erlStream

Operating systems and multicore programming (1DT089)
Project proposal for group 1:

Jeanette Castillo (910929-4208)
Filip Hedman (931022-7476)
Robert Källgren (930405-0611)
Oscar Mangard (920820-4116)
Mikael Sernheim (920329-4179)

Version 2, 2014-05-05

1. Introduction

In todays world, streaming is a highly attractive subject and a constantly growing method of sharing media. It is becoming more valuable than ever to possess the relevant knowledge to understand the technologies behind it. We want to create a simple service to stream data between a server and clients. The final goal is to stream music through a simple, user friendly graphical interface.

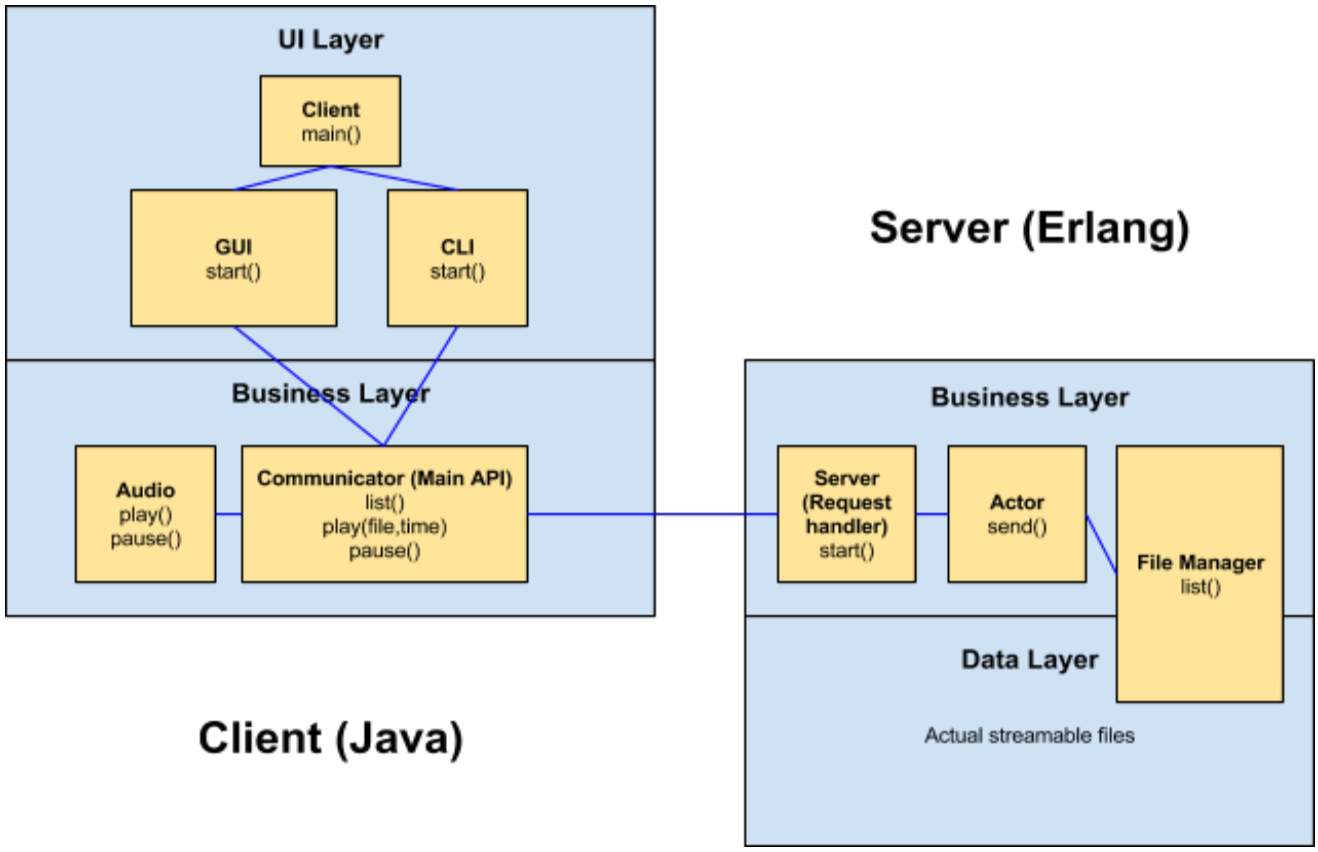
Our plan is to implement the server in Erlang, and the client in Java. Erlang provides us with great server communication via Erlang OTP, while Java will make it easy to use the service on any platform without trouble. We are also quite familiar with Java's Graphical User Interface (GUI) libraries.

One of the challenges we are facing is to combine two programming languages. The communication between them should only be by sending packages over the internet. This means that we have to research what protocol to use, as there are many options. Another challenge would be the file splitting process, which we definitely have to look into. Since we have no experience with managing audio files, we are not familiar with the circumstances for dividing and restoring them into playable music.

Concurrency comes into the picture as the server must have the ability to handle incoming requests from clients, while at the same time sending data to other clients. On the client side you have to be able to receive data, play it back and run this in a GUI.

With this project, we hope to gain a general understanding of streaming, basic network communication, combining multiple programming languages and audio management.

2. System architecture



An overview of the system architecture

We strive for an architecture that is based on the low coupling and high cohesion concept. By doing this, we hope to have a stable system with the ability to swap out modules with ease.

The client will consist of a Communicator class that will handle all the communication with the server, and act as an API for the user interface. The clients will send requests for songs that exist on the server, which will then return the appropriate data. The songs will be divided into chunks of data that the client translates and plays as audio. This method will keep the response time low and allow the user to start listening before the whole song is downloaded.

We plan on providing the user with both a graphical (GUI) and a command line interface (CLI). The preferred option can easily be chosen when launching the client through the Client class, which is the main entry point.

On the server side, handling requests, splitting files, sending data, and managing files are the main functionalities needed. Because of this, it felt natural to split these tasks up in separate modules.

3. Concurrency models

Both the server and the client will have to deal with concurrency. The client has to be able to receive data while playing it back and the server must handle multiple requests while sending data. The server will use the actor model from Erlang, where the requests from clients will result in that the server creates an actor which controls further contact with the client. The concurrency for the client is about handling communication, building audio from the received data, playing the music and displaying all of this at the same time. This will be dealt with using Java Threads.

The actor model is an ideal tool in Erlang to enforce concurrency, this is because the actors are rather independent and have great communication methods. Actors will start the contact with each client that has sent a request and can then work individually to get the data that was asked for.

When coding concurrent programs in Java the only way to go is using Java Threads. Java is not chosen because its concurrency tools but for the advantages it provides to build a client application that is platform independent.

4. Development tools

The following list explains what tools we plan to use during the project.

- **Planning and communication:** We will use Trello to structure the project and keep track of the progress, as well as writing down notes.
- **Source code editor:** Mainly Emacs because we are all familiar with it from previous courses, but for the Java bit we might use Eclipse to ease the development process.
- **Revision control and source code management:** Git, with a hosted repository on GitHub.
- **Build tool:** Make will be used to build our files.
- **Unit testing:** EUnit and JUnit.
- **Documentation generation:** We will use EDoc and Javadoc to automatically generate documentation. We have used them before and know how to use them.

5. Process evaluation

We settled on the idea quite early in the brainstorming process, because we all thought it would be exciting and useful to know more about the different parts involved in the project. Ultimately, our group had no problems working together, as we let everyone express their opinion. We all agreed that during the coding sessions we should be together while the research could be done apart.