A Farm Simulation using Swing and Concurrency

Filipe Pires [85122], João Alegria [85048]

Software Architecture
Department of Electronics, Telecommunications and Informatics
University of Aveiro

March 14, 2020

1 Introduction

This report aims to describe the work developed for the first assignment of the course of 'Software Architecture', explaining the overall architecture and describing its components and respective communication channels and elaborating on the adopted solutions for concurrency. We also mention how the work was distributed amongst the authors.

The Java application has the purpose of conducting harvest simulations on an agricultural farm. Along with the technical aspects of the implementation, we also elaborate on the adopted solutions for concurrency. Efforts on making the UI highly usable and the code readable and well documented are also stated here. All code developed is publicly accessible in our GitHub repository: `https://github.com/FilipePires98/AS/`.

2 The Agricultural Farm

Agricultural farms have well defined seasons where different activities must be executed to maintain the business productive. Tasks must be distributed amongst workers and quantities must be calculated and tracked for the correct functioning of the entire farm. As the whole system grows, its complexity and difficulty in management grows as well, so management tools emerge as valuable assets for farmers.

Amongst the many features of such tools, one offers a particularly interesting view of the farm as it simulates its behavior. Such simulations allow farm owners to plan harvests and test strategies to understand which offer the greatest productivity and profit. These simulator tools may be as complex as the farm itself, but allow manipulation of time and other resources without any cost. So, it is easy to understand that solutions of this nature offer value to the farming audience in general.

With this in mind, it was proposed to us to develop an agricultural farm harvest simulator with very simple features in order to apply the knowledge gained during the course. This simulator isn't meant to serve as a final product for an agricultural business, rather it should show the potential of such solutions. As it implements concurrency by design, it ensures scalability for a potential product and offers realistic aspects on the virtual farmers' behavior.

But how exactly is the system organized? There are two main entities: the Control Center (CC) and the Farm Infrastructure (FI). The CC is responsible for supervising the harvest, while the FI is the infrastructure for the agricultural harvest.

2.1 Control Center

The Control Center is where the number of farmers to be used is defined, along with their maximum speed, the amount of existing corn cobs and a special parameter called timeout. Timeout, defined in milliseconds, is a parameter used for regulation of the simulation's execution time and sets the maximum amount of time for each farmer to make a movement.

It is in the CC that users send orders for the virtual farmers to execute. The commands available are:

- Prepare - the selected farmers move to a Standing Area, ready for orders.
- Start - the actual simulation begins and farmers start moving.
- Collect - farmers collect corn cobs from the Granary (where the cobs initially are).
- Return - farmers return to the Storehouse with the collected corn cobs.
- Stop - farmers stop whatever they are doing and return to the Storehouse.
- Exit - simulation ends and the program closes.

2.2 Farm Infrastructure

The Farm Infrastructure is the part of the system that has the virtual components of the farm. It holds four sections of the farm:

- Storehouse - where farmers rest and corn cobs are stored.
- Standing Area - where selected farmers wait for further orders.
- Path - a representation of the field that farmers must cross.
- Granary - where corn cobs are temporarily stored.

FI also supports virtual farmers, which during their lives transit between the following states:

- Initial - resting (blocked) in the Storehouse.
- Prepare - ready for orders (blocked) in the Standing Area.
- Walk - moving in the Path (one by one, in the same order they entered it) towards the Granary.
- Wait to Collect - waiting for orders (blocked) to collect corn cobs in the Granary.
- Collect - collecting corn cobs from the Granary.
- Return - moving in the Path (one by one, in the same order they entered it) towards the Storehouse, with the collected cobs.
- Store - storing the collected corn cobs in the Storehouse.
- Exit - farmer kills itself.

So, as you can see, there is a direct mapping between the commands made available to users and the farmer states.

When deployed, the system offers a Graphical User Interface (GUI) for each entity. CC's UI allows user interaction and FI's UI allows visualization of the simulation in real time. We added to CC's interface a mirror of FI's to allow the possibility of both entities running in different environments far away from each other, while ensuring that the user on CC's side has knowledge of what is happening during the simulation. This and other interface-related aspects will be mentioned in greater detail further ahead.

3 System Architecture

In this chapter we focus on the implementation of each component and on the architecture of the entire solution. Here, we resort to a class diagram (see Figure 1) to visually aid the reader to understand what is here stated. We also present screenshots of the GUIs to explain how the interaction works.

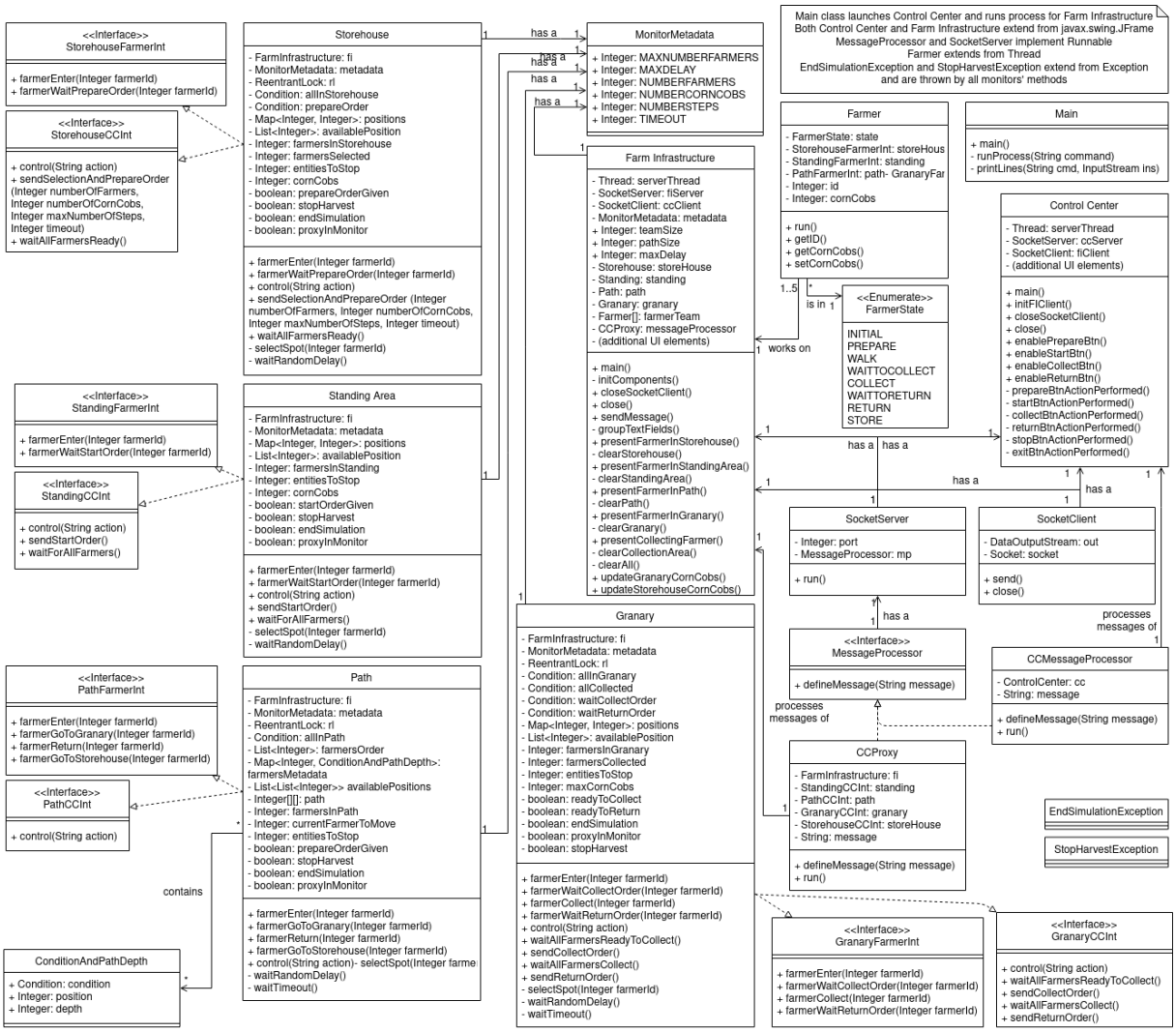

Figure 1: Class diagram of the Farm Simulation application.

Our harvest simulator follows a modular structure. As we will see, each component has a specific purpose and is responsible for one key aspect of the system's functionality. The principle of least knowledge is here applied and code dependency is kept low. This means that there are no unnecessary elements but it is needless to say that without any of the components the system will not work as intended. The system is designed to be extensible as well, through the use of well defined interfaces that allow the insertion of new elements such as new farm

areas or even different types of farmers.

All relevant actions during simulations are printed to a unique terminal, with the due identification of where it comes from, allowing an easy management of what is occurring during execution. To deal with exceptions thrown by user commands, we designed our own exception handling mechanisms. Also, since the user side is the greatest potential source of errors, the interface is confined to its most limited usage, i.e. users are only allowed to do what they can actually do at all times. Although this may seem to reduce the usability of the solution, it actually helps users by guiding them towards what they can and want to do, without leaving margin for errors.

Components belonging to the same logical layer are grouped within a package. Interfaces, classes and methods are all documented and naming conventions are kept throughout the code, focusing on allowing readers to understand what methods do and what variables hold just by reading their names. Communications between independent components are done through well defined protocols. These are mentioned further ahead, in a section dedicated to them.

3.1 Components

.....

3.2 User Interface

.....

4 Concurrency Strategy

..... the two main entities (CC and FI) are both the server and the client to each other

4.1 Communications

.....

4.2 Farmers

.....

5 Documentation

.....

6 Discussion

.....

Regarding the work distribution amongst developers, a close-contact strategy was defined where each worked on a piece of software according to a predefined plan. The project structure and architecture was decided in conjunction, as well as the key concurrency solutions chosen. Both servers were also implemented collectively.

Nevertheless, some relatively independent task distribution was defined: João implemented the Granary and Path monitors, while Filipe did the Storehouse and Standing monitors; João established socket communications and respective message processors, while Filipe designed the user interface and respective interaction with the remaining components. Bug and error solving was made along the development phase by both developers any time it was required.

Once the final version of the application was completed, this report and the code documentation became our primary concern, with both contributing equally.

7 Conclusions

After completing the assignment, we drew a few conclusions regarding the topics here explored and our endeavor to deliver work of quality.

.....

References

1. Óscar Pereira, *SA: Practical Assignment no.1*, University of Aveiro, 2019/20.