Project Cons-Air booking system:
Motherforking Threads On This Motherforking Plane

Operating systems and process-oriented programming (IDT096)

Project proposal for group 04

Andreas Rubensson 940722-6019
Carl Wingårdh 930126-4694
Erik Österberg 930220-5571
Lucas Arnström 920524-0857
Oskar Ahlberg 841012-1498
Wen Ting Jin 870119-1804

Version 1

April 19, 2015

Contents

1 Introduction

At the time of this writing there are 8212 airplanes in the sky according to `http://www.planefinder.net`. There are thousands of airports around the world endlessly shuffling passengers in and out of planes to send people where they need to be. Back in the day the airports would only have to worry about the passengers queuing at their check in terminals, however, nowadays in the age of the internet there can be several people booking the exact same flight at the exact same time.
Our project will investigate a solution to the potential problems that this huge load can cause. We will create a booking system which people all over the world can access and book concurrently.
During this project we will assume the role of system designers with the airline Cons-Air as our client. Our creation will be a proof-of-concept with both a solution for booking and a simulation which will prove its robustness.

2 Concurrency models and programming languages

Our project will contain two different languages, Erlang and Java.

2.1 Erlang and the Actor Model

We will use Erlang as the foundation to our main concurrency solution. The actor model which Erlang uses is the concept that every part of a program is an "actor" which can communicate with other actors, react to messages received, but most importantly, act concurrently with other actors. We will have two modules written in Erlang: the back-end and the simulation.

2.1.1 Back-End

For the back-end we will need to spawn actors which will take care of each booking request. We will need for the actors which are working on the same flight to communicate with each other to prevent double booking and data races.

2.1.2 Simulation

The simulation will implement actors in a way where every actor represents a person. Each person will have a set of needs and wants which will determine how it will book their flight. By using the actor model for the simulation we can overload the back-end without having to worry about the simulation's main function becoming a bottleneck.

2.2 Java and Multithreading

When creating a graphical interface it is important for it to flow efficiently and not have to wait for inputs to update. We want to show real time statistics in the interface, and for this to be handled smoothly we will have to use threads. Our GUI will be designed with both the end-user and the developer in mind. We will have two separate sections, one which will allow the user to book and see real time information about their flight and one section which gives real time statistics about both the back-end's performance and the simulation's data.

3 System architecture

Our project is split into two main parts: The Real World Solution and the Real World Simulation.

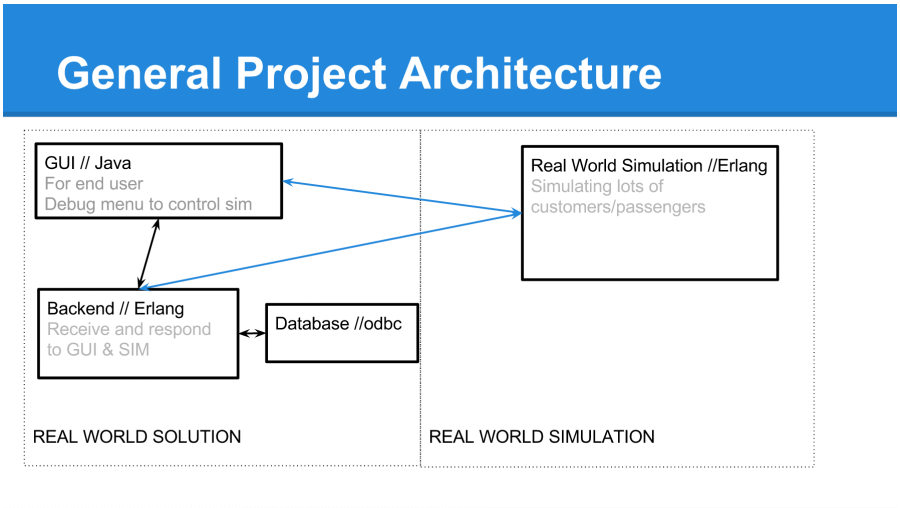

Figure 1: General description of the system modules and how they communicate with each other

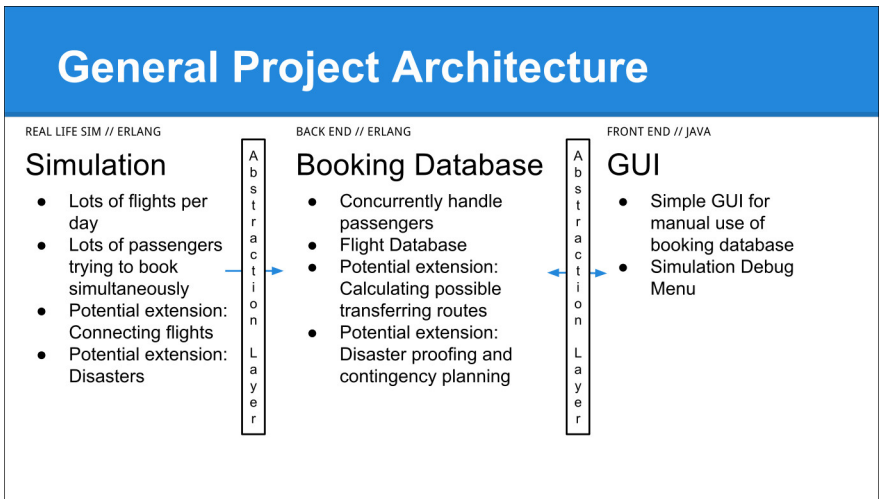

Figure 2: The functionalities of the modules in the system design

3.1 Real World Solution

This module will handle the main simulation of the volume of passengers, and we will use this as well as the main test for the whole system, ea. a system wide test. We do not have a in depth design for this module yet due to the design will progress, with the development of the back-end.

3.1.1 Back-end

As seen in Figure 2, the back-end module of the program will spawn and maintain all processes i.e a passenger, it will also handle some auxiliary processes to maintain connections to the Database as well the connections to the GUI. In the back-end the special case passengers will be handled as separate cases due to the nature of the special case. The global updates of bookings and flights is the main function of this module.

3.1.2 Database

The database module will scope and depth will be design in parallel whit the functionality of the main module (Back end) due to the abstract layers between the modules is to be implemented. The most likely be an odbc driver, and probably some SQL code is needed to sort data.

3.1.3 GUI

The GUI is the control part of the program. This is the Front-end of the program and the part that a passenger will see, we have an rudimentary plan of this interface. As well we are going to use the GUI to control the simulation of the program. We have some design for this as well.

3.2 Real World Simulation

The problem that our project aims to solve includes a large number of people using the service concurrently and because our group only consists of 6 people we need to simulate a number of people who will be using this service concurrently.

4 Development tools

- **Source code editor:** Emacs, Vim and the individually preferred Java IDE
  We will use Emacs and Vim for coding Erlang and Make, and we will use the personal preferred IDE for coding Java. We are using Emacs and Vim mainly because we are used to work with them when coding Erlang programs and Makefiles and IDE's because those are making Java coding more easy and efficient.

- **Revision control and source code management:** Git and Github
  We will use Git together with the Github repository that was opened for us. We will try to use Git as professionally as we can, making use of branches and merges and most importantly: informative commit messages. We have found GIT to be very useful in earlier projects and we are used to work with it.

- **Build tool:** Make
  We will use Make to build our program. Make will be the easiest way to combine Erlang and Java compilation, tests and documentation generation.

- **Unit testing:** Eunit and jUnit as well as our simulation
  Eunit and jUnit will be used to test the inner workings of the different modules. Our simulation will be an external test which will as realistically as possible test our program to its limits.

- **Documentation generation:** Edoc and javaDoc
  The built in documentation of Edoc and javaDoc will help us with our documentation. We might want to find a way to combine these to make the documentation more streamlined. However we need to research some more about Edoc and javaDoc before we can even decide if it is worth to do.

- **Communication:** Trello, Facebook Messenger
  Trello is our main organizational tool; we make use of cards to distribute tasks. Our plan is to have a lot of small tasks that we will decide on weekly meetings and of course allow for the addition of cards throughout the week. The completion of cards will give the rest of the team a good idea of how far we have gotten and how much has been done during the past week.
  Facebook is our main form of communication. It is used to discuss ideas as well as organize our daily meetings.

- **Any other tools:** Google Docs, Overleaf
  We will be using these mainly for sharing files and to write reports concurrently.

5 Conclusion

We are proposing a solution to the problem with people trying to book flights concurrently as many others. This is an example of traditional booking module which can be universally applied to any kind of booking service with limited instances and with the possibilities of special needs.
By using the internal structures of the Erlang actor model and the already existing OTP framework we are able to handle vast amount of bookings. As well as Erlang has no problem with scaling.
One of the major technical difficulties will be to manage the abstraction layers between the multilingual modules as well as countering bottlenecks between them. The system needs to be robust and reliable as well as enduring failures.
As mentioned before, we are using a multilingual solution and in this case the primary front end will be implemented in Java. We have selected to do the front end in Java because it is platform independent and for its well documented API with easily manageable threads.
The main selling point of this product is that it is a general solution for problems with fixed number of booking resources and transport resources. The scaleability and the modular approach makes it easy to adapt for different kinds of applications and markets.