Vojpeh

A simple VOIP project

Jacob Ferm, dt08jf0@student.lth.se
Anton Karlstedt, dt08ak0@student.lth.se
Erik Söderqvist, dt08es8@student.lth.se
Mats Rydberg, dt08mr7@student.lth.se

14 maj 2012

1 Background

The system we have developed, Vojpeh, is a very simple voice over IP system. This means it has functionality that allows voice communication between users. It is split into two parts, a server and a client application. When a client is connected to a server it is able to view all other users connected to that same server and request a call to be made to another user. If the other user accepts, a voice line between the two clients will be opened.

The system can be used by anyone, as we have developed an easy-to-understand GUI. You will have buttons with names explaining their functionality that they represent. The one issue that could confuse users is mentioned in section 4 below, and is about the ports used by Vojpeh. Other than that, anyone with knowledge of the server address can register and start calling someone today!

2 Requirements specification

There was no formal document specifying the functionality of the Vojpeh system, but there was a short description, which follows (in Swedish):

> Vi planerar skapa ett system som låter användare kommunicera med varandra mellan olika datorer med hjälp av ljud. Med andra ord, vi vill skapa en simpel version av Skype. Användare ska kunna se vilka andra personer som finns tillgängliga och ringa upp dessa.
>
> Systemet ska bestå av en adresserver och en klient. Adresserverns uppgift är att hålla reda på vilka användare som finns anslutna till systemet och förmedla samtal mellan dessa. Klienten ska kunna ansluta till en adresserver, hämta en lista på tillgängliga användare från servern samt upprätta samtal till dessa användare. Klienten behöver därmed klara av att spela in ljud från en mikrofon samt spela upp ljud.
>
> Potentiella utökningar till systemet är att möjliggöra gruppsamtal mellan flera användare eller att öka samtalens säkerhet genom att kryptera datan.

The above could be summarized to the following few requirements:

1. There should be an addresserver and a stand-alone client.
2. The server should keep track of registered clients and set up calls between these.
3. A client should be able to register to the server.
4. A registered client should be able to get a list of other clients registered to the server.
5. A registered client should be able to call another registered client.
   (a) A microphone or other sound-input device has to be available.
   (b) A speaker or other sound-output device has to be available.
6. Possible extensions:

(a) Group calls.
(b) Encrypted network communication.

Not part of the text above, but still a planned design feature:

7. The server should not maintain, but only set up calls.
   (a) All set-up calls are completely client-client.
   (b) The server does not know which clients are currently calling eachother.

3 Model

Here the different parts of the system is described in some detail.

Server

The server is composed mainly of three important classes: `Server`, `ConnectionHandler` and `PhoneBook`. The first thread is `Server`, which is started by running the exported .jar package (by main-method). It works mainly through delegation, and does very little actual work itself. It has access to a `PhoneBook` instance, which it sends further down the chain of command, where its methods are invoked. `The Server` thread waits for clients to connect to it, on a given port, and once connection has been set up, it creates and starts a `ConnectionHandler` thread to handle the incoming request. It then proceeds to receive a new clients connection attempt. `Server` is not a subclass of `java.lang.Thread`, but it functions as if it were. `ConnectionHandler` first parses the request received and then invokes methods in `PhoneBook` as needed for the request. It then (always) sends a response to signal to the client how the request was handled and whether it was successful or not. `ConnectionHandler` extends `java.lang.Thread` and handles instances of the class `Connection`, which it received in its constructor. The `PhoneBook` class is the server-side monitor. It stores the common information of the server, which is addresses and names of clients handled with a `java.util.Map` implementing class. It guarantees full thread safety, as it is needed with many `ConnectionHandler` instances invoking its methods concurrently.

Supporting classes in the server package are `Connection`, `MessageType`, `Protocol`, `Request` and `Response`. `Connection` is more or less a wrapped `java.net.Socket`, working solely with `ObjectIOStreams`. Sent through these streams are classes implementing the serializable interface `Protocol`. The `Server` in its current implementation uses only the classes `Request` and `Response`, which both implement `Protocol`. All responses sent by the server will be instances of `Response`. The `MessageType` enum is used to model a type field in `Protocol`. Finally there is also an inner class in `Main`, which handles the server-side shutdown command.

Client

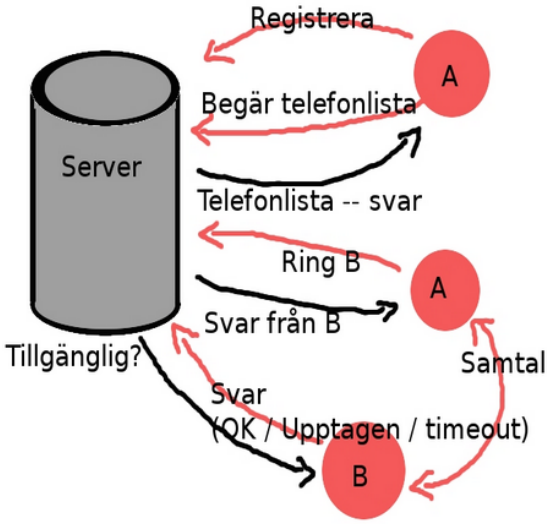
The client application consists of three different parts, the basic gui functionality, a thread that listens for requests from the server and a few threads for managing calls.

The gui is started when the application is first started. This class creates all the interface elements such as buttons, lists, labels and handlers to deal with user input. When connected to a server it uses the thread `UserListUpdater` to periodically create a socket to the server and fetch the list of names of all connected users.

The thread that detects incoming calls, `CallReceiver`, is started by the `GUI` class when you connect to a server. This thread listens for connections from the server with requests that another users wishes to start a conversation. If the user declines the call a NACK is passed on to the server, if he user accepts the third part of the client for managing calls will be started and an ACK is passed on to the server.

The main thread for dealing with calls is `CallManager`. It is started either by the `CallReceiver` thread when an incoming call is detected of by the `GUI` class when an outgoing call is requested. This thread creates a new gui that displays the current status of the call and contains buttons for muting the call and ending it. If the call is incoming the thread will act as a server and wait for the other user to connect. If the call is outgoing it will first send a call request to the server and wait untill it receives the desired users IP address. It will then open a connection to the other user. In both cases, when the connection is created the `CallManager` will create two other threads, `CallInput` and `CallOutput`. `CallInput` will read data from the connection and play it through the computer's speakers. `CallOutput` will read data from the computer's microphone and write to the connection.

All communication with the server is accomplished with the `Protocol` classes in the server package, `Request` and `Response`. These objects are passed through the connection's objectstreams.



Figur 1: This is the pre-project design of Vojpeh (in Swedish). The final design much follows the flow chart, except that Vojpeh does not implement any time-out functionality. The unregistration process is not in the figure.

4 User manual

Here follows a step-by-step guide on how to get Vojpeh up and running.

Server

To start the server run the command `java jar vojpeh_server.jar` from the command line. The server is now ready to accept connections from users. To close the server write ”kill” and press enter in the console.

Client

Setup

1. Run the command `java -jar vojpeh.jar` from the command line.
2. Press the connect button in the window that pops up.
3. Enter the user name of your choosing.
4. Enter hostname:port in the next window. If left empty standard host and port will be used (localhost:45000).
5. If no problems occured you should now be connected to the server! (This can be confirmed by looking at the previously named ”Connect”-button, it should now display ”Disconnect”. Your user name is also shown to the top left if connected)

Calling

First connect to the server as shown in the Setup part of the user guide. To make a call to another user you double click that users name in the list in the user interface. If done correctly a new window will pop up that shows which user you are trying to call. If the other user doesn't wish to speak with you, a window will pop up and display that and you will be returned to the start window. If the other user accepts the window will change so that only the other user name is shown. When a call is in progress you can mute your microphone by pressing the mute button. To unmute it, press the button again. To end a call, press the end call button and the call will terminate. This will return you to the start window.

Ports

For the application to function correctly it must be able to accept incoming connections. This goes for both the server and the client. The server uses port 45000 and the client uses ports 45001 and 45002. If the server or client are located on a local network, extra routing may be required for the application to work.

5 Evaluation

All requirements except #6 above are implemented. These were not implemented due to lack of time. There is no nature of the Vojpeh systems structure that would disallow such extensions, but the current implementation would not be difficult to improve with the mentioned functionality. Group calls would possibly need its own `MessageType` signal, and be handled separately from other types of calls.

The task is well made. It is nice to get free hands to conduct a project like this one without any guidelines as it inspires programmatic creativity and allows for us to exercise all programming patterns and techniques we have learned so far. As a network communcation task, it might be a bit simple, as the Java language provides so many classes and so much abstracted functionality via its public API that the "real" work becomes simply invoking methods on classes written by professionals.

6 Program code

In the project, we have used Google Code Subversion as tool for version control. The project exist on Google's servers at `http://code.google.com/p/simple-voip-project/`, where it is open for anyone to checkout the project.