Group project report

Jacek Czyrnek, Adam Koleszar, Marion Le Guével, Mihaly Lengyel •

March 25, 2015

Contents

Chapter 1

Introduction

Optimisation process is a big part in the domain of engineering. It applies for a very wide amount of subjects, and for many, it can be a very heavy and long treatment requiering a lot of ressources. That is because of these costs, that a monitoring and recovering system is always needed if any problem could happen during the optimisation process. That way, not all the treatment is lost, and it is not necessary to restart it from the beginning.

This project report is dealing with the purpose of its realization in Chapter 1. Then, the previous researches done to define a good recovery system are presented in Chapter 2. Chapter 3 explains what the project needs to provide and Chapter 4 the choices about how to provide it. Finally, Chapter 5 shows what the project actually looks like, Chapter 6 how the testing phase where handled and Chapter 7 what problems where dealed by the team, to end with a conclusion in Chapter 8.

The aim of the project itself, is to design and developp a distributed system for monitoring the progress of the aerofoil design optimisation through an intuitive user interface with a login access. The main point is not the optimisation but the recovery system behind it.

As students working in engineering, the objective is to test all the skills needed to do a project. Indeed, not only working in the subject of optimisation or doing an actual monitoring and recovery system, but also working as a team. This means planning and organizing the tasks to design and code the project with a good communication, and finishing with a presentation of what has been done.

Chapter 2

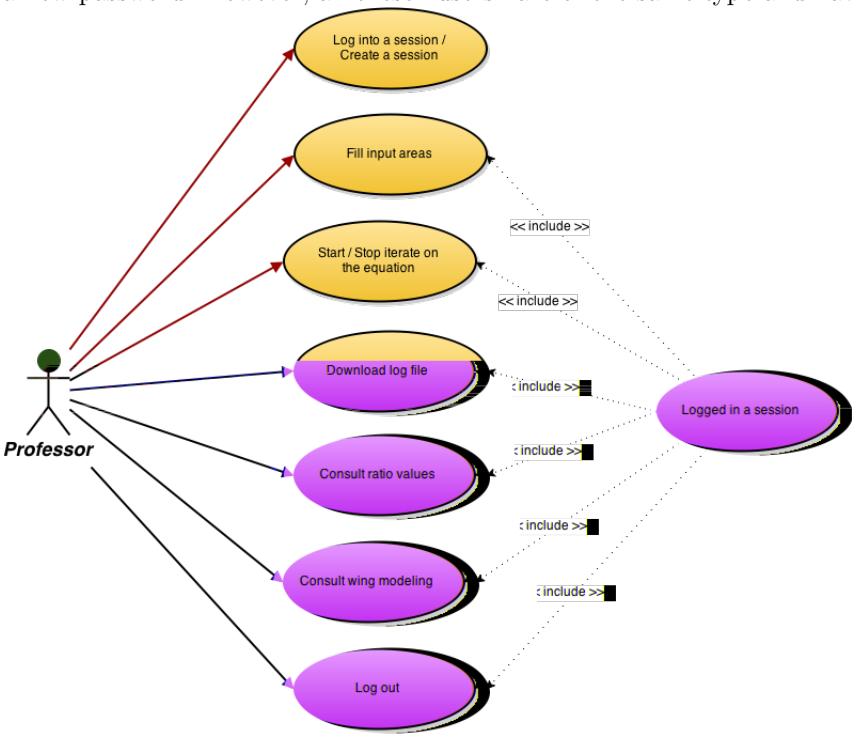
Technical review

Chapter 3

Requirements speci cations

3.1 Usecases

This project handle login as sessions. Therefore each session (a set of iterations) is a new user with a new password. However, all these "users" are of the same type and have the same functionnalities :



Mainly, the purpose of this software is to solve an equation with different given parameters and to display the result graphically.

3.2 Functional and non functional requirements

On the previous usecase, different functionalities are written. This subsection will detailed those functionalities into functional and non functional requirements.

3.2.1 Functional requirements

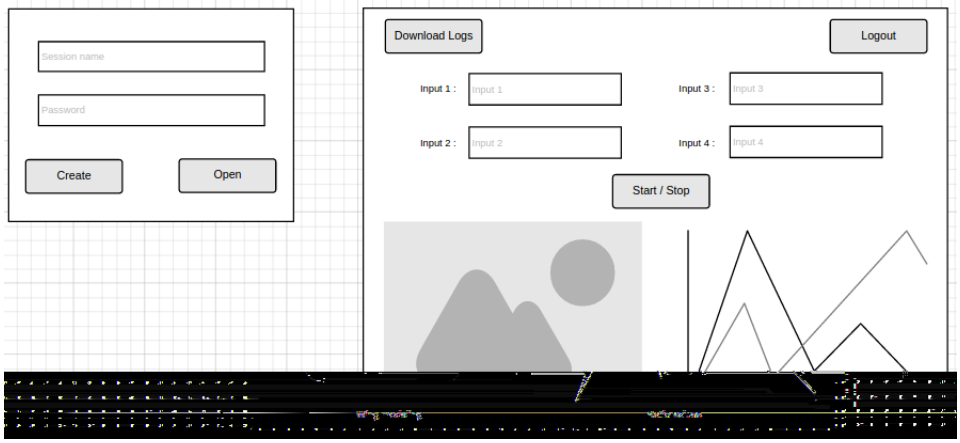
ID	Requirement
1	To log into a session
2	To create a new session
3	As logged in : To enter new inputs
4	As logged in : To start or stop iterating on the equation (exchange of data with the cluster)
5	As logged in : To display the result of the iterations in real time on a graph
6	As logged in : To display the result of the iterations in real time as a wing modeling
7	As logged in : To download the log file (exchange of data with the cluster)
8	As logged in : To logout from a session

3.2.2 Non-functional requirements

Type	Requirement
Security	To log into a session with a password so only people having those information can access to it
Reliability	To have a recovery system to access previous steps of a session
Speed	To use a cluster for solving the equation

3.3 Further information

An example has been given for the software user interface. With the functionalities required, the final product should have two "pages", one for the connexion or creation of a session, and another one once logged in a session, as the mockups below:



Chapter 4

System design

4.1 Component diagram

We began the design process by dividing the system into bigger modules and designing the interfaces between them to conduce parallel work. We split the software into 4 bigger modules and assigned one to each team member:

1. Database & Session: Marion Le Guével
2. Atlas Connector: Ádám Koleszár
3. GUI: Jacek Czyrnek
4. Wing Visualizer: Mihály Lengyel

The following diagram shows the components of the system and the interfaces they interact through.

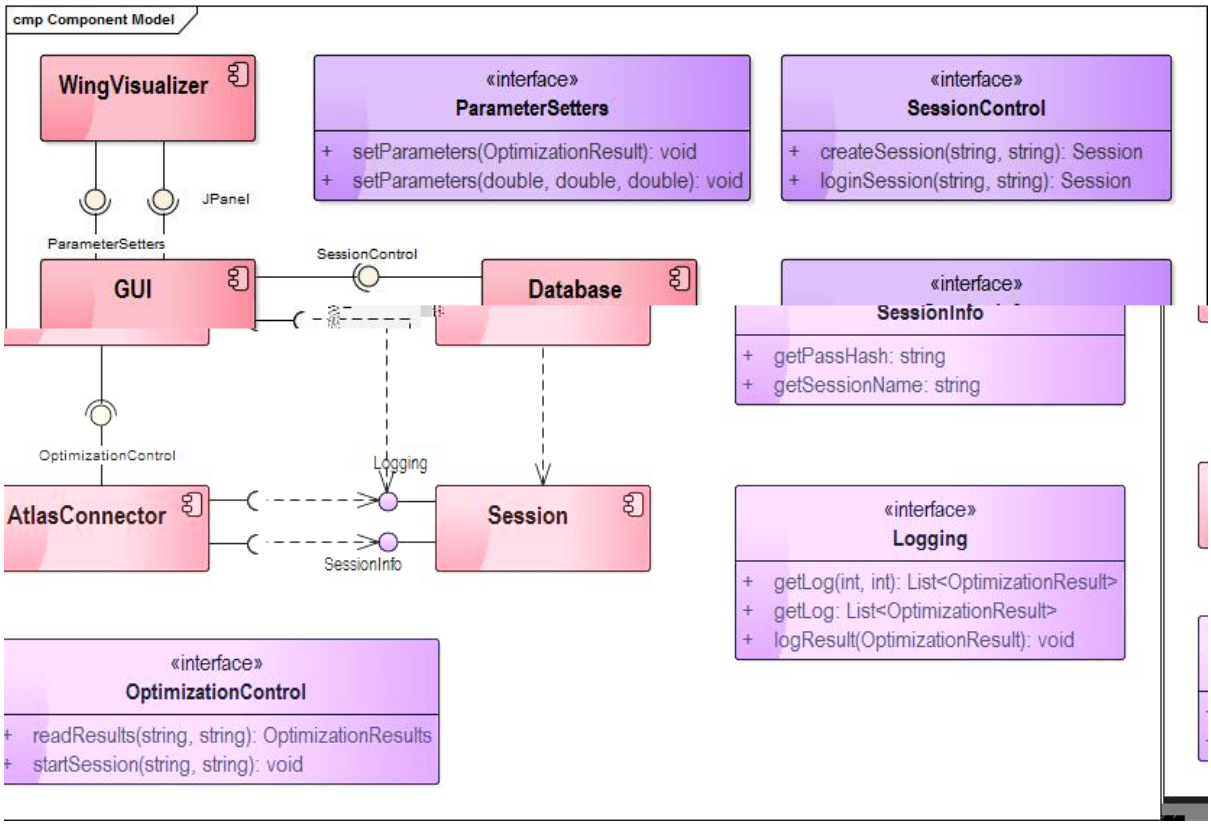


Figure 4.1: Component diagram

4.2 Class diagram

In the following diagram you can see, that the classes in the implemented system are simple reflections of the component design above.

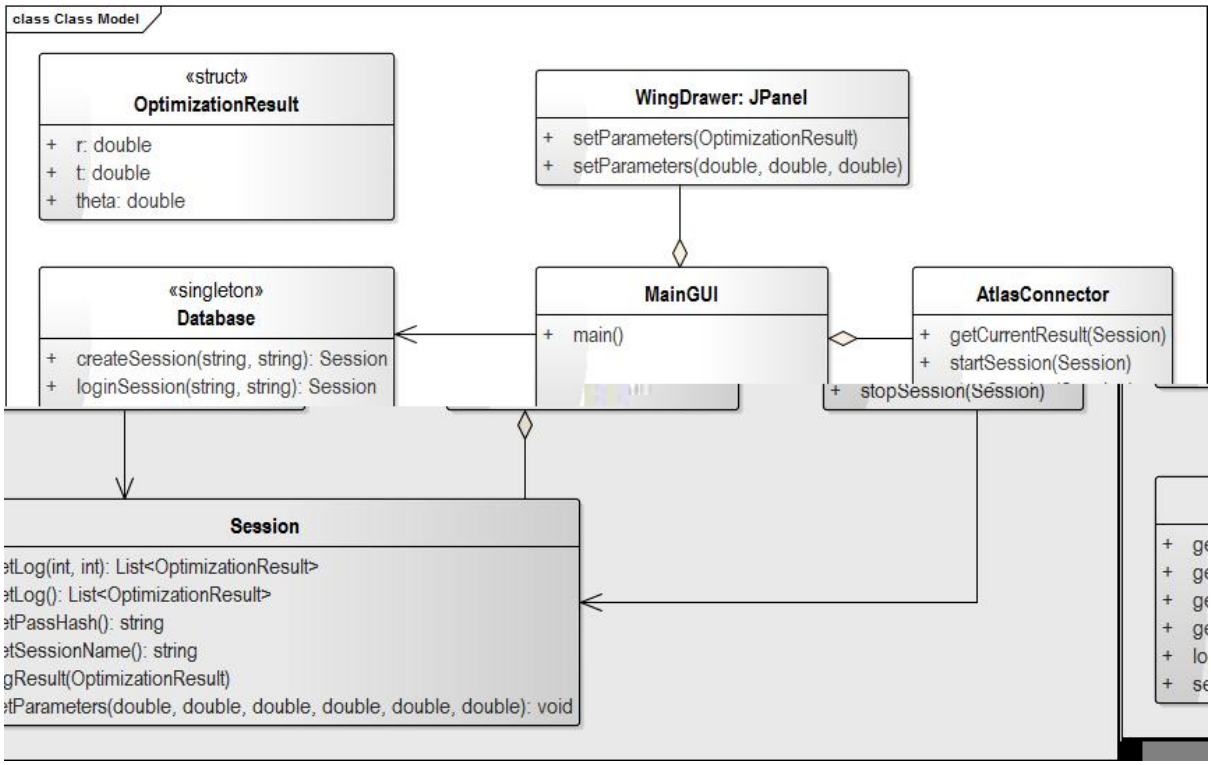


Figure 4.2: Class diagram

4.3 Sequence diagram

To complement the static diagrams above here we provide a diagram to show the dynamic behaviour of the system. This is not an exact sequence diagram but it shows the flow of data through the system.

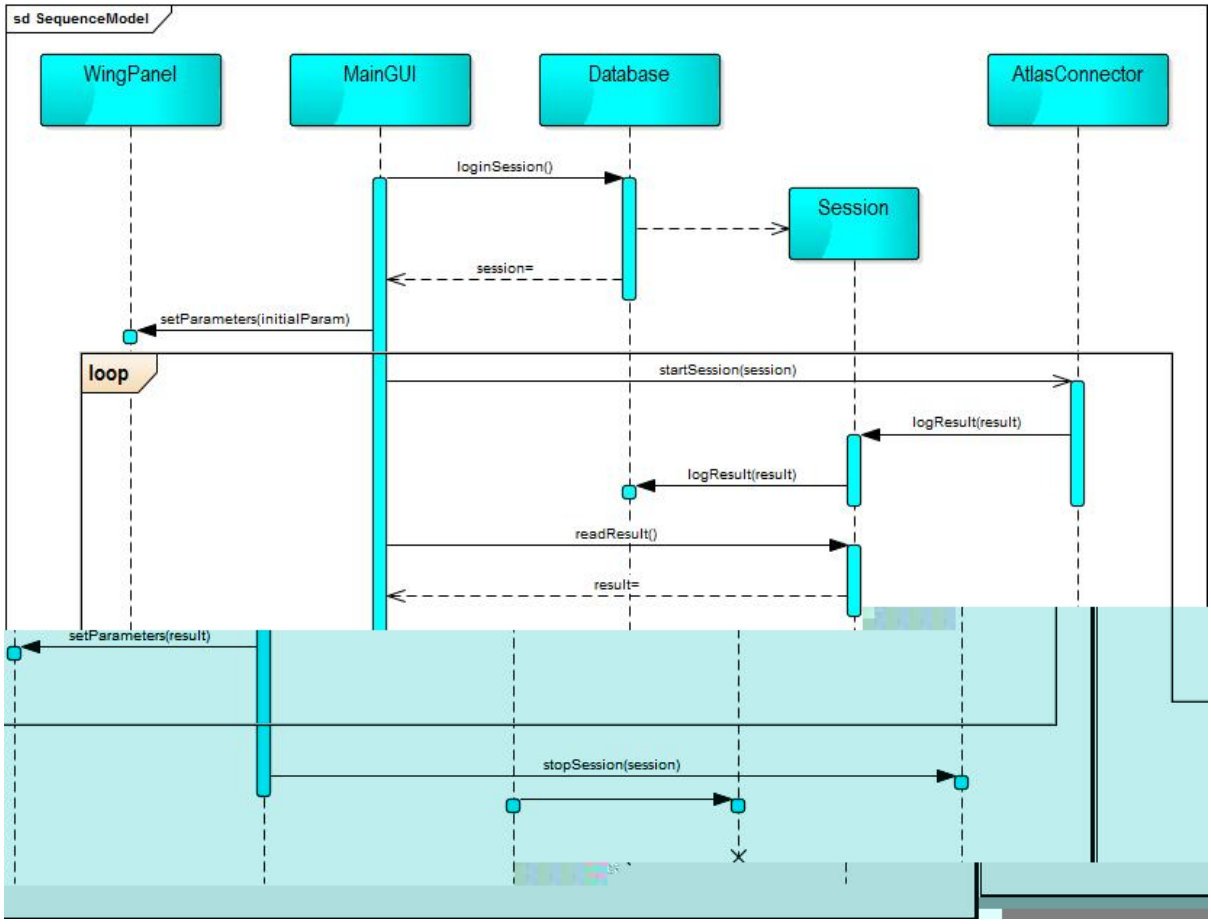


Figure 4.3: Sequence diagram

Chapter 5

Implementation

5.0.1 Database

We used the SQLite single file database solution as our database, and for it to work with our JAVA application we used the open-source SQLite JDBC driver [**?**].

5.0.2 Cluster connection

As a cluster we used the power of the Astral [1] university cluster, which can be reached via SSH protocol. In order to connect to the servers from our software we used the open-source JSch package from JCraft [**?**]. It allowed us the connect to the gate server, copy our solver on it via SCP (secured copy) and start the computation. The result are also coming back on the SSH connection as remote command execution.

For user authentication we created our own handler which uses the interfaces provided by the JSch package. It handles the password prompt and other connection configuration as well silently, so the user only need to type in his or her username and password.

5.0.3 Remote computation

Although our software contains the solver component as well, due Java version differences on the Astral server we decided to create a separate software that handles the computation and can run on the remote machines.

It is a command line application. It has several parameter options. To handle these command line parameters we used the open-source CLI commons package from Apache [**?**].

Solver

It is based on the previously mentioned NASA air foil simulation educational JAVA applet [**?**]. Because it contained computation for several other environment that is different for the normal Earth conditions we removed most irrelevant parts from it. The solver now can compute dimensionless lift and drag forces for given wing geometrics.

Optimizer

The optimizer is also part of the remote program. It takes parameter intervals for the wing geometrics and optimizes the lift drag ratio between those intervals returning the optimized wing geometrics. It also logs whenever a better solution is found.

[1] 10240 nodes

Chapter 6

Failure case studies

In this chapter we are going to write about the various failures that can occur during execution of the program and the way they can be avoided or repaired.

6.1 Communication errors

For a system that uses external connections to transfer data between components (like a network) is bound to have some communication errors in time. These can be disconnection issues, data corruption, security issues. As programmers we have to ensure that the system will properly handle these issues without too much data loss.

Failure to do so will lead to the waste of significant computation effort, which can be ultimately translated to the loss of money and time. To avoid or to be prepared for these cases the following measures were taken.

6.1.1 Disconnection

If a network connection disconnects it is only allowed to lose the data being sent, nothing more. For this, proper and thorough logging system needs to be implemented. In our case it is client and server side logging as well.

6.1.2 Server side logging

On the server side simple log files are created. Every line contains the best solution at that point in time. The solution contains the angle of attack, the camber size relative to the chord and the thickness of the wing relative to the chord. The solution also contains the computed the dimensionless values of the lift and drag forces and the ratio of the two for easier data readability. The help the finding of data on the server the log files are named after the session name.

6.1.3 Client side logging

The sessions and the computed values are also logged on the client side of the application. These data are stored in the SQLite database file attached to the program. From that any computation session and it's result can be restored and even continued if the computation didn't finish.

6.1.4 Transfer error

If the messages between the server and the client side get corrupted for any kind of reason, the messages will get discarded, and user will get a notification about possible communication error. The computation will continue and produce relevant data on the server side. If the data in between is not relevant the results can be gathered from the server log, if they are relevant as well, the computation can continue from the last correct message.

6.1.5 SSH related errors

Before the computation could start the program needs to copy the computing part of the program to the remote machine. During these communications errors can occur either with the connection itself, or with the SSH protocol. Later errors include SSH server downtime, user authentication errors etc. Dealing with these issues is outside of the scope of our program, although the program will do anything so that no data should get lost. The program will notify the users about SSH errors, and computation won't start until such issues are handled.

6.2 Database error

6.3 Software error

6.3.1 Server side error

6.3.2 Client side error

Chapter 7

Conclusion

This group project was a good exercise to make us use all the knowledge we gathered throughout the last semester, and provided a quick introduction into making a complete product through teamwork.

We used JAVA programming language and technologies to implement our solution.