Project Nuntii

Operating systems and multicore programming (1DT089)
Project proposal for group 3: Hampus Adamsson(900921-1492), Erik Andersson (860119-2936), John Davey (930213-0738), Per Albin Mattsson (900321- 1613), Maria Svensson(920821-8041)
Version 1, 2014-04-15

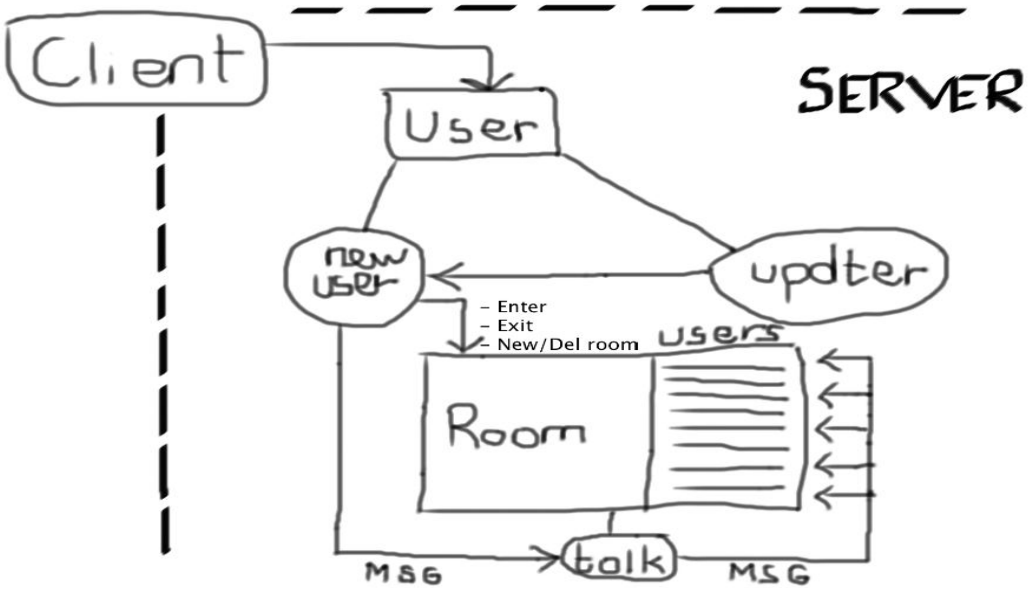
Table of contents

1. Introduction

We have selected to build a chat like an Internet Relay Chat (IRC). An IRC is basically a chat where clients send messages to each other through a server. We didn't have many other suggestions but an IRC-like chat prevailed because of its simple concept and the possibility to extend it with different features. The use of Erlang in a project like this seems to be an interesting learning experience. We plan to make a GUI in a different programming language and by doing this we hope to learn more about how integration between backend and frontend works. We also wish to learn more about how communication with a server works. While we hope to learn these things, they will also be the main challenges in our project.

With many users in the system concurrency will be helpful because we hope to make every user a separate Erlang process. Another concept we aim to implement is the use of rooms where a certain group of users can communicate. We hope to be able to make these rooms processes as well.

2. System architecture



Picture 1a

We are aiming for a trivial and easy-to-implement system where each component is minimal in its requirements. The basic idea is to create a system that works like an onion – with layers and layers of functionality. When we have successfully implemented the basics we can either choose to perfect that module or build more functionality on top of it.

As described in *picture 1a* there are two modules needed for the common functionality of an IRC-server, namely **Room** and **User**. The User-module will care for all the users; creating users once a connection is established and deleting users once a connection is lost or terminated. The Room-module will care for all the rooms where users are located. These rooms can roughly be seen as a sub-list of users as well as the necessary component needed to communicate with the room. Each room (within the Room-module) will be represented by a thread, hence fulfilling the conceptual idea of concurrency, and also not limit the efficiency as the number of rooms grow higher. The same idea goes for the average user as well – once connected to the server the user will be represented by a new thread. This thread only exists as long as the user stays connected to the server.

Furthermore we would like to expand our current model with perks such as security, user-information, AI and login. These are not primary objectives at this point but given time these modules will be implemented.

3. Concurrency models

We have chosen to use Erlang as the programming language for the backend in our project. Primarily because of the design of the language concerning concurrency and how easy it is to make use of concurrency in your programs. But also because of the ease of how Erlang nodes can communicate with each other over networks, which we think will be fitting for a project of this type. The actor model is the standard concurrency model in Erlang, hence this will be the most natural model for us to follow.

For the frontend and graphical interface we have settled on using Java or Python; we are leaning towards Python. The idea being that if Python turns out to be complicated to use we will use Java instead. The choice fell upon python because only two of us have used it before and we want to give it a try, and apparently it should be quite easy to familiarize yourself with.

In the Client, concurrency will appear in the form of a separate thread that handles all the receiving from a TCP socket. As such, no freezes due to blocking will occur in the mainthread that runs the GUI

4. Development tools

Source code editor: We will primarily use Emacs and Sublime during the project. For programming in Python we will most likely use IDLE as this provides good shell integration and is easy to use.

Revision control and source code management: As we have used Git throughout the course already, we will continue with that.

Build tool: We will use Make as we are familiar and comfortable with it, and it is powerful.

Unit testing: We are planning on using Erlang for the server, and Java or Python for the frontend. All of these have tools for testing and we will of course use these, which renders external debuggers redundant.

Documentation generation: As above, all of the languages we plan to use have good documentation generation and we will use these tools.

5. Process evaluation

We carefully listened to everyone's opinions and brainstormed. However, we all swiftly agreed upon making an IRC chat because everyone had different wishes and skills they wanted to develop through this project. A couple of these wishes were:

Easy to get to work and expand
Front-end design
Scalability

An IRC chat combines all of these requirements.
The brainstorming did not work as well as we initially thought. Instead of focusing on what we were going to produce, we focused on what skills we wanted to develop, and from there we came up with the idea of what we wanted to make. We played with the idea of a game, but we chose to discard it because we had no idea of how much work would be involved, how to implement it, nor what level to take it to.

Right now our focus is that everyone must do their part and take their responsibility in this project, and because of this we have chosen to follow the Scrum method. We will change the Scrum master each week so everyone will have the opportunity to have the main responsibility. We think that this will make us more efficient as a group.