Tampereen yliopisto

Vitosta vaille valmis

Pietari Pakarinen
Matti Hoivala
Janne Vikelä
Toni Grönberg

SOFTWARE DESIGN DOCUMENT

Table of Contents

Version number	Who changed	What changed	Date
0.1	Pietari	Document created	25.01.2022
0.2	Everyone	Wrote the initial draft of the document	01.02.2022
0.3	Janne	Component diagram and their descriptions added	04.02.2022
0.4	Pietari	Wrote chapter 3 describing the components and their responsibilities	08.02.2022
1.0	Everyone	Finishing touches. Ready for submission	15.02.2022
1.1	Matti	Modified chapter 6 and made changes to chapter 5 based on that.	21.02.2022
1.2	Janne	Model updated	18.3.2022
1.3	Pietari	Updated chapter 8 refactoring and added chapter 9 self-evaluation	19.3.2022
1.3.5	Toni	Updated chapter 7 and deleted some old info about interval tab	22.04.2022

1. Introduction

This is a design document of a software for monitoring real-time data on greenhouse gas emissions, as well as comparing the current and historical data. The software fetches this data from external data sources. More details on the used databases and API’s can be found in chapter 7.

The software visualizes the data, which is then displayed on the user interface of the software. The user of the software can manipulate the visualization, as well as what data is used to create it, using the provided controls.

The software is to be developed using Python as the programming language. Qt will work as the development environment and the UI library.

2. High level description of the design

The software is a data visualization software that displays data that is fetched from external sources. The data is fetched and plotted according to selections that the user makes on the user interface. The software does not require an account to run, and the instructions for running the software can be found in the readme.md file.

The selections the user can make include the data source used, the type of emission, and the timeframe of the data. The user can also, in addition to viewing the raw data, choose to view the minimum, maximum or average values of the selected data.

2.1 Use Case

A user story:

As a politician I can look at my local emission values, so that I can take that information into account in my decision making.

Use-case field	**Description**
Use case name	Politician creates a data visualization of their local emissions situation.
Subject area	Politics
Business event	The politician must vote on whether to give permission for a company to construct a coal plant in their city.
Actors	Politician, other politicians, voters, coal plant owners
Use case overview	The local power prices have soared recently, and the economy has been stagnating. A coal company discovered that the city was a good location for a coal plant due to the market conditions and requested permission to build. Local environmental activists are in firm opposition, and the topic is hotly debated locally. Now, the politicians must decide on whether to give the permit.
Preconditions	1. The software displays relevant data 2. The software can be used to compare current data to historical data
Termination outcome	Success: The politician is able to create the visualization they needed to make their decision. Failure: The politician failed to create a useful visualization.
Condition affecting termination outcome	Success conditions: - The software is accessible - The software is easy to use

	- The software is easy to understand - The visualizations are clear - The data is correct Failure conditions: - The software is hard to use - The data is useless
Use case description	1. The user runs the software. The software runs. 2. The user chooses both SMEAR and STATFI. The software displays the selections. 3. The user chooses to compare current year's emissions data to data from ten years ago as a scatter chart. The software visualizes the data. 4. The user saves a graph for later use. The graph is saved.
Use case associations	*A list of other use cases that are associated with this use case.*
Traceability to	*A list of other related documents, models, and products that are associated with this use case.*
Input summary	The user selects which databases to use to make the visualization. The user selects which gas emissions to display. The user selects the time periods from which the data is displayed. The user clicks a button to save the produced visualization.
Output summary	The software fetches data from STATFI and SMEAR and produces a visualization as an output.
Usability index	The way the software is used in this use case is one of the intended, most common ways to use this software. It also concerns the key features of the software, so on a scale of 1-5, the usability index is a 5.
Use case notes	*Information that is not directly part of this use case but that the solution developer needs to be aware of while working on the workflow.*

Source for the template: *https://www.ibm.com/docs/en/imdm/11.5?topic=cases-use-case-template*

3. Describing components and their responsibilities

The user interface of the software consists of three large components, their subcomponents and a few buttons.

1. The chart component
2. The sidebar component
3. The controls bar

The buttons that are outside the three main components are

1. The buttons to change between SMEAR and STATFI views
2. The buttons to save to favorites and exporting the plot

The chart component is a container that contains subcomponents

1. Chart
2. MinMaxAvg –Table
3. Chart type –selector
4. “Add historical data” -button

The sidebar component is a container that contains subcomponents

1. Favorites list
2. Search history

The controls bar is a container that contains different subcomponents depending on which view the user is in, SMEAR or STATFI. In the SMEAR –view it contains the following subcomponents

1. Stations drop down menu
2. Variables drop down menu
3. Time selections with calendar
4. Plot –button

The buttons “STATFI” and “SMEAR” allow for switching between the SMEAR and STATFI views. The different views are for fetching and viewing data from the two different sources. There’s also the possibility, that we’ll add more options for different data sources later in the project. The buttons also change the contents of the controls bar container. The SMEAR –view is explained above. In the STATFI –view, the controls bar contains the subcomponents

1. Volume selector
2. Intensity selector
3. Indexed selector
4. Intensity indexed selector
5. Calendar
6. Plot –button

Finally, the button to save to favorites saves the current visualization to the favorites –component. The export plot –button exports the current visualization.

4. Interfaces (external)

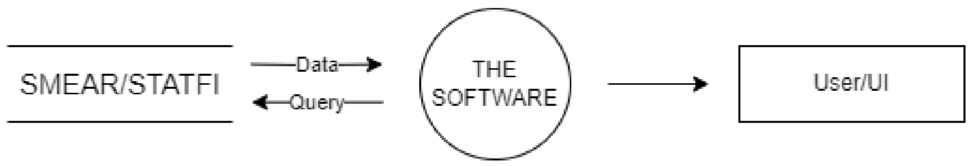

Figure 1. A data-flow diagram of the software

The user makes selections in the user interface of the software. The software makes a query to an external database(s) based on the selections. The database(s) send data to the system as a response, which the system plots into a visualization. The visualization is displayed to the user on the user interface of the software.

5. Components' internal structure and functions

Figure 2 shows the preliminary structure of the software as a UML component diagram. The components will be implemented with Qt as the development environment and using Qt's GUI libraries in implementing the graphical user interface.

One of Qt's peculiarities is that instead of traditional Model-View-Controller (MVC) design Qt uses Model-View (MV) design where the controller modules are integrated into the View components which are based on Qt graphical objects and widgets.

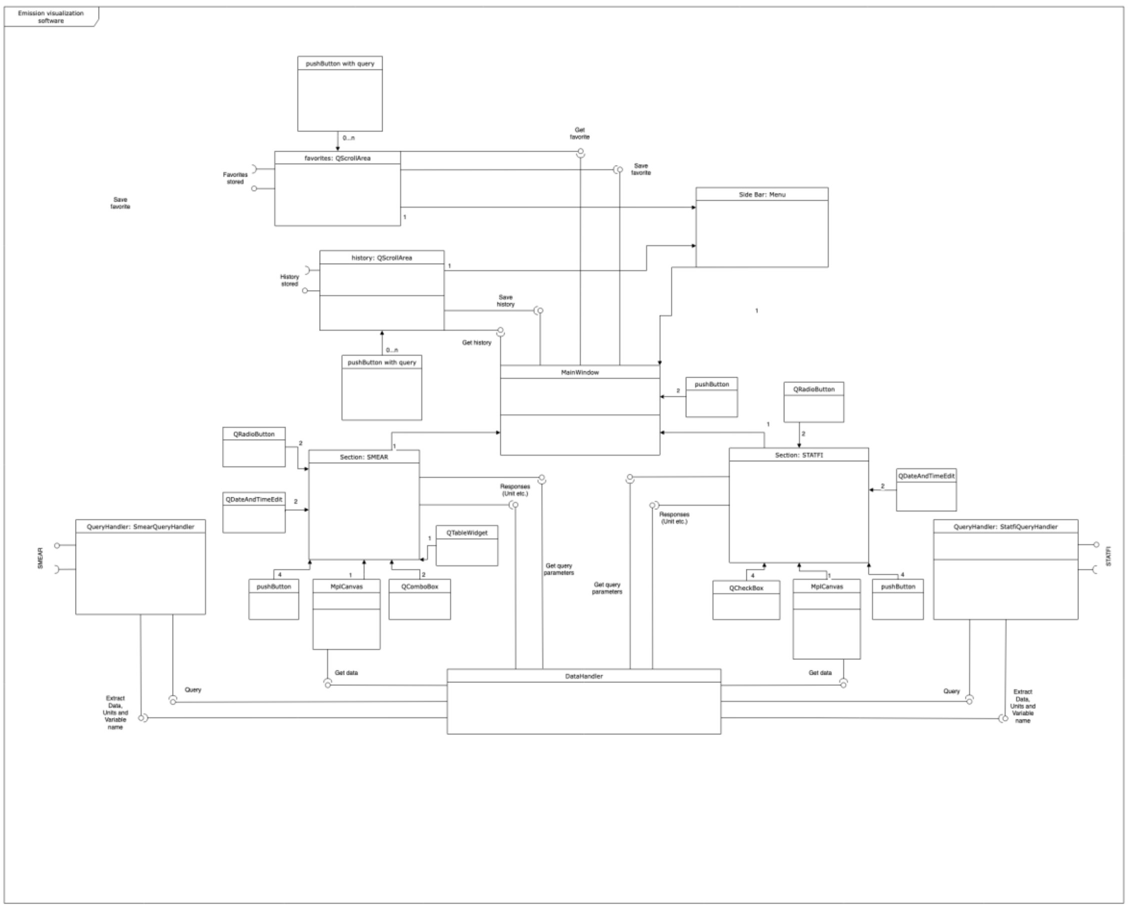

Figure 2. Component diagram in UML notation

The main components in the software are listed in the below sections.

Main window

The main window is implemented as a QMainWindow and it consists of two sections used to display charts for SMEAR and STATFI data. The sections are switched from pushButtons and shown and hidden after a select button is hit. Additionally, the main window features a sidebar including two scroll areas implemented as QScrollAreas. The first one features the favorite searches and the second one the search history (Req 5.1). The user can return a saved query from this scroll areas to the search tabs.

Sections

Two sections are implemented. Both feature query tools, a data graph, a summary statistics area and buttons to save queries to favorites or to save graphs as images (Req 1 & 4). One tab is for SMEAR and another for STATFI (Req 1.1).

Virtually, the only difference between the look of the two sections is that the SMEAR and STATFI search parameters are different. For example, the SMEAR tab allows to select from a dynamic selection of observation stations and variables and the STATFI tab has fixed parameter options.

5.1 SMEAR section

The SMEAR section features a graph used to visualize the query data. The graph is planned to be implemented using MatPlotLib (Req 1.1 - 1.3). Also, the tab features three dropdown menus implemented with QComboBoxes. Then, the time intervals are determined by QTimeAndDateEdits.

The dropdown menu QComboBoxes are used to select first, the observations station and second, the observed variables (Req 1.1 - 1.3). The dropdown menus interact with the SMEAR query handler in the following way:

1) The query handler seeks the available stations and pushes them to the tab for the dropdown menu.
2) The user selects a station, and another query is asked from the query handler for the available parameters on the station. The query handler returns the available parameters. They are pushed to the second dropdown menu.

After determining the stations and the variables with the interval, the user can input the search parameters, including the observation interval from QComboBox and start and end date from QDateAndTimeEdit (Req 1.3). Then, the tab forwards the query parameters to the query handler after plot pushButton is clicked.

3) Query handler receives the query parameters station, variable, interval and time range from the tab and makes the query.

SMEAR returns a JSON result which must be converted into a list that is fed to the MatPlotLibs in the MplCanvases. The JSON is converted into a Python dictionary and then the needed values are extracted as lists.

4) Query handler converts the JSON query result into a Python dictionary from which the needed data arrays are extracted, converted to list QBarSeries and pushed into the graph and displayed.
5) The graph made by MatPlotLib has two plotting functionalities, line chart and bar chart.
6) The query is saved to a history storage file and the query is also shown in the historical queries in the sidebar.
7) If user selects to save query to favorites, the query is saved into favorites file and shown in the favorites tab.

A method in the tab is implemented to create a summary statistic to a statistics box as created as a QTableWidget.

8) The graph is shown.
9) A summary statistics method calculates a summary statistic in the QTableWidget.
10) The user can select the chart type – line vs. bar – from QRadioButtons.
11) User can save the graph by clicking save button.

The user can save a successful query to favorites. Also, the user can save the graph. Additionally, the query is saved to the query history.

5.2 STATFI tab

STATFI tab is somewhat different from the SMEAR tab. The graph area and the summary statistics are similar. The save options are included for the graph and the query history. However, the query creation is different compared to SMEAR tab. The dynamic query menus are replaced with checkboxes created as QCheckBoxes. A similar QTimeAnd Date widget is implemented as in the SMEAR tab. The query process is as follows:

1) The user selects from four STATFI alternatives from the check boxes.
2) The user can select the chart type from QRadioButtons.
3) The user sets time start and end year in the QDateAndTime widget.
4) Query is asked from the query handler.
5) The query is saved into history and by user request into favorites.

Again, the query handler converts the resulting JSON data to a dictionary and converts the needed data into list. The MatPlotLib based MplCanvas visualizes the result. Also, a statistics method in the tab creates a summary statistic as labels.

6) Query handler returns a list containing datapoints and time as lists.

7) The data is displayed on the graph.

5.3 Adding historical data

Both SMEAR and STATFI tab have a pushButton that adds the data from the other section to the chart. In the SMEAR section, the “Add historical data” adds the data from STATFI. Then in the STATFI section, the “Add current data” adds the current data from SMEAR tab.

5.4 Sidebar

Sidebar includes two scroll areas. One is for favorites and second for history. The scroll areas made as QScrollAreas consist of customized pushButtons called pushButtonWithQuery. Clicking the push button will enter the historical query parameters to the corresponding sections.

5.5 Query handlers

The software features two query handlers. They are derived from a shared queryHandler class. One query handler is for SMEAR and the second for STATFI.

5.5.1 SMEAR query handler

The SMEAR query handler is a two-way query handler interacting with the SMEAR tab and the search history and favorites. The queries are built as follows:

1) The query handler asks a list of available stations and converts it to a form that is forwarded to the SMEAR section and the user can pick the station the user wants.
2) The query handler receives a selected station. Another query is made. It asks which variables are available at the station. Then, the result is forwarded to the tab and user can select the variable inspected.
3) The user enters query time to the section and section forwards the information into the query handler. A full data query is made. The resulting JSON is converted into a Python dictionary and then the needed values are extracted as lists.
4) The data handler also creates a SMEAR data summary statistic from the received data to a QTableWidget in the SMEAR section. The statistic is handed over through the data handler.

5.5.2 STATFI query handler

The STATFI query handler is a one-way query handler interacting with the STATFI tab. The queries are built as follows:

1) The user selects from four parameter alternatives in the STATFI tab. Also, the user enters the start and end years.
2) The STATFI tab feeds the query values into the query handler. A query is made by creating a JSON query.
3) The resulting JSON is converted to a list and given to the tab.
4) The tab displays the data on graph.

5.6 Data handler

The data handler is responsible for storing the data from the queries, the units, variable descriptions and also triggering the query handlers. Moreover, the plots request their data from the query handlers. Finally, the query handler’s task is to merge the SMEAR and STATFI time series.

5.6.1 Merging STATFI data on SMEAR data

The STATFI data is combined with the SMEAR data in a following way:

- First, a SMEAR query is made for selected station
- The station and data must be selected so that pre-2016 data is selected
- The SMEAR query is made in the SMEAR tab
- Then, in STATFI tab, the variables to merge are selected
- Add historical data button is pressed in the SMEAR tab
- The data handler extracts the SMEAR data years and queries the selected STATFI items and plots them on top of the SMEAR data

5.6.2 Merging SMEAR data on STATFI data

The SMEAR data is combined with the STATFI data in a following way:

- First, a STATFI query is made for selected variables
- Then, in SMEAR tab, a station and variable are selected
- The station and data must be selected so that pre-2016 data is available
- Add historical data button is pressed in the STATFI tab
- The data handler extracts the dates in January 1 for the STATFI data years and queries the selected SMEAR item and plots it on top of the STATFI data

6. The requirements of the software

The requirements the software needs to fulfil (as we have interpreted)

1. Data visualization
   - 1.1. User can select station(s) from which to visualize data from
   - 1.2. User can select gas(es) for which they want to visualize data for.
   - 1.3. User can set a time period for which to visualize data for.
   - 1.4. User can ask for min, max and average value.
2. Data fetching
   - 2.1. Software needs to fetch data from two APIs, SMEAR and STATFI.
3. Data handling
   - 3.1. Fetched data needs to be stored in a way that it is easily accessible and modifiable.
   - 3.2. Software needs to prepare for errors and null values in the data.
4. UI handling
   - 4.1. Software should change the view depending on user's choices.
   - 4.2. User should be able to select only data which is accessible
5. File IO
   - 5.1. Software needs to store user preferences to be used later.

7. Databases and external API’s

The data to the software is fetched from two different databases: SMEAR and STATFI.

- SMEAR is a data visualization and download tool that gathers, for example, real-time data of greenhouse gases. SMEAR includes multiple measurement stations across Finland, which are hosted by the University on Helsinki and the University of Eastern Finland.
- STATFI is Statistics Finland’s free-of-charge statistical database that holds historical and statistical data. This data includes past data of greenhouse gases.

7.1 External Libraries

7.1.1 Qt-related libraries

- Qt widgets – Used for UI

7.1.2 Python-related libraries

- Json – Used to convert JSON data into Python dictionaries and vice versa
- Requests - Query library used to make the SMEAR and STAFI queries
- MatPlotLib – Used to make the plots
- Unicodedata – Used to modify data to a proper form
- ABC – Used to create abstract classes and abstract methods

8. Documenting the refactoring during the project

Refactoring made

In chapter 5.4 Query handlers, we added inheritance to the design of query handlers. The two query handlers are now inherited from an abstract query handler class.

In chapter 6, we further specified the requirements. Requirement 1 was renamed from data plotting to data visualization. All the sub-requirements are new. Earlier we had 3.1 Clean up, 5.1 Export and 5.1 Saving. Those were changed to more descriptive requirements.

The plotting tool was changed from Qchart to MatPlotLib during the implementation phase. Also, the design of the MainWindow was altered to replace the tabs with hide-and-show-based sections. Moreover, the design of storing of the history and favorites was altered to using customized pushButtons.

GraphQL was dropped, as there was no need for a query language for the data fetching we do in this project. GraphQL would have been beneficial if we needed to make more specific queries, but we decided to make the queries more general and handle data manipulation on the fetched data ourselves.

The queries are now done using python's requests library, as it proves to be sufficient for the purposes of this project. URLlib was thus dropped, as well as Pandas and Numpy.

A class DataHandler was created to store the data and perform SMEAR/STATFI merge operations.

In chapter 7.1 External libraries, we added some needed libraries

9. Self-evaluation

The design we made before we started the implementation was so thorough, that it supported the implementation process very well. There has been very little need to make adjustments or additions to the design during the process. The few that were made are mentioned in chapter 8. Our current design does also cover the implementation of the all the remaining functionalities.

We're currently not anticipating any further changes to our design, though that is something that may always change in projects. Often this is caused by a change in requirements, but that is not a concern in this case. We way find better ways to implement things during the actual process of implementation, though, which can lead to adjustments to the design, like what happened with the way we fetch data.

As for how our design corresponds to quality, we felt that the more thorough we are with the design, the higher the quality of our software will be. That hypothesis seems to hold water.

10. Attachments

1) Component diagram in UML (Component_diagram_v04.pdf)
2) Figma prototype

URL: *https://www.figma.com/proto/feHbz6M1IiLlxnSYAVXDi4/SoftwareDesign-Group-Assignment?node-id=6%3A2&scaling=min-zoom&page-id=0%3A1&starting-point-node-id=6%3A2&show-proto-sidebar=1*