Model-based Optimization and Visualization of Aircraft Noise

Emergent Architecture

2 May 2016

Contents

1 Introduction

This document presents the architecture and current state of the design of our system. It contains diagrams of the architecture design which were continuously extended throughout the sprints. The first chapter defines the design goals we considered for this project. In chapter 2 an high level overview of the designed system is given and explained. Chapter 3 contains a glossary.

2 Design Goals

When designing the product we take the following design goals in consideration:

Broad usage

We try to make the program as broad as possible for usage by the ATO research department: the ATO researchers should be able to deploy the system for both simulated and real flight routes and they should be able to deploy the different models (noise model, optimization model, visualization model) of the program separately from each other. For each model the user should also have the option to turn on (or off) particular output, e.g. the user should be able to select particular contours to be show in Google Earth in the visualization component of the program.

Modular

Our goal is to split the program into different modules. Each model (noise, optimization and visualization) corresponds to one module. The different modules must be as loosely connected as possible so that for example changes within modules would not affect the graphical user interface (GUI) and vica versa.

Quality of product

We aim for the highest quality of the product. To achieve such a high quality we are going to build a good architecture for our program so that the code is easily maintained. We write automatic Unit test cases, so that if we make an enhancement to our program it automatically checks whether we have not broken another part of the code. We aim for a minimum line coverage of 75%. We will couple the continuous integration server Jenkins to our Version Control System (VCS) in order to make sure that the code on the master branch always compiles and passes all the tests.

User friendliness

Another goal is to create an easy to use interface for the user, which is partly dependent on the overall quality of the program. We think that the modularity helps to guide the user through a work flow that works well for data analysis and visualization.

Performance

The data analysis should be performed within a reasonable period of time with a limit of ten seconds. When the results are calculated, the output should be shown directly on screen to ensure that the users don't experience long loading times. During analysis the program shows the user that it is still responding, but busy with doing a calculation. Also, a form of error management including error prevention and error correction (with error messages) is implemented in all layers of the architecture.

Scalability and flexibility

The program is able to process large sets of data with sizes in the gigabyte range and all models should respond within second(s).

Use of Design Patterns

We implemented our application following the Model-View-Presenter (MVP) pattern to separate the backend logic from the frontend that represents the program. This enables us to divide domain objects from the GUI elements to keep the code cleaner and the system more maintainable.

3 Software Architecture Views

This chapter describes what our software architecture looks like. First the subsystems are identified and explained. Then the software to hardware mapping is explained, followed by how we store persistent data. Lastly concurrency between information is explained.

3a Design Principle: Separation of Concerns

Following the design principle of Separation of Concerns for separating a computer program into distinct sections, we decided to implement the three models in such a way that they can be used as separate executables. They will be connected with each other in a pipelined manner in the main tab of the program to make them part of the workflow of the program. But these models can also be seen as standalone application: by specifying a dataset and wanted output you could also use the models outside of our program.

3b Hardware Software Mapping

Our program is designed for a single computer. The program also runs as one process. However, in future extensions we would like to make the program multi threaded. This would enable the user to run the optimization and visualization models in separate threads. This way the program will be able to optimize the entered flight trajectory while already visualizing the noise contours of the original (unoptimized) trajectory. Also reading in the files could be put into a separate thread. This means that all the identifiers that are read could be shown in the select tab, the moment the thread reads them.

3c Persistent data management

For this program it is not required to store information persistently. The output of the program is stored as a KML file for visualization. The format of the output is structured in such a way that it can directly be entered into Google Earth and different visualization objects can be turned on or off in Google Earth.

The state of the program can also be stored easily by saving the intermediate results of the pipeline. For example, the output of the noise model or the optimization model can be stored as a .csv file to be read in at a later point in time. This can be stored in a user specified directory. After the file is loaded in the program, the program will be able to continue where the user left off. This also means that changes to intermediate results, that will be entered as input into the next model in the pipeline, will not require changes to the input file you used at the beginning. More on this can be found in section 2j. External Major Technologies.

Lastly, the possibility to store the KML files for visualization also enables the user to visualize multiple flight routes in Google Earth at the same time.

3d Concurrency

This subchapter is meant to describe how concurrency issues are solved. Because it is not possible that the program is used by multiple people at the same time and the data analysis is not divided over different processes, there are no concurrency issues. However, when multiple threads are going to be used in future projects, it becomes very important to think about concurrency. This could be useful when an ATO researcher wants to read in the next flight trajectory while the previous trajectory is still being optimized or visualized in the program. Our client let us know that this is not one of his requirements.

3e   Diagram of the architecture

Here is a diagram of our layered architecture and the connection between the front-end and the back-end:
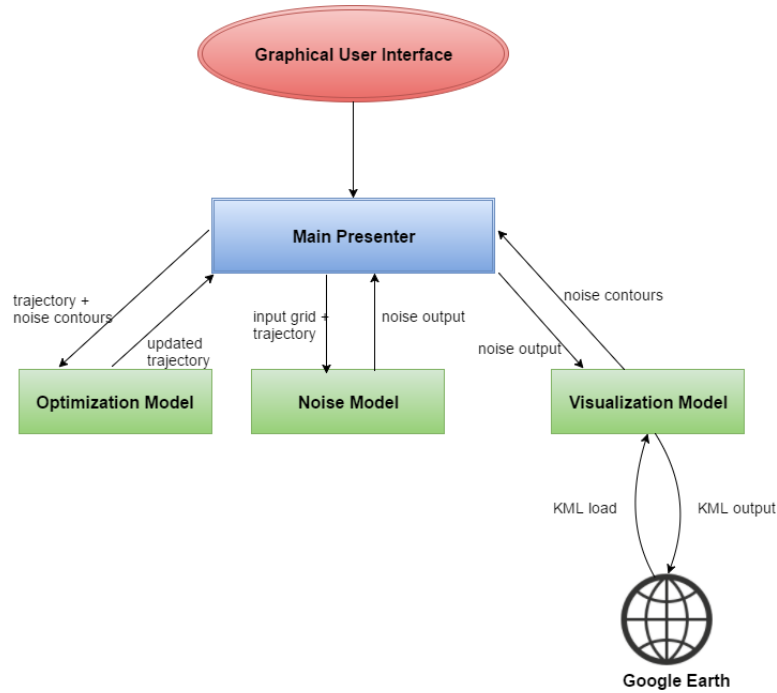
As depicted in the diagram above, the models present the subsystems described in the Subsystem Decomposition section of the first chapter. They will exchange their outputs with the main presenter which uses the output of one system as input for the next one. The main presenter relegates the output of the the noise model to the visualization model. Based on the actual noise values the visualization model constructs the noise contours. The noise contours will be transformed into KML files and entered into Google Earth for visualization. Changes to the KML files will automatically be transferred to Google Earth by a 'KML load' script, which periodically loads the files into Google Earth.

The noise contours will also be outputted tho the main presenter. The main presenter then hands the noise contours together with the current trajectory to the optimization model. The optimization model updates the trajectory to reduce the produced noise and passes the updated trajectory on to the main controller. Now the workflow of the program can start all over again by the main presenter entering the updated trajectory into the noise model.

Following this method the models are executed in a pipelined manner. But given the right input files, the subsystems can also be run separately from each other and function as standalone applications.

4   Major external technologies

Model View Controller

We decided to implement our project following the Model View Presenter (MVP) pattern.