COMP.SE.110 [2022-2023] – Group 6

Software Design Documentation

Introduction

The purpose of this document is to describe how our group has planned to implement the group assignment application. The document will go through requirements, the structure and most important components and interfaces of the software as well as the reasonings for our decisions. A prototype of the application will be submitted alongside the document.

Table of contents

Prototype

1. Requirements

Visualization window

- Timeline
- Coordinates
- Traffic data type
  - Maintenance
  - Road condition forecast *(of 2, 4, 6 or 12 hours)*
  - Messages
  - Select items of interest
    - Visibility
    - Friction
    - Precipitation
    - Winter slipperiness
    - Overall road condition
- Weather conditions
  - must be able to see at least the following and combine them in some way
    - Temperature
    - Obs. Wind
    - Obs. Cloudiness
    - Pred. Wind
    - Pred. Temperature
  - must be able to request calculations and visualization on average daily temperature at certain location in certain month
  - must be able to request calculations and visualization on maximum and minimum daily temperature at certain location in certain month
- Adjust the parameters of the visualization
  - Timeline
  - Location
  - Selected weather information
- Road information integration
  - Must be able to combine weather information with:
    - Road maintenance data
    - Road condition forecast information
  - Data queries done with coordinates *(minimum of 5 predefined locations OR give free coordinates)*
  - User must give a timeline *(in a user-friendly environment)*
- Ability to save data sets and produce visualizations, capability to compare current and saved data sets
- Ability to save preferences for producing visualizations and fetching those preferences will produce a visualization using the most recent data with the given parameters
- The design must be such that further data sources, e.g., Statistics Finland or additional data from existing sources could be easily added

Bonus

- Ability to save visualisations as images
- Ability to choose from several plotting options
- Addition of a third data source *(with meaning)*

2. User Interface

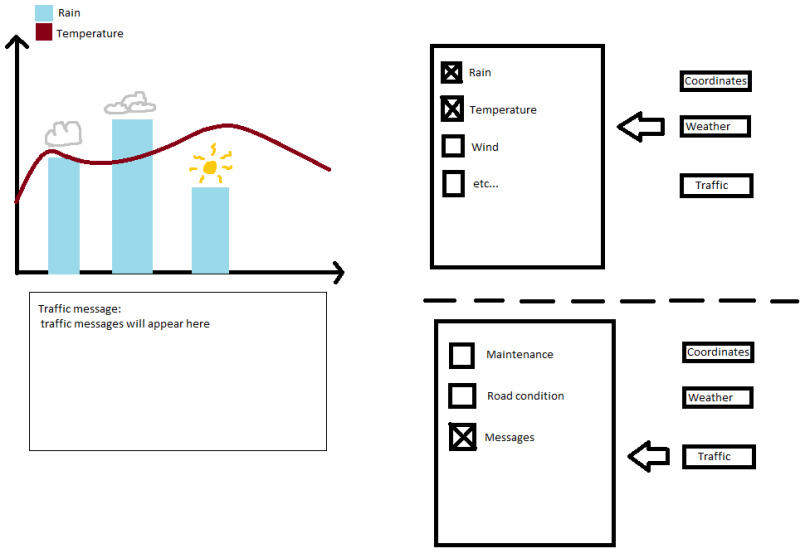


Figure 1: Sketch of data visualization implementation

The user interface consists of the visualization graph on the left side and different menus on the right side of the interface. The menus are hidden before the user clicks on them. From the menus user can modify what kind of data is displayed on the graph.

3. Reasonings

We went with a streamlined approach for the application. We tried to approach the requirements with the mindset of trying to fulfil at least the wanted base requirements. We did our platform decisions based on past experiences, such as choosing Java and JavaFX, as we felt that we could perform in the most optimal way with these tools when dealing with graphical user interfaces. The goal was and is to deliver the requirements in an optimal, timely manner. Our design choices were also inspired by our earlier studies, going for a one main viewport, where we then can display all the needed data in intuitive and compact manner, while avoiding overly complex menu and submenu structures. Everything is clearly labelled, and the user has control over what he/she wants to visualize.

4. High level description

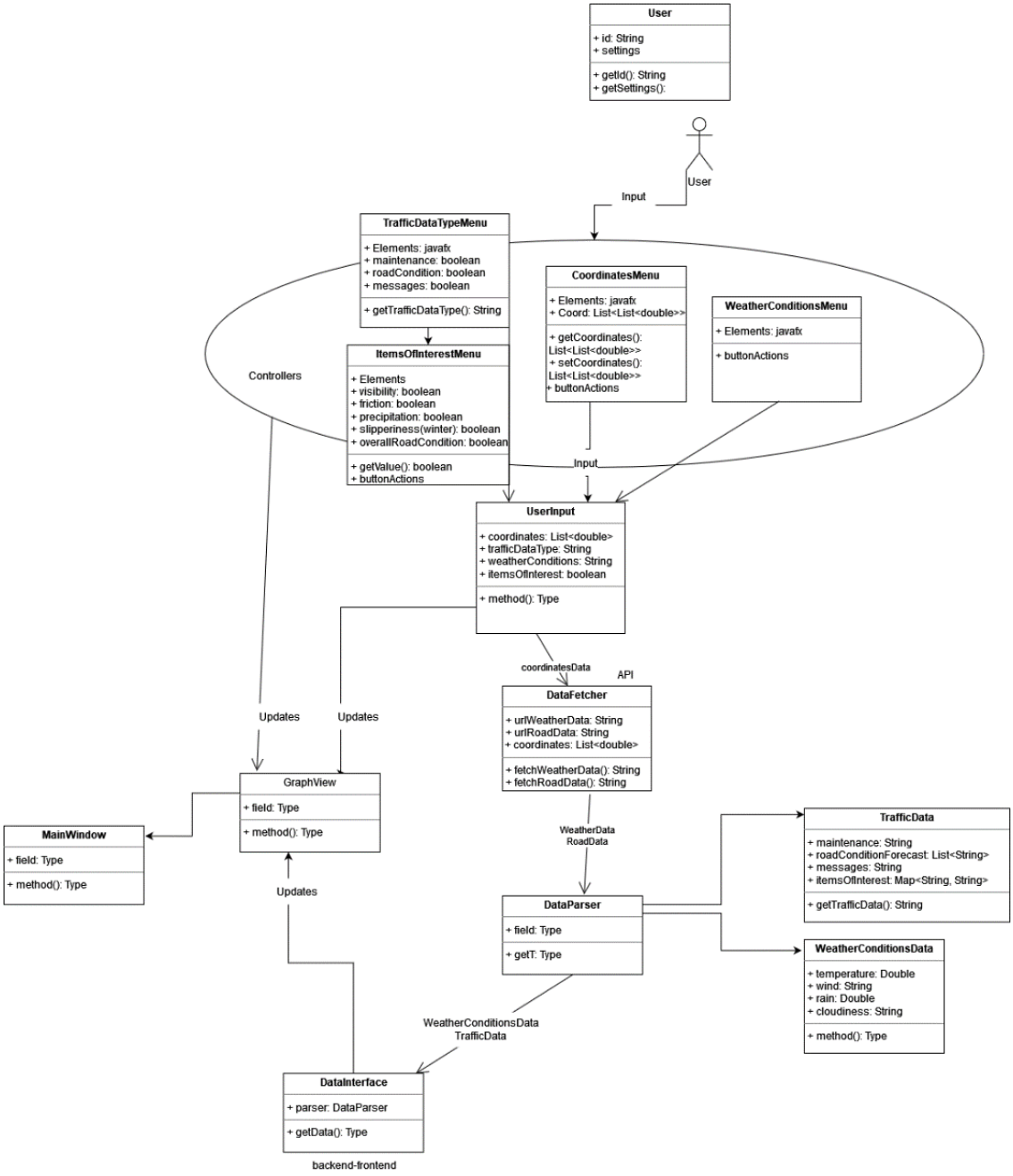


Figure 2. UML diagram of the application.

ClassDiagram.drawio - Google Drive

5. Boundries and interfaces

MainWindow

Is the main highest level-window, where our graphics are rendered in, this is the view that users will be seeing as the ‘app’.

GraphView

Is responsible for generating our graphs onto MainWindow from the gained data, gets UserInput boolean values to decide which things to display/render.

DataInterface

Is the end-receiver of all data, that will be then utilized in the visualization.

DataParser

Parses data when need be. Outputs (through accessors) two JSON trees of TrafficData and WeatherConditionsData and then relays both to DataInterface.

TrafficData

Fetched (and parsed) traffic data in JSON.

WeatherConditionsData

Fetched (and parsed) weather data in JSON.

DataFetcher

Fetches the data from the given APIs.

UserInput

Relays all the user-based data from menu interactions.

User

Stores all relevant data related to user.

Menus inside the app in general have the same interaction of filtering the data that user wishes for. The menus are categorised into two larger menus, where you can find all the fetched weather condition and traffic data items of interest. All of the data fetched is represented in menus as checkbox -esque items, where from you can pick and choose the ones you wish to see visualized on the viewport.

TrafficDataTypeMenu

Responsible for shown traffic data and its filtration, user chooses which data he/she wishes to see or not see with boolean variables.

ItemsOfInterestMenu

Submenu of TrafficDataTypeMenu, handles filtration with items of interest items, user chooses which items he/she wishes to see or not see with boolean variables.

CoordinatesMenu

Coordinates is the base data we want and need for fetching (specific) weather data for a certain area from the API. At base level the coordinates menu will be handling at least the pre-defined 5 locations of a list, the chosen from coordinates menu interaction are utilized in queries.

WeatherConditionsMenu

Responsible for shown weather conditions data and its filtration. User chooses which data he/she wishes to see or not see with the related boolean variables.

6. Timeline

6.1 Prototype

Week 0 [5.9. – 9.9.]

- Group formation
- Requirements definition
- Start of Figma sketch prototype

Week 1 [12 – 16.9.]

- Figma sketch of the general UI

Week 2 [19. – 23.9.]

- Figma sketches of separate UI (requirement) scenes
- Start of documenting design choices

Week 3 [26. – 30.9.]

- Completed Figma sketch (our prototype)
- Completed design document

Week 4 [3. – 7.10.]

- TA prototype meeting

6.2 Midterm

Week 5 [10. – 14.10.]

- Start of software implementation process
    - Start development of the functional features that utilizes one or both of the API

Week 6 [17 – 21.10.]

- Continue implementation of the functional features
- Reflection on done features, do they meet requirements?
    - Start updating the design document on this matter

Week 7 [24 – 28.10.]

- Continue
    - implementation
    - reflection
- Turn in software and updated document

Week 8 [31.10. – 4.11.]

- TA mid-term meeting

6.3 Final

Week 9 [7. – 11.11.]

- Continue implementation

Week 10 [14. – 18.11.]

- Continue implementation

Week 11 [21. – 25.11.]

- Continue implementation

Week 12 [28.11. – 2.12.]

- Final submission deadline (2.12.)
    - *Probably will extend to Sunday (4.12.)*

Week 13 [28.11. – 2.12.]

- Final TA meeting (5 - 9.12.)

7. Prototype

The prototype is also accessible from our group's GitLab.

https://www.figma.com/file/MgLkycQuzlvqfoukqI4FAX/First-draft?node-id=39%3A252

8. Group members

Arttu Lehtola – arttu.lehtola@tuni.fi

Vilma Lahti – vilma.s.lahti@tuni.fi

Mikko Moisio – mikko.moisio@tuni.fi

Aleksi Hasu – aleksi.hasu@tuni.fi

Mid-Term

1. High level description

We chose to use the MVC model and here is our [UML diagram.]

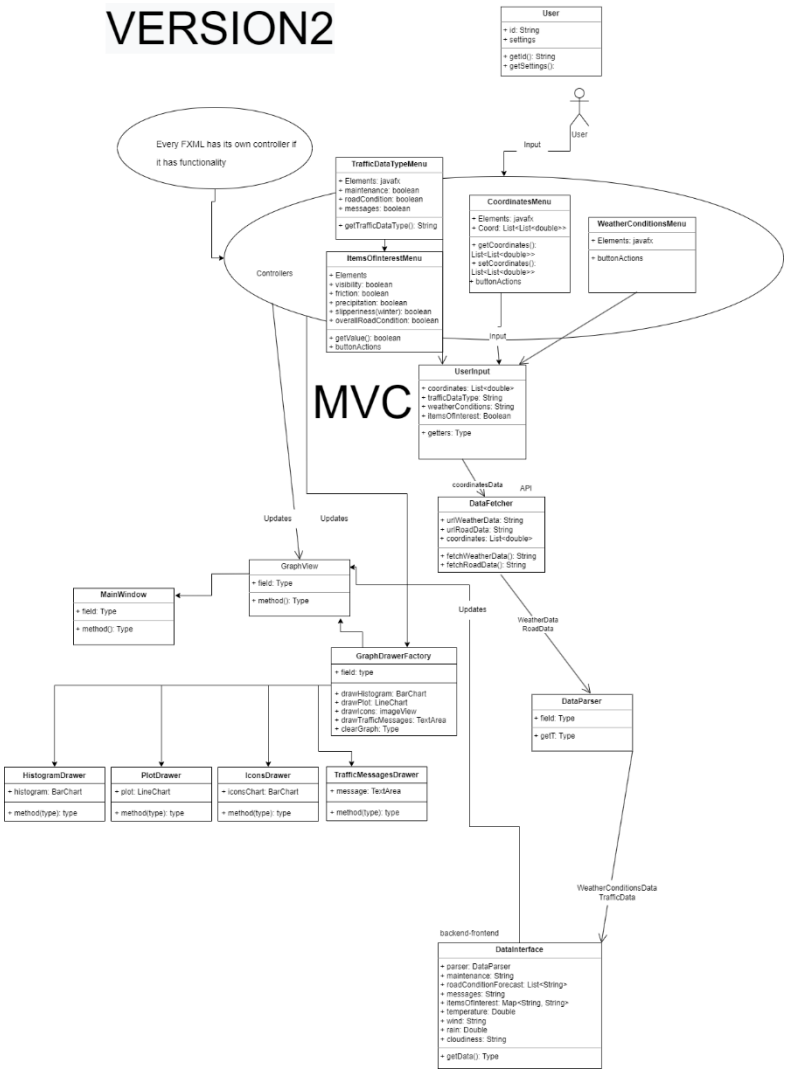


Figure 3 Updated UML diagram

We used SceneBuilder to design our UI. Icons for cloudiness are acquired from flaticon.com. Our code utilizes open libraries like Gson, json and xml.

2. Boundaries and interfaces

UserInput gets data from menu controllers according to what User selects from the menus. DataFetcher then fetches the data that UserInput quires from the APIs. DataParser parses the fetched data if needed and then sends it to DataInterface. GraphView is updated by both DataInterface and menu controllers. Menu controllers also update GraphDrawerFactory which in turn tells Drawers what items User wants to display. 9

GraphDrawerFactory updates into GraphView the content that drawers produce. Finally, MainWindow gets the data what to display from GraphView and it is shown to the User.

3. Describing components and responsibilities

Menus inside the app in general have the same interaction of filtering the data that user wishes for. The menus are categorised into two larger menus, where you can find all the fetched weather condition and traffic data items of interest. All the fetched data is represented in menus as checkbox-esque items, where from you can pick and choose the ones you wish to see visualized on the viewport.

DataFetcher

Fetches data from the given APIs.

DataInterface

The end receiver of all data that will be utilized in the visualization.

DataParser

Parses data when it is needed. Outputs (through accessors) two JSON trees of TrafficData and WeatherConditionsData and relays the data to DataInterface.

GraphDrawerFactory

Responsible for displaying all the graphs.

HistogramDrawer

Responsible for the rain histogram.

IconsDrawer

Responsible for icons for cloudiness etc.

PlotDrawer

Responsible for the temperature plot.

TrafficMessageDrawer

Responsible for traffic messages.

GraphView

Generates graphs onto the MainWindow from the gained data, gets UserInput boolean values to decide what will be displayed/rendered.

MainWindow

Is the highest-level window, where graphics are displayed in. This is the view that the user will be seeing as the “app”.

Menus

CoordinatesMenu

Coordinates are used as the base data for fetching the weather data for a certain location from the API. At base level the coordinates menu will be handling at least the pre-defined 5 locations from a list. The coordinates inputted into the coordinates menu will be utilized in queries.

ItemsOfInterestMenu

Submenu of TrafficDataTypeMenu. User can choose which items they wish to filter out with boolean values.

TrafficDataTypeMenu

Responsible for shown traffic data and its filtration. User can choose which data they wish to filter out with boolean values.

WeatherConditionsMenu

Responsible for show weather conditions data and its filtration. User can choose which data they wish to filter out with boolean values.

User

Stores data related to the user.

UserInput

Relays all the user-based data from menu interactions.

4. Self-evaluation

Our design has supported the implementation well. Having designed most of the project beforehand has made the implementation much easier. We feel that the design will also support making the remaining functionalities well. We will hopefully be able to follow our updated UML diagram well. We have been able to stick to our original plan mostly. TrafficData and WeatherConditionsData were combined into the DataInterface and drawers for the plot, histogram, icons, and traffic messages were added. We have been able to implement features based on our original plan. We made each FXML that has functionality their own controller to improve the quality of the code. There is some repetition on the code at this point, but we are planning to improve this when implementing the remaining features.

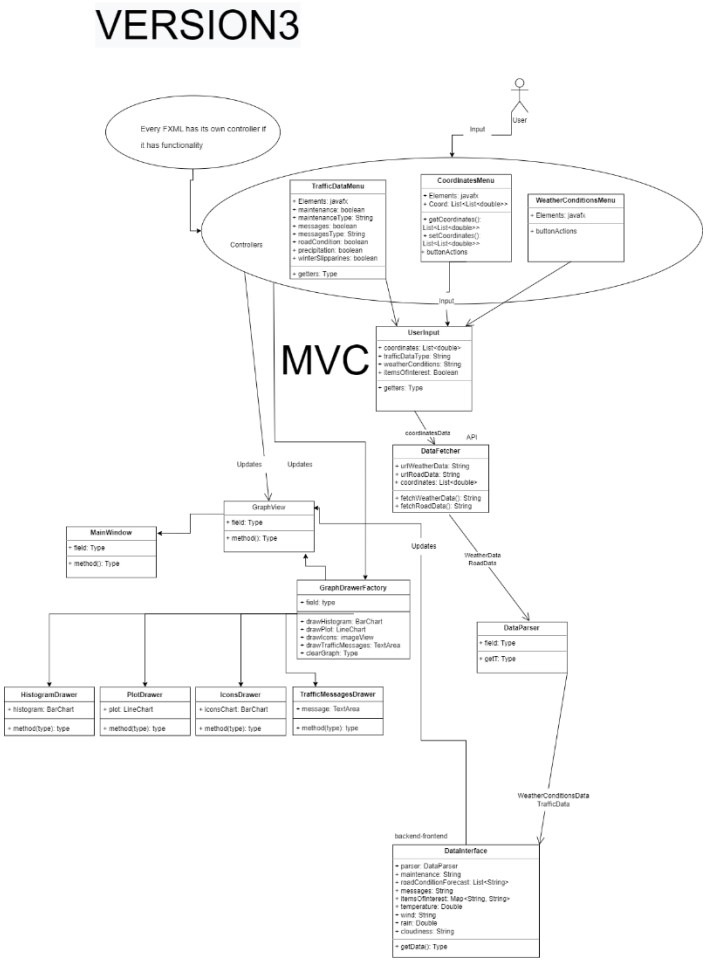
5. Reasonings

We wanted to make the UI as simple as possible, so it's more intuitive for use. We chose to do this by creating a side menu where user can select the items they wish to see, for more tailored experience. Temperature graph will be displayed as a line chart and rain as a bar chart, since we felt that this would be the clearest way to present both information in the same graph view without making it look too crowded and hard to understand. We chose to use MVC model since we felt that fit best to our design. We combined TrafficData and WeatherConditionData into DataInterface from our previous plan because we thought it would be clearer to have all the data in the same place.

Final version

1. High level description

Our UML diagram is displayed below.



We used SceneBuilder to design our UI. Icons for cloudiness are acquired from flaticon.com. Our code utilizes open libraries like Gson, json and xml.

2. Boundaries and interfaces

Controllers get data from the UI according to what the User selects from the menus. DataFetcher then fetches the data for the coordinates that User has selected from the APIs. DataParser parses the fetched data if needed and then sends it to DataInterface. GraphView is updated by both DataInterface and menu controllers. Menu controllers also update GraphDrawerFactory which in turn tells Drawers what items User wants to display. GraphDrawerFactory updates into GraphView the content that drawers produce. Finally, MainWindow gets the data what to display from GraphView and it is shown to the User.

3. Description of components and responsibilities

Menus inside the app in general have the same interaction of filtering the data that user wishes for. The menus are categorised into three larger menus, where you can find all the fetched weather condition and traffic data items of interest as well as a menu for setting the coordinates where the data will be fetched from. All the fetched data is represented in menus as checkbox-esque items, where from you can pick and choose the ones you wish to see visualized on the viewport.

DataFetcher

Calls the weather and road condition fetchers.

WeatherDataApiFetcher

Fetches the weather data from FMI API.

RoadDataApiFetcher

Fetches the road condition data.

DataInterface

The end receiver of all data that will be utilized in the visualization.

PreferenceLoader / DataLoader

Loads saved preferences / datasets.

PreferenceSaver / DataSaver

Saves preferences / datasets.

GraphDrawerFactory

Responsible for drawing some of the graphs.

HistogramDrawer

Responsible for the rain histogram.

IconsDrawer

Responsible for icons for cloudiness as well as wind speed.

PlotDrawer

Responsible for the temperature plot.

PieDrawer

Creates a pie chart for today's tasks.

GraphView

Generates graphs onto the MainWindow from the gained data, gets UserInput boolean values to decide what will be displayed/rendered.

JsonParsing

Parses data when it is needed. Outputs (through accessors) two JSON trees of TrafficData and WeatherConditionsData and relays the data to DataInterface.

LoadingScreen

Creates an animation when the program is fetching data.

MainView

Is the highest-level window, where graphics are displayed in. This is the view that the user will be seeing as the “app”. Users inputs their choices, and they are received from the MainView.

Menus

CoordinatesMenu

Coordinates are used as the base data for fetching the weather data for a certain location from the API. At base level the coordinates menu will be handling at least the pre-defined 5 locations from a list. The coordinates input into the coordinates menu will be utilized in queries.

TrafficDataTypeMenu

Responsible for shown traffic data and its filtration. User can choose which data they wish to filter out with boolean values.

WeatherConditionsMenu

Responsible for showing weather conditions data and its filtration. User can choose which data they wish to filter out with boolean values.

TrafficMessageDrawer

Outputs a count of all selected traffic messages, it also outputs a string of road condition data for use in GUI.

4. Desing decisions

We wanted to make the design as simple as possible. Thus, we decided to make a side menu where user can input the coordinates for the location that they want to view or select from pre-set coordinates for certain locations. Sidebar menu also includes two menus where user can select which items, they want to see in the graph view, traffic and weather menus. All wanted data and graphs are displayed in the graph view tab. Preferences and datasets can be saved and loaded from the settings tab.

5. Self evaluation

We have mostly been able to stick to our original MVC design. The chosen design worked fine for our implementation. We've been able to implement every major functionality as their own respective class to make the project structure clear and the code clean.

We made some changes to the plan from the mid-term.

We added loader and saver for preferences and dataset so user can use the same parameter as before. The ItemsOfInterestMenu was merged with the TrafficData menu to make thing easier to find. We also added a simple loading screen when the program is fetching data as to not confuse the user.