Appendix C

How-to guide

In this section, the required software and its specifications for using both the Kern dispatcher and MOSAIC framework are explained in detail. The version and system parameters used during this thesis are pointed out. This does not exclude other versions or operating systems from being compatible with this implementation.

C.1 System specifications

System:

- Windows 11
- 16 GB RAM
- Intel Core i7-10750H CPU
- WSL with Ubuntu 24.04
- Python v3.13.3

Software specifications for MOSAIC:

- MOSAIC version 25.0 or 25.1
- SUMO v1.24.0; you have to add SUMO to your PATH variables
- IntelliJ Ultimate v2025.2.3
- Java 21 Eclipse Temurin JDK

Software specifications for Kern:

- VS Code

- MySQL v8.0.41
- Rustup: Rust installer and package management tool (includes cargo); you will have to install it in WSL

C.2 How to run and test the new implementation

Before starting, make sure you have read the documentation of the MOSAIC framework. The same applies for the *README* file in Kern's repository. See Appendix B for useful links. The whole implementation was executed and tested in a Windows 11 environment, therefore there is no guarantee that it will work on macOS or Linux without any adjustments.

1. In IntelliJ, after you have cloned the Mosaic repository, you should once run the `mvn clean install` command in order to build all the necessary dependencies and classes.
2. For development and testing, you can start the application using the `MosaicStarter#main` method. The other way is already described in the MOSAIC documentation and was not used for the purposes of this thesis.
3. Do not forget to set the flag '`-w 0`' during simulations, which turns off the timeout watcher, otherwise you will often get timeout exceptions.
4. In the `TaxiDispatchingServer` class there are scenario custom variables, which should be adjusted before starting the simulation. It is very important to set the correct paths to your files, otherwise the execution would not work.
5. Make sure that you have the `python` command added to your PATH variables, because it is used to execute DB-related python scripts as a preparation for the simulation. If not, you should set the full path to the executable in the `ExternalFilesUtil#executePythonScripts` method.
6. Afterwards, you have to check if you have installed your WSL correctly and can access it using the `wsl` command. This is necessary if the Kern dispatcher is started via IntelliJ. However, this approach does not work correctly and is slower, therefore it is better to start the dispatcher manually.
7. Make sure your MySQL database is running and that you have the necessary tables. You could use the `setupTables.py` script from the `pythonScripts` folder in MOSAIC to create them from scratch. The `executeScripts.py` file can also fill the tables with all necessary simulation data.
8. You have to generate the distances file using IntelliJ, by starting a normal simulation (see the last step) and setting the flag `CREATE_DISTANCES_FILE_AND_TERMINATE`

to true. It will create the file in the Kern project and after it is done, the simulation will terminate. Do not forget to add the path to the Kern project in the `TaxiDispatchingServer` class.

9. Build and run the Kern project once in the WSL using the `cargo run --release` command in order to start the dispatcher faster for the simulation. You have to adjust the `kern.toml` file according to your needs before using the dispatcher.

10. If all the aforementioned steps were executed correctly, feel free to start the simulation. You can also implement other scenarios, but you have to change the name of the `SCENARIO_NAME` variable in the `TaxiDispatchingServer` class. Keep in mind that you have to keep the same structure as the currently available scenario folders.

C.3 Error handling

- If an error occurrs in MOSAIC, the exception can be found in the log files `rti/mosaic-starter/logs` directory. Look inside the `MOSAIC.log` and the `Traffic.log` files.
- Sometimes, it is possible that the WSL quits unexpectedly while running the Kern dispatcher with higher levels of demand. This is likely caused by the fact that it is run on a Windows Subsystem for Linux instead of directly using a Unix-based operation system. It has to be further explored.
- Look also in the `TaxiDispatchingServer.log` from the `rti/mosaic-starter/logs/<logName>/apps/server_0` directory for eventual errors or warnings that are logged instead of throwing an exception. This might explain an unexpected behavior.