Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

User Interface and User Accessibility Functions for RoadGraphTool, a Road Network Processing Tool

Jan Trávníček

Supervisor: Ing. David Fiedler
Field of study: Open Informatics
Subfield: Software
May 2021

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Trávníček Jan**	Personal ID number:	**483686**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Software**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

User interface and user accessibility functions for RoadGraphTool, a road network processing tool

Bachelor’s thesis title in Czech:

Uživatelské rozhraní a funkce pro uživatelskou přístupnost pro program RoadGraphTool na zpracování silničních sítí

Guidelines:

Road Graph Tool (RGT) is a tool for processing road network graphs. It is able to filter the map, find the largest strongly connected component, compute various road and edge parameters, and many more. The tool is currently configured by a text file configuration, which is cumbersome, considering the number of configurable parameters. To make the configuration more comfortable, we need to implement a user interface (UI) for the tool configuration, which is the goal of this thesis. Required tasks:
1) Gather the requirements for the RGT UI.
2) Design the UI according to requirements, and choose a proper UI framework.
3) Implement the UI.
4) Perform a usability testing, evaluate the testing results, and Incorporate the user feedback into UI.

Bibliography / sources:

[1] D. A. Norman, The Design of Everyday Things. USA: Basic Books, Inc., 2002.
[2] R. Unger and C. Chandler, A Project Guide to UX Design: For user experience designers in the field or in the making, 2nd ed. USA: New Riders Publishing, 2012.
[3] https://docs.oracle.com/javase/8/javase-clienttechnologies.htm
[4] J. Vos, W. Gao, J. Weaver, S. Chin, and D. Iverson, Pro JavaFX 8: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients, 1st ed. USA: Apress, 2014.
[5] E. Klimczak, Design for Software: A Playbook for Developers, 1st edition. Hoboken, N.J: Wiley, 2013.

Name and workplace of bachelor’s thesis supervisor:

Ing. David Fiedler, Artificial Intelligence Center, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **12.02.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. David Fiedler Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I want to thank my family and my supervisor Ing. David Fiedler, for motivating me into writing this thesis.

I also want to thank the Stack Overflow community for sharing their knowledge publicly and for free use.

Declaration

I declare that this work is all my own work, and I have cited all sources I have used in the bibliography.

Prague, May 10, 2021

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 10. května 2021

Abstract

Road Graph Tool is a program for processing road networks and their extraction from publicly accessible geographical data. Given the wide range of road networks uses, we want to improve the existing uncomfortable way the program is configured so that it is suitable for a broader audience. This bachelor thesis aims to create a full-fledged user interface, which can be effectively used without deeper knowledge of map processing problematics. We are also focusing on functions that improve the program's usability and make it more user-friendly. The program's usability is also tested by volunteers, whose feedback is used for further improvements. The output program is developed in Java using the JavaFX library and the Model-view-viewmodel architectural pattern.

Keywords: User Interface, GUI, User Experience, Road Graph Tool, OSM Processing, JavaFX, User Testing

Supervisor: Ing. David Fiedler
Artificial Inteligence Centre,
Department of Computer Science, FEE

Abstrakt

Road Graph Tool je program určený ke zpracování silničních sítí extrahovaných z veřejně dostupných geografických dat. Vzhledem k širokému spektru využití silničních sítí chceme vylepšit současnou nevyhovující konfiguraci programu tak, aby byl použitelný širším okruhem uživatelů. Cílem této bakalářské práce je vytvoření plnohodnotného uživatelského rozhraní, které umožňuje použití programu i bez větších předchozích znalostí problematiky zpracování map. Zároveň se zaměřujeme na funkce, které zvyšují uživatelskou přívětivost a použitelnost programu v praxi. Použitelnost programu je také testována s pomocí dobrovolníků a jejich zpětná vazba využita k dalšímu zlepšení. Výsledné rozhraní je vyvinuto v jazyce Java s využitím knihovny JavaFX a architektonického návrhového vzoru Model-view-viewmodel.

Klíčová slova: Uživatelské rozhraní, GUI, Uživatelská přívětivost, Road Graph Tool, Zpracování OSM, JavaFX, Uživatelské testování

Překlad názvu: Uživatelské rozhraní a funkce pro uživatelskou přístupnost pro program RoadGraphTool na zpracování silničních sítí

Contents

Figures

Tables

Chapter 1
Introduction

Road Graph Tool (RGT) is a program for processing road networks obtained from Open Street Map (OSM) geographical data. The processed graph of road networks can be used in various further applications, e.g. for navigation, traffic modelling, or distance measuring. To define the RGT process, the user needs to specify a configuration, listing the procedures that should be performed. The configuration is passed to the program on launch, and the procedures are executed with the specified parameters and in the specified order.

However, the existing approach to creating a configuration requires knowledge of the specific procedures, parameters, and keywords (generally, more profound knowledge of the documentation) the user might not have. It is also very vulnerable to user's mistakes (typographical errors, wrong indentation etc.). Such mistakes invalidate the whole configuration, and for the user, it is complicated to locate and fix them because the program is not providing sufficient feedback about it.

The existing configuration method is therefore very uncomfortable and might dissuade potential users of the program. We also expect the program to be developed further and the number of configurable options to grow, making this problem even more serious.

1.1 The Aims of This Thesis

Since the RGT is meant for a broader audience, we want to create a user interface (UI) operating the RGT providing a more comfortable way than the existing approach. First, we focused on the analysis of RGT and similar programs, which we used for collecting requirements for the development. The development itself aims to create an intuitive environment that is well accepted by both new and professional users. From the technical side, the UI is meant to provide an interface for easy addition of other functionality, primarily for adding and editing Procedures and their parameters. It is also required to test the developed UI on subjects similar to the future users of the RGT. We want to gather feedback from testers and use it for further improvements of the interaction between the user and the program.

Chapter 2
Road Graph Tool

Road Graph Tool (RGT) is a Java software developed by the Artificial Intelligence Centre (AIC) of the FEE CTU. Its purpose is to filter and process road networks from publicly accessible Open Street Map (OSM)[1] data. The road networks are used further in traffic simulations, route planning and other traffic-related tasks. In the context of AIC, the RGT is developed primarily to prepare maps for multi-agent traffic simulations.

2.1 Processing Open Street Map Data

As mentioned, the RGT acquires the road network from the OSM, which provides geographic data for free use. The OSM representation of the map is built on five main entities: nodes, ways, areas, relations, and closed ways. These entities can carry tags, metadata represented by a key-value pair providing additional information about the entity [1]. Except for roads, the OSM includes information about other places of interest such as hills, rivers, restaurants, or parks. The RGT filters out the unnecessary information and also unifies the data in tags. This unification of tag values is needed because the OSM database obtains the data from various contributors (users, companies etc.) [2], which leads to inconsistencies, for example, in speed units[2]. The RGT can also compute and add values that are missing. A typical example is a projection of GPS coordinates for more accurate distance computing.

2.2 RGT Procedures

Processing the map with RGT is carried out by a sequence of procedures. The procedures are given, and the user specifies which to use, their order and parameters using the program configuration. There are three main types of procedures: Input, Output, and Step. RGT process starts with an Input procedure that loads the data and ends with an Output procedure that saves

[1] `https://wiki.openstreetmap.org/wiki`

[2] Conflict of units of the metric and the imperial system, sometimes even text constants such as CZ:urban

them. The process must include exactly one Input and exactly one Output procedure, but there can be multiple different Step procedures.

The Step procedures perform additional changes of the road network graph, mainly changing metadata and filtering entities. Steps also have a sub-category, validation steps that check whether the entity metadata is in defined bounds.

RGT currently contains nine procedures: one Input, one Output and seven Steps:

- Input
    - OSM File - Loads the data from a .osm file.
- Output
    - Database importer - Saves the data to a database.
- Step
    - Node processing - Filters nodes and computes additional properties.
    - Way processing - Unifies speed units and filters roads according to used allowed types of vehicles.
    - Whole graph processing - Simplifies the graph: deletes the nodes that are not crossroads nor ends of the roads, and splits edges so that each edge contains exactly two nodes.
    - Graph IDs generator - Generates new IDs for entities
    - Major Strongly Connected Component - Finds the major strongly connected component of the graph and deletes the entities outside of it.
    - Validation Step
        - Max speed validation - Checks the maximum speed values.
        - Elevation validation - Checks the elevations of nodes and gradients of roads.

Some procedures require to be preceded by another procedure (or multiple of them), so the data already include the metadata necessary for processing. Therefore, each procedure contains two lists declaring its *added* and *needed* properties (Graph Procedure Properties). The prerequisites are validated before the execution of the process. Each procedures' *needed* properties are checked, whether they are included in previous procedures' *added* properties. If not, it means the prerequisites are not met, and the process is not launched. The procedures are visualised in the simplified class diagram in the Figure 2.1.

A detailed overview of all procedures, including parameters and prerequisites, is in the Table C.1 in the appendix.

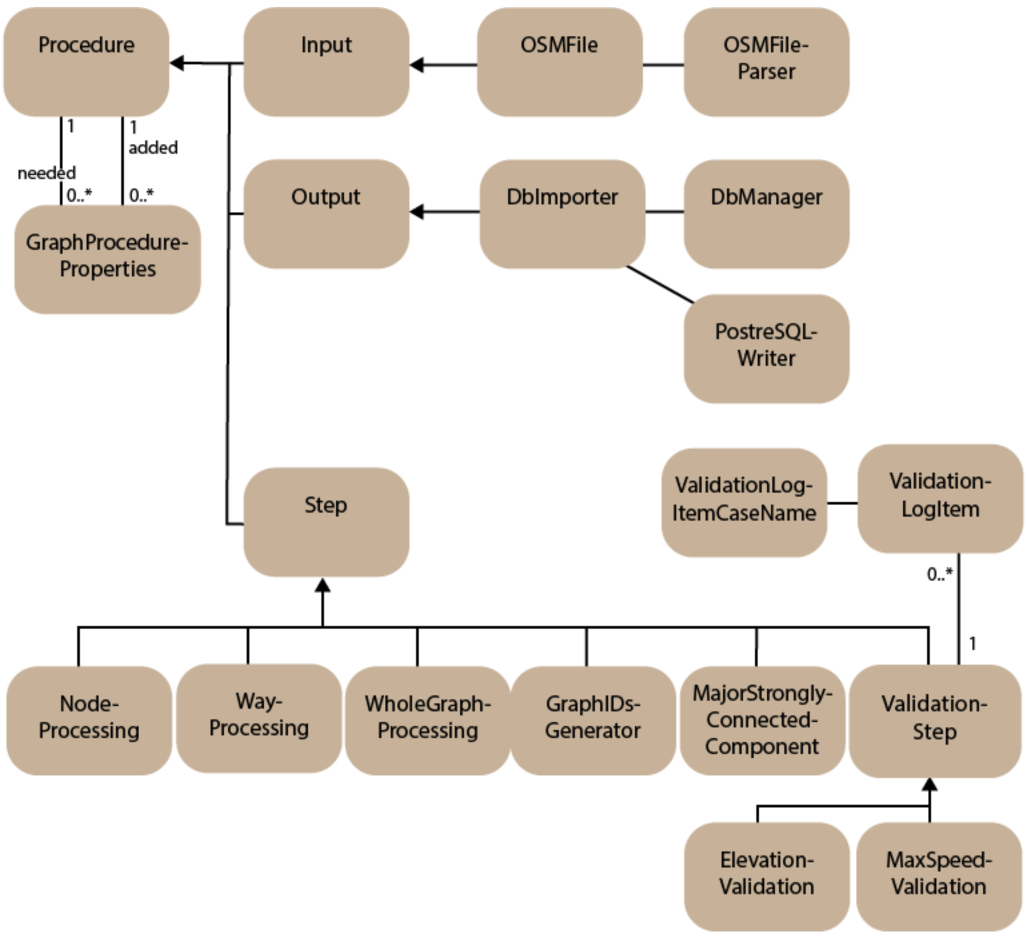


Figure 2.1: Class diagram of RGT Procedures (before changes).

2.3 RGT Process

RGT currently contains only one Input and one Output procedure, which offer one option: load the data from an OSM file and save it to a PostgreSQL database. The principle of the RGT process using these two procedures is visualised in the Figure 2.2.

Although the RGT was designed to contain minimum dependencies, it still needs to use external libraries and executables. The most noticeable example is the Input procedure OSM File, which uses two external executables: Osmfilter[3] and Osmconvert[4] to parse and filter the map.

The current version of the RGT is launched with one argument: a path to YAML[5] file containing a configuration. This configuration is deserialised to a Java object (Config Model), which is passed to Graph Processing Executor, which executes the procedures in the specified order. For better understanding

[3]https://wiki.openstreetmap.org/wiki/Osmfilter
[4]https://wiki.openstreetmap.org/wiki/Osmconvert
[5]YAML is a serialisation format, which is easy to read by humans. https://yaml.org/

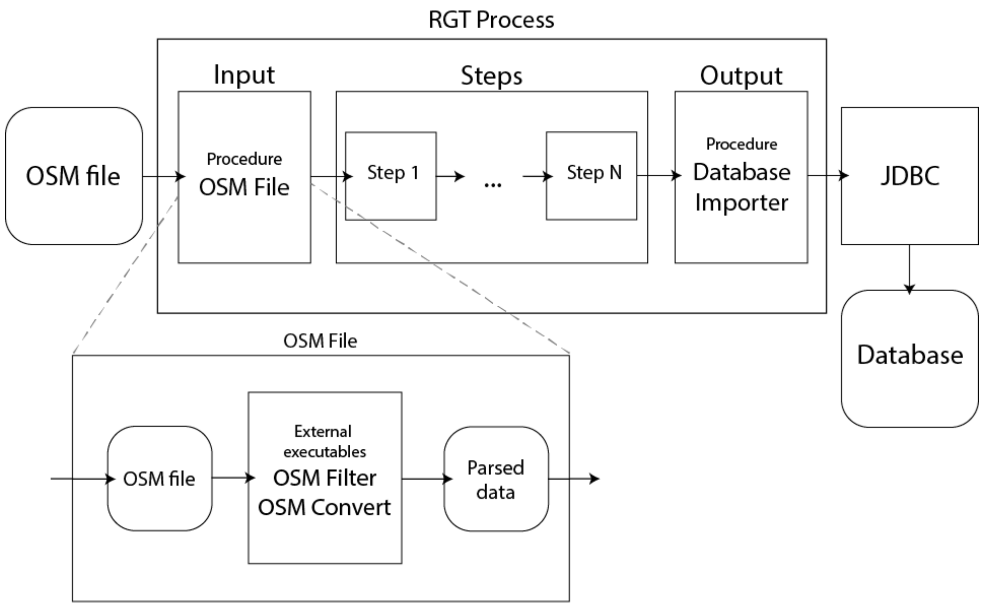


Figure 2.2: Visualisation of RGT process.

of the relationships between classes, see the simplified class diagram in the Figure 2.3.

2.3.1 Problems

As previously mentioned, the RGT initialises the process from a user-defined YAML configuration. A valid configuration must follow the exact format[6] expected by the parser, which requires the user to know the names of procedures and parameters. Since this program is intended for a broad audience, we cannot expect the users to have this knowledge.

Although the RGT is available from a private repository on GitHub, there is no description page provided, which complicates the situation even for the developers.

The second problem is connected with the software architecture. As stated in the class diagram in Figure 2.3, the procedures are currently initialised with a variable number of Parameters. Since the Parameters do not specify a type of value they contain, this approach is very error-prone because the users might not know what is expected. Hence, to learn the needed parameters, the users must analyse the source code, which some might find difficult.

[6]Exact indenting, using case sensitive keywords, etc.

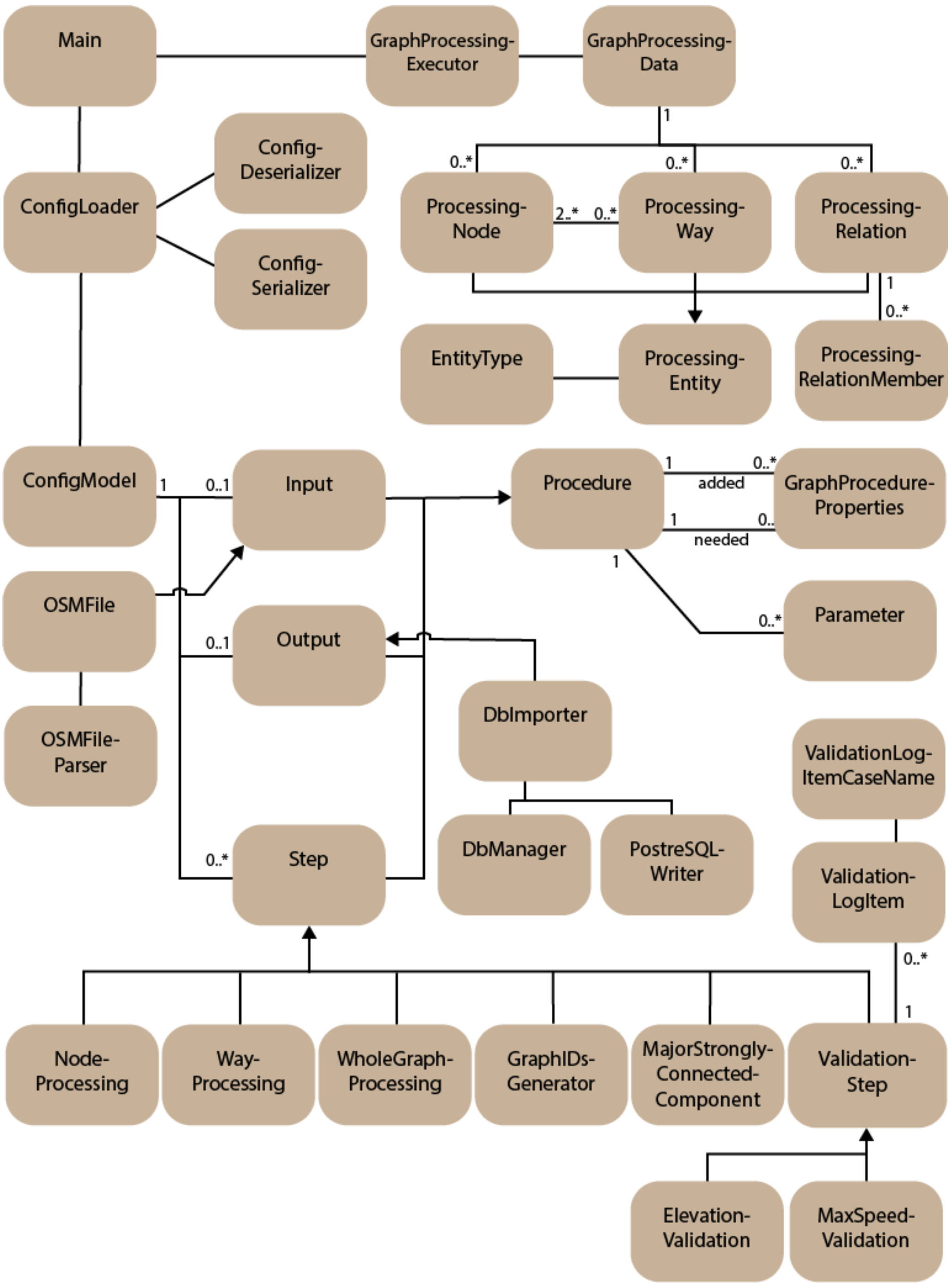


Figure 2.3: Class diagram of RGT (before changes).

Chapter 3
Analysis

This chapter focuses on requirements that specify the desired form of the user interface and the program's features. The general goal is to make the program more user-friendly and add functions that could be practical in connection with the main functionality.

First, we analyse similar programs to gain inspiration about design and explore commonly used features. Based on analysis of similar programs and the RGT (Section 2), we formulate requirements describing the desired UI and its functionality. Once we have the requirements, we decide which technology to use and how to integrate the user interface into the existing part of RGT.

3.1 Similar Programs

There are many other open-source programs processing OSM data. Analysing these programs is a good help for collecting the requirements for this project but also an inspiration for adding new features. Apart from functionality, this also helps us understand the users' expectations [3].

Most of the other OSM processing tools (Osmosis[1], Osmium[2], Osmnx[3], etc.) are distributed as libraries or console applications. Their approach is very similar to the current RGT version, where the users need to memorise the commands and documentation. Although this may be sufficient for the developers, it is not suitable for users using the program occasionally or users with limited background in computer science.

Hence, we focused on analysing programs providing a graphical user interface. Some of them are mentioned with a short description.

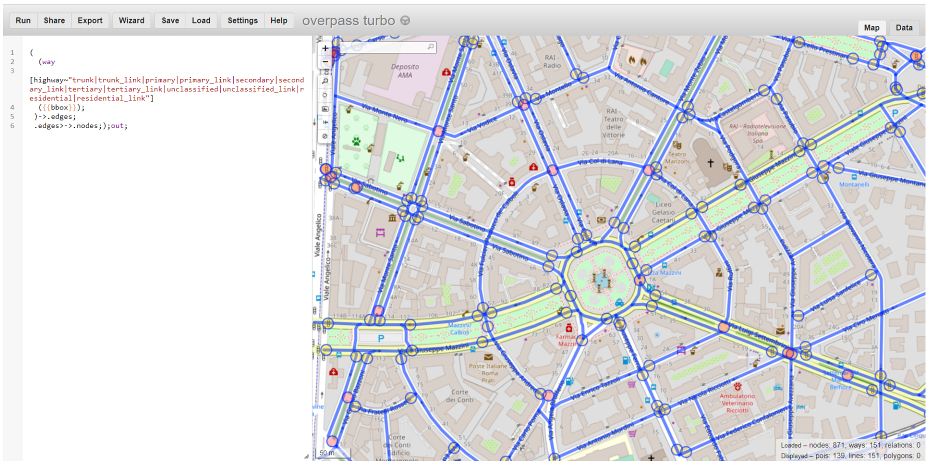
3.1.1 Overpass Turbo

One of the most popular programs associated with the OSM is Overpass Turbo[4], which serves as a frontend to Overpass API. The Overpass API pro-

[1]`https://github.com/openstreetmap/osmosis`
[2]`https://osmcode.org/osmium-tool/`
[3]`https://github.com/gboeing/osmnx`
[4]`https://overpass-turbo.eu/`

vides custom parts of the OSM data specified by the user's request formulated in the Overpass query language (Overpass QL) [4].

The Overpass Turbo contains a console for the user to formulate their query. After executing the query, the result is displayed on an interactive map (Figure 3.1) and may be downloaded. The user can export the map to different formats or use the OpenStreetMap account to store them [5]. The Overpass API also can be queried directly using the QL command in an HTTP request [4].



```
(
  (way
[highway~"trunk|trunk_link|primary|primary_link|secondary|secondary_link|tertiary|tertiary_link|unclassified|unclassified_link|residential|residential_link"]
  ({{bbox}});
 )->.edges;
 .edges>->.nodes;);out;
```

Figure 3.1: Overpass Turbo after running a query for specific road tags.

3.1.2 QGIS

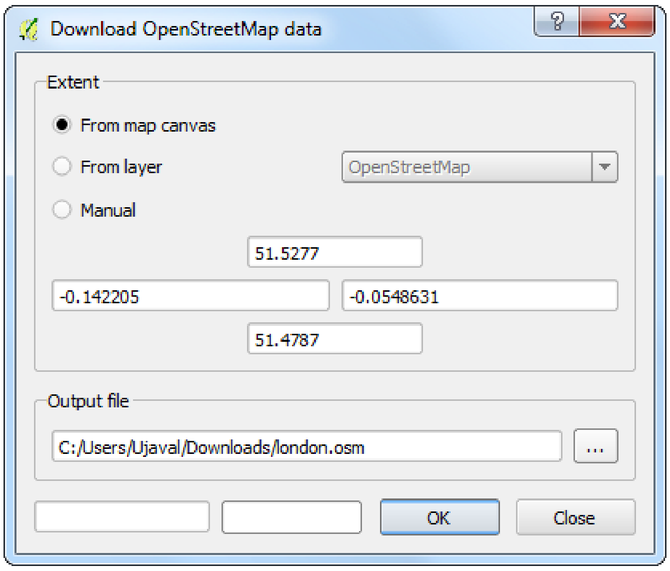
Quantum Geographic Information System[5] (QGIS) is an open-source program for editing and visualising various spatial data types. Although its main functionality is aimed at visualisation and measuring, the QGIS also provides a possibility to crop maps or edit values of single entities. QGIS can operate with multiple layers from various sources (for example, from PostGIS database, .osm file or raster image).

QGIS also supports plug-in modules that the users can use to extend its functionality. The plugins may also be used for downloading OSM data or previewing a map of the whole planet without downloading the full OSM file [6]. A particular example of downloading a map using map envelope[6] is shown in Figure 3.2.

3.1.3 Osm2garmin

Another program similar to RGT is Osm2Garmin[7], which generates maps for updating Garmin GPS receivers. Osm2Garmin provides a way to download free maps covering the whole globe or smaller regions according to the user's

[5]`https://github.com/qgis/QGIS`
[6]a rectangle defined by two latitude and two longitude values
[7]`https://osm2garmin.mantlik.org/`



From map canvas [✓]
From layer: OpenStreetMap
Manual
51.5277
-0.142205
-0.0548631
51.4787
Output file: C:/Users/Ujaval/Downloads/london.osm

Figure 3.2: Downloading a map from OSM using OSM Place Search plugin in QGIS [6].

specification. Program is based on the functionality of Osmosis, Mkgmap[8] and Tile Splitter[9], which process data downloaded using BitTorrent [7].

The main window of the application is shown in Figure 3.3.

3.1.4 Jeography

Jeography is a Geographic information system (GIS) created in Java. Its GUI (Swing) allows users to browse maps and choose areas of specified shape. The user can create unions, intersections and other set operations on these areas. The project also contains several scripts providing additional functionality, such as saving into a database. The functionality is separated into several modules, which can be used independently as a library in other projects (for example, module *tiles* for previewing maps). The main window of the program is shown in the Figure 3.4.

[8]https://wiki.openstreetmap.org/wiki/Mkgmap
[9]https://www.mkgmap.org.uk/doc/splitter.html

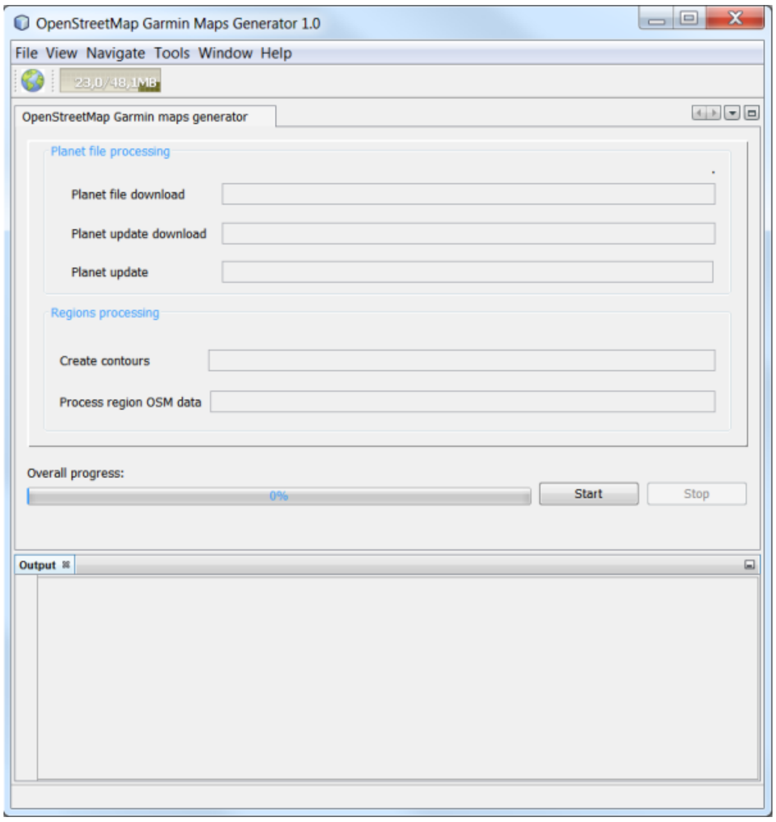
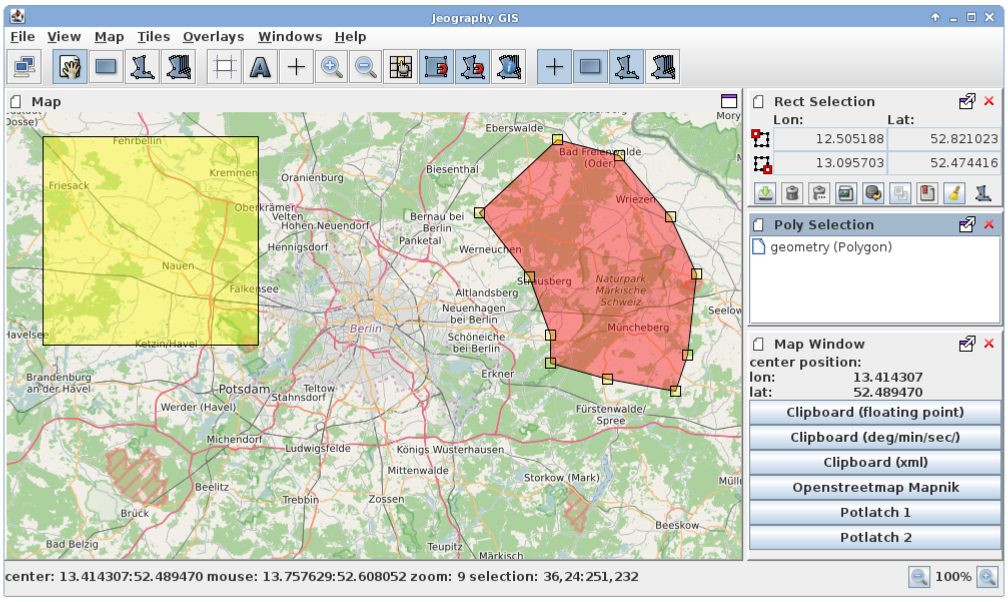


Figure 3.3: Main window of Osm2Garmin displaying progress of the downloading.



Lon:	Lat:
12.505188	52.821023
13.095703	52.474416

center position:
lon: 13.414307
lat: 52.489470

center: 13.414307:52.489470 mouse: 13.757629:52.608052 zoom: 9 selection: 36,24:251,232

Figure 3.4: Main window of the Jeography GIS [8].

3.1.5 Summary

Most of the programs for OSM processing are distributed either as libraries or as console applications. Such user interface is very similar to the current approach, and therefore we decided to focus on programs providing a graphical user interface (GUI). The analysis has shown most of the GUI programs are able to visualise and filter the map entities but do not provide such complex metadata processing as the RGT (computing additional entity parameters, unification of units etc.). We, therefore, could not follow the approach of particular programs because the visualisation of the map is irrelevant, since the RGT does not provide any sort of interaction with it.

Out of the programs we have analysed, the Osm2Garmin is the most similar to RGT. We decided to build RGT GUI in a similar way: listing all the functionality but not visualising the changes performed on the data.

A widespread feature is downloading maps from within the program. Apart from programs specifically built for it (Overpass Turbo, Osm2garmin), this option is available in QGIS or Osmnx. For improving user comfort, the map download option should also be integrated into the RGT.

Other features regarding the design, usability and functionality are referred in the following sections.

3.2 Requirements

Creating a list of requirements is a software design standard that helps us specify the tasks, schedule the work, and agree with our client on what is needed. The main goal of this thesis is to create a program that helps prepare a map of road networks for any traffic-related purposes. Therefore, we want to keep the advantages of the current RGT version: the possibility to run on any device with Java Runtime Environment (JRE) and keep it available as open-source. We also want to minimise the difficulties with using the program so that the users can focus on using the map instead of processing it.

We formulated the following requirements based on the analysis of RGT and its competitors and consultation with the supervisor and the users of the program.

3.2.1 Non-functional Requirements

Non-functional requirements (NFRs) are defining qualitative attributes of the software. Since the processing part of the program is already developed, the requirements were aimed at GUI and its behaviour on fields of extendibility, robustness and usability [9].

- NFR001 - Cross-platform - The program is available on desktop for all three leading platforms (MS Windows, Linux, and macOS).
- NFR002 - GUI - The program provides a graphical user interface (GUI) that informs the user about a current program's state and all exceptions

in a straightforward form, so the user understands what is happening.

- NFR003 - Declarative GUI - The GUI is defined using a declarative language, so it is easy to read and modify.
- NFR004- Open-Source - All of the used libraries and frameworks must be available under a license permitting unrestricted use of this component for non-commercial purposes.
- NFR005 - Current technology - Program uses technologies, frameworks, and libraries that are being actively maintained.
- NFR006 - Extendibility - GUI provides an interface for easy addition of new procedures and parameters or other features that may appear in future development.
- NFR007 - Exceptions handling - The program informs the user about any failure, so they are not confused about RGT's behaviour.
- NFR008 - English language - The whole program, including the documentation, is in English so that most users can understand it.

3.2.2 Functional Requirements

Functional requirements (FRs) are defining the actions a software must be able to perform. In our case, the UI must provide a possibility to create and edit a configuration, run the process with this configuration and also check all the values inputted by the user and warn them in case of invalidation.

The program does not distinguish particular user roles, all of the stated functions are available to any user.

- FR001 - Creating configuration - The user may use a graphical environment to create a configuration. The user may add and remove procedures to/from the process and change their execution order and parameters.
- FR002 - Real-time validation - Program warns the user if the value inputted in a parameter is in an unexpected format, so they can immediately fix the mistake.
- FR003 - Configuration import - The user may load a YAML configuration file to initialise the data in the GUI, so they do not have to input the same values multiple times.
- FR004 - Configuration export - The user may save the configuration from the GUI environment to a YAML file, so they can reuse it in the future or share it with other users.
- FR005 - RGT execution - The user may execute the RGT process from within the GUI with the configuration specified in the environment, so they do not have to use the command line.

  - The program validates all input data before executing the parsing process and informs the user in case of any irregularities.
  - The user may interrupt the process while it is running.

- FR006 - RGT monitoring - The program informs the user about the progress of the map processing, so the user always knows what is happening.

  - The program informs the user once the process ends via dialog. In case of failure, the dialog contains an error message describing the reason for failure so that the user may fix it.
  - The progress is represented by a progress bar, so it is evident in what phase the parsing process is.
  - The program displays program logs, so the user has detailed information about the current program's activity.

- FR007 - Map downloader - The user may download a desired map from within the GUI, so they do not need to learn about other programs.

- FR008 - Help page - The user may open a help page if they do not understand some of the program's features properly.

3.3 Selecting Language and Framework

The selection of the framework is defined by the non-functional requirements. As stated, the UI must support multiple platforms to be available for a broad spectre of users. Also, the GUI must be defined using a declarative language, and all of the technologies used must be currently maintained.

Based on these criteria, we selected three candidates for further analysis: Electron[10], NW.js[11] and JavaFX[12]. As listed in the Table 3.1, all of the three frameworks fulfil the mandatory requirements. Other considered options were Java Swing and Node.js, which are now understood as deprecated, and the .NET WPF, which is limited to Microsoft devices[13]. .

3.3.1 Electron

Electron is a cross-platform framework based on web technologies: Chromium and Node.js. It provides a desktop wrapper for a web application build with JavaScript, HTML, and CSS. Electron is distributed under MIT license and is maintained by OpenJS Foundation[14], which is supported by GitHub, Microsoft, and other software giants. The Electron powers many programs, such as Facebook Messenger, Visual Studio Code, or Slack.

[10]https://www.electronjs.org/
[11]https://nwjs.io/
[12]https://openjfx.io/
[13]https://devblogs.microsoft.com/dotnet/announcing-net-5-0-preview-1/
[14]https://openjsf.org/

Mandatory requirements			
Requirement	Electron	NW.js	JavaFX
Cross platform	YES Own executable for each platform	YES Own executable for each platform	YES Running in JRE
Open source	YES Available under MIT license	YES Available under MIT license	YES Available under GPL
Currently maintained	YES By OpenJS foundation	YES By Intel	YES By community
Declarative GUI	YES HTML, CSS	YES HTML, CSS	YES FXML, CSS

Table 3.1: Comparison of selected frameworks on mandatory requirements.

3.3.2 NW.js

NW.js, previously known as Node-Webkit, is a cross-platform framework very similar to Electron. It is also web-technology based, uses Chromium and Node.js, and works as a desktop wrapper for programs written in JavaScript, HTML, and CSS. Currently, NW.js is supported mainly by Intel and is available under MIT license.

In comparison to Electron, the NW.js is less popular, having 38 thousand stars on GitHub[15], while Electron over 91 thousand[16] (to 24-04-2021). The Electron has a larger community, which positively impacts documentation, user-created libraries, and tutorials. As for the technical aspects, the main difference is the usage of JavaScript contexts[17]. The NW.js also runs on older operating systems (such as Windows XP), offers a source code protection[18] and supports a broader range of Chrome APIs.[19]

3.3.3 JavaFX

Next to it, JavaFX, also known as OpenJFX, is a Java library for GUI development licensed under the General Public License (GPL). It can be used in connection with a markup language FXML and styled by CSS stylesheets. The main advantage of choosing JavaFX is the easy integration to the existing project written in Java. Hence we can avoid inter-process communication because the GUI and the parsing may run in the same process.

[15]https://github.com/nwjs/nw.js
[16]https://github.com/electron/electron
[17]https://github.com/nwjs/nw.js/wiki/Differences-of-JavaScript-contexts
[18]http://docs.nwjs.io/en/latest/ForUsers/Advanced/ProtectJavaScriptSourceCode
[19]https://www.electronjs.org/docs/development/electron-vs-nwjs

3.3.4 Selecting the Best Candidate

Summary of comparison of the selected options (Electron, NW.js, and JavaFX) is presented in the Table 3.2[20]. Based on these observations, the JavaFX was selected for development. It is a library that we can add to the existing project, and all dependencies will be downloaded automatically by Maven.

Suitability for the project			
Requirement	Electron	NW.js	JavaFX
No additional installations	NO Chromium, Python, npm, and other programs needed for development	NO Chromium, Ninja, npm and other programs needed for development	YES same as base program (+dependencies in maven repositories)
Easy integration to RGT	NO Another process. Problem with monitoring the parsing	NO Another process. Problem with monitoring the parsing	YES Could be part of the RGT process, just separate threads
Accessible documentation	YES big community, user-friendly environment	NOT COMPLETELY community is not that big, documentation is chaotic	YES Javadoc, FXML documentation, CSS reference guide, OpenJFX documentation.

Table 3.2: Comparison of considered frameworks in terms of suitability for the project.

3.4 Architecture

As previously stated, the processing part of RGT is still in development, and we expect source code changes, mainly in procedures and their parameters. Hence we cannot access the procedures directly from the GUI, because such approach makes the data and its visualisation too dependent, meaning that any change of procedures will require a change of the visualisation part.

To avoid this problem, we used the Model-View-ViewModel (MVVM) architectural pattern. Its great advantage is the loose coupling between the visualisation and the application logic [10]. The MVVM pattern divides the program into three components: Model, View, and ViewModel (VM). The View component visualises the data and communicates with the user. The changes the users perform in the View properties[21] are forwarded to the relevant variables in ViewModel via data binding. This way, the VM preserves the program's current state and works as a mediator between the

[20]NW.js tutorials are aimed for Linux users, various quick start guides among the official pages, some are missing prerequisites.

[21]Properties are a special data type. If two properties are bound together, they automatically share the same value [10].

View and the Model. The Model component holds the application data and logic and, as mentioned, is entirely independent of the View [10][11].

Chapter 4

Implementation

This chapter summarises programming the RGT GUI, encountered problems, selected solutions and the most critical decisions.

The GUI is implemented directly in the existing RGT project; with the usage of the JavaFX library. All of the windows and components are defined in FXML[1], a markup language based on XML. The advantage over the standard JavaFX declaration[2] is a more readable hierarchy and cleaner code. We also selected the MVVM architecture, which uses JavaFX components' properties and binds them to corresponding values of ViewModel [12].

GUI was implemented in RGT's private GitHub repository in the user_interface[3] branch.

4.1 Integrating GUI into the Existing Project

Since the program is still in development, the GUI must provide an interface for easy visualisation of procedures added in the future. The GUI is designed to accept a various number of procedures so that a future developer does not need to change the core of the GUI dramatically if a new procedure is added. Also, the possibility to run the program without the GUI must be preserved so that the program may run on devices with a console environment only, for example, using remote access on servers.

The whole program is visualised in the class diagram in Figure 4.1.

4.1.1 Procedure Configuration

Currently, the Procedures are initialised with its *init*() method that accepts one argument: a map of key-value pairs, where the *key* is a string containing the name of the parameter, and the *value* is an unidentified Java object. The initialising method then searches for expected keys and loads the paired values to corresponding parameters.

[1]`https://openjfx.io/javadoc/11/javafx.fxml/javafx/fxml/doc-files/introduction_to_fxml.html`

[2]Creating and connecting instances of the objects and setting their parameters

[3]`https://github.com/aicenter/road-graph-tool/tree/user_interface`

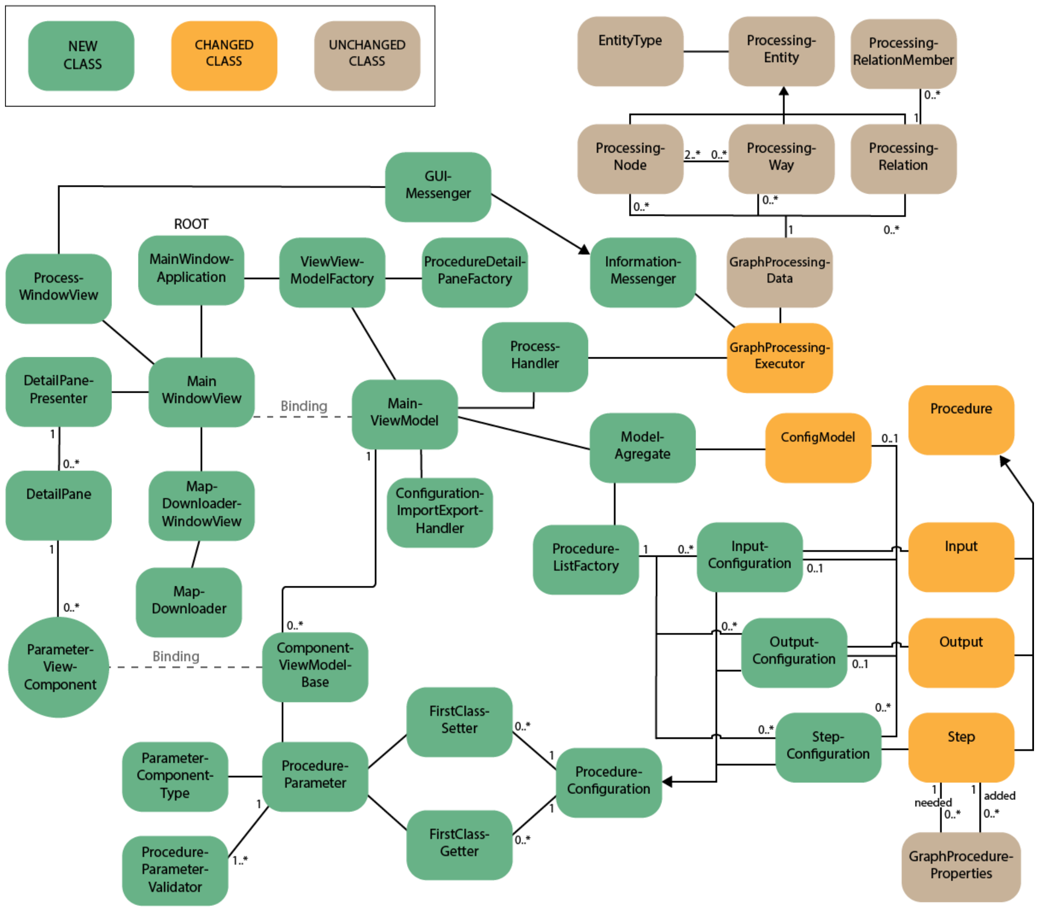


Figure 4.1: Simplified class diagram of the RGT (current state).

This is a very error-prone approach because the procedures do not declare (on the outside) what parameters are required, and the users might not know what is expected. To learn the needed parameters, the users must analyse the source code.

To avoid this problem, the Procedure Configurations were created. The Procedure Configurations store a copy of Procedures' parameters and display them on the outside[4]. Procedure Configurations also handle validation and parameter changes and provide an interface for visualising the Procedure in the View. Once the RGT process is launched, the data of Procedure Configurations are used to create instances of Procedures[5], which can be executed. All available Procedures and corresponding Procedure Configurations are listed in a class diagram in Figure 4.2.

[4]Using getters and setters
[5]The *init* method is now replaced with a constructor.

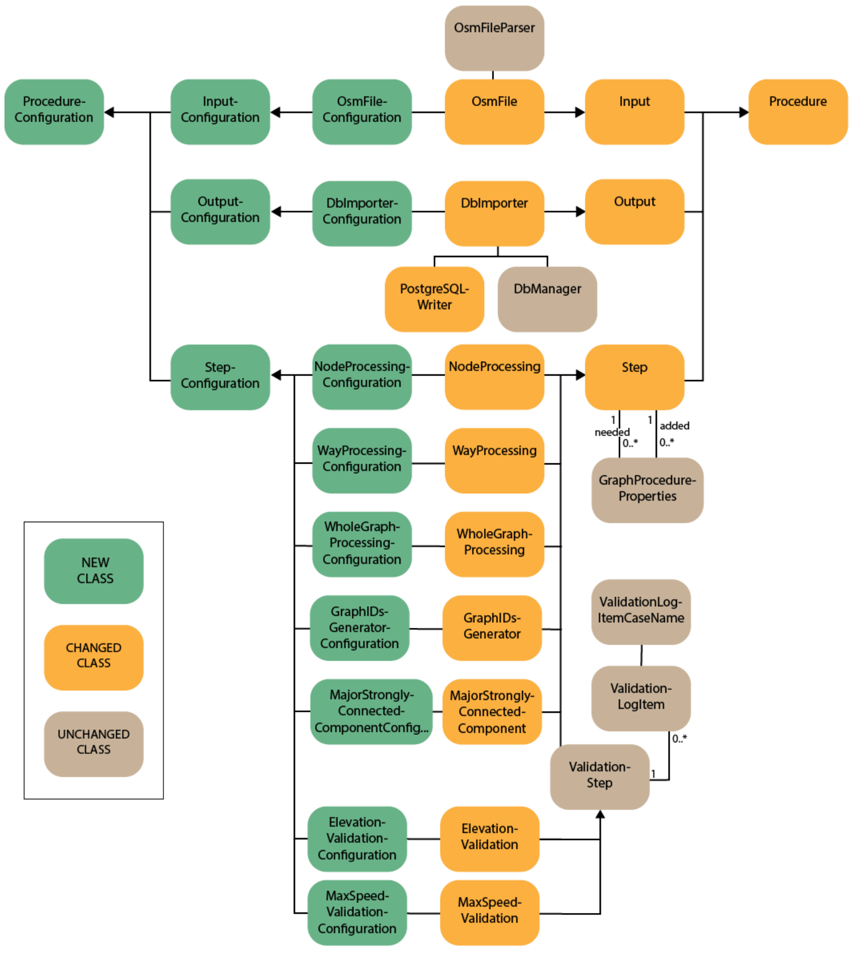


Figure 4.2: Class diagram of Procedures and Procedure Configurations in the RGT.

4.1.2 Custom Components

We have noticed that the Procedures contain only a small amount of parameter types: integer, real number, text, path to a file, and a list of enumeration values.

Hence, we can create a custom component[6] (Parameter View component) accepting each of these types and use it as its GUI representation. Inputting some of the types is also very similar, and therefore, we can merge them. The three resulted Parameter View components (PV components) are shown in the Table 4.1.

4.1.3 Usage of MVVM in RGT

Following the MVVM architecture [11], the GUI is divided into three components: Model, View and ViewModel (VM). The Model creates and stores

[6]Predefined GUI component with specific behaviour, which can be reused in multiple places in the program.

PV component	Type of parameter
LabelledTextBox	Integer, Real number, Text
InputFile	Path to a file
SelectableList	List of enum values

Table 4.1: Custom components representing the procedures' parameters.

the Procedure Configurations and also represents the configuration of the process The View presents the data in the form of components that the user can change. The VM then controls a data flow between the Model and the View, providing the necessary validations and conversions. The number of procedures and parameters may change, which requires the GUI to create a representation of the procedures dynamically.

The used architecture of MVVM is demonstrated in the schema in Figure 4.3 and in the class diagram in Figure 4.4. Procedure parameters, meant to be visualised, are defined in the Procedure Configuration and include several attributes: name, type, reference to a getter, reference to a setter, and 1 to N Validators. According to the parameters given by the Procedure Configuration, the program builds a panel (procedure's Detail Pane), which contains the PV components of the given type.

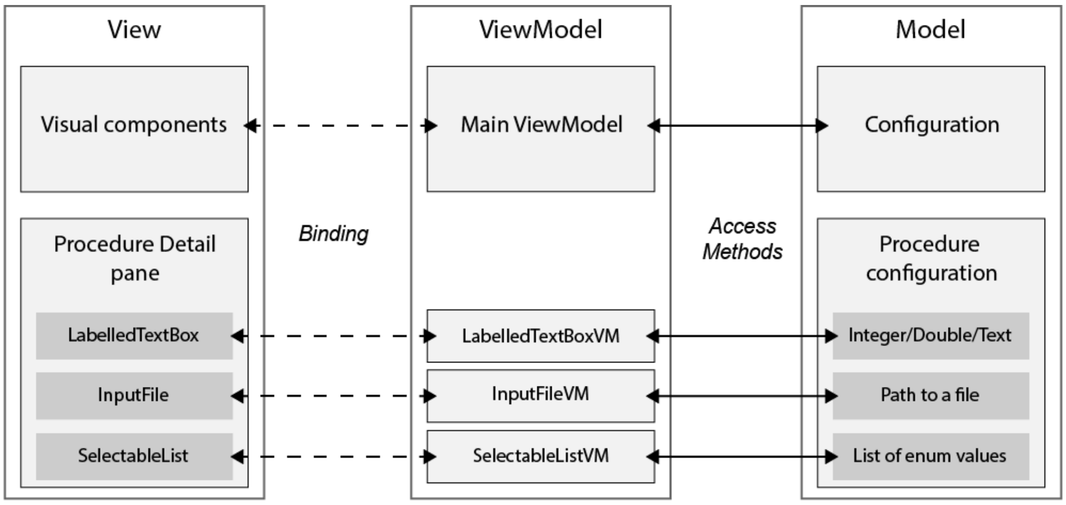


Figure 4.3: Communication of the MVVM components.

The algorithm of initialising the GUI consists of three steps:

- Step 1 - A Model component is created. The Model creates the Configuration and all Procedures Configurations.
- Step 2 - A View component is created.
- Step 3 - For each Procedure Configuration in the Model:
  - Step 3.1 - The procedure's Detail pane is created.

- Step 3.2 - For each parameter defined in the current Procedure Configuration:
    - Step 3.2.1 - A PV component and corresponding VM are created, its values are bound together.
    - Step 3.2.2 - The PV component is added to the procedure's Detail Pane.
- Step 3.3 - The Detail Pane is added to the View.

Once the user changes the value of the PV component in the Detail Pane (see Figure 4.4), the change is automatically propagated to the corresponding VM, which validates the input using Validators. If the validation is correct, forwards it to the procedure using the first-class function setter. The VM also performs the necessary value conversions.

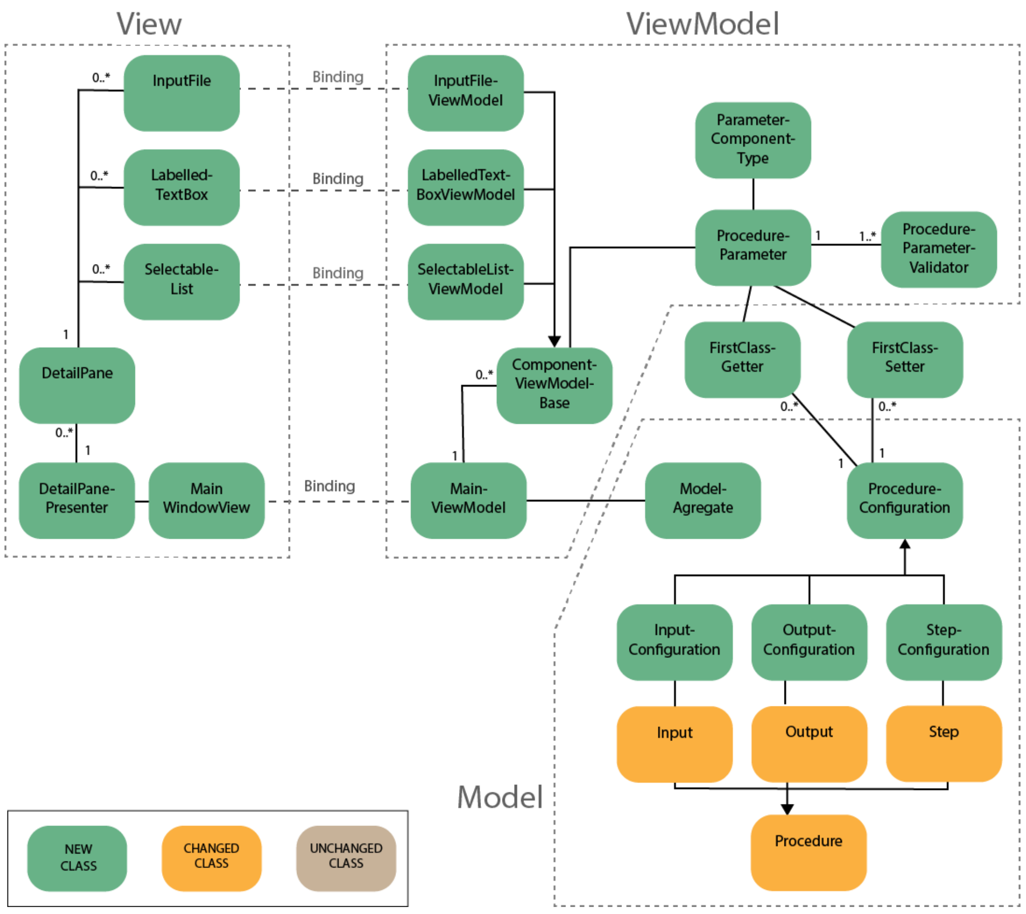


Figure 4.4: Class diagram of the MVVM architecture in the RGT.

4.1.4 Displaying Other Data

Other data (shared data) not dependent on particular procedures do not require to be initialised dynamically. This includes lists storing the procedures and the visual components such as buttons labels and other containers displayed in the View. These data are defined in the main window and bound to the Main ViewModel (MVM), which provides a connection to the Model and defines the functions of buttons etc.

The overall solution is described in the class diagram in the Figure 4.4.

4.2 User Experience Design

User Experience (UX) design is a process of defining the interaction between the user and the product. Since the UX is also affected by other aspects, e.g. communication with the provider or overall brand experience, [13], we are limiting this term to usability. Usability is a characteristic of a UI, describing how easy is this interface to use [14]. According to Nielsen [15], usability is defined by five main components:

- Learnability - How easy is it for a new user to learn and perform basic tasks.
- Efficiency - How quickly are the averagely skilled users able to operate and perform tasks.
- Memorability - If a user does not use the program for a while, how quickly does the knowledge of the UI recover once he starts using it again.
- Errors - How many and how severe errors can a user make in the program.
- Satisfaction - How pleasant is the user interface to use.

To know which of the usability characteristics to prioritise, we first need to understand the context of the user and a typical use case [3]. Since the RGT is used for processing maps, we can presume the user will primarily use the RGT in the process of preparations for other projects. Such user will probably use the RGT occasionally and prepare more maps at once, following a similar pattern. We can expect a typical user to be a researcher, student or enthusiast with average or strong IT background. Still, the user's knowledge of OSM and map representation can be narrow or none.

Considering the program is to be used by non-proficient users and only occasionally, we must prioritise the GUI's learnability. Therefore, we followed an approach of Klimczak [3] and his Design to learnability. Its primary idea is to provide a lot of hints and create a simple layout displaying the main functionality in the user's main area of focus. Therefore we firstly focused on the layout of the components.

4.2.1 Program Windows and Layout

The program contains three windows: the main window, the process window and the map downloader window. The main window provides an environment for creating a configuration and options to save/load it. The main window also works as an intermediary to other windows: it allows the user to run the parsing process and open the map downloader window. As the names of the other two windows imply, the process window displays information about the running process, and the map downloader window presents an option to download a map that the user can process later.

There are four main containers to be displayed in the main window (Table 4.2). All existing procedures are presented in the Procedure List, from which the user can move them to the Process List or vice versa. Details of the selected procedure are shown in the Procedure Detail Pane, which presents the custom components according to the procedure's parameters type. Following standards of other programs, other functionality, e.g. other windows or links to important websites, are listed in the Main menu.

Component name	Description
Procedure List	A list of all procedures implemented in the RGT.
Process List	A list representing the RGT process. The RGT executes procedures that are in the process list. Procedures can be moved between this and the Unused Step list.
Procedure Detail Pane	Panel displaying component, which user can edit to change the parameters of the currently selected procedure.
Main menu	Menu containing options to load and save configuration, open map downloader window, and open relevant web pages.

Table 4.2: Containers displayed in the main window.

The first draft of the main window layout was created using Figma[7]. The prototype, shown in Figure 4.5, was only made to visualise the layout and essential functions to agree on how the program will look and what components should be displayed.

The layout itself follows the way users are scanning websites to put the component, which is to be used first, to the user's primary area of focus. One of the most common scanning patterns is called F-shaped. The principle is that a user on the first encounter with an application or a web is starting to scan an area shaped like the letter F [16].

Therefore, the left side is filled by the Procedure list, and the menu (only indicated in the prototype) is displayed at the top. Other components are positioned along the window's borders, the Process list horizontally to copy a real-life image of the process[8] [14]. Process button placed on the bottom right corner, following a standard placement of confirmation buttons [17].

[7]`https://www.figma.com/`

[8]A process is in the human mind visualised as a production line, which is horizontal.

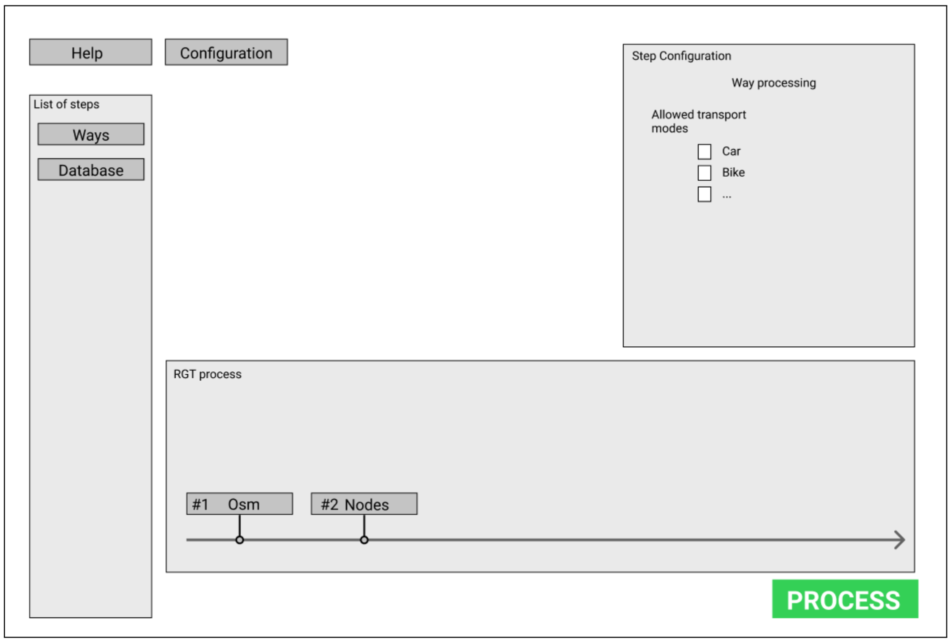


Figure 4.5: Prototype of the main window layout.

4.2.2 Main Window

The second version of the main window design (Figure 4.6) was made in JavaFX and styled using a CSS stylesheet. Since this is a final version, more effort was put into styling and looks.

One of the main differences from the prototype (Figure 4.5) is the layout of lists divided for better partition of the Input, Output, and Step procedures. Also, top buttons were moved to a menu bar at the top of the window to follow the standard approach used in similar programs. We also decided to move the 'Process' button to improve its visibility.

The functionality of the components is described in Section 4.3.

4.2.3 Communication with the User

To provide sufficient feedback and present the current state, the GUI communicates with the user on multiple levels: verbal and nonverbal. An example of nonverbal communication in the main window (Figure 4.6) is the colour highlights of matching containers (green for input, orange for output) or arrows in the Process list indicating the direction of processing [3]. The creating of such hints was based on assumptions.

For more complex messages, the program is using system dialogs and tooltips[9] (Figure 4.10) that provide hints to the user. The system dialogs are used to notify the user in serious situations, mostly about errors. Error messages, following Nielsen's heuristics [14], describe the error and suggest a solution so that it is clear what the user made wrong and how to react.

[9]Hints that appear if the cursor is idle in a defined space

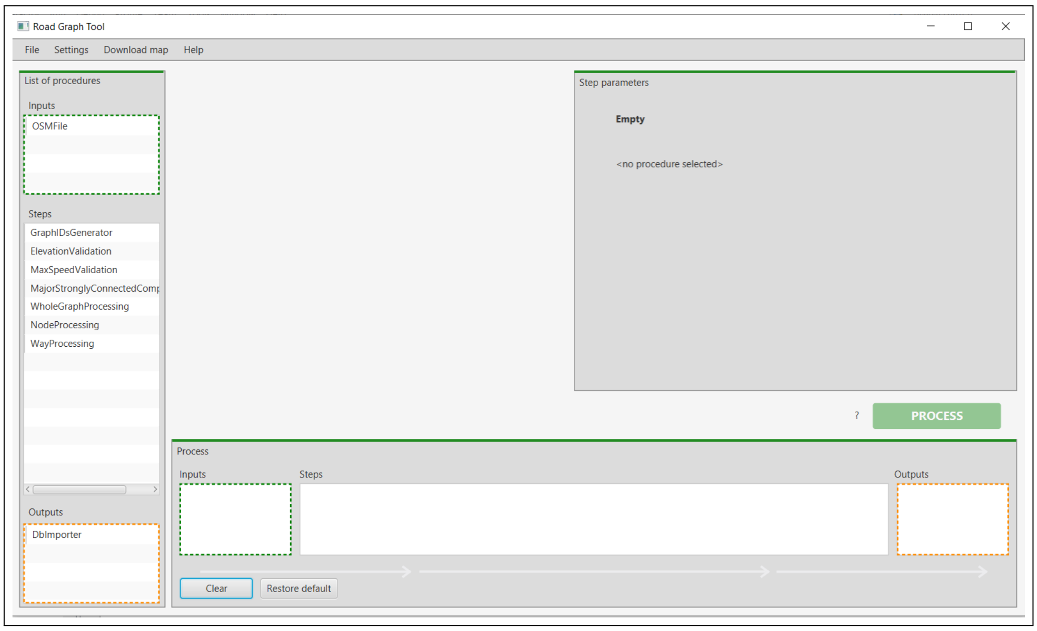


Figure 4.6: Main window of the RGT.

4.3 Program Features

Program features are an overall name for the program's functionality and behaviour. The functionality represents actions the program is able to perform, which are defined in the functional requirements (Section 3.2.2).

The core functionality of the GUI is to provide an environment for the two main operations: creating a Configuration and executing it. The environment for creating a Configuration is contained in the main window, which displays both the current state of the Configuration and Procedures that may be added or edited. The main window also contains the Detail Panes of the Procedures, where the user can adjust the parameters so that the Procedure process the data as desired. The parameters are validated so that the Procedure does not process invalid data.

Once the user is satisfied with the configuration settings, they may execute the process. The process is executed in the background, and the GUI displays its current state in the process window.

GUI also contains functions that are not necessary for processing but have a positive impact on usability. An example of such a function is the map downloader that enables users to download a map they may immediately use.

4.3.1 Configuration Environment

A principle of creating a Configuration is that the user moves Procedures between the Procedure List in the left, containing all procedures, and the Process Lists, at the bottom of the main window (Figure 4.6). The lists, which represent the same type of Procedure, are highlighted by matching colour, and the user can move the Procedures only between the pair (Figures 4.7 and 4.8).

A Procedure can be dragged by mouse and moved to a specific position in the Process List (Figure 4.7) or moved to the end by a double click. Since there can be only one Input and Output in the process configuration, the program replaces the Procedures in the Process List if the user tries to add another.

The lists also support keyboard shortcuts. The user may use `WSAD` or arrows to navigate and select the procedures and move them from/to the process by pressing `Space`. The whole configuration can be cleared by pressing the 'Clear' button or `Shift+C`. The default configuration, which can be easily adjusted in the source code, can be renewed using the 'Restore default' button.

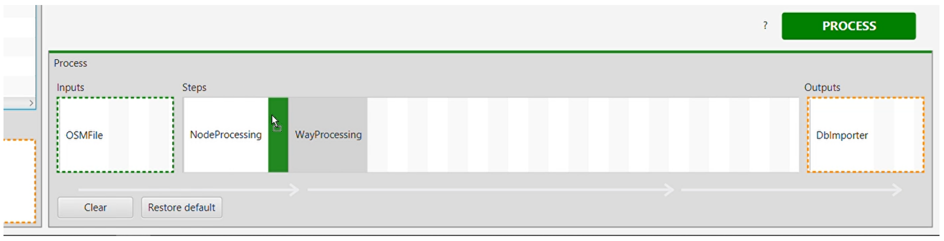


Figure 4.7: Dragging a Step Procedure to a second position in the Step Process List.

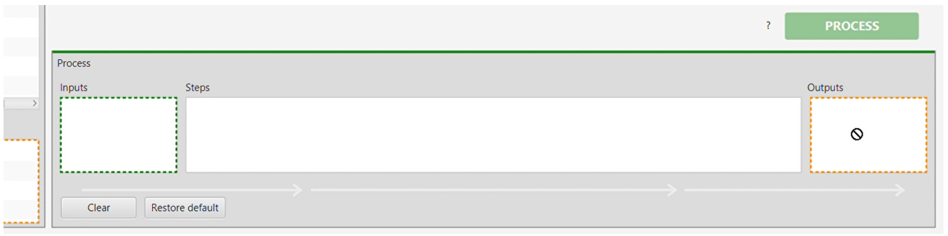


Figure 4.8: Dragging Input procedure to Output container shows an error.

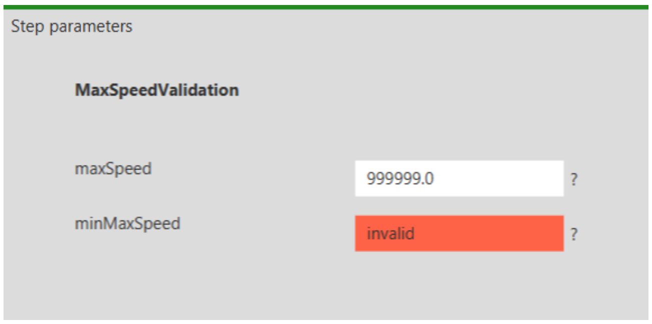
If the user wants to keep configurations for future use, they may export it to a YAML file using either option in the menu or keyboard shortcut `Ctrl+S`. Similarly, a YAML file can be imported to the program, which overwrites the current configuration, changes settings of procedures and values of parameters. The importing and exporting is also automatically used by the program itself: the program imports the last used configuration on startup.

4.3.2 Procedure Parameters

Parameters are displayed in the Detail Pane Presenter component on the right side of the main window. The Presenter switches the Detail Panes so that it always displays details of the currently selected Procedure.

The parameters are connected to a ViewModels that validate the inputted values in real-time. In the case of invalid values, the parameter component displays an error. For example, text components change colour to red (Figure

4.9). Therefore, the user is notified immediately after making a mistake, which improves efficiency and learnability because the user is not afraid to input various values [14].


Step parameters

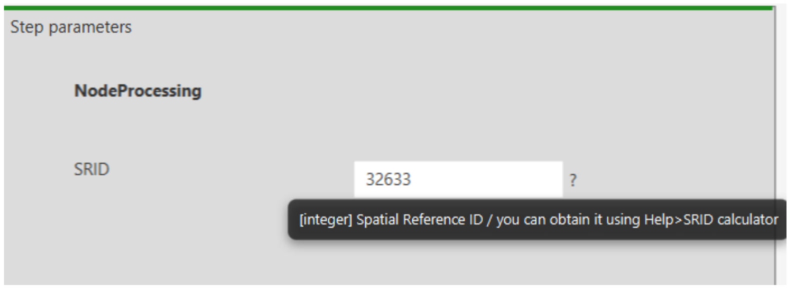
MaxSpeedValidation

maxSpeed 999999.0 ?

minMaxSpeed invalid ?

Figure 4.9: Invalid value inserted into a parameter.

Some of the parameters also include hints providing information about the type of the value and its meaning. The Figure 4.10 shows an example of a description of *the SRID* parameter.


Step parameters

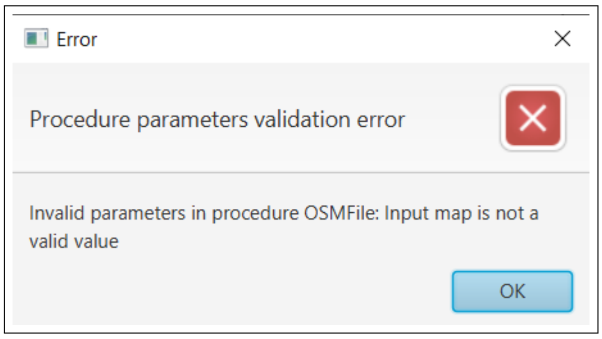
NodeProcessing

SRID 32633 ?

[integer] Spatial Reference ID / you can obtain it using Help>SRID calculator

Figure 4.10: Hint describing meaning of the *SRID* parameter.

Procedures are also validated before the parsing process is launched. If the value in the parameter is still invalid, the program notifies the user with a system dialog (Figure 4.11).


Error

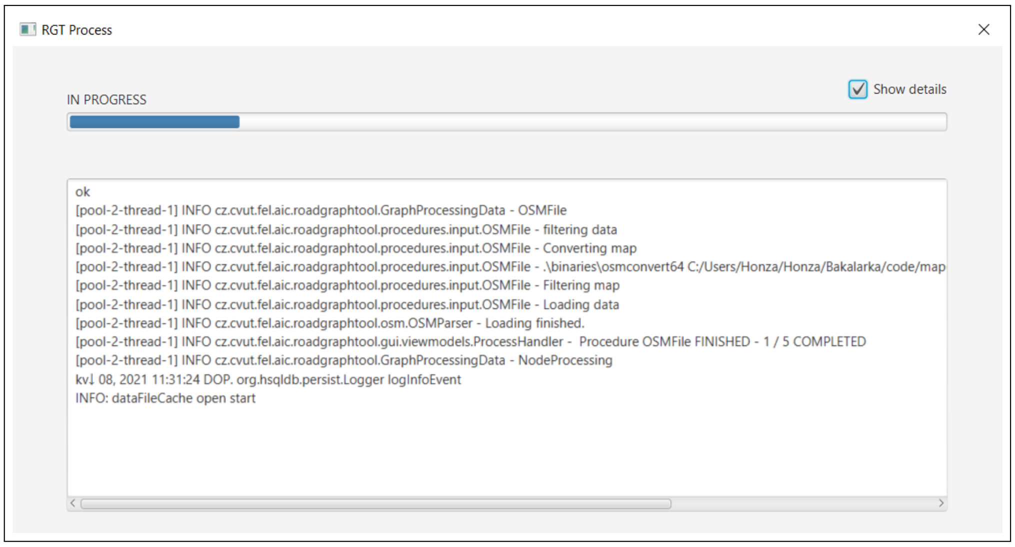
Procedure parameters validation error

Invalid parameters in procedure OSMFile: Input map is not a valid value

Figure 4.11: Error in parameters discovered during the process launch.

4.3.3 Monitoring the Process

Once the user clicks the 'Process' button in the main window and the program verifies none of the parameters includes an error, the RGT process is launched. The process is represented by the process window, displaying progress on a progress bar and showing program output in a console (Figure 4.12).



```
ok
[pool-2-thread-1] INFO cz.cvut.fel.aic.roadgraphtool.GraphProcessingData - OSMFile
[pool-2-thread-1] INFO cz.cvut.fel.aic.roadgraphtool.procedures.input.OSMFile - filtering data
[pool-2-thread-1] INFO cz.cvut.fel.aic.roadgraphtool.procedures.input.OSMFile - Converting map
[pool-2-thread-1] INFO cz.cvut.fel.aic.roadgraphtool.procedures.input.OSMFile - .\binaries\osmconvert64 C:/Users/Honza/Honza/Bakalarka/code/map
[pool-2-thread-1] INFO cz.cvut.fel.aic.roadgraphtool.procedures.input.OSMFile - Filtering map
[pool-2-thread-1] INFO cz.cvut.fel.aic.roadgraphtool.procedures.input.OSMFile - Loading data
[pool-2-thread-1] INFO cz.cvut.fel.aic.roadgraphtool.osm.OSMParser - Loading finished.
[pool-2-thread-1] INFO cz.cvut.fel.aic.roadgraphtool.gui.viewmodels.ProcessHandler -  Procedure OSMFile FINISHED - 1 / 5 COMPLETED
[pool-2-thread-1] INFO cz.cvut.fel.aic.roadgraphtool.GraphProcessingData - NodeProcessing
kv↓ 08, 2021 11:31:24 DOP. org.hsqldb.persist.Logger logInfoEvent
INFO: dataFileCache open start
```

Figure 4.12: Process window displaying the state of map processing running in background.

The progress bar changes colour according to the state of the process. It colours green if the process ends successfully and red if the process fails. Validating the parameters before launch minimises the chances of process failure. Yet, the process can still fail in case of a mistake in data or other edge cases, such as lost connection during saving to database.

4.3.4 Map Downloader

The map downloader is a component enabling the user to download a custom map defined by bounds (rectangle bounded by longitude and latitude values) or the area's name. The map downloader is contained in a separate window, which can be launched from the main window using a menu option shown in the Figure 4.13 or by pressing `Ctrl+D`.

To obtain maps, the map downloader uses the Overpass API. According to the selected option, the inputted values (bounds or name of the area) are placed in a prepared Overpass QL query and sent to the server. Although the authors of the API assure us, we can safely use up to 5GB of downloads per day without disturbing others [4], the maximal size of the rectangle is limited to 10 000 km^2.

The user is also encouraged to use other providers, for example, Geofabrik[10] for downloading bigger areas or the Overpass Turbo for their own queries.

[10] `https://www.geofabrik.de/`

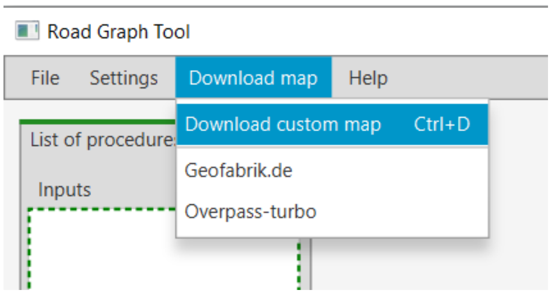
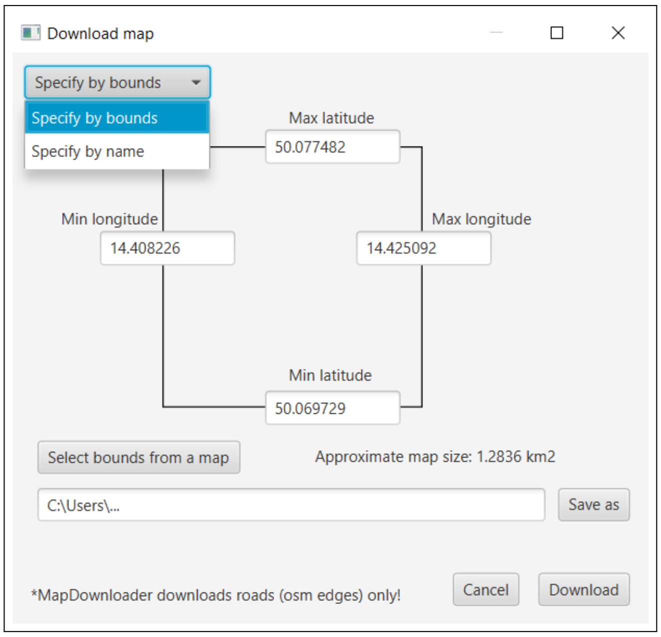


Figure 4.13: Opening map downloader from menu.

The map downloader window shows two options: specify map by bounds and by name. Both options download maps containing roads only to a folder specified by the user. Map downloader also does not overwrite the existing files with the new ones. If a file with a matching name exists in the selected folder, the new one adds a number at the end[11].

The 'specify by bounds' option is shown in the Figure 4.14. It contains a rectangle of latitudes and longitudes, where the user can input the values directly or select a rectangle in map preview. The layout is inspired by the window of the QGIS OSM plugin (Figure 3.2) we have analysed .



Max latitude: 50.077482
Min longitude: 14.408226
Max longitude: 14.425092
Min latitude: 50.069729
Approximate map size: 1.2836 km2
C:\Users\...
*MapDownloader downloads roads (osm edges) only!

Figure 4.14: Map Downloader Window.

The map preview uses a JX Map Viewer 2 library[12]. The user can move the map using the left mouse button and select a rectangle using the right mouse button (Figure 4.15). Once the user confirms the selection, the longitude

[11]If the user is downloading multiple maps, he does not have to create unique names, the program automatically numbers them: map.osm, map1.osm etc.
[12]`https://github.com/msteiger/jxmapviewer2`

and latitude values are inserted into the text boxes in the map downloader window.

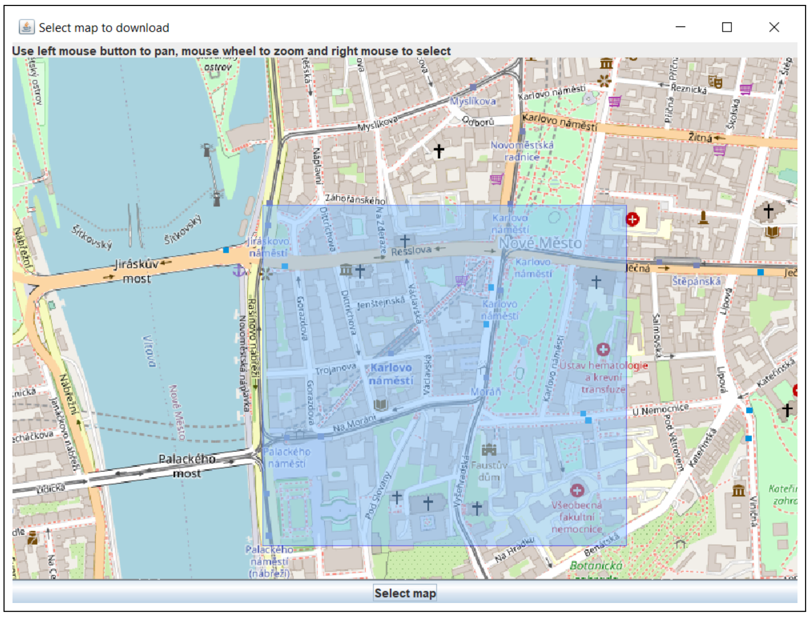

Select map to download
Use left mouse button to pan, mouse wheel to zoom and right mouse to select
Select map

Figure 4.15: Selection of the map to download previewed using JX Map Viewer 2.

The second option is to download the area by its name. This approach is handy for downloading irregularly shaped maps (cities, districts). Yet, the comparison of names while querying is case sensitive, and therefore the user may need to try different variants to obtain the map they want. The general idea is that the entities are named in the local language and use the local name conventions [18]. For example, the user must use the Czech name ’Praha’ to download a map of the Czech capital, the query for ’Prague’ downloads Prague, Oklahoma, US[13].

[13]`https://en.wikipedia.org/wiki/Prague,_Oklahoma`

4.4 Documentation

This section includes information for users about how to use the program and for developers about how to extend it further. The full documentation is available at the project's GitHub page `https://github.com/aicenter/road-graph-tool/tree/user_interface`.

4.4.1 Program Controls

The program control focuses on users with an average or advanced knowledge of computer science. Therefore the GUI is aimed to be controlled with mouse and keyboard simultaneously. The keyboard is obviously used for inputting values, but the user can also accelerate selecting the components using keyboard shortcuts. The summary of keyboard shortcuts is listed in the Table 4.3. The used keys are concentrated on the left side of the keyboard so that the user may use the left hand only.

Shortcut	Action
`W` / `Up`	Navigate up in the list of procedures
`S` / `Down`	Navigate down in the list of procedures
`A` / `Left`	Navigate left in the process list
`D` / `Right`	Navigate right in the process list
`Space`	Move selected procedure from/to the process
`Alt+A` / `Alt+Left`	Move selected procedure one position to the left in the process list.
`Alt+D` / `Alt+Right`	Move selected procedure one position to the right in the process list.
`Ctrl+S`	Export (Save) configuration
`Ctrl+O`	Import (Open) configuration
`Ctrl+D`	Open map downloader
`Shift+Enter`	Start the parsing process.
`Shift+C`	Clear the configuration

Table 4.3: Table of the supported keyboard shortcuts.

4.4.2 Launching program without GUI

The RGT might be launched and used without the user interface. To do so, the user must use the original `Main` class with a path to existing YAML configuration passed as a program argument. If no argument is provided, RGT automatically uses the default configuration. We recommend creating a configuration in the GUI environment and then use it as stated.

A more detailed description is available on the project's GitHub page: `https://github.com/aicenter/road-graph-tool/tree/user_interface`.

4.4.3 Adding Procedures and Parameters

A new Procedure may be added to the GUI by completing the following steps:

- Step 1 - Create a Procedure P implementing either interface Input, Output or Step.
- Step 2 - Create a Procedure Configuration C representing procedure P. The C must implement one of the interfaces InputProcedure, Output-Procedure or StepProcedure according to the type of P.
- Step 3 - List the configuration C in the header of its interface for correct loading and saving.
- Step 4 - List the configuration C in the `ProcedureListFactory` (see Figure 4.1).

The parameters are then defined in the Procedure Configurations. The Procedure Configuration also defines which of these parameters should be displayed in GUI and what type of PV component should be used. A parameter only needs to be declared in Procedure Configuration's `getProcedureParameters()` method and is visualised automatically.

A full description of adding procedures and parameters, including the used methods, is stated on the project's GitHub page: `https://github.com/aicenter/road-graph-tool/tree/user_interface`.

4.4.4 CSS Styling

The program uses a CSS stylesheet to adjust the look of the components. This stylesheet can be found in the project resources and is loaded on GUI initialisation. The developers may use the stylesheet to adjust the application's look, for example, the program currently uses *forestgreen* CSS colour for highlights, which can be easily replaced with another.

The program also includes an option of changing the theme to dark mode. However, the dark mode is still not completed and was added more as a possibility for future development.

Chapter 5
Testing

This chapter describes the methods that were used to test the RGT. According to the tested features, the testing is divided into two groups: functionality and usability testing. The functionality tests have been performed as the program was developed and typically covered the newly added feature. At first, the program was tested only by the author. The supervisor and other members of AIC tested the GUI in later stages of the development. The usability testing was performed by volunteers. Its purpose was to check how the users with no connection to the development are able to perform tasks and orientate themselves in the layout.

This approach was selected since the GUI testing is generally complicated to be performed automatically, and using it may not be worth it on such a small application.

The testing revealed that the procedure *Node Processing* is incomplete since it does not compute elevations. Therefore the procedure *Elevation Validation* had to be removed from the program temporarily because it was causing errors that the testers may be confused about.

5.1 Usability Testing

To make the program intuitive and user-friendly, the testing contained a phase of testing by volunteers. The user-testing was realised when the program already contained most of the main features, missing only the map downloader, which was being implemented in parallel with the testing. The primary goal of the usability testing was to see how are unskilled users able to operate the GUI.

The usability testing itself was divided into two phases: Informal Walkthrough and Scenario Testing. The Informal Walkthrough was aimed to test the general intuitiveness and learnability of the program. The Scenario Testing was then focusing on specific features and tasks.

Each testing phase was followed by a Google form, that testers filled. Transcription of the feedback from and answers is listed in the appendix B.

5.2 Informal Walkthrough

An Informal Walkthrough is a form of usability testing, where the users explore program features on their own. In order to successfully apply this method formulated by Riihiaho [19], the program must be finished or nearly finished, and the users must be familiar with its concept. The session is lead by a moderator, who interferes minimally with the tester. Still, the tester is encouraged to share his expectations, struggles, and thoughts, which can be helpful in further program development. The moderator also can encourage the tester to explore the features by assigning them a task. The benefits of this approach is a realistic evaluation of how intuitive and easy to understand the program is [19].

The Informal Walkthrough testing was realised in the form of a video conference with screen sharing. The recipients were given a detailed description of the RGT's purpose and principle, and each session also started with an additional explanation. The users were asked to explore the program features but were told nothing about the actual program controls. Once they were finished exploring, they were asked to perform the following task:

> "Parse road network of Andorra[1] to your database. Create a projection of points with SRID 32644 and select only roads for cars and bicycles."

Six recipients participated in the Informal Walkthrough testing. Three of them described themselves as average, two as an expert and one as a beginner level computer user. After the testers were acquainted with the RGT parsing principle and functions of the procedures, they started exploring the program (Table 5.1). All users managed to find and use the main features: assigning procedures to process, changing their parameters, and running the process.

Feature name	On their own	With help
Selecting Input and Output enables process	6	-
Clicking the procedure displays details	6	-
Dragging procedures to other list by mouse	6	-
Importing and exporting a configuration	5	1
Parameter changes color on invalid value	3	3
Tooltips	3	2
Selecting procedure by using WSAD	1	2
Other keyboard shortcuts	2	3

Table 5.1: Amount of users (out of six) who discovered and used the program functions on their own or with help of the moderator during the informal walkthrough.

At the end of each session, the recipients were asked to fill in feedback acquiring information about them and their program ratings. The users

[1]The recipients were provided with a map of this country

rated the program design and layout with an average of 8.5 points out of 10. However, the program hints and guidance was rated with an average of 6.0 points (Figure 5.1).

All of the feedback questions are listed in the Section B.1 in the appendix.

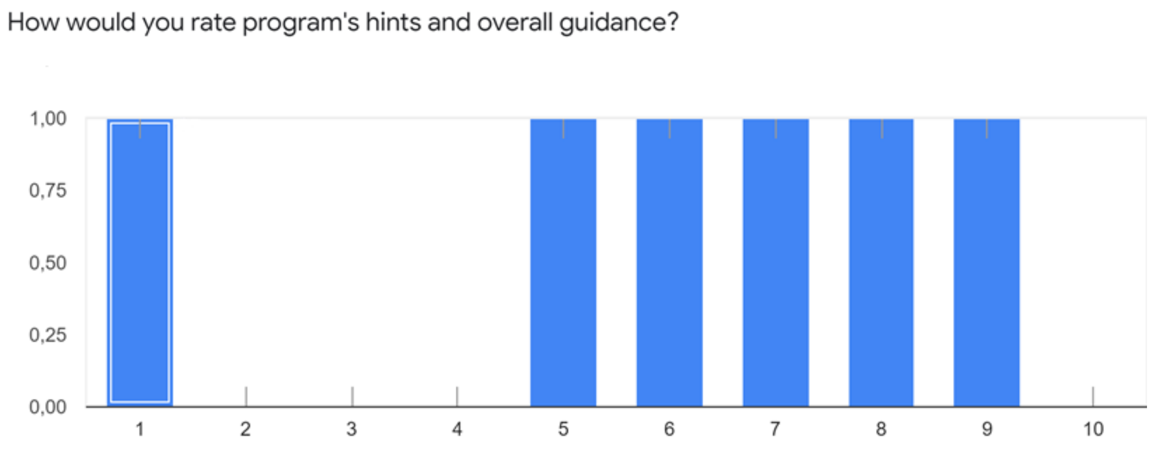


Figure 5.1: User ratings of program guidance.

Three users encountered a problem with launching the program from the jar file; two of them were caused by unsupported Java version[2]. Since the information was contained in the instructions, we highlighted it so that future users will not be confused. The third user was not able to launch the jar on the macOS for unknown reasons. However, the program worked without any problems when launched from IntelliJ.

The testing also revealed that the parsing process on macOS fails to launch the external executables osmfilter and osmextract, which only work on Windows and Linux. Since this issue is not GUI related, we included this information on the help page but did not examine it further.

5.3 Scenario Testing

The Scenario testing was initiated after the Informal Walkthrough. The recipients were described the program controls and asked to execute scenarios: multi-step tasks discovering how different users on different platforms will deal with the assignment. Creating the scenarios is based on a program flow-chart, a directed graph in which each node represents a particular state and each edge represents a user action. Scenarios are finite paths in this graph, which are selected according to the desired depth of testing [20].

In our case, we selected four scenarios representing the most frequent actions the users will perform (Table 5.2).

[2] Java 13 or higher is required

Scenario	Description
1	Load a configuration -> Change parameters -> Change order of procedures -> Process configuration
2	Clear configuration -> Create a new configuration -> Save configuration -> Change configuration in external text editor -> Load configuration -> Change configuration in GUI -> Save configuration -> Repeat from the fourth step
3	Create a configuration using mouse and then using keyboard shortcuts
4	Create a configuration -> Process configuration -> Stop process -> Change configuration -> Process configuration

Table 5.2: Description of the testing scenarios.

Based on Teague, De Jesus and Ueno [21], in order to achieve relevant and unbiased answers, the users filled feedback immediately after completing a scenario. The feedback questions, along with detailed scenarios' description, are listed in the Section B.2 in the appendix.

A total of five recipients attended the scenario testing on two platforms: MS Windows and Linux[3]. None of the recipients has worked with RGT before.

5.3.1 Scenario 1

In this scenario, the testers were asked to load a prepared configuration and run the process specified by it. However, the configuration includes mistakes, which users must fix based solely on error messages. The particular mistakes were two:

- the input file path is set incorrectly,
- the specified order of procedures does not follow the prerequisites.

The program does not include any information about the prerequisites[4]. Therefore, we wanted to test whether this information is needed[5]. To measure the quality of the communication, we also asked the testers to track time, which we can compare with the reference time of 2 minutes 27 seconds (2:27 in shortened form). Except for testing the communication, this scenario is also aimed at testing the reordering of the lists.

[3]Testing on the macOS did not continue after the error occurred during the Informal Walkthrough.

[4]Prerequisites are not stated in the program, only in the documentation.

[5]Too much information in procedure detail pane may become chaotic, and therefore we want to test if the prerequisites are needed there.

The recipients' average time was 3:44 (152 % of reference time) with a maximum of 4:21 (177 % of reference time). Most of the users were satisfied with the program communication (Figure 5.2); the average rating of reordering the list was 8.2 out of 10 (Figure 5.3).

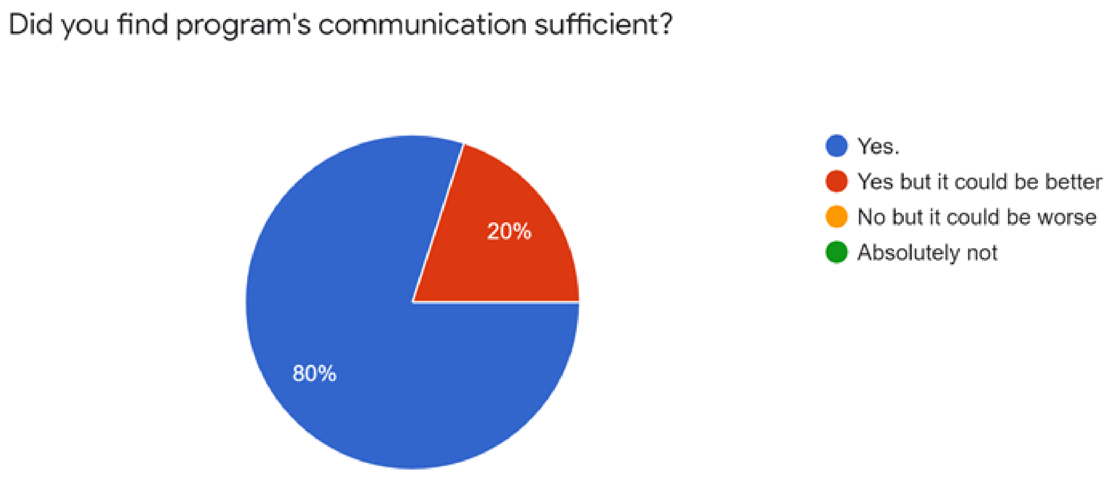


Figure 5.2: User's rating of the program error communication regarding the invalid configuration.

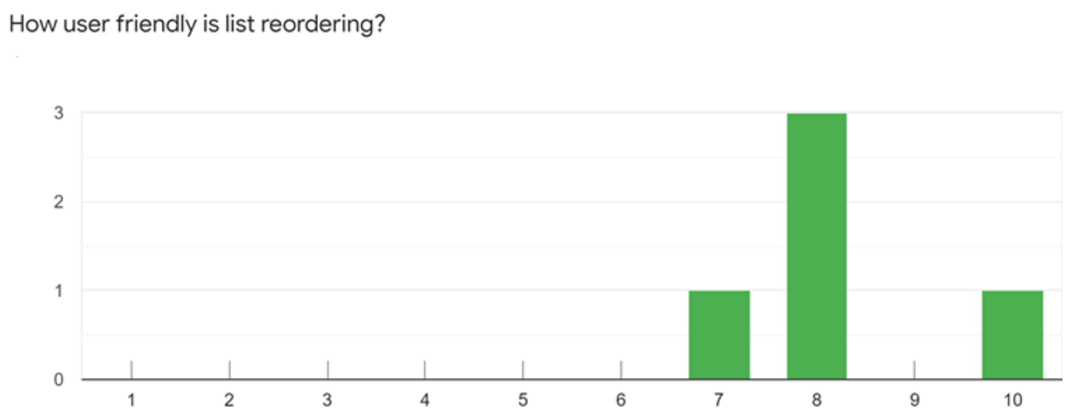


Figure 5.3: User's rating of list reordering feature on scale from 1 to 10.

The only difficulty, the users expressed, was with an error message that shows when the input file does not exist. This error was fixed immediately.

5.3.2 Scenario 2

The second scenario is aimed at loading and saving the configuration. The testers were asked to create a configuration, save it and then change it using a text editor. After the change, the users loaded the configuration back and adjusted it in the GUI. This process was repeated several times, and the users were asked about their preferences regarding changing the configuration. All of them prefers using the GUI to using the text configuration and would prefer so even in everyday use. According to their responses, it is also unnecessary to include the text preview in the main program, which was planned initially but postponed.

5.3.3 Scenario 3

The third scenario required the user to create a series of configurations using either mouse or keyboard only. The recipients stated that their preferred way is using a mouse rather than a keyboard. Mouse drag and drop also achieved a slightly better rating: average 9.0 out of 10, while using the keyboard: 8.6 on average. Creating the configuration is also very friendly to the users, reaching an average user-friendliness score of 9.6 out of 10.

5.3.4 Scenario 4 and Overall Feedback

The task in this scenario was to create a specific configuration, stop its processing, adjust it and then rerun the process. This scenario also tested how the users read the messages. At the time of testing, closing a process window terminated the process after completing the currently processed procedure. Although information about this was presented in the system dialog, none of the users answered correctly when asked about it in the feedback form. Therefore, the RGT process termination method was changed to follow the expectations so that the parsing is terminated immediately.

After completing the fourth scenario, users filled in overall feedback. The recipients stated that although they do not know anything about OSM, they could use the program and understand its operations. The GUI's overall feedback achieved an average rating of 9.0 points out of 10.0 for user-friendliness and the same value for layout and readability.

5.4 Results

The testing has proven that the RGT can be built and run on two platforms: MS Windows and Linux. Although the GUI can also be launched on the macOS, processing the OSM is not available because this platform is not supported by external executables that the processing part uses.

Although none of the recipients had previous experience with OSM processing, they were able to quickly learn the basics from the documentation and use the RGT to perform the desired tasks. They were also able to discover and use the program's main features without the help of the moderator during the Informal Walkthrough.

In comparison with the reference solution, the testers were able to perform a task and fix the mistakes (scenario 1) in 152 % of reference time on average, which we accept as sufficient.

As shown in the Table 5.3, the users were generally pleased with using the program. The worst rating was given to program guidance.

Field name	Average rating
Mouse control	8.6
Keyboard control	8.6
Program communication	9.4
Real-time validation	9.6
Layout and readability	9.0
Monitoring the process	8.8
Hints and guidance	6.0

Table 5.3: Summary of average ratings obtained during user testing.

The user testing was also a good source of inspiration. During the Informal Walkthrough, users were encouraged to speak and share their thoughts. Their ideas significantly helped to enhance the program's communication with the user and revealed several errors in the design.

Chapter 6
Conclusion

The aim of this thesis was to create a user interface for the program Road Graph Tool. After analysing other OSM parsers and the Road Graph Tool itself, we managed to create requirements for this task. Based on the requirements, the JavaFX library was selected for the development of the GUI, using a Model-View-ViewModel architectural pattern.

The program is currently available on two platforms: Linux and Microsoft Windows. Although the GUI may be launched on the macOS, the external executables needed by the processing part of the program do not support this platform. All of the used components are available for free use for non-commercial purposes and are currently being maintained.

The GUI currently provides an environment for the user to create a configuration and execute an RGT process specified by it. The user interface is prepared for future modifications of the code, and developers may add other functionality using a simple interface.

Usability of the user interface was tested by volunteers proving it is easy to learn and therefore suitable even for users with little knowledge of the problematics of map processing. The users' feedback was taken into account and used for further development. However, the second round of the testing was not realised due to time reasons. Hence, some of the program functionality remains untested and may need further attention.

6.1 Future Work

In future, we would like to focus on the second round of user testing to ensure the added functions work as expected and are understood by different users. We would also like to improve the help page on GitHub to attract more people to use the RGT once it is made a public project.

Bibliography

[1] *OSM Beginners Guide 1.3.* URL: https://wiki.openstreetmap.org/wiki/Beginners_Guide_1.3 (visited on 04/20/2021).

[2] *"Source" tag page.* URL: https://wiki.openstreetmap.org/wiki/Key:source (visited on 04/20/2021).

[3] Erik Klimczak. *Design for software. a playbook for developers.* 1st ed. Online: Wiley, 2013. ISBN: 9781119943693.

[4] *Overpass Turbo.* URL: https://wiki.openstreetmap.org/wiki/Overpass_API (visited on 05/08/2021).

[5] *Overpass Turbo.* URL: https://wiki.openstreetmap.org/wiki/Overpass_turbo (visited on 01/07/2021).

[6] *Searching and Downloading OpenStreetMap Data.* URL: https://www.qgistutorials.com/en/docs/downloading_osm_data.html (visited on 05/16/2021).

[7] František Mantlík. *OpenStreetMap Garmin Maps generator User Guide.* URL: https://osm2garmin.blog.mantlik.cz/downloads/ (visited on 01/07/2021).

[8] *Jeography Main Page.* URL: https://jaryard.com/projects/jeography/ (visited on 05/14/2021).

[9] "IEEE Trial-Use Standard for Software Non-Functional Sizing Measurements". In: *IEEE Std 2430-2019* (2019), pp. 1–87. DOI: 10.1109/IEEESTD.2019.8870263.

[10] Waseem Sheikh and Nadeem Sheikh. "A Model-View-ViewModel (MVVM) Application Framework for Hearing Impairment Diagnosis - Type Dependency Architecture". In: *2020 Intermountain Engineering, Technology and Computing (IETC).* IEEE, 2020, pp. 1–6. ISBN: 978-1-7281-4291-3. DOI: 10.1109/IETC47856.2020.9249181. URL: https://ieeexplore.ieee.org/document/9249181/ (visited on 04/25/2021).

[11] Raffaele Garofalo. *Building Enterprise Applications with Windows Presentation Foundation and the Model View ViewModel Pattern.* 1st. USA: Microsoft Press, 2011. ISBN: 0735650926.

[12] Johan Vos et al. *Pro JavaFX 9. A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients.* 4th ed. ProQuest Ebook Central: Apress L. P., 2017. ISBN: 9781484230428. URL: `https://ebookcentral.proquest.com/lib/cvut/detail.action?docID=5199561` (visited on 04/29/2021).

[13] Justin Misfud. *Difference between usability and user experience.* URL: `https://usabilitygeek.com/the-difference-between-usability-and-user-experience/` (visited on 05/07/2021).

[14] Afifa Lodhi. "Usability Heuristics as an assessment parameter. For performing Usability Testing". In: *2010 2nd International Conference on Software Technology and Engineering* (2010). DOI: `10.1109/ICSTE.2010.5608809`. URL: `http://ieeexplore.ieee.org/document/5608809/`.

[15] Jakob Nielsen. *Usability engineering.* San Francisco: Morgan Kaufmann, 1993. ISBN: 01-251-8406-9.

[16] Kara Pernice. *Text Scanning Patterns: The Eyetracking Evidence.* URL: `https://www.nngroup.com/articles/text-scanning-patterns-eyetracking/` (visited on 04/30/2021).

[17] John Moore Williams. *10 Essential UI Design Tips.* URL: `https://webflow.com/blog/10-essential-ui-design-tips` (visited on 04/30/2021).

[18] *Name tags in OSM entities.* URL: `https://wiki.openstreetmap.org/wiki/Names` (visited on 05/08/2021).

[19] Sirpa Riihiaho. "User Testing When Test Tasks Are Not Appropriate". In: ECCE '09. Helsinki, Finland: VTT Technical Research Centre of Finland, 2009. ISBN: 9789513863401.

[20] J. C. Huang. "An Approach to Program Testing". In: *ACM Computing Surveys* 7.3 (1975), pp. 113–128. ISSN: 0360-0300. DOI: `10.1145/356651.356652`. URL: `https://dl.acm.org/doi/10.1145/356651.356652`.

[21] Ross Teague, Katherine De Jesus, and Marcos Nunes Ueno. "Concurrent vs. post-task usability test ratings". In: *CHI '01 extended abstracts on Human factors in computing systems - CHI '01* (2001), pp. 289–. DOI: `10.1145/634067.634238`. URL: `http://portal.acm.org/citation.cfm?doid=634067.634238`.

Appendix A
Acronyms

AIC	Artificial Intelligence Centre
AND	Automotive Navigation Data
API	Application Programming Interface
CSS	Cascading Style Sheets
CTU	Czech Technical University
FEE	Faculty of Electrical Engineering
FR	Functional Requirement
GIS	Geographical Information System
GPL	General Public License
GUI	Graphical User Interface
IT	Information Technology
JBDC	Java DataBase Connectivity
JRE	Java Runtime Environment
HTTP	Hypertext Markup Language
MVM	Main ViewModel
MVVM	Model-View-ViewModel
NFR	Non-Functional Requirement
OSM	Open Street Map
OQL	Overpass Query Language
QL	Query Language
RGT	Road Graph Tool
SRID	Spatial Reference Identifier
UI	User Interface
UX	User Experience
VM	ViewModel

Appendix B

User Testing Feedback Forms

This appendix includes transcriptions of Google Forms used for gathering feedback. Numbers in brackets represent the number of people voting for this option. For example, '(5x)' means five recipients selected this answer. If the brackets are missing, none of the recipients selected this option. Some of the questions required a rating on a scale of 1-10. The 10 is the highest mark. The 1 is the lowest. Please note the answers where the recipients used their own words may be simplified.

B.1 Informal Walkthrough

Transcription of a feedback form used to gather feedback after Informal Walkthrough testing. Six recipients attended the Informal Walkthrough testing.

- How would you describe the level of your computer skills?
  - Advanced (3x)
  - Average (2x)
  - I am a beginner (1x)
- Do you have any experience using computer networks?
  - Yes, I have worked with road networks.
  - I have heard about it but never used it. (2x)
  - No. (4x)
- What operating system are did you use for testing?
  - Microsoft Windows (4x)
  - MacOS (1x)
  - Linux (1x)
- Did you launch the program on the first try?
  - Yes. (3x)
  - No. (3x)
- If not, what was the problem?
  - Old Java version. (2x)

  - Problem launching from the jar on MacOS. (1x)
- Did you understand that the list of procedures on the left represents all procedures, and the list at the bottom represents the process?
  - Yes, without problems. (1x)
  - Yes, but I would appreciate more hints. (5x)
  - No.
- How would you rate the program's hints and overall guidance? `[scale 1-10]`
  - 1 (1x)
  - 5 (1x)
  - 6 (1x)
  - 7 (1x)
  - 8 (1x)
  - 9 (1x)
- How would you rate the program's design? `[scale 1-10]`
  - 8 (3x)
  - 9 (3x)
- What would you like to improve in the program?
  - "When trying to open a new map, it would be useful to remember the previous path, so the user doesn't have to click to the maps again."
  - "I'd prefer if the steps can be reorganized using arrows instead of WASD."

B.2 Scenario Testing

The feedback form for Scenario Testing also included the instructions for performing specific scenario. For clarity, are the scenarios separated into sections. Each section is starting with instructions followed by questions and answers. Five recipients attended the testing.

The feedback started with gathering information about the volunteers:

- What operating system are did you use for testing?
  - Microsoft Windows (4x)
  - Linux (1x)
- Have you worked with RGT before?
  - Yes
  - No (5x)

B.2.1 Scenario 1

- Instructions:
  - This scenario requires time tracking. Start the stopwatch once you open the program. Try to keep precision +-10s.Load configuration config-scenario1.yml from <rgt-folder>/scenarios. This configuration parses the map of Andorra from <rgt-folder>/maps subfolder, so please update the input map first. The configuration includes some errors. Follow the program's error messages and fix and run the process with this configuration. Stop the time tracking once you see a "Successfully finished" dialog.

- How much time [minutes:seconds] did it take to create a configuration and run it?
  - 4:01 (1x)
  - 3:40 (1x)
  - 3:21 (1x)
  - 4:21 (1x)
  - 3:17 (1x)
  - Average time: 3:44
- Did you find the program's communication sufficient?
  - Yes. (4x)
  - Yes, but it could be better. (1x)
  - No, but it was not terrible.
  - Absolutely not.
- Did you use keyboard shortcuts?
  - Yes.
  - No. (5x)
- How friendly is the list reordering? `[scale 1-10]`
  - 7 (1x)
  - 8 (3x)
  - 10 (1x)
- Is there something you would like to add? You may do it here.
  -

B.2.2 Scenario 2

- Instructions:
  - Clear previous configuration. Now create another one with at least 5 procedures and save it in a file. Open this file via text editor and change the order of procedures. Try also changing parameter's values. Now load the configuration. If you kept the expected type of parameters, the configuration is loaded. If not, an error message appears. Adjust this configuration and save it again. Repeat this process 5 times.
- Does the configuration loading work as you expected?
  - Yes. (5x)
  - No.
- Do you prefer GUI or text configuration?
  - GUI. (5x)
  - Textual configuration.
- If you were working with RGT every day, what would you prefer, GUI or text configuration?
  - GUI. (5x)
  - Textual configuration.
  - I am not sure.
- Would you appreciate a preview of textual configuration in GUI?
  - Yes. (1x)
  - No. (4x)
- Is there something you would like to add? You may do it here.

B.2.3 Scenario 3

- Instructions:
  - This scenario is not exactly specified. Please, try to brute force test the program control. Start with using mouse only, then keyboard only and then combine these two approaches.
- Please, rate using keyboard shortcuts. `[scale 1-10]`
  - 6 (1x)
  - 8 (1x)
  - 9 (1x)
  - 10 (2x)

- Please, rate mouse drag and drop.`[scale 1-10]`
  - 8 (2x)
  - 9 (1x)
  - 10 (2x)
- What is your preferred way to control the RGT?
  - Using keyboard shortcuts. (1x)
  - Using the mouse. (4x)
- Is there something you would like to add? You may do it here.

B.2.4 Scenario 4

- Instructions:
  - Create a configuration parsing the map jihocesky.osm.pbf (can be found in /maps subfolder) and saving it to your database. Also add procedures NodeProcessing(srid = 32633, WayProcessing (only roads for pedestrians), WholeGraphProcessing. Close the process window approximately in the middle of the process and add a GraphIDsGenerator procedure. Then process it again.
- When you close the process window, what happens with the running parsing process?
  - I am not sure. (2x)
  - Procedure will stop after completing the procedure. `correct`
  - Process will stop immediately (the thread is killed). (3x)
  - Process will stop as soon as possible but does not have to complete the procedure.
- Do you like the current way the process window displays over the main window? Should it remain modal?
  - I do not mind the way it is. (5x)
  - I think the Process window should NOT be modal so I can work with the Main window while the process is running.
- How friendly is the environment for creating configurations? `[scale 1-10]`
  - 9 (2x)
  - 10 (3x)
- How do you like the information overview about the parsing process? `[scale 1-10]`
  - 7 (1x)
  - 8 (1x)
  - 9 (1x)

  - 10 (2x)
- Is there something you would like to add? You may do it here.
  - "The modal process window forbids me from, for example, minimising the program, which is annoying but nothing I cannot handle."

B.2.5 Overall feedback

At the end of the Scenario testing, the users were asked to fill in overall feedback .

- Did you find any unhandled errors during the testing?
  - Yes.
  - No. (5x)
- How were you satisfied with the overall program's communication? `[scale 1-10]`
  - 9 (3x)
  - 10 (2x)
- Was there a situation when you didn't know what was happening due to the lack of information from the program's side? If yes, please describe it.
  -
- Did you understand all of the error messages?
  - Yes, without problems. (5x)
  - Yes, but with some problems.
  - No, some of them were very unclear.
- How do you rate parameter validation? `[scale 1-10]`
  - 9 (2x)
  - 10 (3x)
- How easy is the user interface to learn and use?
  `[1 - very difficult to use, 10 - very easy to use]`
  - 8 (1x)
  - 9 (3x)
  - 10 (1x)
- Please rate the program's layout and readability. `[scale 1-10]`
  - 8 (1x)
  - 9 (3x)
  - 10 (1x)
- Is there something you would like to add? You may do it here.
  - "Add some textual descriptions to the process list, the colour highlights are not enough." (1x)
  - "I would add the navigation using `WSAD` also to Arrows." (3x)

Appendix C

Detailed Overview of Procedures

Inputs	
OSMFile	Filters and loads road network from .osm files into program's graph representation. Uses two external executables, osmfilter and osmconvert, to process the map.
Parameters	inputFile [string]

Outputs	
DbImporter	Provides connection and saving to a PostgreSQL database representation.
Parameters	Connetion parameters - *Address*, *Port*, *Database name*, *User*, *Password*

Steps	
NodeProcessing	Creates a spherical projections of GPS coordinates so Earth curvature and elevation is taken into account during the computation of distances
Prerequisites	-
Parameters	srid (Spatial Reference ID) - code representing coordinate system and type of projection
WayProcessing	Unifies speed units, keeps roads selected types of vehicles can use
Prerequisites	NodeProcessing
Parameters	*Allowed transport modes* - means of transport that the user will use in the simulation. Roads for omitted means of transport are deleted.
WholeGraphProcessing	Simplifies graph, removes nodes that are not crossroads, recomputes length of roads.
Prerequisites	NodeProcessing, WayProcessing
Parameters	-
GraphIDsGenerator	Generates new IDs for entities (from 1 to N).
Prerequisites	-
Parameters	-
MajorConnectedComponent	Creates a major strongly connected component of the graph. Finds the biggest connected part of the map and deletes others.
Prerequisites	-
Parameters	-
MaxSpeedValidation	Logs all entities, where max speed is not in interval <minmaxspeed; maxspeed> to a file.

Prerequisites	WayProcessing
Parameters	*maxspeed* - upper limit of maximum permitted speed on a road *minmaxspeed* - lower limit of maximum permitted speed on a road
ElevationValidation	Logs all entities, where max speed is not in interval <minmaxspeed; maxspeed> to a file.
Prerequisites	NodeProcessing, WholeGraphProcessing
Parameters	*maxElevation* - upper limit of maximum permitted speed on a road *minElevation* - lower limit of maximum permitted speed on a road *maxGradient* - maximal slope of the road *minEdgeLength* - minimal length of the road

Table C.1: A list of procedures, their parameters and prerequisites.

Appendix D
List of Attachments

- `./source-code/` - Source code of the program.

the