bild 3** – Client GUI

Die Client Oberfläche ist wieder in zwei Flächen aufgeteilt (Textfeld, Eingabefläche). Im Textfeld werden die empfangenen Nachrichten dargestellt. Die Eingabefläche beinhaltet Eingabefelder für die Server-Informationen (IP-Adresse/Host-Namen, Port-Nummer), Eingabefelder für den Benutzernamen und die Nachricht, Buttons zum Verbinden, Schließen der Verbindung, Senden einer Nachricht, Anzeigen aller Teilnehmer und Beenden der Anwendung.
Für das leichtere Bedienen wurden die Eingabefelder mit Tastatur-Events ausgestattet (Enter → Verbinden bzw. Nachricht senden). Nach dem Drücken des Connect-Buttons wird ein Thread

gestartet, welcher die Kommunikation mit dem Server übernimmt. Mit Disconnect wird dieser Thread wieder sicher terminiert. Der Thread arbeitet alle empfangenen Nachrichten selbstständig ab und fügt im Textfeld, über eine Delegate-Methode (durch Invoke mit Form-Thread synchronisiert), die Nachrichten des Servers ein. Durch Drücken des Senden-Buttons wird eine neu generierte Nachricht an einer, mittels Mutex synchronisierten, Queue angehängt. Der Thread überprüft diese Queue regelmäßig auf vorhandene Nachrichten, und schickt sie gegebenen falls dem Server.

**Detailbetrachtung – Thread-Synchronisation zum Form-Thread**

Damit ein Worker-Thread die Form-Umgebung verändern kann, müssen die aufgerufenen Funktionen zum Form-Thread synchron sein. Dies wird mit Hilfe von Invoke erreicht. Das Member-Delegate mPrintConsole, welche für die Ausgabe im Textfeld verwendet wird, und das Synchronisations-Objekt mSyncObject, wird im Constructor vom Form-Thread übergeben.

```
public delegate void AddStringToConsoleDelegate(string str);
private AddStringToConsoleDelegate mPrintConsole;
private ISynchronizeInvoke mSyncObject;

...

protected delegate void PrintConsoleDelegate(string str);
private void PrintConsole(string str)
{
    if (mSyncObject.InvokeRequired)
        mSyncObject.BeginInvoke(new PrintConsoleDelegate(PrintConsole),
        new object[]{str});
    else
        mPrintConsole(str);
}
```

**Detailbetrachtung – Delegate für Verbreitung empfangener Nachrichten (Server)**

Der Server verwaltet mehrere ConnectedClient-Threads, die selbstständig Nachrichten der Clients empfangen und verarbeiten. Damit eingehende Nachrichten den anderen Clients weiter geleitet werden können, werden diese dem Server mittels Delegate-Methode übertragen, welcher die aktuelle Nachricht, den einzelnen Clients, an die Sende-Queues anhängt.

ConnectedClientThread.cs:

```
ServerThread.SpreadClientMessageDelegate mMsgDelegate;

...

        case ChatMsg.eMsgCmd.CHAT:
            msg.mServerTime = DateTime.Now;
            msg.mUID = mUID;

            mMsgDelegate(msg);
            break;
```

ServerThread.cs:

```
public delegate void SpreadClientMessageDelegate(ChatMsg msg);

...

        // add new client
        mClients.Add(new ConnectedClientThread(client, newUID,
                     new GetUsersDelegate(GetUsers),
                     new SpreadClientMessageDelegate(SpreadMsg),
                     new AddStringToConsoleDelegate(PrintConsole)));

...

public void SpreadMsg(ChatMsg msg)
{
    CleanClientNotConnected();

    foreach (ConnectedClientThread client in mClients)
    {
        client.AddOtherClientMessage(msg);
    }
}
```

## Ergebnisse

Der Test im Debug-Modus innerhalb der Visual Studio Umgebung brachte bereits viele kleine Implementierungs-Fehler zum Vorschein, dennoch war der reale Einsatz, auf einem nicht Entwicklungsrechner, sehr wichtig. Es traten zwei schwerwiegende Probleme auf. Zu Beginn wurde der Client auf einen anderen Rechner übertragen, der kein .NET Framework 3.5 SP1 installiert hatte. Da die Anwendung zwar startete, aber nach dem Versuch eine Mutex-Methode aufzurufen, eine unklare Fehlermeldung lieferte, dauerte es etwas länger, bis das eigentliche Problem gefunden und behoben wurde.
Nach der Installierung des Frameworks funktionierte die Software zum großen Teil, außer die Terminierung der Threads (funktionierte bereits in der Debug-Version). Dabei stellte sich heraus, dass in der Release-Version ein Timing-Unterschied vorlag und der Thread somit nicht terminiert werden konnte.
Ein abschließender Test der fehlerfreien Chat-Messenger-Umgebung ist durch die eingebunden Screenshots ersichtlich.

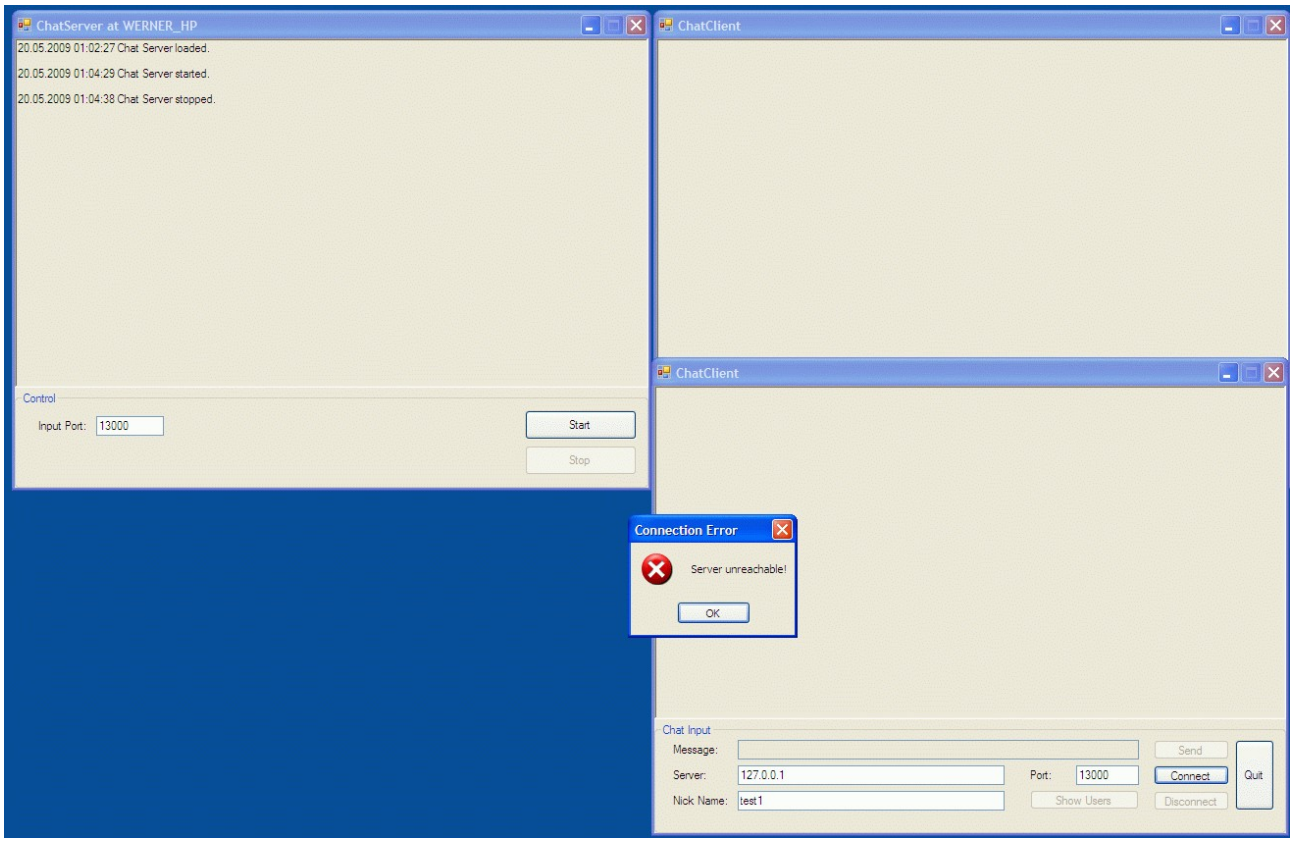


**Bild 4** – Server nicht gestartet, Client Verbindungs-Fehler

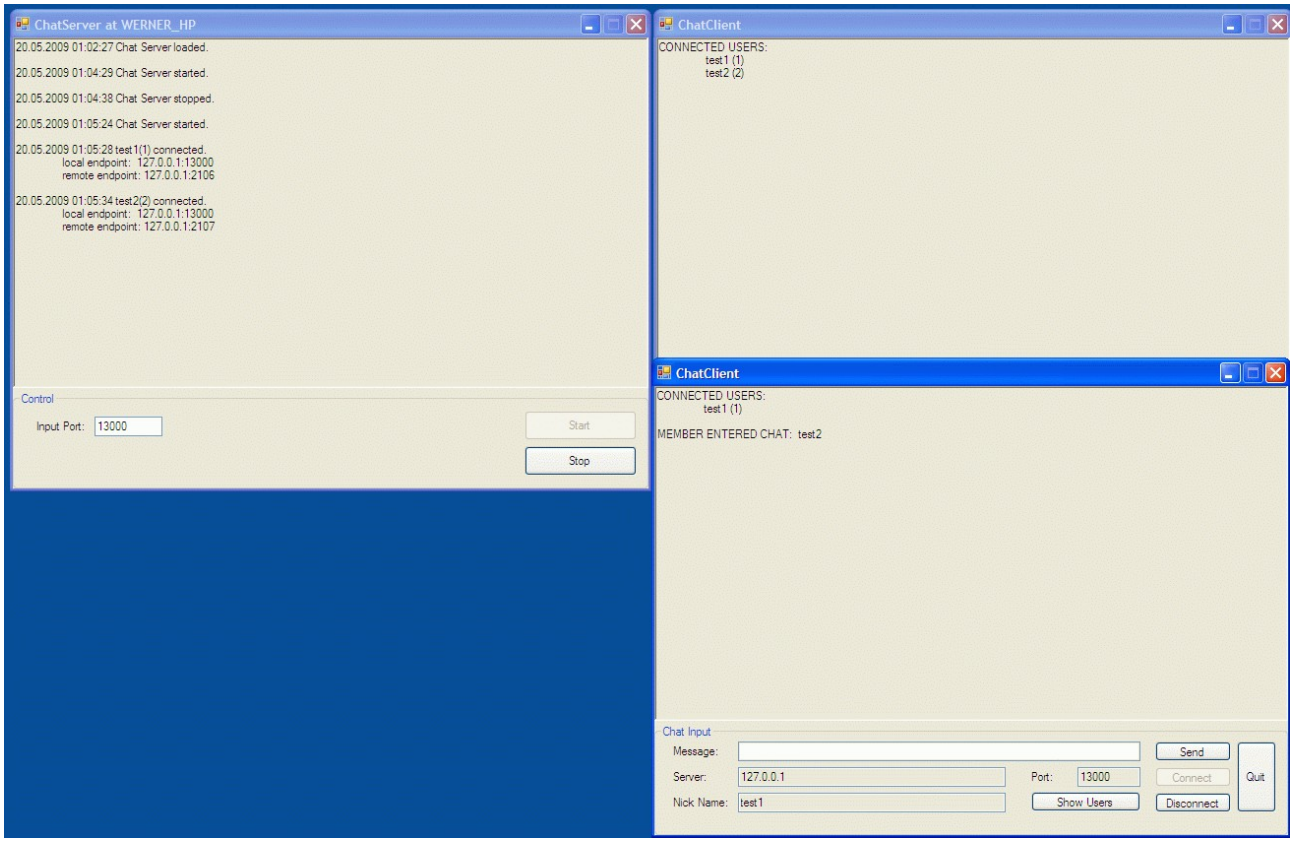


**Bild 5** – Server gestartet, Clients verbunden

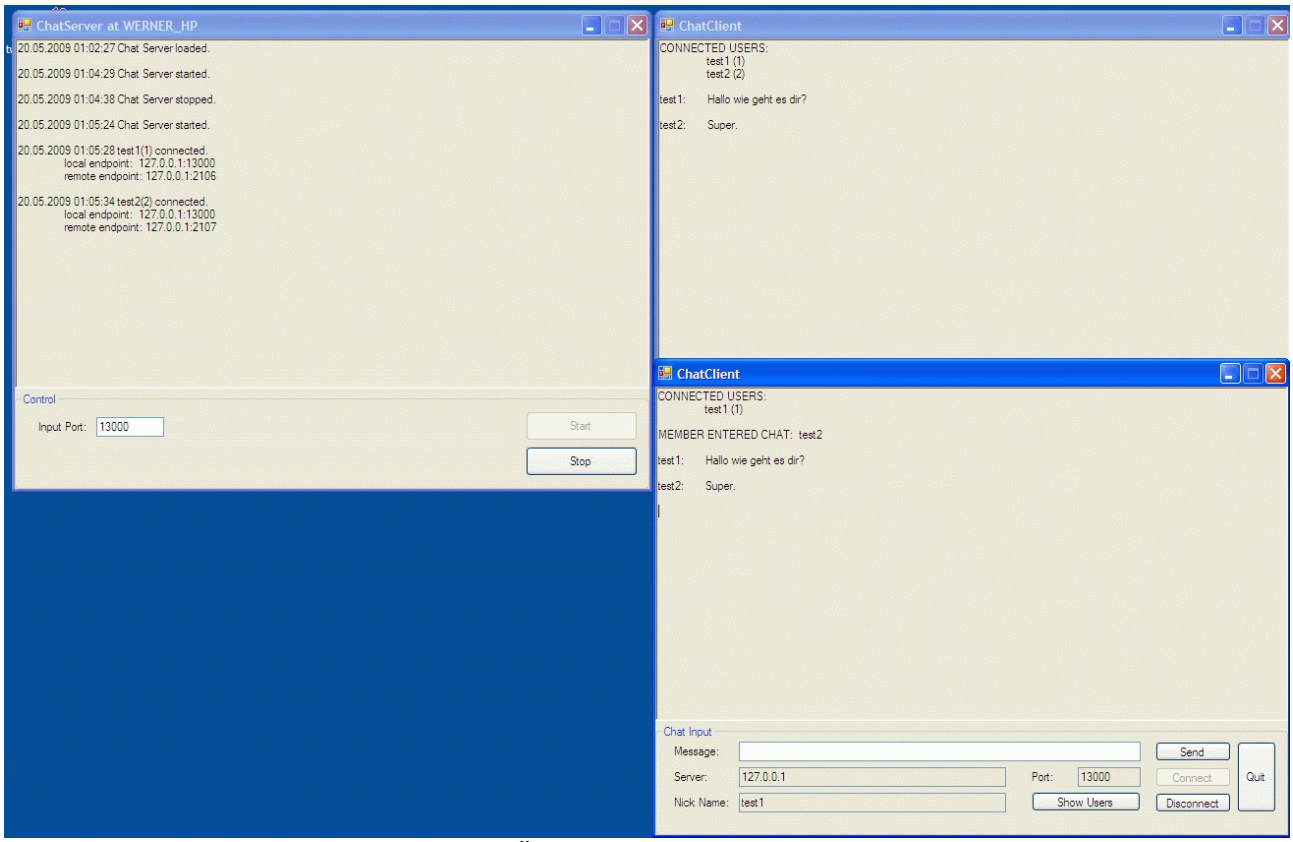


**Bild 6** – Übertragung von Nachrichten

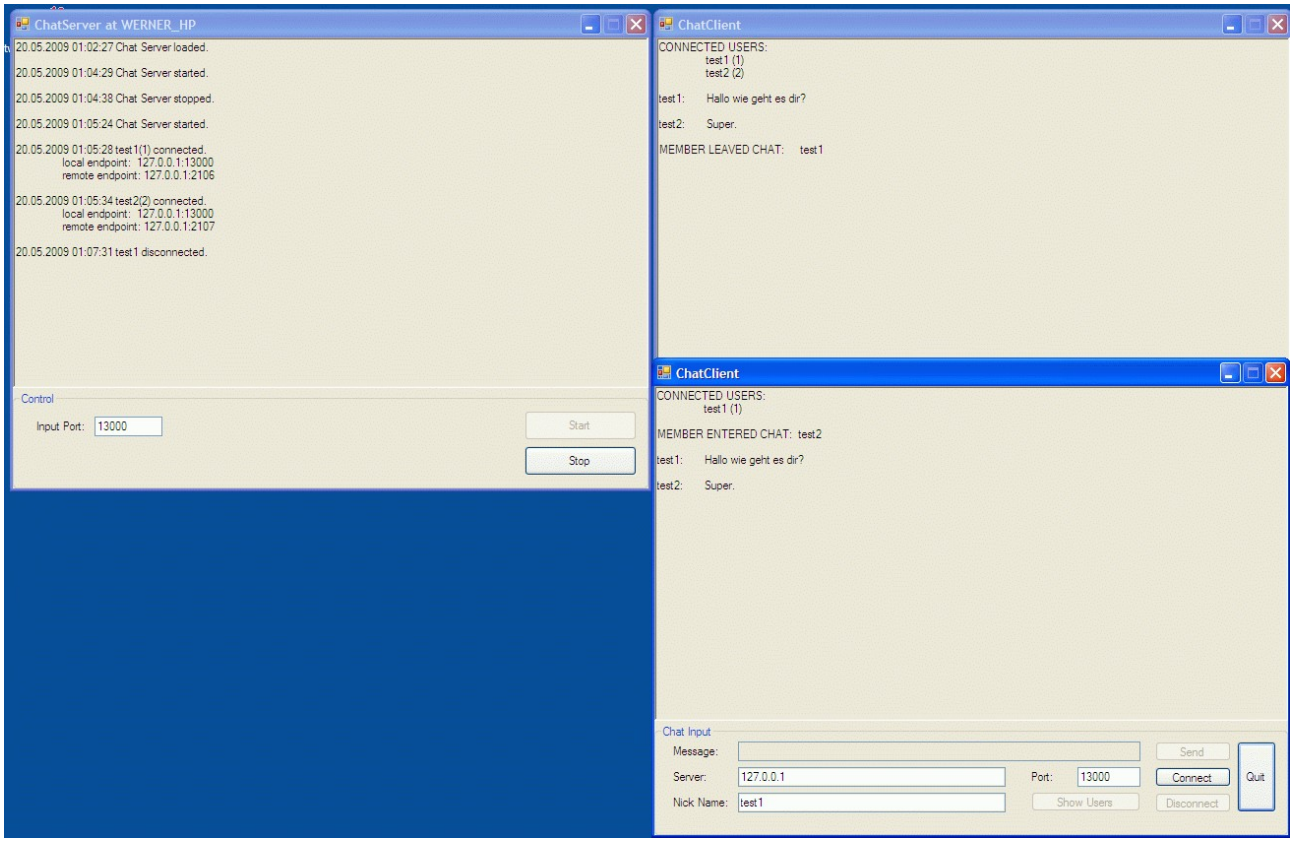


**Bild 7** – Client verlässt Chat

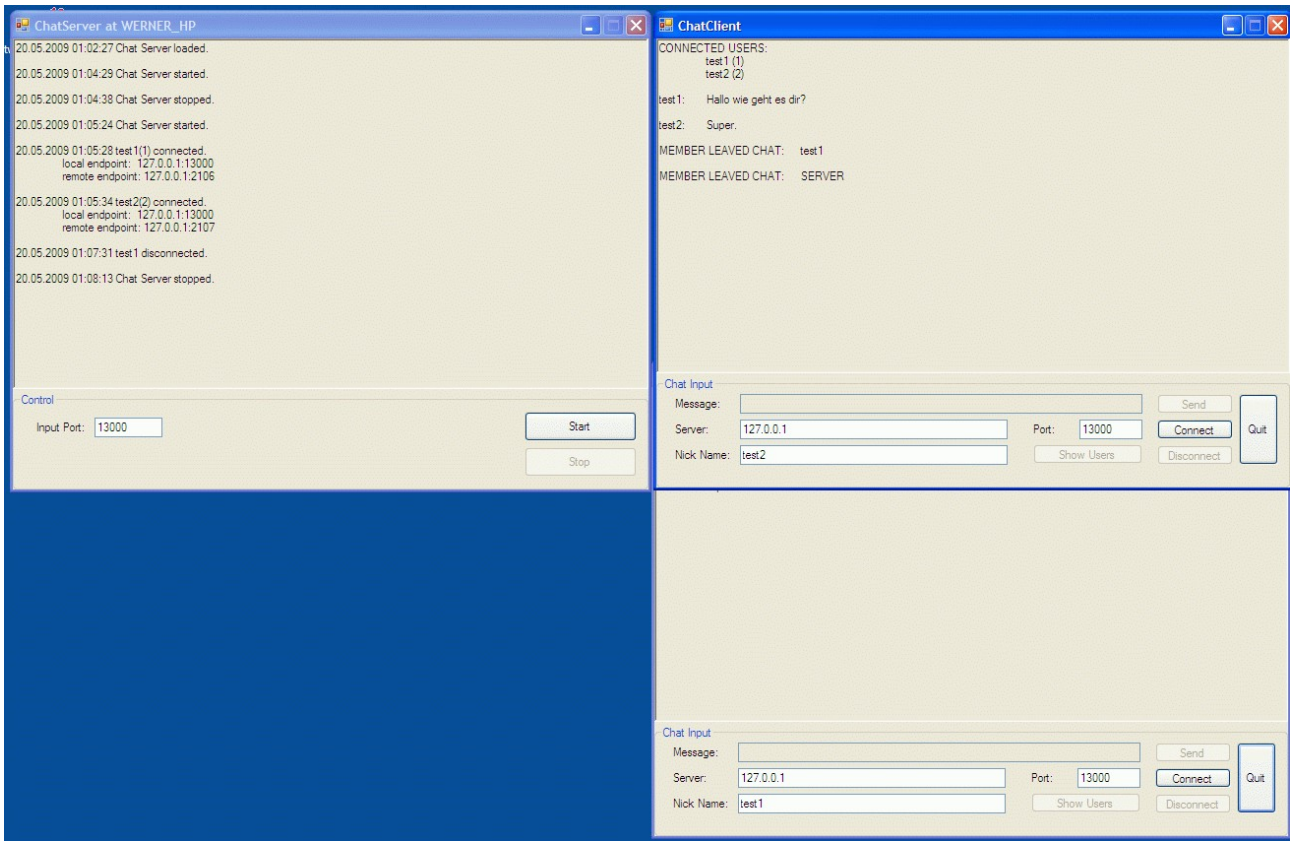


**Bild 8** – Server wird gestoppt

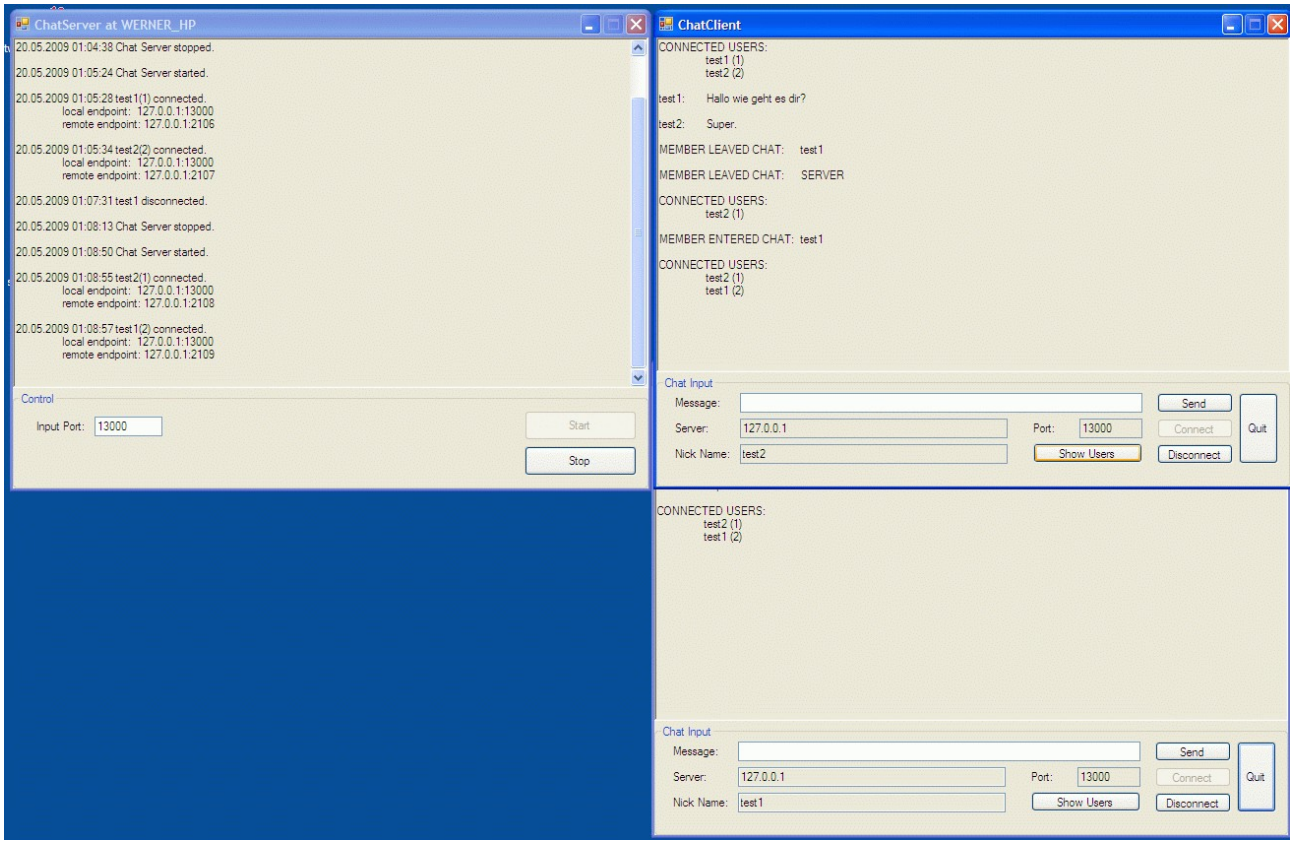


**Bild 9** – Server Neustart, Anzeige der Teilnehmer