KID Is Dirty

Because he is playing in the MUD

Process Oriented Programming (1DT049), project proposal for group 14, version 1, 2012-04-24

Eric Arnerlöv	900406-0092
Michael Bergroth	910123-4194
Magnus Lång	910409-2672
Mikael Wiberg	840526-0194

1 Brainstorm

This project proposal started with a brainstorm.

1.1 Preconceptions

A brainstorm is a useful tool to come up with idéas without restrictions. Everybody is encouraged to participate and no ideas are bad ideas.

1.2 Summary

These are the ideas that came up during the brainstorm.

Games

Picture guessing game

Multiple users are connected to a session with a chat. A picture is gradually revealed to all players, and the first player to type the correct answer (what the picture depicts) gets a point.

We plan to explore concurrency by making the server multithreaded (or multiprocess in erlang)

Pictionary

Multiple users are connected to a session with a chat. One person is selected to draw a randomly selected word, and the other players are shown his drawing and the first player to type the word gets a point.

We plan to explore concurrency by making the server multithreaded (or multiprocess in erlang)

Pong

Multiplayer pong.

We plan to explore concurrency by making the server multithreaded (or multiprocess in erlang)

Chess

Multiplayer chess

We plan to explore concurrency by making the server multithreaded (or multiprocess in erlang)

Rouge-like

Single player ASCII graphics dungeon explorer game

No gain in introducing concurrency, see the next idea instead

MUD

Multi User Dungeon. Multiplayer text-based dungeon explorer game.

We plan to explore concurrency by making the server multithreaded (or multiprocess in erlang)

Multiplayer Breakout

Cooperative play breakout clone, where the roof of one players playing field is the bottom of another.

We plan to explore concurrency by making the server multithreaded (or multiprocess in erlang)

Creative tools

Multi User Text Editor

A text editing service allowing multiple users editing the same text.

We plan to explore concurrency by making the server multithreaded (or multiprocess in erlang). The project will have a heavy focus on synchronization.

Multi User Sequencer

A sequencer playing a fixed length loop, where several users connect and create their own tracks, and the entire arrangement playing on all users machines.

We plan to explore concurrency by making both the server and the client multithreaded (or multiprocess in erlang).

Multi user image editor

A drawing canvas where several users can draw.

We plan to explore concurrency by making the server multithreaded (or multiprocess in erlang).

Services

Chat server

Chat server.

We plan to explore concurrency by making the server multithreaded (or multiprocess in erlang).

Ticket reservation system

Multiple users can reserve tickets from a limited supply.

We plan to explore concurrency by making the server multithreaded (or multiprocess in erlang). The project will have a heavy focus on synchronization.

1.3 Conclusions

The brainstorm worked well. All members contributed and several interesting ideas was spawned in the discussions. We could have improved the process by having prepared individually to have more ideas to work from, because the slowest part was the beginning of the brainstorm. We learned that one easily gets stuck in one line of thought, and has problems to thinking outside the box.

2 Project selection

We selected MUD as our idea for our project. We three projects, Picture guessing game, MUD and Multi User Sequencer as the projects we felt particularly interested in. We finally chose the MUD because everybody was excited about it and it seemed easy to scale in scope.

We found the MUD especially interesting because it is different to the kinds of programming tasks we are usually assigned, and because of it's larger scale we think it will be a more engaging project.

We hope to during this project learn how to write larger systems in erlang, and how they fit together. We also hope to learn from each other and the dynamics of working in a group. Also we think the scalability requirement of a server will be interesting to fulfill.

The main challenges will be the network programming, because we have the least experience in that area, and the synchronization because we do not really know what will be required of it.

3 System Architecture

A MUD (Multi User Dungeon) is a text-based online game where players perform actions by typing them (such as “go north” or “attack goblin”). The world is partitioned into *zones*, and each player exists in one zone at any one time, and can move to neighboring zones. Zones can be occupied by several players as well as one or more NPC (Non player characters) that can be interacted with (e.g. attacked).

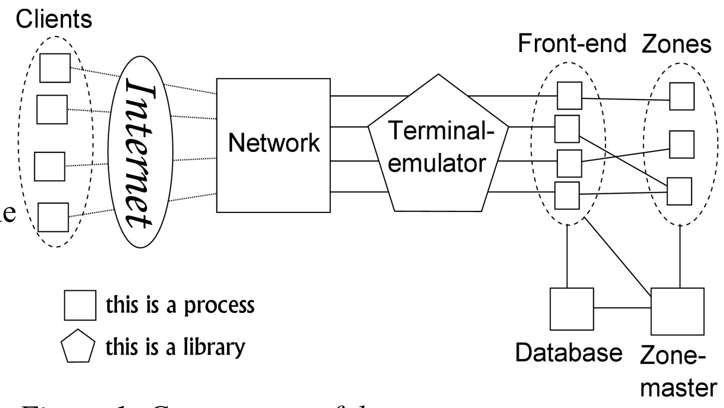

Figure 1: Components of the system

The system was broken down into the following parts, and interface as indicated in 1Figure 1.

3.1 Network

Listening at incoming connections. Spawns a front-end process for each connection.

3.2 Terminal Emulator

Read input from the user and sends it to the “Front End”. Allows the user to write an action whilst messages are outputted to his screen.

3.3 Front End

The “Front End” will handle all the inputs and interpret information which it will send to the right module. The front also simulates player behaviour such as moving or dying.

3.4 Zones

Each zone will have their own process to handle information.

The zones are active only if there is player in that zone.

When a zone is active, it will have their own process to handle that zone.

When a zone is inactive, it will be stored in the database.

3.5 Zone-Master

When the player connect to the game, the "Zone-Master" will spawn the player in the correct zone. If that zone is inactive it will start up a process to handle that zone.

3.6 Database

The database will contain the information about the player, the zones and the monsters.

3.7 Rationale

We chose this particular decomposition because it was an intuitive solution to the requirements we had on the system. We believe this decomposition will allow us to make our solution highly concurrent and scalable, without making it overly complicated.

4 Concurrency and programming languages

We have chosen Erlang as the language for this project. We believe it has the concurrency tools we need to make our solution scalable. By using the highly integrated message-passed concurrency model, the concurrency will take minimal effort compared to more traditional programming languages, such as C or Java.

5 Development tools

We plan to use the following tools (see Table 1) for our project.

Source code editor	Emacs
Revision control	Git
Source hosting	GitHub
Build tool	Rebar
Unit testing	Eunit
Documentation generation	EDoc

Table 1: Tools

5.1 Source code editor

We will use Emacs (or similar text editor) for our project. We did try Erlide[1] but came to the conclusion that it wasn't for us, and that the obstacle of learning a new editor wasn't worth the advantages.

5.2 Revision control and source code management

We have chosen to use Git for our source control, because it is fast, popular, distributed and there is several free hosting services for it. We plan to use Git[2] as a tool to enable us to work at the same time without having to worry. For hosting we plan to use GitHub[3]. We plan to use it as a central repository for our source code.

5.3 Build tool

For automated building and testing we plan to use Rebar[4], which is a Erlang specific build tool. We believe that it is better suited to the nature of Erlang projects than Make, and thus will be less of a

1 http://erlide.org/

2 http://git-scm.com/

3 http://github.com/

hassle to use. It also integrates into Git and Eunit automatically.

5.4 Unit testing

For unit tests we will use Eunit[5] as we are already somewhat familiar with it, and it is the most used unit testing library for Erlang. We plan to use Eunit for regression testing so that we do not unintentionally break something that is already working. This includes adding failing unit tests as the first step of bugfixing.

5.5 Documentation generation

We plan to use EDoc[6] for documentation generation as it is very easy to use.

4 https://github.com/basho/rebar/wiki

5 http://www.erlang.org/doc/apps/eunit/chapter.html

6 http://www.erlang.org/doc/apps/edoc/chapter.html