VSimRTI: Vehicle-2-X Simulation
Runtime Infrastructure

User Documentation

Michalis Adamidis     Chris Kay Baumann     Kai Lennert Bober
Manuel Bock           Roland Cristea        Sebastian Dunkel     Daphne Grape
Jan Henning           Adrian Herrmann       Jiajun Hu            Karl Hübner
Sven Erik Jeroschewski Franz Kage           Bernhard Ladenthin    Nico Naumann
Robert Protzmann      Tobias Queck         Stefan Reichel       David Rieck
Denis Röper           Stefan Rulewitz       Aleksandr Sandalov   Severin Sander
Alexander Scheck      Erik Schleiff         Björn Schünemann     Julia Ullrich
Sascha-Gaetano Urso   Michael Weiner

Version 19.1
October 18, 2019
Abstract

The Vehicle-2-X Simulation Runtime Infrastructure (VSimRTI) enables the preparation and execution of Vehicle-2-X (V2X) simulations. It is a flexible system which simulates traffic flow dynamically. VSimRTI couples different simulators and thus allows the simulation of the various aspects of intelligent transportation systems. The easy integration and exchangeability of simulators enables the utilization of the most relevant simulators for a realistic presentation of vehicle traffic, emissions, wireless communication and the execution of V2X applications.

The developer alliance

The developer alliance consists of Fraunhofer-Institut für Offene Kommunikationssysteme (FOKUS), Daimler Center for Automotive Information Technology Innovations (DCAITI) and Automotive Services and Communication Technologies (ASCT).

Contact information

VSimRTI Mailing List (developer team)

vsimrti@fokus.fraunhofer.de

VSimRTI: Smart Mobility Simulation

http://www.dcaiti.tu-berlin.de/research/simulation

Fraunhofer FOKUS: ASCT Competence Center

https://www.fokus.fraunhofer.de/asct

DCAITI

http://www.dcaiti.tu-berlin.de
## Contents

1 Introduction .................................................. 1
   1.1 Overview .................................................. 1
   1.2 Download and install ..................................... 2
   1.3 Update .................................................... 2
   1.4 Compatibility Matrix ...................................... 2
   1.5 License ................................................... 3
   1.6 Concept ................................................... 4
      1.6.1 Federates and Ambassadors ......................... 4
      1.6.2 Scenario configuration .............................. 4

2 Run simulations .............................................. 5
   2.1 Run a single simulation via CLI ......................... 5
      2.1.1 VSimRTIEmbeddedStarter ............................ 7
   2.2 Results ................................................... 8
      2.2.1 Logging ............................................... 8
   2.3 Run a simulation set via CLI ............................. 9

3 Simulators ................................................... 10
   3.1 Kinds of simulators ...................................... 10
      3.1.1 Network simulators .................................. 10
      3.1.2 Traffic simulators ................................. 11
      3.1.3 Application simulators ............................. 11
      3.1.4 VSimRTI communication simulators ................. 11
      3.1.5 Environment simulators ............................. 12
      3.1.6 Electricity simulators .............................. 12
   3.2 OMNeT++ .................................................... 13
      3.2.1 omnetpp-ambassador folder structure ............... 13
      3.2.2 Installation ........................................... 14
      3.2.3 Installation in Docker environment ................. 20
      3.2.4 Configuration ......................................... 16
   3.3 ns-3 .......................................................... 17
      3.3.1 ns3-ambassador folder structure ................... 18
      3.3.2 Installation .......................................... 18
      3.3.3 Installation in Docker environment ................. 20
      3.3.4 Configuration ......................................... 21
   3.4 SUMO ........................................................ 23
      3.4.1 sumo-ambassador folder structure .................. 23
      3.4.2 Installation .......................................... 23
      3.4.3 Configuration ......................................... 23
      3.4.4 Using the SUMO GUI with VSimRTI ............ 24
      3.4.5 Routes and Vehicle Types ........................... 24
      3.4.6 Further information ................................ 25
      3.4.7 VSimRTI specific information ..................... 25
   3.5 VSimRTI Application Simulator .......................... 26
      3.5.1 applicationNT-ambassador folder structure .......... 26
      3.5.2 Installation .......................................... 26
      3.5.3 Libraries ............................................. 26
      3.5.4 Basic functionality ................................ 26
7 Tutorial LuST
  7.1 Execute the LuST scenario with VSimRTI .................................. 66
  7.2 Current limitations ................................................................. 66

8 Tutorial Traffic Management
  8.1 Scenario Overview ................................................................. 67
  8.2 Detector configuration ........................................................... 67
  8.3 Configuration of Traffic Management Centers ............................ 68
  8.4 Traffic Management Center Applications .................................. 69

9 Create a new scenario
  9.1 Road network data ................................................................. 70
  9.1.1 Using OpenStreetMap data ................................................. 70
  9.2 Vehicles and Routes .............................................................. 72
  9.3 VSimRTI .......................................................... 73

10 Visualizers
  10.1 Kinds of Visualizers ............................................................ 76
  10.2 VSimRTI WebSocket Visualizer .............................................. 77
  10.3 VSimRTI File Visualizer ....................................................... 79
    10.3.1 Configuring the File Visualizer ...................................... 79
  10.4 Integrated Test and Evaluation Framework (ITEF) ....................... 83
  10.5 VSimRTI 3D Visualization (PHAPMap) .................................... 85
  10.6 VSimRTI Statistics Visualizer .............................................. 86
    10.6.1 Configuration of Statistics Visualizer .............................. 86
    10.6.2 Specification of the Statistics Visualizer .......................... 86
    10.6.3 Application of the Statistics Visualizer ............................ 87

11 Run simulation series
  11.1 Simulation Runner ............................................................. 89
    11.1.1 Usage .............................................................. 89
    11.1.2 Configuration ....................................................... 90
    11.1.3 Parametrization ...................................................... 92
    11.1.4 Additional Information ............................................. 94

12 Additional tools
  12.1 scenario-convert ............................................................. 95

13 VSimRTI configuration
  13.1 Overview ................................................................. 100
  13.2 Federates configuration ..................................................... 100
    13.2.1 Federate priorities .................................................. 101
  13.3 Host configuration .......................................................... 101

14 Additional information
  14.1 PATH configuration ........................................................... 102

15 List of Acronyms

A VSimRTI deployment
  A.1 VSimRTI Folder Structure .................................................... 106
  A.2 File listings ................................................................. 108

B Example scenario Barnim
  B.1 Folder Structure ............................................................ 118
  B.2 File listings ................................................................. 119

C Example scenario Tiergarten
  C.1 Folder Structure ............................................................ 134
C.2 File listings ................................................................. 135

D Package Additional Examples ....................................... 137

E Configuration Schemata .............................................. 140

F Message Dictionary .................................................... 146

G References ............................................................... 151
1 Introduction

1.1 Overview

This documentation is part of the current release of VSimRTI and aims at supporting users in their first steps with the software. We try to explain most of VSimRTI's features and configurations by providing two tutorials named Barnim (see chapter 5) and Tiergarten (see chapter 4).

If you need more information regarding VSimRTI or a specific configuration, we ask you to take a look at the Doxygen documentation provided alongside the current release. Documentation is available for all JSON style configs. Apart from that the appendix provides information for the respective XML style configurations.

If you have any questions or need further support, please feel free to contact our support team via our mailing list. Please attach relevant log files and scenario files to your inquiry. We look forward to receiving your feedback concerning further improvements of the VSimRTI documentation as well as the installation and configuration process.

VSimRTI at a glance

VSimRTI was built to provide users with the flexibility to perform various V2X simulations with their own choice of simulators. In order to provide the flexibility to exchange a simulator without changing the underlying infrastructure, VSimRTI offers several interfaces for the integration of different simulators, e.g. for network, traffic, and environment simulations. For the synchronization and the communication purposes, the implemented infrastructure uses common concepts defined in the Institute of Electrical and Electronics Engineers (IEEE) standard for modelling and simulation (M&S) High Level Architecture (HLA). Thus, the runtime infrastructure VSimRTI allows a flexible combination of time-discrete simulators for V2X simulations. Based on the (possibly differing) requirements of a specific scenario, arbitrary simulators can be added to VSimRTI and will be executed together.

VSimRTI is written in the programming language Java and deployed as Java Archive (JAR) files. Consequently, a compatible Java Runtime Environment (JRE) for your operating system must be installed to execute VSimRTI. We recommend the Java 8 version of AdoptOpenJDK, which can be found at https://adoptopenjdk.net/releases.html?variant=openjdk8&jvmVariant=hotspot.
1.2 Download and install

- Download vsimrti-bin-19.1.zip from the DCAITI website.
- The package vsimrti-bin-19.1.zip has to be extracted to an arbitrary path. This installation path is referenced as vsimrti throughout the entire document.

1.3 Update

In order to update VSimRTI to a new version, please perform the following steps manually:

- Backup your personal simulation scenarios from VSimRTI's scenarios directory.
- Remove your old VSimRTI installation completely.
- Install VSimRTI according to the previous section.
- Copy your simulation scenarios back to VSimRTI's scenarios directory.
- Be aware of possible updates of the used software and simulators from third party (see the Compatibility Matrix below).

1.4 Compatibility Matrix

The following table gives an overview of supported environments and simulators. Please make sure that you install those versions only.

<table>
<thead>
<tr>
<th>Component</th>
<th>Required</th>
<th>Operating System</th>
<th>Java</th>
<th>SUMO</th>
<th>OMNeT++</th>
<th>INET</th>
<th>ns-3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Windows</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating System</td>
<td>yes</td>
<td>yes yes</td>
<td>Java 7 and below no</td>
<td>0.32.0 and below no</td>
<td>4.6 and below no</td>
<td>3.5 and below not tested</td>
<td>3.27 and below no</td>
</tr>
<tr>
<td>Java</td>
<td>yes</td>
<td>Java 7 and below no</td>
<td>Java 8 yes</td>
<td>1.0.1 to 1.2.0 yes</td>
<td>5.3 yes</td>
<td>3.6 yes</td>
<td>3.28 yes</td>
</tr>
<tr>
<td>SUMO</td>
<td>yes</td>
<td>0.32.0 and below no</td>
<td>1.0.1 to 1.2.0 yes</td>
<td>above 1.2.0 not tested</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OMNeT++</td>
<td>optional</td>
<td>4.6 and below no</td>
<td>5.3 yes</td>
<td>5.4 and above no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INET</td>
<td>optional</td>
<td>3.5 and below not tested</td>
<td>3.6 yes</td>
<td>4.0 and above no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ns-3</td>
<td>optional</td>
<td>3.27 and below no</td>
<td>3.28 yes</td>
<td>3.29 and above not tested</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1.5 License

The licensing mechanism was introduced for a better release control. It improves user support whilst helping our developer team to deactivate outdated releases which cannot be maintained anymore. Every user needs to be registered at the license server to obtain a license.

1. Running the `firstStart` script in the VSimRTI root folder causes the file `vsimrti/systeminfo.txt` to be generated. This file is also generated when VSimRTI is run without a valid license file or an invalid user.

2. The created file contains basic information in plain text about the machine on which VSimRTI was executed and is used to identify the user. The system info needs to be sent to the VSimRTI mailing list. A staff member registers the new user at the VSimRTI license server as soon as possible, usually within one workday. When your license is active, an email containing your user id will be sent to you.

3. The following information is stored to identify your machine:
   - central processing unit (CPU) model id
   - CPU cores
   - CPU architecture
   - Media Access Control (MAC) addresses
   - operating system name
   - operating system version
   - random-access memory (RAM) size
   - hashed user id
   - VSimRTI version

4. Your license will be activated directly after confirmation by the VSimRTI team. On the next VSimRTI run with an available Internet connection to the VSimRTI license server a valid license is generated and a local copy is stored in the VSimRTI folder.

5. The local license files `vsimrti/vsimrti-license.lcs` and `vsimrti/vsimrti.dat` are valid for the next 14 days. In this period no Internet connection is needed for the execution of VSimRTI. Every time when an Internet connection to the license server can be established, the local license file will be renewed for another 14 days.

   ☢️ **Notice:** You should not backup the license files. If you have a registered account and a valid license is present on the license server, you will always receive a new license file.

   ☢️ **Notice:** If you encounter any problems with corrupt license files, you can also use the `firstStart` script to reset the local license files and retrieve a new copy during the next run.
Notice: If the firstStart script does not result in a vsimrti/systeminfo.txt file, please check if you have sufficient rights on your machine. You can also try to start the vsimrti script instead, by calling vsimrti.sh -u <YOUR-EMAIL-ADDRESS> -s Barnim. This command will always fail, due to the missing license, however, it may generate the required systemInfo.txt.

### 1.6 Concept

In contrast to existing fixed simulator couplings VSimRTI allows easy integration and exchangeability of simulators. Depending on your specific requirements, the high flexibility of VSimRTI enables you to couple the most appropriate simulators for a realistic representation of vehicle traffic, emissions, wireless communication, and the execution of V2X applications.

#### 1.6.1 Federates and Ambassadors

VSimRTI uses the federate-ambassador concept inspired by the concept of the HLA. Using this concept, it is possible to couple different simulation systems with a remote control interface. Attaching an additional simulator only requires implementation of the appropriate ambassador interface and the possibility to execute its specified commands.

#### 1.6.2 Scenario configuration

Any scenario needs to be configured with a configuration file. The specific path to this file is vsimrti/scenarios/<scenarioName>/vsimrti/vsimrti_config.xml.
2 Run simulations

<i>Notice:</i> Install the required simulators for the specific scenario and have a valid license (see section 1.5).

2.1 Run a single simulation via CLI

To run a single simulation via Command Line Interface (CLI), call the **VSimRTI** start script with the following command line arguments. The **VSimRTI** start script supports the following arguments.

- **-c, --configuration** The primary **VSimRTI** configuration file which is scenario dependent and located in the according scenario folder. This file transitively includes other necessary configuration files. Usually you will use the file `vsimrti/scenarios/<scenarioName>/vsimrti/vsimrti_config.xml`.

- **-s, --scenario** If the **VSimRTI** configuration file of your scenario is located in the default scenario directory of **VSimRTI** (i.e. in `vsimrti/scenarios/<scenarioName>/vsimrti/vsimrti_config.xml`), this option can be used instead of the **-c** option by passing only the scenario name (`-s <scenarioName>`).

- **-u, --userid** The id of the **VSimRTI** user which was assigned with the authentication for the **VSimRTI** release area. For better release control and support of **VSimRTI** users, a user id is needed to start a simulation. The user id is connected to a dedicated **VSimRTI** license.

- **-w, --watchdog-interval** The interval of the internal alive check (in seconds) which is used by **VSimRTI** to detect execution stalls. This parameter is not mandatory and it is also possible to turn off the watchdog (`-w 0`) for debug sessions.

- **-o, --loggingLevelOverride** Override all specified logging-levels. This option is useful for debugging simulations. For example logging every possible event would be done with `-o TRACE`.

- **-b, --realtimeBrake** With this parameter, the simulation will be slowed down to a desired Real Time Factor, if possible. When simulations already run slower than real time, this factor will have no effect. For example, use `-b 1` to execute the simulation in real time.

- **-v, --start-visualizer** Opens a page in your default browser which visualizes all vehicle movements of the simulation on a map. This option only works, if your scenario has configured the websocket visualizer (see section 10.2).
GNU/Linux

Listing 2.1: VSimRTI GNU/Linux start command.
```
./vsimrti.sh -c ./.scenarios/<scenario name>/vsimrti/vsimrti_config.xml -u userid
```

Microsoft Windows

Listing 2.2: VSimRTI Microsoft Windows start command.
```
vsimrti.bat -c .\scenarios\<scenario name>\vsimrti\vsimrti_config.xml -u userid
```

While VSimRTI is running, it prints some information on the command line:

```
[user@gnulinux vsimrti]$ ./vsimrti.sh -u userid -s Barnim
2019-03-28 15:52:32,217 INFO  ROOT - Running VSimRTI 19.0 on Java JRE v1.8.0_162 (Oracle Corporation)
2019-03-28 15:52:33,896 INFO  LicenseCheck - writing license file
2019-03-28 15:52:33,908 INFO  g - License valid
2019-03-28 15:52:33,917 INFO  FederationManagement - Federation with id=Barnim is started
2019-03-28 15:52:33,919 INFO  FederationManagement - join federate with id sns
2019-03-28 15:52:33,936 INFO  FederationManagement - Start sumo local in ./tmp/tmp_Barnim_1553784750183_0/sumo
2019-03-28 15:52:33,937 INFO  FederationManagement - join federate with id applicationNT
2019-03-28 15:52:33,937 INFO  FederationManagement - join federate with id battery3
2019-03-28 15:52:33,937 INFO  FederationManagement - join federate with id mapping3
2019-03-28 15:52:33,937 INFO  FederationManagement - join federate with id cell2
2019-03-28 15:52:33,937 INFO  FederationManagement - join federate with id visualizer
15:52:38 - Simulating: 25000000000ns (25.0s)-62.0% (RTF:1,96, ETC:15s)
```

The current simulation progress is shown in the following format.

- current wall clock time
- current simulation time in [ns] and [s]
- progress in %
- Real Time Factor (RTF) and Estimated Time to Completion (ETC)

The RTF is the ratio of simulated time to simulation duration in wall clock time, e.g. a real time factor greater than 1.0 means, the simulation is running faster than real time. Both RTF and ETC are calculated based on the performance of the last five seconds of the simulation and should only give a rough overview, how long a simulation can take. Depending on the simulation setup, the values can differ heavily between start and end of a simulation.

Gather results

All active simulators as well as the according ambassadors generate certain logging output, depending on their configured logging level. Therefore, these logs are very helpful to retrace and understand the individual states during the simulation time.

However, these logs do not conform to a generic formatting. For uniformly formatted or visually prepared results, VSimRTI offers different Visualizers. Please refer to Chapter 10 for more details regarding all visualizers, and more specifically to Section 10.3 for formatting possibilities of simulation results with the FileVisualizer.
### 2.1.1 VSimRTIEmbeddedStarter

Notice: The following options are only available with a commercial license of VSimRTI. For further information on licenses, please refer to our mailing list.

The VSimRTIEmbeddedStarter is a customizable Java start program. You can also use this program as a template to embed VSimRTI in your Java application. The embedded starter has a user friendly CLI and the most of the options as the usual start scripts.

Listing 2.3: VSimRTI VSimRTIEmbeddedStarter start command.
```
  -s scenarioId
```

Listing 2.4: VSimRTI VSimRTIEmbeddedStarter help command.
```
usage: java -javaagent:VsinrtiEmbeddedStarter.jar -jar VsinrtiEmbeddedStarter.jar 
-b,-- realtimeBrake <REALTIMEFACTOR> Set value for real time brake.
-c,-- config <PATH> Path to Scenario configuration file (vsimrti_config.xml). Can be used instead of "-s" parameter. (mandatory).
--classpath Class paths. Default is "." (working directory).
-d,-- defaults-file <PATH> Path to VSimRTI configuration file (default: etc/defaults.xml)
-e,-- externalWatchDog <PORTNUMBER> Specific external watchdog port number
-h,-- hosts <PATH> Path to host configuration file (default: etc/hosts.json)
--help Shows this help information.
--logback path <PATH> Use specific logback.xml
-o,-- loggingLevelOverride <LOGLEVEL> Overrides the log level to new value (e.g. DEBUG)
--pipe Use pipe redirect (only for --process-builder
--print-jar-files Print the found jar files.
--print-parameter Print the parameter before start.
-s,-- scenario <NAME> The name of the VSimRTI scenario. Can be used instead of "-c" parameter. (mandatory)
--simrun Use the VSimRTI simulation runner.
--strict-configuration-path The starter pass-through the configuration path as it is.
-u,-- user <USERID> Your UserId (mandatory). Please read the user manual if you do not have a UserId yet.
-v,-- start-visualizer Starts the web socket visualizer.
--verbose Print information at start.
--version Print the version information and exits.
-w,-- watchdog-interval <SECONDS> Kill VSimRTI process after n seconds if a federate is not responding. 0 disables the watchdog. (default: 30)
2.2 Results

**VSimRTI** generates logfiles for each simulation run. Logs are generated for the ambassadors of each coupled federate respectively simulator and for **VSimRTI** itself.

The logfiles are stored in the folder `vsimrti/logs/log-<timestamp>` For each simulation run a new folder is created.

In addition to standard logging output for each federate there is a `statistics.csv` file which contains detailed information about sent and received **VSimRTI** messages. This file can be used to trace a simulation run.

```
log-<timestamp>
  appsNT ...................................................... Detailed application specific logs for each unit.
  <unitType>_<unitId> .................. Detailed application specific logs for each unit.
  OperatingSystem.log .................. Detailed operating system logs for the unit.
  ExampleApp.log .................. Detailed application specific logs for each application.
  ApplicationNT.log ................ Information about the application ambassador.
  Battery.log .................. Battery ambassador log for simulation of electric vehicles.
  Cell2.log .................................................... Cellular network log.
  ChargingStation.log .................. ChargingStation ambassador log.
  Communication.log .................. (Ad hoc) network simulation ambassador log.
  CommunicationDetails.log ........ Detailed output of network simulator (ns-3 or OMNeT++).
  Environment.log .................. Logs of the environmental eventserver.
  Mapping.log .................. Mapping configuration logs.
  Navigation.log .................. Detailed logs about navigation component in the application ambassador.
  statistics.csv .................. Simulation overview in comma separated value-format.
  Traffic.log .................. Traffic simulation log (SUMO or others).
  visualizer.csv ................. Recorded data of the **VSimRTI** File Visualizer.
  VSimRTI.log ............ General **VSimRTI** information, e.g. startup sequence information.
```

Figure 2.2: **VSimRTI** log folder structure

### 2.2.1 Logging

The main configuration file is `vsimrti/etc/logback.xml`. In this file, each output can be configured in great detail. This file can be adjusted to your needs, e.g. you can set up a more detailed logging for communication components but set a less verbose output for **VSimRTI** internal messages or traffic simulation depending on your simulation focus.

**VSimRTI** uses LOGback as logging framework and it is suggested to use LOGback for the other simulators as well. LOGback offers a lot of parameters to adapt the output to your needs. Please refer to [http://logback.qos.ch/manual/layouts.html#ClassicPatternLayout](http://logback.qos.ch/manual/layouts.html#ClassicPatternLayout) for a detailed overview of all parameters you can use in the logback.xml file.

Please note that you can adjust the output to your needs by setting different log levels (`ERROR`, `INFO`, `DEBUG` etc.) for each component in the file at `vsimrti/etc/logback.xml`. This might also influence the simulation performance because of a possibly high amount of data to be logged.
**Federate specific logging**

Depending on the simulation purpose, further configuration possibilities for federate specific logging may be of interest.

For instance, OMNeT++ exhibits an elaborated logging concept. The `omnetpp.ini` in the scenario folder includes options to adjust the logging levels. The outputs of this federate are written to `Communication-Details.log`.

### 2.3 Run a simulation set via CLI

**Notice:** The following options are only available with a commercial license of VSimRTI. For further information on licenses, please refer to our [mailing list](#).

A common objective when running simulations is to assess the impact of different parameter settings for a specific scenario on the results of the simulation. To this end, the simulation-runner is a tool to apply different configurations to a scenario, after which a series of simulations is executed via CLI by calling the simulation-runner start script. Please refer to 11.1.
3 Simulators

VSimRTI couples different simulators and can't be run alone. Therefore, it requires pre-installed simulators. In this chapter we give an overview of common simulators already supported as well as basic configuration help. For further information and configuration options, please see the javadoc documentation provided on the VSimRTI website. To run other simulators than the provided ones, an additional component has to be written, which couples the simulator to VSimRTI. If you have any questions or need further support, please feel free to contact our support team via our mailing list.

3.1 Kinds of simulators

The figure 3.1 gives an overview of the currently available simulators for VSimRTI

```
vsimrti
  | network simulators .................................................. (see 3.1.1)
  |   Objective Modular Network Testbed in C++ (OMNeT++) ............... (see 3.2)
  |   network simulator 3 (ns-3) ........................................ (see 3.3)
  | traffic simulators .................................................. (see 3.1.2)
  |   Simulation of Urban Mobility (SUMO) ................................ (see 3.4)
  | application simulators ............................................... (see 3.1.3)
  |   VSimRTI Application Simulator (built in) ......................... (see 3.5)
  |   VSimRTI Mapping (built in) ........................................ (see 3.5.15)
  |   VSimRTI navigation simulator (built in) ........................... (see 3.5.14)
  | VSimRTI communication simulators ................................... (see 3.1.4)
  |   VSimRTI Cellular Simulator (built in) ............................. (see 3.6)
  |   VSimRTI Simple Network Simulator (SNS) (built in) ............... (see 3.7)
  | environment simulators .............................................. (see 3.1.5)
  |   VSimRTI Eventserver (built in) ................................... (see 3.8)
  | electricity simulators .............................................. (see 3.1.6)
  |   VSimRTI Battery Simulator (built in) ............................ (see 3.9)
```

Figure 3.1: VSimRTI simulator structure

3.1.1 Network simulators

A network simulator is a piece of software modeling the behavior of a network by calculating the interactions between all participating entities. For VSimRTI and, more specifically, the communication in a V2X environment this refers to simulations of the wireless transmissions taking place between the various simulated entities. The provided simulators are prepared to simulate a communication stack basing on the IEEE 802.11p network interfaces with IP and UDP on top. However, according to their model base,
alternative set ups are possible. Currently, VSimRTI supports the commonly known network simulators OMNeT++ and ns-3, and the built in communication simulator SNS.

### 3.1.2 Traffic simulators

A traffic simulator is a software modeling the movements of users in a traffic system. Users can mean cars, pedestrians, trains, ships, planes etc. Traffic simulators can be discriminated by various measures, of which one is the used scope:

**Microscopic models** Simulate each car individually and through the interaction of multiple cars also traffic flows. They are commonly used for situations such as a traffic crossing or an on-ramp situation.

**Macroscopic models** Models of a traffic flow without the modelling of individual cars. Instead the traffic flow is computed using models derived from fluid dynamics or gas-kinetics. By this the simulation is computationally far less expensive, so more cars and wider areas can be simulated. An example would be the prediction of traffic jams.

**Mesoscopic models** Try to bridge the gap between macroscopic and microscopic simulation using individual vehicles that are being actuated by macroscopic control variables.

**Sub-Microscopic models** Used to simulate a car in as much detail as possible. The prediction of the behavior is the most precise of all models, but also computationally the most expensive.

The currently supported traffic simulator SUMO is a microscopic traffic simulator.

### 3.1.3 Application simulators

An application simulator is an important component enabling the simulation and creation of V2X applications. It follows the European Telecommunications Standards Institute (ETSI) standards for Vehicle-to-Vehicle (V2V) communication. These standards contain message definitions like Decentralized Environmental Notification Messages (DENM) and Cooperative Awareness Messages (CAM) and also a general application container design.

### 3.1.4 VSimRTI communication simulators

For different reasons of convenience or extended analysis capabilities, VSimRTI ships with two additional communication simulators:

- The Cellular Simulator is a special case of network simulator to simulate the communication taking place within a cellular network, such as Universal Mobile Telecommunications System (UMTS) and Long Term Evolution (LTE).

- The Simple Network Simulator is written in Java and already tightly integrated in VSimRTI. It offers a lightweight and fast solution for simulation of ad hoc communication.

The provided VSimRTI communication simulators can be used as an alternative (SNS) or addition (Cell2) to classic network simulators.
3.1.5 Environment simulators

This kind of simulator simulates certain environmental events such as rain, fog, snow, etc..

3.1.6 Electricity simulators

In connection with VSimRTI, electricity simulators enable investigations in the field of electric mobility. For this purpose, VSimRTI ships with the built in VSimRTI Battery Simulator.
3.2 OMNeT++

OMNeT++ itself is solely the simulation platform for discrete-event systems. Even though it is primarily targeted at simulating computer networks and distributed systems, it cannot be used without any extensions for wireless communication. For this kind of simulations, external model frameworks have to be included. Currently there are two prominent model frameworks which cover whole model suites for according focus of wireless research. These are the Mobility Framework and the OMNeT++ - Model library for wired, wireless and mobile networks (INET) Framework. As INET provides all models necessary for simulating Vehicle-2-X communication, it is selected for the integration to VSimRTI.

For more information on the INET extension you should look closer on the website https://inet.omnetpp.org.

<table>
<thead>
<tr>
<th>Software information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer(s)</td>
</tr>
<tr>
<td>Written in</td>
</tr>
<tr>
<td>Operating system</td>
</tr>
<tr>
<td>License</td>
</tr>
</tbody>
</table>
| Website               | http://www.omnetpp.org/  
                       | https://inet.omnetpp.org |
| Supported version(s)  | OMNeT++ 5.5.1  
                       | INET 4.1 |
| Dependencies          | libprotobuf 3.3.0  
                       | python < 3.0.0 |
| Installation          | via released installation script |

Table 3.1: Software information: OMNeT++

3.2.1 omnetpp-ambassador folder structure

The omnetpp.ini file has to be located in the omnetpp folder of the scenario configuration.

```
<scenarioName>
  \omnetpp ..........................................................            
  \omnetpp_config.json ..................... Ambassador configuration file 
  \omnetpp.ini ................................. OMNeT++ federate configuration file 
```

Figure 3.2: omnetpp-ambassador folder structure
3.2.2 Installation

The VSimRTI all-in-one package comes with an installation script for the bash-shell, which automates the task of the OMNeT++/INET installation.

Prerequisites

The OMNeT++/INET federate furthermore depends upon:

- libprotobuf 3.3.0
- python < 3.0.0

Please make sure these libraries are installed before running the installation script.

Installation shell script

```
vsimrti
  └── bin
      └── fed
          └── omnetpp
              ├── Dockerfile ......................... Docker file for OMNeT++/INET federate
              └── omnet_installer.sh ................ Installation script for OMNeT++/INET
```

Figure 3.3: OMNeT++ folder structure

Please execute the following steps for installing the OMNeT++/INET federate:

1. Follow the link and download the source code of OMNeT++ 5.5.1 for your current operating system (omnetpp-5.5.1-src.tgz):  
   🌐 https://gateway.ipfs.io/ipns/ipfs.omnetpp.org/release/5.5.1

2. Follow the link and download the source code of INET 4.1 for your current operating system (inet-4.1-src.tgz):  
   🌐 https://inet.omnetpp.org/Download.html

3. Follow the link and download the source code of VSimRTI OMNeT++ Federate (omnetpp-federate-19.1.zip):  
   🌐 https://www.dcaiti.tu-berlin.de/research/simulation/download/

4. Make the installation script executable:

   ```bash
   chmod 755 omnet_installer.sh
   ```

5. Make sure all dependencies are met by your system.

6. Run the script to install OMNeT++/INET:

   ```bash
   ./omnet_installer.sh
   -o /path/to/omnetpp-5.5.1-src.tgz
   -i /path/to/inet-4.1-src.tgz
   -f /path/to/omnetpp-federate-19.1.zip
   ```
### 3.2.3 Installation in Docker environment

**Notice:** This is an experimental feature. Please refer to our mailing list if you experience any problems.

This guide gives instructions to execute the OMNet++ federate inside a docker container. If you already installed OMNet++ on your machine following the steps before, you can skip this section.

Docker\(^1\) is a new approach to execute software. More precisely, it "wraps software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, and system libraries". As a result, the software is executed within a container and its execution does not rely on the environment the container is running in.

In context of VSImRTI, this approach allows to execute OMNet++ within a docker container. The user does not need to manually install OMNet++ and can even run OMNet++ on Windows hosts.

1. Install Docker \(\geq 1.13\) on your machine.

2. To get everything to work, please make sure to execute the following steps depending on your operating system:
   - **Windows** - In the settings, share the drive where VSImRTI is installed on. You may need to restart docker in the reset tab.
   - **Linux** - Make sure your user account belongs to the unix-group "docker". You may need to restart your machine.

3. Switch to the location of the Dockerfile in `vsimrti/bin/fed/omnetpp`

4. Follow the link and download the source code of OMNet++ 5.5.1 (omnetpp-5.5.1-src.tgz) and place the downloaded file into the `vsimrti/bin/fed/omnetpp` directory:
   
   ![Download Link](https://gateway.ipfs.io/ipns/ipfs.omnetpp.org/release/5.5.1)

5. Execute the following command on command line:
   ```bash
   docker build -t omnetpp-federate:19.1 .
   ``
   This could take a while to finish.

6. Enter the name of the docker image `etc/defaults.xml` in the `omnetpp`-section into the tag `dockerImage`:

   ```xml
   <federate class="...">
   <id>omnetpp</id>
   ...
   <dockerImage>omnetpp-federate:19.1</dockerImage>
   ...
   </federate>
   ```

   You can test the installation of your docker image with the Tiergarten scenario, by activating `omnetpp` in the `vsimrti_config.xml`.

\(^1\)https://www.docker.com/
3.2.4 Configuration

The whole OMNeT++ specific configuration is done via the omnetpp.ini file. It covers static parts for the simulator coupling such as the specific VSimRTI Event Scheduler and the ScenarioManager. Furthermore, logging configurations and the typical parameters for the communication layers (MAC, PHY and Radio Channel) are addressed. The communication parameters are different for vehicles and RSUs. Please refer to the OMNeT++ documentation on the OMNeT++ homepage for further information about the structure of the omnetpp.ini file.
3.3 ns-3

The ns-3 is a discrete-event network simulator that was developed as a replacement for the popular network simulator 2 (ns-2) and mainly focuses upon improving the core architecture, software integration, models, and educational components for real-world network devices and protocols. It simulates both unicast and multicast protocols and is used extensively in research on mobile ad-hoc networks.

Regardless, ns-2 still remains in active use and will continue to be maintained in the near future. For VSimRTI coupling, only ns-3 will be available.

Like other network simulators, ns-3 has a relatively steep learning curve, especially compared to GUI-based simulators. If you have no prior experience with ns-3, we recommend to familiarize yourself with the ns-3 simulation environment and the ns-3 simulation basics first. The ns-3 documentation can be found under:

https://www.nsnam.org/documentation

To take your first steps with ns-3, you might want to download\(^2\) the latest version, build a copy of ns-3 (it uses the Python-based build-system waf) and take a look at the examples, that are shipped within most of the ns-3 source code repositories and packages. You might also examine the simulation output and try to adjust it.

Typically, a ns-3 user will work under a Linux environment. For those running under Windows, there do exist environments which simulate the Linux environment to various degrees. The ns-3 project has in the past (but not presently) supported the Cygwin environment for these users (see http://www.cygwin.com for details on downloading). MiniGW is presently not officially supported, however there are also some people who managed to use it with ns-3. For detailed information of how to setup ns-3, please refer to their Wiki:

https://www.nsnam.org/wiki/Installation

For more information on how to setup ns-3 with VSimRTI, please just refer to the following section. We prepared an installation script, which manages most of the required work.

⚠️ Important: As stated above, ns-3 is primarily developed on and for GNU/Linux platforms. Since Windows is such a widely used platform and Cygwin is not a perfect emulation of any Linux, we highly recommended for non-Linux users to consider the installation of a Linux virtual machine with a virtual machine environment, such as VMware\(^3\) or VirtualBox\(^4\).

\(^2\)Please note, that downloading ns-3 via tarball is simpler than the Mercurial process since all of the pieces are pre-packaged for you.

\(^3\)http://www.nsnam.org/wiki/index.php/HOWTO_use_VMware_to_set_up_virtual_networks_(Windows)

\(^4\)http://www.nsnam.org/wiki/index.php/HOWTO_use_VirtualBox_to_run_simulations_on_Windows_machines
### Software information

| Developer(s) | Tom Henderson, Mathieu Lacage, George Riley, Mitch Watrous, Gustavo Carneiro, Tommaso Pecorella and others |
| Written in | C++ (core) and Python (bindings) |
| Operating system | GNU/Linux FreeBSD Mac OS X |
| License | free software: GNU GPLv2 |
| Website | http://www.nsnam.org/ |
| Supported version(s) | 3.28 |
| Dependencies | libprotobuf 3.3.0 |
| | libxml2 |
| | libsqlite3 |
| Deployed in VSimRTI all-in-one | no (and need a patch to link) |

Table 3.2: Software information: ns-3

#### 3.3.1 ns3-ambassador folder structure

```
<scenarioName>
  ns3
    ns3_config.json .................................. Ambassador configuration file
    configTechnologies.xml .......................... ns-3 federate configuration file
    confWifi.xml ................................. ns-3 federate configuration file
```

Figure 3.4: ns3-ambassador folder structure

#### 3.3.2 Installation

VSimRTI offers support for the current stable release of ns-3 (3.28), that was released in March 2018. Older versions of ns-3 (prior to 3.28) are not supported. However, also for newer versions we cannot guarantee the correct operation. The coupling to VSimRTI is designed in a manner of minimal code changes on the ns-3 simulator itself to keep the update capabilities for future versions.

### Prerequisites

For GNU/Linux platforms, the minimal requirements to run basic simulations are a gcc or clang compiler and a Python interpreter. At least you need the following packages to be installed:

```
gcc
g++
python
python-dev
```

Listing 3.1: ns-3: Minimum requirements
For a complete list of required packages for different distributions, please refer to the [ns-3 installation guide](http://www.nsnam.org/wiki/index.php/Installation):

Please make sure the following libraries are installed before running the installation script:

- libxml2
- sqlite3
- libprotobuf 3.3.0

**Run the installation script**

⚠️ **Important:** ns-3 requires several packages to be installed on your computer. You will need to ensure, that all required libraries are present on your system before proceeding. You may need superuser permissions to install packages.

⚠️ **Important:** If your local protobuf version does not fit the required one, the installation may fail with an error. In that case, you can run the install script with the `-p` flag. This will rebuild the protobuf files during installation and allow it to proceed correctly.

To ease the installation of ns-3 for VSimRTI, the installation process has been delegated to an installation script, that can be found in the associated ns-3 federate folder.

```
vsimrti
  |- bin
  |  |- fed
  |     |- ns3
  |        |- Dockerfile.sh ...................... Dockerfile for ns-3 federate
  |        |- ns3_installer.sh ................. Installation script for ns-3
```

Figure 3.5: ns3-ambassador federate folder structure

The ns-3 installation script accomplishes the following tasks:

1. Download ns-3 tarball from the official sources
2. Download the ns-3 federate for VSimRTI.
3. Apply a patch to ns-3 in order to make it work with VSimRTI.
4. Add the ns-3 federate to the waf build system.
5. Configure and build the patched ns-3 with the ns-3 federate.

In order to start the simulation, the following steps need to be performed:

1. Set up the confWifi.xml file in the scenario folder (see section 3.3.4). An example confWifi.xml file is shipped with the Tiergarten scenario.
2. For reasonable result logging, the logger-configuration in vsimrti/etc/logback.xml has to be adapted to support the ns-3 ambassador and federate.
3. At last ns-3 has to be activated in the vsimrti_config.xml and the simulation can be started.
3.3.3 Installation in Docker environment

Notice: This is an experimental feature. Please refer to our mailing list if you experience any problems.

This guide gives instructions to execute the ns-3 federate inside a docker container. If you already installed ns-3 on your machine following the steps before, you can skip this section.

Docker\(^5\) is a new approach to execute software. More precisely, it "wraps software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, and system libraries". As a result, the software is executed within a container and its execution does not rely on the environment the container is running in.

In context of VSimRTI, this approach allows to execute ns-3 within a docker container. The user does not need to manually install ns-3 and can even run ns-3 on Windows hosts.

1. Install Docker ≥ 1.13 on your machine.

2. To get everything to work, please make sure to execute the following steps depending on your operating system:
   - Windows - In the settings, share the drive where VSimRTI is installed on. You may need to restart docker in the reset tab.
   - Linux - Make sure your user account belongs to the unix-group "docker". You may need to restart your machine.

3. Switch to the location of the Dockerfile in vsimrti/bin/fed/ns3

4. Execute the following command on command line:
   docker build -t ns3-federate:19.1 .
   This could take a while to finish.

5. Enter the name of the docker image etc/defaults.xml in the ns3-section into the tag dockerImage:

   ```xml
   <federate class="...">
     <id>ns3</id>
     ...
     <dockerImage>ns3-federate:19.1</dockerImage>
     ...
   </federate>
   ```

You can test the installation of your docker image with the Tiergarten scenario, by activating ns3 in the vsimrti_config.xml.

\(^5\)https://www.docker.com/
3.3.4 Configuration

The whole ns-3 specific configuration is done via the confWifi.xml and configTechnologies.xml files.

Listing 3.2: confWifi.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wifi>
    <!-- IPv4 address generator -->
    <ipConfiguration>
        <ip address="192.168.0.0" mask="255.255.0.0"/>
    </ipConfiguration>
    <!-- Calculate a propagation delay -->
    <propagationDelayModel>
        <delay model="ns3::NonePropagationDelayModel"/>
    </propagationDelayModel>
    <!-- Modelize the propagation loss through a transmission medium -->
    <propagationLossModel>
        <loss model="ns3::FriisPropagationLossModel"/>
    </propagationLossModel>
    <wifiConfiguration>
        <!-- Create non QoS-enabled MAC layers -->
        <wifiMac property="type" value="ns3::AdhocWifiMac"/>
        <!-- Wifi PHY node -->
        <wifiManager property="phyMode" value="OfdmRate54Mbps"/>
        <!-- Wifi manager -->
        <wifiManager property="type" value="ns3::ConstantRateWifiManager"/>
        <!-- The energy of a received signal should be higher than this threshold (dbm) to allow ->
        <!-- the PHY layer to detect the signal -->
        <wifiPhy property="EnergyDetectionThreshold" value="-81.0"/>
        <!-- The energy of a received signal should be higher than this threshold (dbm) to allow ->
        <!-- the PHY layer to declare CCA BUSY state -->
        <wifiPhy property="CcaMode1Threshold" value="-99.0"/>
        <!-- Transmission gain (dB) -->
        <wifiPhy property="TxGain" value="0.0"/>
        <!-- Reception gain (dB) -->
        <wifiPhy property="RxGain" value="0.0"/>
        <!-- Number of transmission power levels available between TxPowerStart and TxPowerEnd -->
        <wifiPhy property="TxPowerLevels" value="1"/>
        <!-- Maximum available transmission level (dbm) -->
        <wifiPhy property="TxPowerEnd" value="17.0"/>
        <!-- Minimum available transmission level (dbm) -->
        <wifiPhy property="TxPowerStart" value="17.0"/>
        <!-- Loss (dB) in the Signal-to-Noise-Ratio due to non-idealities in the receiver -->
        <wifiPhy property="RxNoiseFigure" value="0.0"/>
        <!-- Channel center frequency = Channel starting frequency + 5 MHz * (nch - 1) -->
        <wifiPhy property="ChannelNumber" value="1"/>
    </wifiConfiguration>
</wifi>
```

The IP configuration information includes the network address and network mask. The ns-3 propagation module defines two generic interfaces, namely PropagationLossModel and PropagationDelayModel, for the modelling of propagation loss respectively propagation delay.

In the default confWifi.xml, the Wi-Fi device uses the ns-3 standard propagation delay model ns3::ConstantSpeedPropagationDelayModel and the ns-3 standard propagation loss model ns3::FriisPropagationLossModel. Radio propagation models in ns-3 can easily be exchanged with
the ns-3 class registration system (see the ns-3 documentation for details). The Wi-Fi configuration includes additional parameters, like sending power and antenna gain.

Listing 3.3: configTechnologies.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ns3Configuration>
  <installers>
    <installer type="ns3::WifiVehicleInstaller" name="WifiVehicle" file="confWifi.xml" />
    <installer type="ns3::MobilityModelInstaller" name="Mobility" default="true" />
  </installers>
</ns3Configuration>
```

The configuration manager of the ns-3 federate defines, which installer should be loaded for the Wi-Fi device (refering to the confWifi.xml) and the mobility model. Usually, you don't need to take any changes and simply use the default configuration file, that ships with the ns-3 federate.
3.4 SUMO

SUMO is an highly portable, microscopic and continuous road traffic simulation package. It is designed to handle large road networks faster than real-time. Each vehicle has an own route and is simulated individually.

<table>
<thead>
<tr>
<th>Software information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer(s)</td>
</tr>
<tr>
<td>Written in</td>
</tr>
<tr>
<td>Operating system</td>
</tr>
<tr>
<td>License</td>
</tr>
<tr>
<td>Website</td>
</tr>
<tr>
<td>Supported version(s)</td>
</tr>
<tr>
<td>Deployed in VSimRTI all-in-one</td>
</tr>
</tbody>
</table>

Table 3.3: Software information: SUMO

⚠️ Important: Please note that VSimRTI only supports SUMO 1.0.1 or higher.

3.4.1 sumo-ambassador folder structure

This ambassador can be configured with a configuration file. The specific path is vsimrti/scenarios/<scenarioName>/sumo/sumo_config.json

<scenarioName>
  | sumo ................................................................. Node file
  | <scenarioName>.nod.xml ............................................ Node file
  | <scenarioName>.edg.xml ............................................ Edge file
  | <scenarioName>.con.xml .......................................... Connection file
  | <scenarioName>.net.xml .......................................... Network file
  | <scenarioName>.rou.xml .......................................... Route file
  | <scenarioName>.sumo.cfg ............................................ sumo configuration file
  | sumo_config.json ............................................ ambassador configuration file

Figure 3.6: sumo-ambassador folder structure

3.4.2 Installation

For Microsoft Windows operating systems download and extract sumo_winbin.zip from the SUMO website. For GNU/Linux operating systems download and extract the provided GNU/Linux binaries or build sumo by yourself. Please refer to the SUMO Wiki for further information.
In order to run SUMO with VSimRTI you need to make the SUMO binaries available system wide by adding the SUMO binary folder to your PATH environment variable (see section 14.1). Alternatively, the environment variable SUMO_HOME can be used to define the installation directory of SUMO.

### 3.4.3 Configuration

⚠️ **Notice:** The documentation for the VSimRTI specific component is freely available on the DCAITI website, explaining all available options.

The SUMO configuration consists of sumo specific config files and the sumo-ambassador configuration file. vsimrti/scenarios/<scenarioName>/sumo. Your main configuration file name must end with the suffix *.cfg. The SUMO Wiki covers more information and also a tutorial about the different configuration files. Route-, network- and SUMO configuration files have been generated (using scenario-convert or by yourself). The last step is the creation of a sumo-ambassador configuration file.

In contrast to standard traffic simulations using SUMO only, VSimRTI handles the SUMO files in a slightly different way, i.e. the route file for a simulation is created at runtime to enable dynamic routing. To get correct simulation results, we recommend using the scenario-convert tool to create the SUMO files out of OpenStreetMap data (see section 12.1).

### 3.4.4 Using the SUMO GUI with VSimRTI

It is also possible to use the Graphical User Interface (GUI) of SUMO in order to visualize and interact with the simulation. To achieve this, VSimRTI can be configured to start the GUI process of SUMO as the federate rather than the command line interface.

In order to use the SUMO GUI with VSimRTI, please open the file vsimrti/etc/defaults.xml and replace the entry `com.dcaiti.vsimrti.fed.sumo.ambassador.SumoAmbassador` with `com.dcaiti.vsimrti.fed.sumo.ambassador.SumoGUIAmbassador`.

Afterwards VSimRTI can be executed as usual. However, the traffic simulation has to be triggered manually within the GUI window of SUMO. Additionally, the field Delay within the GUI can be used to slow down the simulation. For further information about the graphical interface please consider the SUMO documentation.

⚠️ **Notice:** Keep in mind to launch VSimRTI with the argument `-w 0` in order to disable the watchdog timer. Otherwise it would shutdown VSimRTI if you pause the simulation in the SUMO GUI.

### 3.4.5 Routes and Vehicle Types

For the simulation the traffic simulator SUMO is started as a background process. In preparation, VSimRTI automatically generates all files needed, such as routes and vehicle type definitions. In particular, VSimRTI generates a *.rou.xml which contains all vehicle types from the mapping3 configuration and all routes from the scenario database. This also means, that any route referenced by the mapping3 configuration must be available in the scenario database. Additionally, the *.rou.xml in the scenario configuration of SUMO will always be overwritten by VSimRTI at the start of the simulation.
If you want to use SUMO specific parameters for the vehicle types (e.g. to define the car following model or the emission class), you can predefine them in the file `<scenarioName>.rou.xml` of your scenario. All parameters you define there will be used when the route file is generated for the simulation (except all parameters which are defined in the mapping3 configuration such as `accel`, `decel`, `tau`, etc.). The Tiergarten scenario provides such an example file.

### 3.4.6 Further information

Notice: We recommend to use the 64 bit version of SUMO if you want to simulate scenarios with a big traffic network.

### 3.4.7 VSimRTI specific information

Notice: The SumoAmbassador broadcasts `VehicleMovements` in fixed time steps and also sends a `TrafficLightUpdate` message, when the tl-state change was initiated in the previous step. For dynamic change of vehicle behavior during simulation, SUMO subscribes to the messages `VehicleTypesInitMessage`, `VehiclePathsInitMessage`, `ChangeSpeed`, `SlowDown`, `PropagateNewRoute`, `ChangeStaticRoute` and `ChangeTrafficLightsState`.

Important: Even though the SumoAmbassador is able to receive `ChangeRoute` messages which will result in using the internal Re-route feature of SUMO, it is strongly recommended to use the navigation functionality of the ApplicationNTAmbassador by using the provided API.
3.5 VSImRTI Application Simulator

The application simulator is a sophisticated simulator to provide simulation units (e.g. vehicles, road side units, traffic lights, charging stations) an interface to run their logic. The latest application simulator has been developed from scratch and is called ApplicationSimulatorNT.

3.5.1 applicationNT-ambassador folder structure

This ambassador can be configured with a configuration file. The specific path is
vsimrti/scenarios/<scenarioName>/applicationNT/applicationNT_config.json

```
<scenarioName>
  applicationNT ..........................................................
  applications ..........................................................
  applicationNT_config.json ............... Configuration file for the ambassador.
  <scenarioName>.db ........................... Database file for navigation.
  mapping3 ......................... Configuration file for the ambassador.
```

Figure 3.7: applicationNT-ambassador folder structure

3.5.2 Installation

This simulator does not need to be installed. It is delivered as part of the VSImRTI-installation package.

3.5.3 Libraries

To develop applications you need to reference to jar files you will find in the following directories:

- vsimrti/bin/ambassadors/applicationNT-ambassador-19.1.jar
- vsimrti/bin/ambassadors/lib/*.jar
- vsimrti/bin/lib/*.jar
- vsimrti/bin/core/vsimrti-19.1.jar

3.5.4 Basic functionality

Each logical unit is recognized as a separate simulation unit. The application simulator can load different applications for each simulation unit. For each application there is a general class and an extension for the particular type of the simulation unit (e.g. Vehicle extends SimulationUnit).

This application may request information from an object and wants to control this. For each simulation object, an operating system will be created. For each application, there is an operating system and an extension of the operating system for the particular type (e.g. VehicleApplication extends AbstractApplication<VehicleOperatingSystem>).
3.5.5 Gather information in an application

To gather information in an application from a simulation unit, there are typically two different ways.

React on changes

Applications will be notified on changes. All applications must implement methods that are called before and after a change. Typically, these methods are named before* and after*. Please note that not all changes of information notify applications through this mechanism. Some information must be evaluated by sampling. For example, the method `getStateOfEnvironmentSensor` could not notify the application by changing its state.

Sample information

The sampling of information at specific times can be done by requesting future events.

Self events

An event is a notification with information at a future point in the simulation. With a self-event, applications can wake up themselves at certain time steps. An event can be requested at any simulation point to trigger an event execution.

3.5.6 Equipped units

All units are generally classified into two types, units that have application logic or not. Equipped vehicles can also use a predefined ETSI application.

3.5.7 Technical details

- The simulator does not use concurrent processing.
- The logs are based on the application name.

3.5.8 Configuration

Notice: The documentation for the VSimRTI specific component is freely available on the DCAITI website, explaining all available options.

Most of the actual configuration of the applications is done by the mapping-simulator. The mapping-simulator is responsible for spawning the entities of the simulation and therefore also decides which application to map onto which entity.

To deploy an application you have to put it into the appropriate subdirectory for the simulated unit.
3.5.9 Development of applications

Overview

Developing custom applications in VSimRTI is rather easy. The best way to learn it is by looking at the source code of actual applications. For this purpose, we provide the source code of all tutorial applications and further examples. You can find the source code on the DCAITI website.

For an easy understanding of the application API, the following questions and their answers should help:

- **What is required to get my own application to run in VSimRTI?**
  
  In VSimRTI it is very easy to build your own application. First, it needs to inherit from the `AbstractApplication` class (see following section). Secondly, the application must be mapped to a vehicle (or RSU, or traffic light, ...) via the mapping configuration (see section 3.5.15). Finally, the application must be compiled as a Jar-File and placed into the application directory of your scenario.

- **How can I access vehicle functions from within the application, such as sending V2X messages?**
  
  Every applications has access to the `OperatingSystem` of the underlying unit which allows to change its state or to initiate actions, such as sending messages to other vehicles.

- **How can I react to events during the simulation, such as receiving V2X messages?**
  
  For each application you decide, which events the application should listening to. For example, if your application needs to react upon incoming V2X messages, it simply implements the `CommunicationApplication` interface. In the following section you can find all available interfaces applications can implement.

Create a 'Hello world' application using eclipse

Create a new project in eclipse (File → New → Java Project). Name your application `HelloWorldApp`. Confirm the creation of the project. Right click on the project and choose "Properties" to open the project specific configuration options. In the dialog window select "Java Build Path" and click on the Libraries tab. Choose "Add external JARs" and navigate to your vsimrti installation folder. Choose the following JARs and add them to your project:

- `vsimrti/bin/ambassadors/applicationNT-ambassador-19.1.jar`
- `vsimrti/bin/ambassadors/lib/*.jar`
- `vsimrti/bin/lib/*.jar`
- `vsimrti/bin/core/vsimrti-19.1.jar`

Right click on the "Hello world" Project and choose (New → Package). Name the package something like `com.mycompany.vsimrti.applications`. Right click on the package and chose (New → Class). Name the class `HelloWorldApp`, extend from the Class `[..]applicationNT.ambassador.simulationUnit.applications.AbstractApplication<OS>` and confirm. Afterwards, you need to specify the type of operating system `<OS>` your application is
running on by parameterizing the extended AbstractApplication<OS> by one of the following classes:\(^6\):

- VehicleOperatingSystem - for applications mapped to normal vehicles.
- ElectricVehicleOperatingSystem - for applications for vehicles with electro mobility features.
- RoadSideUnitOperatingSystem - for applications mapped to RSUs.
- TrafficLightOperatingSystem - for applications mapped to traffic lights.
- TrafficManagementCenterOperatingSystem - for applications mapped to TMCs.
- ChargingStationOperatingSystem - for applications mapped to charging stations.

Furthermore, your application can implement the following interfaces\(^7\) in order to get informed on specific events:

- VehicleApplication - get informed when information about the vehicles state is updated.
- ElectricVehicleApplication - get informed on electric vehicle specific events.
- CommunicationApplication - react on incoming V2X messages.
- VSimRTIApplication - get informed on VSimRTI internal events.
- TrafficLightApplication - get noticed when the traffic light program is changed.
- ChargingStationApplication - react on state changes of the charging station.
- TrafficManagementCenterApplication - get informed on state changes of road infrastructure.

You can now continue and make modifications. To deploy the application you have to export the project (File \(\rightarrow\) Export \(\rightarrow\) Java/Jar File) and put the JAR in the folder vsimrti/scenarios/<scenarioName>/applicationNT/applications.

**Notice:** You can download the example Application HelloWorldApp, which is part of the AdditionalExamples package, from the website.

In the following picture you can find the example application HelloWorldApp opened in eclipse.

\(^6\)See package com.dcaitti.vsimrti.fed.applicationNT.ambassador.simulationUnit.operatingSystem.*

\(^7\)See package com.dcaitti.vsimrti.fed.applicationNT.ambassador.simulationUnit.applicationInterfaces.*
3.5.10 Debugging of applications

To debug an application, remote debugging needs to be used. The following steps need to be performed in order to debug the application:

1. Open the application in your IDE

2. Modify your vsimrti.sh or vsimrti.bat, depending if you are on windows or a Unix-like system. Examples can be found below.

3. Add a new debug profile for remote debugging, make sure to correctly configure port 4100 (or whichever was provided in the call to VSimRTI)

4. Launch VSimRTI with the argument \(-w 0\) to disable the watchdog timer otherwise it will interfere with debugging
3.5 VSimRTI Application Simulator

Listing 3.4: "vsimrti.sh on Unix-like systems"

```bash
#!/bin/bash
set -e

source common.sh
javaMemorySizeXms=""
javaMemorySizeXmx=""
javaProxySettings="-Djava.net.useSystemProxies=true"
javaDebugSettings="-agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=4100"

cmd="java $javaProxySettings $javaMemorySizeXms $javaMemorySizeXmx $javaDebugSettings $javaMemorySizeXmx $javaProxySettings -cp .:${core}:${libs}:${ambassdor_libs}:${ambassador_libs}:${etc}:Djava.library.path=./lib "

$cmd
```

Listing 3.5: "vsimrti.bat on windows systems"

```bash
@ECHO OFF
SetLocal EnableDelayedExpansion

call common.bat

set javaMemorySizeXms=
set javaMemorySizeXmx=

set javaProxySettings=-Djava.net.useSystemProxies=true

set javaDebugSettings=-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=4100

java %javaProxySettings% %javaDebugSettings% %javaMemorySizeXms% %javaMemorySizeXmx% -cp !libs/ !com.dcaiti.vsimrti.start.XMLConfigRunner %*

EndLocal

exit /b %errorlevel%
```

3.5.11 Internal ETSI applications

We provide ETSI conform applications which implement specific CAM generation rules. You can find them in the package `com.dcaiti.vsimrti.fed.applicationNT.etsiApplications.impl`.

ETSI Application (Vehicle)

For example, to provide ETSI functionality for your vehicle, you can load the ETSI application `com.dcaiti.vsimrti.fed.applicationNT.etsiApplications.impl.Vehicle`

This application generates ETSI data for its simulation unit (e.g. heading, position, speed and time for vehicles). According to its configuration, the application then sends out CAMs to other vehicles in range. Note that the messages are only send when the time lies between the configured minimum and maximum interval.
Default configuration  Without explicit configuration for the ETSI a default configuration is loaded. For the ETSI interceptor have a look at the class `CEtsi`.

**Notice:** For more information about this component, please have a look at the javadoc (doxygen) documentation.

Currently, the default configuration for the ETSI application looks like this:

```json
{
    "maxStartOffset": 1000000000,
    "minInterval": 100000000,
    "maxInterval": 1000000000,
    "positionChange": 4,
    "headingChange": 4,
    "velocityChange": 0.5
}
```

In general, the numeric values should be interpreted as time delta. E.g., if the ETSI values change by the given amount, a new CAM is sent. The only exceptions are the minimum and maximum intervals. If the time lies over the maximum interval, a CAM is sent even if the difference between the ETSI values isn't high enough. If the difference is high enough but the minimum interval isn't surpassed, no message will be sent.

### 3.5.12 Sending Cooperative Awareness Messages (CAM)

The following section will show how CAMs are implemented in VSimRTI and how you can alter them. CAMs consist of four parts:

- Header with generic information
- MessageBody
- ServiceList
- TaggedValue list

First of all generic information like protocol version and creation time stamp are transmitted. Normally this data set follows a network beacon, already containing data like position and speed. Nevertheless this functionality must be implemented in the network layer, that means in the network simulator. At the moment this is not supported and is therefore compensated in the next message part, the message body.

The body can contain either RSU or Vehicle awareness data. In the first case, only the RSU type is submitted, but this will probably change with a new CAM specification. In the second case, we provide data like vehicle width, length, position, speed, type and heading. However, the specification is not completely implemented, especially acceleration data, as well as light and brake status are missing. The third part of the CAM specification, the service list, is also not implemented. This will probably change, when a list of services is defined by the ETSI.

Last but not least a message can contain optional data. This is fully implemented and is used for our traffic light CAM messages, which provide the traffic light status.

The CAM sending interval can be configured, more information can be found in the configuration section of the application simulator.
3.5.13 Access SUMO TraCl from applications

If SUMO is used as a traffic simulator and a special functionality is required, the `sendSumoTraciByteArrayMessage` function in the OperatingSystem can be used.

The function expects a string (a unique identifier which will be assigned to the response) and a byte array (representing the complete Traffic Control Interface (TraCI) request including the header). The message identifier can be an empty string.

In all cases the command will trigger a response. The application can receive the response from the method `onSumoTraciByteArrayMessageResponse`. This method will receive a `SumoTraciByteArrayMessageResponse` object. This response contains the specified identifier. The application must handle the identification process of the response itself.

⚠️ Notice: A response is delivered to every application. To prevent unintentional delivery each application request should use an immutable universally unique identifier (UUID).

⚠️ Important: Be careful when using this interface and the TraCI commands. The commands are delivered to TraCI without any prior checks.

⚠️ Notice: You can find the example application `TestSumoTraciByteArrayMessageApp` in the additional examples bundle on the DCAITI website.

3.5.14 Navigation

The navigation of vehicles is handled completely by the application simulator. Each vehicle is equipped with a navigation system which provides all required information and functions for navigational purposes:

- Retrieve the current position and heading of the vehicle.
- Get the target of the vehicle.
- Calculate various routes from the current position to an arbitrary target.
- Choose a suitable route out of existing ones from the current position to an arbitrary target.
- Switch onto a specific route.

In order to provide routing functionality, a map model based on Open Street Map data is used, which needs to be transformed before the simulation using scenario-convert. The map data including initial routes for vehicles is provided with the database file which needs to be located in `vsimrti/scenarios/<scenarioName>/applicationNT/<scenarioName>.db`.
Configuration

If the database needs to be located somewhere else, the path can be specified in
vsimrti/scenarios/<scenarioName>/applicationNT/applicationNT_config.json:

```json
{
    ...
    "navigationConfiguration": {
        "databaseFile": "path/to/scenario.db"
    }
}
```

Usage

The following snippet show, how the navigation system can be used within an application:

```java
// get navigation module
INavigationModule navigationModule = getOperatingSystem().getNavigationModule();

// choose target position to current target
RoutingPosition targetPosition = new RoutingPosition(navigationModule.getTargetPosition());

// set routing parameters to fastest route search
RoutingParameters params = new RoutingParameters().costFunction(IRoutingCostFunction.Fastest);

// calculate routes
RoutingResponse response = navigationModule.calculateRoutes(targetPosition, params);

// switch to best route
if(response.getBestRoute() != null){
    boolean routeSwitched = navigationModule.switchRoute(response.getBestRoute());
    ...
}
```

3.5.15 Mapping 3

The mapping configuration is used to define how many vehicles of a specific type should drive in a simulation. You can choose between a deterministic mapping (producing the exact same sequence of mapped vehicles in every simulation run with regard to the given ratios at each point in time the simulation) and a stochastic mapping (resulting in a random order of mapped vehicles).

The new mapping ambassador was built to simplify the configuration of complex scenarios. It is not as tightly linked to the database as its predecessor and relies on JSON rather than Extensible Markup Language (XML) for configuration.

The configuration for the Mapping3-Federate is located in
vsimrti/scenarios/<scenarioName>/mapping3/mapping_config.json.

In this file an object is described in JavaScript Object Notation (JSON), which will at runtime be parsed using the MappingFramework-Class.

⚠️ **Notice:** The documentation for the VSimRTI specific component is freely available on the DCAITI website, explaining all available options.
The Mapping-Framework-Class

The main class for the Mapping3-Configuration is the MappingFramework-Class. It is divided into six parts: prototypes, rsus, tls, tmcs, vehicles, matrixMappers and config.

In prototypes you can define models for other objects, which can later be reused in the other sections. This makes it possible to reuse the definition of certain vehicles or the combination of multiple applications, reducing redundancies.

The rsus-section offers the possibility to define instances of Road Side Units (RSU)s. To reuse a prototype simply specify its name in the name-property. All relevant properties will then be automatically filled in. At least, you must define a position for every RSU.

In the tls-section traffic lights can be defined. The traffic lights themselves are already specified in the database of the scenario. Here applications can be mapped to them. Two different modes are offered for that: You can map the entities to individual traffic lights by specifying the name of the traffic light in the tlName-property. Alternatively you can define weights for the specified traffic lights, which will then be the base for a random distribution through all traffic lights. The weights do not have to add up to 100 or 1. Consequently all traffic lights will be mapped using the specified prototypes as soon as one weight differs from zero. So in case you don't want all traffic lights to have applications running on them you have to define one traffic light without any applications and add a weight to it.

The tmcs-section offers the possibility to define instances of Traffic Management Center (TMC). A TMC provides the possibility to interact with the infrastructure of the road, i.e. retrieving traffic properties from detectors (e.g. traffic flow), and changing properties from the road (e.g. speed limits).

The vehicles-section is the centerpiece of the configuration. Here the departures, routes and types of the vehicles are defined. It is divided into VehicleSpawner. These spawners are designed to produce a traffic stream with the given traffic density (in vehicles/hour). By specifying a maximum number of vehicles to be created you can also spawn individual vehicles. A VehicleSpawner offers the possibility to combine multiple vehicle types. By specifying weights for the different vehicle types the ratios between them can be defined. If no weights are defined they are assumed to be equal. Note: If at least one vehicle type has a weight defined this does not apply! In that case all types without a defined weight are ignored. Instead of specifying a route you can also choose two points (given in long/lat). If no radius for the point is specified the closest point to the given position will be chosen. Otherwise a point inside the radius will be select randomly.

It is also possible to define and use OD-Matrices. To do so add a ODMatrixMapper to the matrixMappers-list. Each MatrixMapper consists of a list of points, the vehicle-types to be used and the actual flow-values between each of the points. It is possible to define multiple matrices. This way distinctively different compositions of the vehicle flows can be achieved. The MatrixMapper will be called before the actual execution of the simulation and will generate vehicle-spawners for the flow between each of the points. Thus the MatrixMapper is just a way to save work. If you want to use the extra-option that multiple vehicleSpawners offer you should use them.
Mapping and Identifier

Every traffic object in VSimRTI has a globally unique string identifier. These identifiers are used to identify a traffic object in VSimRTI as well as in different ambassadors. From user’s aspect, these identifiers will be seen in the log files which are generated after a simulation. Therefore, it should be explained, which identifier belongs to which traffic object.

<table>
<thead>
<tr>
<th>Traffic Object</th>
<th>VSimRTI Internal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle</td>
<td>veh_[seqNr]</td>
</tr>
<tr>
<td>RSU</td>
<td>rsu_[seqNr]</td>
</tr>
<tr>
<td>Traffic Light</td>
<td>tl_[groupId]</td>
</tr>
<tr>
<td>TMC</td>
<td>tmc_[seqNr]</td>
</tr>
</tbody>
</table>

Table 3.4: Mapping aspects

- `seqNr` is the sequence number of simulated vehicles, RSUs, TMCs, each starting from zero.
- `groupId` is the group id of the traffic light.

**Notice:** For more information about this component, please have a look at the javadoc (doxygen) documentation.
3.6 VSimRTI Cellular Simulator

The VSimRTI built-in Cellular Simulator enables the applications to use cellular network communication. The simulation of cellular communication in VSimRTI consists of two parts:

1. The Cellular Simulator itself and
2. The applications that can communicate over cellular networks in the Application Simulator

These changes are done in a generic way, making the cellular simulator exchangeable. Users interested in a different kind of simulation of cellular communication may use other simulators and develop ambassadors connecting them to VSimRTI.

The Cellular Simulator in the current state consists of three main modules:

1. Uplink Module
2. Geocaster Module
3. Downlink Module

The Geocaster module simulates a mandatory component for ITS communication. It is inspired by the several architectures from research projects as simTD or CONVERGE to enable ITS use cases over cellular networks. It mainly takes over the task of an addressing and routing component with geographic knowledge to support geo-addressing. However, it also supports regular topological addressing.

The Uplink and Downlink Module are responsible for the transmission simulation. They account for the aspects of transmission delays, packet losses and available data rates. In this context, Uplink and Downlink always refer to the direction towards respectively from the Geocaster. For instance, a transmission from an Internet-based server towards a vehicle would include an Uplink between the server and the Geocaster and a Downlink between the Geocaster and the vehicle. While the Uplink direction only allows point-to-point communication (Unicast), the Downlink supports point-to-point (Unicast) as well as point-to-multipoint (Multicast).

3.6.1 cell2-ambassador folder structure

The VSimRTI cellular simulator can be configured via three distinct configuration files, which can be found within the scenarios folder structure:

```xml
<scenarioName>
  cell2
    cell2_config.json  ......................... Cellular ambassador configuration file
    network.json  ............................... Network configuration file
    regions.json  ........................... Regions configuration file
```

The network and regions configuration files are referenced in the cellular ambassador configuration file.
### 3.6.2 Installation

This simulator does not need to be installed. It is delivered as part of the VSimRTI-installation package.

### 3.6.3 Configuration

We provide a cellular configuration file in the example scenarios of Tiergarten and Barnim. Please note that in the default configuration of this scenario the Cellular Simulator is deactivated. To activate the cellular simulator just add

```xml
<!-- Cell federate -->
<federate id="cell2" active="true" />
```

<vsimrti_config.xml (or, if this line already exist, make sure ‘active’ is set to ‘true’). With this done you now can deactivate all network simulators if only cellular simulation is intended in the investigated scenario. However, it is also possible to use multiple communication technologies (of ad hoc and cellular) in a hybrid scenario.

The central configuration for the cellular simulator in the file

```
vsimrti/scenarios/ <scenarioName>/cell2/cell2_config.json
```

could look like this:

```json
{
   "networkConfigurationFile": "network.json",
   "regionConfigurationFile": "regions.json"
}
```

The configuration for the global network in the cellular simulator in the file

```
vsimrti/scenarios/ <scenarioName>/cell2/network.json
```

could look like this:

```json
{
   "globalRegion": {
      "uplink": {
         "delay": {
            "type": "ConstantDelay",
            "delay": 100,
            "prpl": 0,
            "retries": 2
         },
         "capacity": 28000000
      },
      "downlink": {
         "unicast": {
            "delay": {
               "type": "ConstantDelay",
               "delay": 50,
               "prpl": 0,
               "retries": 2
            }
         },
         "multicast": {
            "delay": {
               "type": "ConstantDelay",
               "delay": 100,
               "prpl": 0
            },
            "usableCapacity": 0.6
         }
      },
      "capacity": 42200000
   }
}
```
Note that all bandwidths are given in bit per second and all delays in milliseconds. Also, every json configuration goes through a minifier, so comments are allowed in the configuration files. An example would be the comment before the `globalRegion` option.

**Delay regions**

Additionally the user has the option to define regions with individual delays. This can be used to simulate areas with bad reception, crowded areas etc.

The regions should be stored in `vsimrti/scenarios/<scenarioName>/cell2/regions.json`. An example definition for a single region called *Ernst-Reuter-Platz* could look like this:

```json
{
  "regions": [
    {
      "id": "Ernst-Reuter-Platz",
      "area": {
        "nw": { "lon":13.3249, "lat":52.5131 },
        "ne": { "lon":13.3273, "lat":52.5125 }
      },
      "uplink": {
        "delay": {
          "type": "SimpleRandomDelay",
          "steps": 4,
          "minDelay": 50,
          "maxDelay": 200,
          "prpl": 0.8,
          "retries": 2
        },
        "capacity": 28000000
      },
      "downlink": {
        "unicast": {
          "delay": {
            "type": "SimpleRandomDelay",
            "steps": 3,
            "minDelay": 100,
            "maxDelay": 200,
            "retries": 2
          }
        },
        "multicast": {
          "delay": {
            "type": "SimpleRandomDelay",
            "steps": 3,
            "minDelay": 120,
            "maxDelay": 220,
            "prpl": 0.8
          },
          "usableCapacity": 0.6
        },
        "capacity": 42200000
      }
    }
  ]
}
```
Note that \textit{nw} represents the upper-left (north-west) point of the rectangle and \textit{se} the lower-right (south-east). For further information about the possible options, please refer to the VSimRTI API documentation.

The actual configuration of the uplink and the downlink modules for each region exhibits the identical format as configuration of the globalRegion in the \texttt{network.json}.

When no regions are found, or if a node (a vehicle) is not within a specified region, the globalRegion defined in the \texttt{network.json}-File will be used as the delay model.

\textbf{Transmission simulation}

One of the most important features of the cellular simulator is an estimation of the delay experienced through the transport over the cellular network.

The cellular simulator offers various modes to estimate the delay of the transmissions. The type of estimation is specified with the \texttt{delayType} for the uplink and downlink for each region.

- \texttt{delay.type = 'ConstantDelay'}: The message is transmitted with the latency being exactly equal to \textit{delay}.
- \texttt{delay.type = 'SimpleRandomDelay'}: The latency can assume different (randomly generated and uniformly distributed) values between \textit{minDelay} and \textit{maxDelay}. The number of different values is determined by \texttt{steps}.
- \texttt{delay.type = 'GammaRandomDelay'}: A gamma distribution is used to estimate the latency, with $\alpha = 2$ and $\beta = 2$. The minimum delay \textit{minDelay} is added to the result. The curve is fitted so that the maximum likelihood for the delay is exactly equal to \textit{expDelay}.
- \texttt{delay.type = 'GammaSpeedDelay'}: This mode closely resembles the GammaRandomDelay. Additionally, a penalty for the velocity with which the node is moving is calculated. This penalty is then added to the original delay. The GammaRandomDelay and the GammaSpeedDelay are derived from a measurement campaign during a research project at the DCAITI.

The two different modes for the downlink are \textit{unicast} and \textit{multicast} which are configured separately. Multicast aims to simulate the features of Multimedia Broadcast Multicast Service (MBMS). The main difference in terms of the transmission for unicast and multicast is the handling of undeliverable messages. For unicast, the options \texttt{prpl} and \texttt{retries} are used. Pr is short for packet retransmit and denotes the probability for a failed delivery and a subsequent retransmit. The maximum number of retries for the retransmit is configured through the \texttt{retries}-option. The probability of a successful retransmit is recalculated on every try.

In case of multicast the \texttt{prpl} is used as packet loss rate. The value is factored into the delay calculation. In contrast to the unicast, just one transmission attempt is made for multicast.
3.6.4 Operation

Beside the transmission simulation, the Addressing and Routing is the other important aspect of the Cellular Simulator. This task is enabled by the Geocaster.

The Geocaster evaluates the message headers for cellular messages, which are created by the communicating applications in the Application Simulator.

It supports the following addressing and casting schemes.

**CellTopocast** is the *normal* unicast, where the Geocaster simply resolves the single receiver via the IPResolver. Hence, the CellTopocast directly routes the message further. Currently, Topocast doesn’t allow broadcast or anycast addresses, but any transmission protocols (tcp, udp).

**CellGeoUnicast** addresses every node in the destination area individually. In this way it takes a geographic address and results in a loop to generate multiple unicasts.

**CellGeoBroadcast**, which is basically MBMS, uses one broadcast to all nodes in the destined regions. The MBMS uses the different transmission mode of multicast in the downlink. CellGeoUnicast as well as CellGeoBroadcast require broadcast, but don’t allow tcp (as ack for broadcasts is denied).
3.7 VSimRTI Simple Network Simulator

As well as the network simulators OMNeT++ and ns-3, the VSimRTI built-in Simple Network Simulator (SNS) aims at simulating ad hoc communication for V2X scenarios. In contrast to the other simulators, SNS is already bundled with VSimRTI and can be used directly out of the box, and due to Java language, in an operating system independent way.

SNS implements abstract communication models to support the following important communication principles for V2X applications:

**TopoCast** currently implemented as Singlehop Topocast, to communicate either to all or an individual destination node in the direct communication range of the sender.

**GeoCast** to communicate, similar to Georouting, to nodes in a certain geographic destination area, with the options of unicast (to address one node), broadcast (to address all nodes) and anycast (to address the first one of all reachable nodes).

SNS does not implement a full featured IEEE 802.11p communication stack, as the other simulators could do. It does not consider particular detailed properties of the communication, e.g. signal fading models, PHY Layer reception, MAC Layer scheduling, detailed Geo Routing protocols etc. Due to this, SNS is an interesting alternative several reasons and suited for users aiming for quick and easy use without the need of detailed models.

However, when detailed communication characteristics need to be considered, the other simulators are the better choice.

### 3.7.1 sns-ambassador folder structure

The SNS can be configured via the sns_config.json which can be found within the scenarios folder structure:

```plaintext
<scenarioName>
  sns
    sns_config.json ................................. SNS configuration file

Figure 3.10: sns-ambassador folder structure
```

### 3.7.2 Installation

This simulator does not need to be installed. It is delivered as part of the VSimRTI-installation package.

### 3.7.3 Configuration

The SNS configuration basically consists of the two components for **singlehop** (direct Topocast) and **multihop** (Georouting) transmission. Each of these transmission modes base on the delay configuration, which is already known from the Cell2 Simulator (3.6).
The singlehop component additionally includes the singlehop \texttt{radius}, which actually is the direct communication range of the sender - meaning that all nodes within this distance can successfully receive the according message. In the context of the fundamental Freespace Model, which is implemented by the classic network simulators OMNeT++ and ns-3 this value results from the carrier frequency, transmission power, receiver sensitivity and antenna properties (gains) etc. Additionally, the delay supports the value for packet retransmission and packet loss (\texttt{prpl}), which allows the simulation of packet errors within the communication range. However, in contrast to fading models as the Rayleigh or Rice Fading, the simulated packet errors in SNS are distributed uniformly over the whole communication range and do not, for instance, increase with higher distances (which would be more realistic).

The multihop component only consists of the delay configuration including the value for prpl and according transmission retries. In this case, the destined communication area is determined by the dissemination area in the \texttt{GeocastDestinationAddress}.

An example definition for SNS, with relatively short delays in the order of microseconds or low milliseconds for direct singlehop and slightly higher delays for multihop could look like this:

```json

{"version": "0.16.0",
"singlehop": {
   
   "_singlehop radius in m": 0,
   "radius": 509.4,
   "_delay config according to vsimrti - communication": 0,
   "delay": {
      "type": "SimpleRandomDelay",
      "steps": 5,
      "minDelay": 0.4,
      "maxDelay": 2.4,
      "prpl": 0,
      "retries": 0
   }
 },

"multihop": {
   
   
   "_delay config according to vsimrti - communication": 0,
   "delay": {
      "type": "GammaRandomDelay",
      "minDelay": 10,
      "expDelay": 30,
      "prpl": 0,
      "retries": 2
   }
 }

```

3.7.4 Operation

The individual steps of operation for the two modes of Topocast (direct singlehop) and Geocast (georouting with multihop) transmission are as following.

**Simulation of direct singlehop transmission**

1. get transmission radius from configuration (to simulate tx power, carrier frequency and antenna properties etc.)
2. determine potential receiver nodes in the sending radius (without original sender)
3. calculate equal delay for all nodes
4. simulate successful message transmission (via prpl) for each node

**Simulation of geocast routing transmission**

1. get destination area from geocast header in message (to simulate georouting)
2. determine potential receiver nodes in the destination area (including original sender due to re-broadcasting in geocast)
3. calculate specific delay for each node
4. simulate successful message transmission (with packet loss and possible retransmissions via prpl) for each node
3.8 VSimRTI Eventserver

3.8.1 eventserver-ambassador folder structure

This ambassador can be configured with a configuration file. The specific path is
vSIMRTI/scenarios/<scenarioName>/eventserver/eventserver_config.json

```
<scenarioName>
  └── eventserver .................................................................
      └── eventserver_config.json  .......................... Eventserver configuration file
```

Figure 3.11: eventserver-ambassador folder structure

3.8.2 Installation

This simulator does not need to be installed. It is delivered as part of the VSimRTI-installation package.

3.8.3 Configuration

Notice: The documentation for the VSimRTI specific component is freely available on the DCAITI website, explaining all available options.

The root node of the configuration is a list of environment events. Each event require the type of the event, a rectangle area, a strength and the time window.
3.9 VSimRTI Battery Simulator

3.9.1 battery-ambassador folder structure

This ambassador can be configured with a configuration file. The specific path is 
vsimrti/scenarios/<scenarioName>/battery/battery_config.json

```
<scenarioName>
  battery .................................................................
  battery_config.json ........................................... ambassador configuration file
```

Figure 3.12: battery-ambassador folder structure

3.9.2 Installation

This simulator does not need to be installed. It is delivered as part of the VSimRTI-installation package.

3.9.3 Introduction

The VSimRTI Battery Simulator is used to simulate electric vehicles. It takes environment, vehicle characteristics and the battery itself into account. The main feature of the battery ambassador is that it can utilize dynamic class loading to use a battery model provided by the user, depending on the given needs. This also holds true for the environment model and the vehicle model. However, simple models for vehicle, environment and the battery are provided and will be briefly explained in the following sections.

3.9.4 Vehicle model

The vehicle model holds the general properties of a vehicle. Examples would be the weight of the vehicle and the voltage at which the electric engine operates. However, as with the other models, the provided class for the vehicle model directly affects what can be configured here. If other properties are needed for the vehicle, this is the right place to put them. To implement an own vehicle, the class AVehicleModel has to be extended.

3.9.5 Battery model

The battery model defines the battery used by the vehicle itself. Useful properties could be the number of cells and their capacity. As with the vehicle, this class can be freely defined and use configuration values placed in the batteryModelConfig-area. To implement an own battery model, the class ABatteryModel needs to be extended.
3.9.6 Environment model

Normally environmental factors like rolling resistance of the given underground or air drag go into this section. At the current state, just a minimal environment model is bundled with the battery ambassador, mainly to show what is possible. To implement a custom environment model, AEnvironmentModel has to be extended.

3.9.7 Sample configuration

```json
{
    "vehicleModelClass": "com.dcaiti.vsimrti.fed.battery.sim.models.vehicle.ElectricSmart",
    "vehicleModelConfig": {
        "Mass": 1060,
        "ReferenceArea": 1.95,
        "DragCoefficient": 0.375,
        "TankToWheelEfficiency": 0.7,
        "ElectricMotorOperatingVoltage": 350,
        "ConsumerOperatingVoltage": 12
    },
    "batteryModelClass": "com.dcaiti.vsimrti.fed.battery.sim.models.battery.VerySimpleBatteryModel",
    "batteryModelConfig": {
        "NumberOfCells": 93,
        "CellVoltage": 4.3,
        "CellCapacityInAh": 50,
        "eff_Summer": 0.8448,
        "RechargingType": 2,
        "MinSOC": 0.30,
        "MaxSOC": 0.70
    },
    "environmentModelClass": "com.dcaiti.vsimrti.fed.battery.sim.models.environment.DefaultEnvironment",
    "environmentModelConfig": {
        "Gravity": 9.81,
        "FluidDensity": 1.293,
        "RollingResistanceCoefficient": 0.01
    },
    "consumers": [
        { "Radio": 10 },
        { "HeadLight": 100 }
    ]
}
```

This listing shows how the vehicle, environment and battery model classes for the bundled models are configured. Additionally, an arbitrary number of consumers can be configured that draw additional power from the battery. In this case, headlights and a radio are pre-defined.
This tutorial aims to provide a general overview of the VSimRTI application concept and shows two examples that involve ad hoc communication via IEEE 802.11p between different participants and message passing from one application to another that run on the same vehicle. Additionally, a short introduction will be given on Traffic Lights, with a more in-depth description following in Chapter 6. The tutorial is split up into five main parts:

1. How to equip a vehicle with one or more applications. This process is called mapping.
2. An example application that shows how to implement communication between different vehicles via a wireless ad hoc network.
3. The next part of the tutorial shows how to accomplish message passing between two applications that run on the same vehicle.
4. An overview of the traffic light application and a summary about what can be done with the traffic light from within the application.
5. The last part of the tutorial shows how to retrieve the results of a simulation run.

The scenario itself consists of three vehicles that drive down a road in consecutive order and pass a Road Side Unit (RSU) that emits messages in a fixed interval. The vehicles will receive the messages from the RSU as soon as they are in communication range. After completing this tutorial the reader should be able to deploy own applications according to his needs and make use of ad hoc communication among vehicles and intra-vehicle communication among applications on the same vehicle. The VSimRTI cell simulator that is used to simulate cellular network communication will be covered in tutorial 2.
4.1 Mapping Configuration

In order to use applications they have to be assigned (mapped in VSimRTI terminology) to a simulation entity. In this tutorial, we will assign applications to a RSU that is placed along the road (the red symbol in picture 4.1) and to the vehicles. In order to do this the following steps are necessary:

1. Navigate to the mapping3 folder of the tiergarten tutorial scenario.
2. Edit the mapping_config.json to let it point to the correct classes for your application, define prototypes and to add entities to the simulation.

The mapping is already configured correctly, so we will continue with a description of the important parts of the mapping configuration.

Prototype Section

This section contains the properties of the simulated entities, including their applications. You can, for example, configure the maximum speed and the length of a vehicle. Normally, the default values are fine here. In order to map one or more applications you have to fill in the complete class identifier including the package in the applications array. In this tutorial, we have mapped two applications to the vehicles and one application to the RSU.

Mapping for the vehicles:
"applications": ["com.dcaiti.vsimrti.app.tutorials.tiergarten.TiergartenVehicle",
"com.dcaiti.vsimrti.app.tutorials.tiergarten.TiergartenVehicleSlave"]

Mapping for the RSU:
"applications": ["com.dcaiti.vsimrti.app.tutorials.tiergarten.TiergartenRSU"]

In order for VSimRTI to be able to locate these classes, the resulting .jar archive should be placed in the applicationNT folder of your scenario. For our tutorial we have packaged the needed classes TiergartenVehicle, TiergartenVehicleSlave and TiergartenRSU into one .jar file. However, this isn't a strict requirement. What these classes actually do and how they are implemented will be covered in the next two parts of this tutorial.

Vehicle and RSU sections

This part of the mapping is used to actually bring entities like vehicles and RSUs into the simulation. For example, we placed the RSU that is equipped with the TiergartenRSU application at the specific geo coordinate with latitude = 52.5130607 and longitude = 13.328910. Usually, this and the mapping of its application is all that is necessary to make use of an RSU. For this tutorial we go with just one RSU but analogous to adding multiple vehicles we could also add more of them if the need arises.

Spawning vehicles is a little bit more complex than adding a RSU. Vehicles are configured as groups from which we have three in this tutorial scenario. Exemplary, we will describe one of these groups in more detail:
```
"vehicles": [
  {
    "startingTime": 1.0,
    "route": 0,
    "maxNumberVehicles": 1,
    "types": [{"name": "PKW"}]
  },
  ...
]
```

Listing 4.1: Example Vehicle Group

- **startingTime**: Defines at which second the vehicle is spawned, in this case one second after the beginning of the simulation.
- **route**: Describes which route these vehicle group should use. For now it is sufficient to know that there is one route with the id 0 that leads the vehicles near the RSU that is already defined in this tutorial scenario.
- **maxNumberVehicles**: Used to determine how many vehicles should be in that group. Here, we go with just one single vehicle.
- **types**: This array maps to the defined PKW (german for passenger car) prototype from the prototypes-section of the mapping.

### 4.2 Inter-Vehicle Communication

This second part describes the TiergartenVehicle and TiergartenRSU applications which are used for inter-vehicle communication in more detail. As a coarse overview, the TiergartenRSU application sends out a defined message at a fixed interval. These messages are received and written to the log file by the TiergartenVehicle application. First, we start off with the class definition and will run through the methods we have to implement in order to get the communication working.

**Class definition**

```java
public class TiergartenVehicle extends AbstractApplication<VehicleOperatingSystem> implements VehicleApplication, CommunicationApplication
```

Listing 4.2: TiergartenVehicle: Class definition for a vehicle application

In general, every vehicle application will extend the AbstractApplication abstract class with the VehicleOperatingSystem type parameter. The VehicleOperatingSystem is the main way to interact with the underlying vehicle and gives a wide array of functionality to the user. Also, depending on the needs of the application, several interfaces have to be implemented. For our scenario we implement VehicleApplication to denote that the application is intended to run on a vehicle and CommunicationApplication to be able to use the actual communication facilities we need for this tutorial. In general, it makes sense to let the IDE generate the bodies of the methods that need implementation and fill them with functionality if needed or letting them empty otherwise.
### Application initialization

During the initialization procedure of communicating applications, the communication modules (WLAN-Module or CellModule) need to be activated. For example, activating the WLAN module can be achieved with the following code snipped:

```java
@Override
public void setup () {
    getLog ().infoSimTime (this , " Initialize application ");
    getOperatingSystem ().getAdHocModule ().enable (new AdHocModuleConfiguration ()
        .addRadio ().channel (AdHocChannel .CCH).power (50).create ()
    );
    getLog ().infoSimTime (this , " Activated AdHoc Module ");
}
```

Listing 4.3: TiergartenVehicle activate WLAN module

Since we want to communicate over ad hoc Wifi we have to turn on the Wifi-module of the car. Here you can specify the mode of operation. The first argument says if the vehicle should be able to receive messages, how strong the Wifi antenna is (50mW in this case) and which ad hoc channel to use.

**Note:** The honoring of these arguments depends on the underlying network simulator. Currently, **ns-3** and **OMNeT++** make use of these arguments. For **SNS**, the method `distance(250)` shall be used instead of `power(50)`.

### Receiving the V2X messages

In order to act upon received messages from other simulation entities, the `receiveV2XMessage` from the `CommunicationApplication` interface is used. In our tutorial scenario we don't do something special upon message receiving and just write the name of the sender into the log file.

```java
@Override
public void receiveV2XMessage ( ReceivedV2XMessage receivedV2XMessage ) {
    getLog ().infoSimTime (this , "Received V2X Message from ",
        receivedV2XMessage .getMessage ().getRouting ().getSourceAddressContainer ().getSourceName ());
}
```

Listing 4.4: TiergartenVehicle: Class definition for a vehicle application

This is basically all it takes to receive messages from another simulation entity. Normally, the application author uses an own message class that derives from V2XMessage and casts it to his specific class in order to handle messages easier.

### Sending the V2X messages

Since the receiving application is now set up we move on to the sending side of this tutorial. The sending takes place from the RSU via a broadcast, so every vehicle in transmission range will receive the message sent by it. Picture 4.2 shows the communication range of the RSU which is dependent from the settings in the communication simulator. The actual sending takes place in the `TiergartenRSU` application which is equipped to the RSU via the mapping configuration.
ApplicationNT event model

In order to send messages at a fixed interval we make use of the event based model of the VSimRTI application simulator. A high level description in what we need to do in order to send messages at a specific interval can be summarized as follows:

1. Line up an event every X seconds. For this tutorial an interval of two seconds was chosen.
2. Once the event gets processed in the `processEvent()`-method the actual sending of the message will be triggered.
3. All information necessary for the sending will be assembled in `sendAdHocBroadcast()` and the message gets actually sent out.

Application Setup

```java
@Override
public void setUp () {
    getLog ().infoSimTime (this, "Initialize application");
    getOperatingSystem () .getAdHocModule () .enable (new AdHocModuleConfiguration ()
        .addRadio () .channel (AdHocChannel.CCH) .power (50) .create ()
    );
    getLog ().infoSimTime (this, "Activated WLAN Module");
    sample ();
}
```

Listing 4.5: TiergartenRSU: Setup for the RSU application

The setup for the RSU is the same as for the vehicle where we activate the Wifi module. Additionally, the `sample()` method gets called which is responsible for the events.
Event scheduling

The next step is to line up the event at the interval we desire. The method we use for that, `sample()`, looks like this:

```java
public void sample () {
    final Event event = new Event ( getOperatingSystem (). getSimulationTime () + TIME_INTERVAL , this);
    getOperatingSystem (). getEventManager (). addEvent ( event);
    getLog (). infoSimTime (this , " Sending out AdHoc broadcast ");
    sendAdHocBroadcast ();
}
```

Listing 4.6: TiergartenRSU: Event sampling

Here, an `Event` is created which will be received at the current time in simulation plus the defined interval which is set to two seconds. The second argument for the event creation is a reference to the originating class instance. For our tutorial, the `this` reference is sufficient here. After we created the event, it is added to the event queue via `addEvent()` which will result in a call to `processEvent()` at the given interval.

Message sending

After we made sure the method that does the actual sending of the message is called at the specified interval, we take a closer look at the `sendAdHocBroadcast()` method we defined in the `TiergartenRSU` class:

```java
private void sendAdHocBroadcast () {
    final MessageRouting routing =
        getOs (). getAdHocModule (). createMessageRouting (). topoBroadCast ( AdHocChannel . CCH );
    final InterVehicleMsg message = new InterVehicleMsg ( routing , getOs (). getPosition () );
    getOs (). getAdHocModule (). sendV2XMessage ( message );
}
```

Listing 4.7: TiergartenRSU: Event sampling

This method has basically three tasks to solve:

1. Firstly, a routing description has to be created, which defines the communication type, the source and destination for the message, and additional properties like the maximum number of hops and the communication channel. In this specific case, `topoBroadcast(AdHocChannel.CCH)` creates a message routing which includes the IP address of the `RSU` as the source, the broadcast address as the destination, and a communication path using the present ad-hoc topology with only one hop.

2. Secondly, the message to be send is created. The previously created message routing description is passed to this custom `V2X` message.

3. Finally, the message is send using the ad-hoc module of the `RSU`.

These steps conclude the sending of the message. The procedure is the same for sending messages from a vehicle instead of a `RSU`. 
4.3 Intra-Vehicle Communication

This part of the tutorial describes the steps necessary for letting two applications to communicate with each other on the same vehicle. Here, we use the TiergartenVehicle application as the sender and TiergartenVehicleSlave as the receiver. In general, the approach is similar to the sending of a V2X message and also makes use of the event system.

Message sending

First, we start off with the sending side. The event code should look familiar:

```java
@Override
public void afterUpdateVehicleInfo() {
    final List<? extends Application> applications = getOperatingSystem().getApplications();
    final IntraVehicleMsg message = new IntraVehicleMsg(getOperatingSystem().getId(),
        random.nextInt(MAX_ID));

    for (Application application : applications) {
        final Event event = new Event(getOperatingSystem().getSimulationTime() + 1, application/
        \, message);
        this.getOperatingSystem().getEventManager().addEvent(event);
    }
}
```

Listing 4.8: TiergartenVehicle: Sending intra application message

One noteworthy thing is that we use the `afterUpdateVehicleInfo()`-method to line up a new event. This method is called automatically if the vehicle info (for example, speed or heading) gets updated, which usually takes place at every time step of the simulation.

The general approach works like this:

1. Get a list of all applications that run on the vehicle using `getApplications()`.
2. Create a new message. Here we use our own custom message called `IntraVehicleMsg` which takes an randomly generated id and the name of the vehicle as payload. Again, this is for tutorial purposes and could be anything.
3. After that we iterate over every applications that runs on the vehicle, in this case two: TiergartenVehicle and TiergartenVehicleSlave.
4. Then, an event is constructed for each app running on the vehicle and added to the event queue the same way as in the inter-vehicle example.

Receiving the message

The actual receiving of the message takes place in the TiergartenVehicleSlave application. Since the intra vehicle messages are basically treated as an event, the code is straight forward here:

```java
@Override
public void processEvent(Event event) throws Exception {
    Object resource = event.getResource();
    if (resource != null && resource instanceof IntraVehicleMsg) {
        final IntraVehicleMsg message = (IntraVehicleMsg) resource;
        if (message.getOrigin().equals(getOperatingSystem().getId())) {
```

VSimRTI: Vehicle-2-X Simulation Runtime Infrastructure 54
4.3 Intra-Vehicle Communication

The general concept here is that the payload is wrapped as an object in the actual event. The procedure here is as follows:

1. Retrieve the resource from the event using `getResource()`
2. Check if there is actually some kind of payload
3. Make sure the payload is of the expected type
4. Cast the payload to an actual class and do something with it.

### 4.3.1 The traffic light application

As can be seen in the mapping configuration there is an additional prototype defined for traffic lights:

```json
{
  "applications":["com.dcaiti.vsimrti.app.tutorials.tiergarten.trafficLight/uni2926/curvesarrowse/TrafficLightApp"],
  "name":"TrafficLight"
}
```

Listing 4.10: Traffic Light Prototype

These prototype is used to map the referenced application onto two specific traffic lights, as shown in the following listing.

```json
tls=[
  {
    "tlName": "27011311",
    "name":"TrafficLight"
  },
  {
    "tlName": "252864801",
    "name":"TrafficLight"
  }
]
```

Listing 4.11: Traffic Light Mapping

The use case shown for tutorial purposes here is simple: The two traffic lights referenced in the mapping default to always red. Their application then waits for a V2X message with a "secret" payload in it (the `GreenWaveMsg`) and switches the traffic light program to always green upon receiving that message. The application mapped onto the traffic light then switches back to its previous program after 20 simulation time steps have passed. An in-depth explanation of traffic light functionality can be found in Chapter 6.
4.4 Interpretation of simulation results

The last part of the tutorial describes how to retrieve the actual simulation results. For this tutorial, the results are quite simple and simply show the arrival of the Inter- and IntraVehicle messages. So after executing the simulation using the following command:

```
./vsimrti.sh -u <user> -s Tiergarten
```

Listing 4.12: Running the simulation with the Tiergarten scenario (Unix machine)

```
vsmrti.bat -u <user> -s Tiergarten
```

Listing 4.13: Running the simulation with the Tiergarten scenario (Windows machine)

Afterwards, in the log directory of VSimRTI a new folder should be created containing all log files of the simulation run. Within, the sub-folder `appsNT` contains the log files for each simulation unit and its application. For example, for the vehicles we end up with two log files: `TiergartenVehicle.log` and `TiergartenVehicleSlave.log`.

This following snippet shows the receiving of the V2X messages that were sent by the RSU:

```
INFO - Initialize application (at simulation time 6.000,000,000 s)
INFO - Activated AdHoc Module (at simulation time 6.000,000,000 s)
INFO - Received V2X Message from rsu_0 (at simulation time 18.000,400,000 s)
INFO - Received V2X Message from rsu_0 (at simulation time 20.000,400,000 s)
```

Listing 4.14: Snippet from TiergartenVehicle.log

Next, we see the contents of the `IntraVehicleMessage` that originates from another app on the vehicle:

```
INFO - Initialize application (at simulation time 2.000,000,000 s)
INFO - Received message from another application: IntraVehicleMsg { origin='veh_0', id=631} ...
```

Listing 4.15: Snippet from TiergartenVehicleSlave.log

The following log was generated by the RSU application and logs the sending of each ad hoc broadcast:

```
INFO - Initialize application (at simulation time 0.000,000,000 s)
INFO - Activated WLAN Module (at simulation time 0.000,000,000 s)
INFO - Sending out AdHoc broadcast (at simulation time 0.000,000,000 s)
INFO - Sending out AdHoc broadcast (at simulation time 2.000,000,000 s)
```

Listing 4.16: Snippet from TiergartenRSU.log

This concludes the first tutorial and hopefully gave an idea on how to use the VSimRTI application simulator to send out and receiving messages to other simulation entities or inside the same vehicle.

The `OperatingSystem.log` files do not contain specific application output and is mainly used for debugging purposes and won't be discussed in this tutorial.
5 Tutorial Barnim

The following tutorial shows some of the more advanced features of VSimRTI and will take a closer look at different federates. The term federates describes the High Level Architecture and basically denotes a dedicated simulator with whom the simulation infrastructure can communicate. The following topics will be covered in this tutorial:

- How to use and react to DENMs Decentralized Environmental Notification Message.
- Enabling and using a different communication simulator on the basis of the VSimRTI cell simulator.
- The integration of electric mobility aspects into a simulation.

**Note:** The topics from the last tutorial, especially how to map applications and the basics of how to implement them are not covered in this tutorial. If you are not yet familiar with these concepts you should have a look at the preceding tutorials.

### 5.1 Overview

The scenario used for this tutorial is more complex than the Tiergarten tutorial; thus, we will provide you a short explanation. Picture 5.1 shows an overview of the Barnim scenario, exhibiting an icy road section along the highway.
In this scenario, several cars drive on the blue route and are forced to slow down in a specific section due to icy conditions. The rest of the scenario can be described as follows:

1. A car (Car A) which is equipped with ad hoc communication (WiFi) capabilities detects an environmental hazard - in this case an icy section of the road.
2. Car A now sends out a DENM which reaches other cars in its (relatively small) dissemination area. With multi hop routing, the DENM message is transported upstream towards other vehicles.
3. Cars that do not have any form of communication equipment are not warned and drive towards the icy part of the road. Since they have careful and responsible drivers they slow down to avoid accidents.
4. Cars that are equipped with the appropriate communication equipment are able to receive the DENM, which induces them to use a different route (green) which is safer and faster due to the lack of ice on it.
5. Last but not least, the WeatherServer (technically implemented as an RSU) propagates information over the cellular network and could therefore be located virtually everywhere.

We also have some electric vehicles here, mainly to demonstrate how the VSimRTI battery ambassador works. Later in this tutorial, we will show you how to access statistics such as the state of charge of the battery from within an application.

### 5.2 Mapping configuration

This section gives a short explanation of the mapping we use in this scenario. First of all we use five different types of entities. One RSU which acts as the WeatherServer and four types of cars, each of them loaded with different applications. As usual, the configuration takes place in mapping3/mapping-config.json in your scenario folder.

1. Normal vehicles which cannot communicate shall demonstrate how to use electric vehicles in a scenario (60% of all vehicles). They are only equipped with the com.dcaiti.vsimrti.app.tutorials.barnim.SlowDownApp-application which induces them to reduce their speed as soon as their onboard sensors detect hazardous conditions. After leaving the hazardous area, they will resume by increasing their speed again.
2. Vehicles equipped with ad hoc wifi. They might not be able to receive the DENM due to range limitations and drive into the icy section nonetheless. They are equipped with the com.dcaiti.vsimrti.app.tutorials.barnim.WeatherWarningApp-application and the com.dcaiti.vsimrti.app.tutorials.barnim.SlowDownApp-application (20% of all vehicles).
3. Cellular communication enabled vehicles which are able to communicate with the WeatherServer. These vehicles are equipped with the com.dcaiti.vsimrti.app.tutorials.barnim.WeatherWarningAppCell-application, which is a specialized form of the normal weather warning application that can make use of cellular communication. They are also equipped with the com.dcaiti.vsimrti.app.tutorials.barnim.SlowDownApp-application (10% of all vehicles).
4. Electric vehicles cannot communicate since they are only equipped with the com.dcaiti.vsimrti.app.tutorials.barnim.SlowDownApp-application. Apart from that, they also incorporate an electrical battery which empties during driving (10% of all vehicles).

5. The WeatherServer is only equipped with cellular equipment. Despite the greater distance it is able to warn vehicles that can also make use of cellular communication.

### 5.3 VSimRTI configuration

Since several applications are mapped, we need to make sure that the corresponding simulators are enabled. This is done in the main VSimRTI configuration file that resides in vsimrti/vsimrti_config.xml in your scenario. The last part of that configuration file is used to enable and disable certain simulators according to the needs of the user.

Per default, any simulation of cellular communication is disabled. In order to enable communication via cellular networks in this scenario, you need to enable the cell2 simulator by setting the field active to true:

```xml
<!-- Cellular network simulators -->
<federate id="cell2" active="true"/>
```

Listing 5.1: VSimRTI configuration for barnim example scenario

### 5.4 Applications

This section deals with the specific details of the WeatherWarningApplication. We will cover in particular how to enable and use cellular communication, how to react to DEN-Messages and how the user can achieve a re-routing of vehicles.

**Overview**

Here we will explain the application logic and the main ideas behind it in general terms and will go into the implementation details in the following sections. All of the subsequent points stay the same regardless of what form of communication is used.

- If the sensor picks up an event, the vehicle sends out a DEN-message (via cellular communication or ad hoc) with the position that was recorded in order to warn other vehicles.

- The application also contains code to react upon DENMs received from other vehicles. Depending on the configured behavior of the application, either a slow down or a re-routing is performed.

Most of the things that are happening here like re-routing, interval-handling and sending out ad hoc messages were already covered in the first tutorial and will not be discussed in detail in this tutorial.
5.4.1 DEN-Message handling

Receiving

First of all, we look at the receiving side of the DENM here. Since the message is a normal V2X message it is received through the `receiveV2XMessage()`-method which is part of the application interface.

```java
if (!(msg instanceof DENM)) {
    getLog().info("Ignoring message of type: {}", msg.getClassName());
    return;
}
```

Listing 5.2: Testing whether or not message is a DENM

Analogous to a normal message, we check with `instanceof`, if it is of a type that we are interested in, in this case `DENM`. Afterwards, we perform a potential route change if not already done.

Sending

In case the sensor detects an environmental hazard the vehicle sends out a DEN-message to warn other vehicles. If `WeatherWarningAppCell` is mapped, cellular communication is used, in case of `WeatherWarningApp` ad hoc WiFi is used as described in the Tiergarten tutorial.

5.4.2 Cellular Communication

We will now take a closer look at how the communication via the VSimRTI cellular simulator works. The application we point to in the mapping is a specialized form of the normal WeatherWarning Application and can be found under its full identifier `com.dcaiti.vsimrti.app.tutorials.barnim.WeatherWarningAppCell`. It overloads the `useCellNetwork`-method which returns a constant true in order to indicate that we want to use cellular communication inside the base application. The reason for this overloading is to provide a distinct class to which we can point in the mapping. The actual logic that is required to send messages via cell is done in the base WeatherWarning application, so the overloading just signals that we want to use cellular communication.

The method where the actual sending takes place is called `reactOnEnvironmentData()`. The following code snippet shows the steps necessary to send a DEN-message via the cellular network:

```java
GeoCircle dest = new GeoCircle(v_longLat, 3000);
MessageRouting mr;
if (useCellNetwork()) {
    mr = getOperatingSystem().getCellModule().createMessageRouting().geoBroadcast(dest);
} else {
    mr = getOperatingSystem().getAdHocModule().createMessageRouting().geoBroadcast(dest);
}
DENM denm = new DENM(mr, getOperatingSystem().getSimulationTime(), v_longLat, roadId, type, \
    strength, newSpeed, 0.0f);
getOperatingSystem().getAdHocModule().sendV2XMessage(denm);
```

Listing 5.3: Example for sending over cellular networks
The sending is the same as in the Tiergarten tutorial, except that we decide via an if-statement whether or not cellular communication or ad hoc WiFi is used. Note the two different methods - one is ending with `Cell` the other with `AdHoc`, depending on what we want to use in the application.

### 5.5 Conclusion

This concludes the description of the VSimRTI tutorial scenarios Tiergarten and Barnim. The following list sums up the things we have covered in this tutorial series:

- Getting a general idea on how to configure VSimrti, especially how to enable and disable different, coupled simulators.
- How to map different applications to a vehicle or a road side unit
- Achieving inter- and intra-application communication...
- ... via cellular networks or AdHoc wifi
- Receiving and handling of messages
- Basic interaction with the OperatingSystem by means of changing a route
- How to find and process the results of a simulation run with the help of generated log files.
6 Tutorial Traffic Lights

In this brief tutorial we will discuss how to interact with traffic lights via VSimRTI. Usually the position and the lanes they control are coming from the OpenStreetmap-data, which we will also assume during this tutorial. Note that this tutorial is intended to be used with the Tiergarten scenario described in the first tutorial in Chapter 4, so creating a new scenario for is not necessary here and the Traffic Light data is already exported.

At first we cover the general steps necessary to enable traffic lights and eventually “map” applications onto them. Each step is described in more detail later on.

1. Tell scenario-convert to export the traffic light information into the database
2. Determine the node ids of the traffic lights which should have applications on them
3. Adjust the mapping configuration of VSimRTI accordingly and assign the desired application(s) onto one or more traffic lights.

Additional information on how to use traffic lights, how they are modeled within sumo and how their phases work can be found in the sumo wiki under http://sumo.dlr.de/wiki/Simulation/Traffic_Lights

6.1 Use scenario-convert to export traffic lights

Although the Tiergarten scenario which we are focusing on in this tutorial already has the traffic lights exported from the .osm file, this step is important if you desire to set up an own scenario. For the Tiergarten example, the scenario-convert command line looks like this:

```
scenario-convert-19.1.jar --osm2sumo -d tiergarten.db -1 tiergarten.osm -l -n
```

Listing 6.1: Export of traffic lights using scenario-convert

Note the -l switch here that actually exports the traffic light information from the .osm file to the tiergarten.db database. If this command line switch is omitted the traffic lights will not be available in the scenario. More information on how traffic lights are handled by Open Streetmap can be found in their wiki: http://wiki.openstreetmap.org/wiki/Tag:highway%3Dtraffic_signals
6.2 Determine the traffic lights that should be equipped with applications

The next step is to decide which traffic lights should have applications on them. Since their ID(s) have to be referenced in the mapping, we will have to determine them first. There are several ways to do this, e.g. taking them directly from the database using a tool like sqlitebrowser or extract them directly with a text editor from the .net.xml file that gets generated by scenario-convert for usage with sumo. However, here we will focus on a graphical approach using sumo-gui since it is the easiest way to determine the exact traffic lights we want to map applications too.

![Figure 6.1: Locating traffic lights in sumo gui](image)

Figure 6.1 shows how to locate a specific traffic light, or more precise, a traffic light group. The process is as simple as scrolling to the desired traffic light, right click on it and write down the number behind tlLogic in the drop down menu or use the copy to clipboard feature of the sumo gui. This id is then used in the next section which explains how to actually map applications to a traffic light.

6.3 Configure the mapping for traffic lights

Now that we have the ids of the traffic lights we can modify the mapping to assign application(s) to them. Note that the tutorial scenario already has applications assigned to two arbitrary traffic lights, so the mapping file bundled with it can also be used as a starting point here. In general, the mapping looks the same as for vehicles or RSUs. First, we define a prototype that is used for all subsequent traffic lights:

```json
{
    "prototypes": [
        {
            "applications": ["TrafficLightControlApp"],
            "name": "TrafficLight"
        }
    ]
}
```

Listing 6.2: Prototype for traffic light
The listing 8.2 shows an additional prototype called TrafficLight that gets associated with the TrafficLightExample application. Note that the application that should run on the traffic light has to extend AbstractApplication<TrafficLightOperatingSystem> in order to work. The next step is to use this prototype and assign it to a traffic light we found in the section before using the sumo gui:

```
"tls": [
  {
    "tlName": "27011311",
    "name": "TrafficLight"
  },
  {
    "tlName": "252864801",
    "name": "TrafficLight"
  }
]
```

Listing 6.3: Example mapping of traffic lights

Listing 6.3 shows as an example how this prototype is assigned via the name attribute to specific traffic lights. The important aspect here is that tlName refers to the id of the traffic light we found via the sumo gui in section 6.2 which must exactly match each other.

### 6.4 The Traffic Light API

In order to access the traffic light functionality from within an application it must extend the AbstractApplication<TrafficLightOperatingSystem> abstract class. There are two main functionalities provided by it, reachable from the application operating system that can be acquired by using the getOperatingSystem() method:

- Change the currently active traffic light program via setProgramById()
- Adjust the remaining time of the currently active phase via setPhaseRemainingDuration()

However, adding a new traffic light program from within the application at runtime using the VSimRTI application interface is currently not possible and has to be done beforehand via sumo as described here: [http://sumo.dlr.de/wiki/Simulation/Traffic_Lights](http://sumo.dlr.de/wiki/Simulation/Traffic_Lights). The two main ways of doing this is to use an additional file or the tls_csv2SUMO.py tool bundled with sumo. However, it is possible to add a new program by using the TraCI interface as described here: [http://sumo.dlr.de/wiki/TraCI/Change_Traffic_Lights_State](http://sumo.dlr.de/wiki/TraCI/Change_Traffic_Lights_State).
7 Tutorial LuST

The Luxembourg SUMO Traffic (LuST) scenario is a traffic simulation scenario which aims to provide realistic traffic patterns of a common mid-size European city during an average day [4]. This traffic simulation scenario for SUMO has been developed by Lara Codecá et al. and is publicly available via GitHub¹. The following tutorial shows, how the LuST scenario can be integrated into VSimRTI in order to assess your own mobile application at large scale.

Notice: Also, you can use this scenario as a blueprint for integrating your own prepared SUMO scenarios into VSimRTI.

![Figure 7.1: Overview of the simulation site of the LuST scenario [4]](https://github.com/lcodeca/LuSTScenario)

- Area: **156 km²**
- Total length of all roads: **930 km**
- Total length of highways: **89 km**
- Number of intersections: **4.473**
- Number of traffic lights: **203**
- Number of vehicles during 24h of simulation time: **215,526**

¹https://github.com/lcodeca/LuSTScenario
7.1 Execute the LuST scenario with VSimRTI

You can find a prepared VSimRTI scenario for the LuST scenario in the pre-installed scenario directory. However, to get this scenario working, you need to execute the following steps:

1. In the subdirectory `sumo` of the LuST scenario directory, you can find several scripts which will help you to download the actual SUMO scenario from the official sources on GitHub. Depending on which operating system you use and which version control system client you have installed, you can use one of the provided batch or shell script files. If you do not have a git or svn client installed, you can also manually download and unzip the SUMO scenario from https://github.com/lcodeca/LuSTScenario. Please note that the SUMO scenario must be placed inside the `sumo` subdirectory of the scenario.

2. In order to execute VSimRTI with the scenario, you need to pass the given `defaults.xml` to the executable. On Windows, this can be done by executing VSimRTI as follows:

   
   ```
   vsimrti.bat -u <user-id> -s LuST -d scenarios/LuST/defaults.xml -w 0
   ```

   Please note that the execution of this scenario might take a long time, due to the massive amount of vehicles simulated in this scenario. If you want to assess your mobility applications in combination with further simulators (e.g. Cell2), the simulation might slow down even more.

Finally, you can map your own applications onto vehicles using the prepared `mapping_config.json`. Please note that applications can only be mapped for each vehicle type (see limitations below). Furthermore, you can activate any communication simulator, such as Cell2 or SNS, to integrate communication simulation into your assessment.

7.2 Current limitations

The integration of the LuST scenario is currently a work in progress and therefore not all features are available. The following limitations currently exist:

- **No mapping of applications on single vehicles.** It is currently not possible to map applications on single vehicles. Instead, only vehicle types can be mapped with them. Of course, the applications are running solely on each simulated vehicle which belongs to the respective vehicle type.

- **No definition of own vehicle spawner.** Currently, all simulated vehicles are spawned by the SUMO traffic scenario itself. There is no way to add additional vehicles via the `mapping_config.json`.

- **No routing capabilities in applications.** Applications cannot use the navigation module of the vehicle. Furthermore, detailed information about the current route or the road position (upcoming nodes, type of the current road) is also not available to the applications.

- **No valid support of traffic light applications.** The mapping of applications onto traffic lights has not yet been tested.

  - The LuST scenario has been developed for SUMO 0.32.0 and we can not ensure that newer versions of SUMO still support this scenario. Please be also aware, that VSimRTI does not support SUMO 0.32.0 any more, but only version 1.0.1 or higher.
8 Tutorial Traffic Management

Next to vehicles, road side units, and traffic lights, VSimRTI provides also a simulation unit for traffic management, called Traffic Management Center (TMC). With a TMC, road infrastructure can be integrated and controlled. For example, detectors placed along the road can be used to measure traffic properties, such as traffic flow or traffic density. Furthermore, those measures can be used to open and close lanes, or to adjust the speed limits of roads.

This tutorial provides information about TMC applications and how they can interact with the traffic simulator SUMO.

8.1 Scenario Overview

The following image gives an overview about the scenario Highway, provided in the VSimRTI scenario directory.

![Figure 8.1](image)

- Vehicles drive on the highway from east to west (blue line)
- At one location (red rectangle), induction loop detectors measure traffic properties, (e.g. traffic flow).
- In the area of the orange rectangle, two lanes are closed from simulation time 10s until 200s, resulting in a temporary traffic jam.

8.2 Detector configuration

In order to measure traffic properties, such as traffic flow, or traffic density, detectors have to placed in the scenario. This is achieved by setting up induction loop detectors or lane area detectors in the SUMO scenario.
In the *.sumo.cfg file of the scenario, a so-called "additional file" has to be referenced:

```
<configuration>
  <input>
    <net-file value="highway.net.xml"/>
    <route-files value="highway.rou.xml"/>
    <additional-files value="highway.add.xml"/>
  </input>
  ...
</configuration>
```

Listing 8.1: Configure detectors in SUMO

This additional file highway.add.xml now defines the induction loop detectors which should be placed within the road network:

```
<additional>
  <e1Detector id="detector_0" lane="319292433_3257049721_1288275815_3257049721_0" pos="700" file="detectors.txt"/>
  <e1Detector id="detector_1" lane="319292433_3257049721_1288275815_3257049721_1" pos="700" file="detectors.txt"/>
  <e1Detector id="detector_2" lane="319292433_3257049721_1288275815_3257049721_2" pos="700" file="detectors.txt"/>
</additional>
```

Listing 8.2: Configure e1detectors in SUMO

Currently, VSimRTI supports e1Detector and laneAreaDetector. To get more detailed information on this topic, please consider the SUMO user documentation.

### 8.3 Configuration of Traffic Management Centers

In the mapping_config.json of the scenario, a traffic management center can be equipped with an application. Each traffic management center is therefore connected with one or more detectors and retrieves updates from those detectors during simulation. A valid configuration of a TMC looks as following:

```
"tmcs": [
  {
    "name": "HighwayManagement",
    "applications": [
      "con.dcaiti.vsimrti.app.tutorials.highway.HighwayManagementApp('3', 2)"
    ],
    "inductionLoops" : [ "detector_0", "detector_1", "detector_2" ]
  },
  {
    "laneAreaDetectors" : []
  }
],
"vehicles": [ ... ]
```

Listing 8.3: Mapping a tmc with an application

Each TMC retrieves a name (e.g. for logging purposes). Secondly, a list of applications can be mapped onto the TMC. Those applications have to implement a specific interface (see next section). Furthermore, a list of induction loops is given, whose values the TMC applications will retrieve during the simulation. The ids given here must refer to e1detectors of the SUMO scenario. The same applies for the list of lane area detectors, provided by the list laneAreas.
8.4 Traffic Management Center Applications

The implementation of a TMC application follows the same scheme as every other application. The following class declaration is used for this purpose:

```java
public class MyTmcApplication
    extends AbstractApplication<TrafficManagementCenterOperatingSystem>
    implements TrafficManagementCenterApplication {
    @Override
    public void afterUpdateLaneDetectors () {
        // is called whenever a lane detector (induction loop) has new values
    }

    @Override
    public void afterUpdateLaneSegments () {
        // is called whenever a lane segment (lane area detector) has new values
    }
}
```

Listing 8.4: Implementing TrafficManagementCenter application

In order to access values from lane detectors (= e1Detector), the following methods can be used:

```java
LaneDetector detector = getOs().getLaneSegment("detector_0");
double avgSpeed = detector.getAverageSpeedMs();
double flow = detector.getTrafficFlow(); // in veh/h aggregated over the last 1500 updates
```

Listing 8.5: Retrieving values from lane detectors

Values from lane segments (= laneAreaDetector) can be accessed as follows:

```java
LaneSegment detector = getOs().getLaneSegment("lane_segment_0");
double avgSpeed = detector.getMeanSpeed(); // in m/s
double density = detector.getTrafficDensity(); // in veh/km
int vehiclesInSegment = detector.countVehiclesOnSegment();
```

Listing 8.6: Retrieving values from lane segments

Properties of lanes (e.g. allowed vehicle classes) can be changed as well:

```java
// close lane completely
getOs().changeLaneState(edgeId, laneIndex).closeForAll();

// open lane completely
getOs().changeLaneState(edgeId, laneIndex).openForAll();

// close lane for trucks only:
getOs().changeLaneState(edgeId, laneIndex).closeOnlyForVehicleClasses(VehicleClass.HeavyGoodsVehicle);

// change maximum speed of lane (in m/s)
getOs().changeLaneState(edgeId, laneIndex).setMaxSpeed(10);
```

Listing 8.7: Change state of lanes
9 Create a new scenario

This chapter is intended to act as a checklist for all tasks, needed to create a new scenario. Basically, a new scenario means that the simulation is performed in a different road network where vehicles have different routes. Moreover, the number of vehicles can also vary.

9.1 Road network data

The first step always is to prepare a road network the simulation should take place in. While in general it is not relevant where the network data is coming from, we only provide ready to go import functions for OpenStreetMap (OSM) data which is freely available and comprehensive. Also if data in the target area is not up to date or needs to be adapted to the specific needs this can be done without problems.

In case there are ready to go scenarios for supported traffic simulators like SUMO we also try to support that. Nevertheless, we cannot guarantee to support all tool specific functions, so we really recommend using OpenStreetMap data as source.

If you already have data from other sources or want to use another data provider, additional steps might be necessary. If you have any questions or need further support, please feel free to contact our support team via our mailing list.

9.1.1 Using OpenStreetMap data

Preparing the data source

For this approach, simply download the *.osm file of the desired region using the openstreetmap.org website or any other tool that supports *.osm file export (e.g. josm\(^1\) or merkaartor\(^2\)) and save it. Be aware that we currently do not support the binary version *.pbf or any compressed versions.

Make sure that the file only contains complete ways. This means the file should not contain ways which reference nodes that are outside the target areas bounds. Usually this only has to be checked if the downloaded data was manually cropped with tools like osmosis\(^3\). Such tools usually also have an option that will remove such ways (for osmosis this is completeWays=yes).

It is best practice to not let the source file contain roads clearly not usable (like bicycle or pedestrian ways). This is to not clutter the simulation with unnecessary data resulting in a serious slow down for some aspects of the simulation like routing. There are several ways to filter OpenStreetMap files, the most convenient way is included in the tool we provide to process the source file, we will get back to that

---

\(^1\)https://josm.openstreetmap.de/
\(^2\)http://merkaartor.be/
\(^3\)http://wiki.openstreetmap.org/wiki/Osmosis
in a minute. If you need more control over the filter process we recommend to use osmosis or one of the editors already mentioned.

**Processing the data**

**VSimRTI** uses a database for internal storage of map and route data. Therefore it is necessary to convert OpenStreetMap data into the **VSimRTI** specific database format. While converting we filter out unnecessary information about the road network and prepare the data in a way, that allows us to provide routing functionality and create a common base for other components in the simulation like the traffic simulator.

To be able to do this we provide the tool scenario-convert which can be found in vsimrti/bin/tools. As already mentioned this tool also provides a basic filtering for OpenStreetMap input. The usual command to use in this case is:

```java -jar scenario-convert-19.1.jar --osm2db -i path/to/source.osm -o```

In this command `-o` provides the default filtering which filters out every street below type tertiary. Executing this will generate you a new file with file ending `.db` which will be named the same as your `*.osm` file. So for the given command it would generate a `source.db` in the directory `source.osm` is found. Additionally a second `*.osm` file with the suffix `_cleaned` will be created which is the filtered source data actually used in the import process. With `-d` you can provide a custom file name for the database alongside a path where it should be placed. If you provide a name please do so without the file extension, the correct one will be added automatically.

Now you should have a brand new database which contains the road network. In the next step you need to export the data in a format that can be understood by the traffic simulator to be used and still matches the information in the database. To do so export that data from the database like this:

```java -jar scenario-convert-19.1.jar --db2sumo -d path/to/database.db -n```

The example execution should have created a number of SUMO specific network files (nodes, edges and connections). The parameter `-n` automatically executes SUMOs netconvert command after the export to create the needed netfile (requires SUMO to be in the systems path). If you want to have traffic lights generated by netconvert, you need to add the option `--export-traffic-lights` to the command above.

Now you have processed the road network to be in formats usable by VSimRTI and its connected simulators.

Regarding files, the output of these two steps should be the following:

- `<scenarioName>.db` - scenario database (including road network)
- `<scenarioName>.nod.xml` - SUMO node file
- `<scenarioName>.edg.xml` - SUMO edge file
- `<scenarioName>.con.xml` - SUMO connection file
- `<scenarioName>.net.xml` - SUMO network file (after executing netconvert)
- `<scenarioName>.sumo.cfg` - SUMO configuration file
9.2 Vehicles and Routes

**VSimRTI** contains a navigation component which supports the usage of different routes, vehicles can be mapped to. In order to make the routes accessible for the simulation in **VSimRTI**, they have to be available in the scenario database. There are two ways to achieve this:

- Generate routes with **scenario-convert**
- Import an existing **SUMO** route file

### Generate routes

**scenario-convert** offers possibilities to generate routes for the simulation. With the following command you can generate routes from one geo-point (latitude, longitude) to another:

```
java -jar scenario-convert-19.1.jar -d path/to/database.db -g --route-begin-latlon LAT, LON --
\% route-end-latlon LAT, LON
```

If geo-points are given as input, the route generation might fail due to unmatchable points. In this case, you may use the following command in order to calculate routes from one street node to another:

```
java -jar scenario-convert-19.1.jar -d path/to/database.db -g --route-begin-node-id NODE_ID --
\% route-end-node-id NODE_ID
```

The required node ids can be determined from the source **OSM** file or the generated **SUMO** network file. For more than one route between the given points/nodes, the parameter `--number-of-routes NUMBER` can be used. The calculated routes are directly written into the database. If you need to export them into a **SUMO** readable format, you can use the command:

```
java -jar scenario-convert-19.1.jar --db2sumo -d path/to/database.db -r <scenarioname>.rou.xml
```

### Import routes from **SUMO** route file

If you want to use the tools provided by **SUMO**, e.g. **duarouter**[^4], to calculate a set of routes for the simulation, you need to import the resulting routes into the scenario-database of **VSimRTI**. Note, if you use such tools, please always use the network file generated by **scenario-convert**. The route file (*rou.xml) generated by those tools can be directly imported into the database by using **scenario-convert**. For this purpose, you can use the following command:

```
java -jar scenario-convert-19.1.jar --sumo2db -r routefile.rou.xml -d path/to/database
```

This command will read the routes in the route file and write them in the supplied database file. The database file needs be the same you used to export the network file. If the route file also contained vehicle definitions these will be converted into a mapping3 configuration file. You will need to adapt this file later on to achieve a meaningful mapping between vehicles and applications. For more information regarding this please refer to chapter 3.5.15.

[^4]: [http://sumo.dlr.de/wiki/DUAROUTER](http://sumo.dlr.de/wiki/DUAROUTER)
9.3 VSimRTI

Now you have everything you need to create a configuration set that VSimRTI will recognize as a scenario. A scenario is generally a folder structure that reflects the different components usually found in a simulation. The example tutorials of Barnim and Tiergarten also follow this specific folder structure.

The complete set of components is depicted in the Listing 9.1.

```
<scenarioName>
 _ applicationNT
 _ battery
 _ eventserver
 _ cell2
 _ mapping3
 _ ns3
 _ omnetpp
 _ sources
 _ sns
 _ sumo
 _ visualizer
 _ vsimrti
```

Figure 9.1: Scenario: folder structure for complete component set

However, a subset of all components already allows reasonable simulations. The most important components are mentioned in the following part.

**Applications and Mapping (applicationNT, mapping3)**

Usually you want the simulated vehicles to be equipped with some kind of applications that influence the vehicles behavior. To do that you copy the jar files of your applications to the folder `<scenarioName>applicationNT`. To further configure the behavior of the application simulator have a look at chapter 3.5.

Having the applications in place you will have to copy your mapping file to the folder `mapping3`. If you don't have one by now have a look at the configuration for the default simulations provided with VSimRTI.

For further information on controlling the mapping see chapter 3.5.15.

**Navigation (applicationNT)**

The generated database file `<scenarioName>.db` needs to be moved into the `applicationNT` folder. As long as you only have one database file in this folder you don't need to do anything more. VSimRTI will recognize the file automatically.

For further info on navigation in the application simulator have a look at chapter 3.5.14.
Traffic Simulator (sumo)

The generated files for the used traffic simulator go into the folder named after that simulator e.g. SUMO. In the folder should be a configuration file that needs to be adapted, to point to the correct simulator specific network file, e.g. `<scenarioName>.sumo.cfg`.

For further info on this type of simulators have a look at chapters 3.4.

Communication Simulator (cell2, ns3, omnetpp, sns)

There is an extensive default configuration provided in the simulators folder in the tutorials of Barnim and Tiergarten for all communication simulators.

Depending on the simulator you will need to tell the simulator the geographic extend of the simulation area. You can find that data in the traffic simulators network file, e.g. SUMOs `*.net.xml` contains this information in the `convBoundary` attribute of the `location` tag.

- For OMNeT++, it concerns the values of `constraintArea` in the `omnetpp.ini`.
- For the CELL simulator, the expansions do not need to be configured directly. However, the areas of the configured regions (in `regions.json`) have to be in line with the scenario location.
- The SNS also comes without an additional expansion definition.

⚠️ Important: Larger values for the spatial expansion of the simulation area are also usually possible for the communication simulators. However, much larger this might decrease the performance. **Smaller values lead to wrong behavior and will crash the simulation!**

For further information on the communication simulators, have a look at Chapters 3.2, 3.3, 3.6, and 3.7.

VSimRTI config

Having done that you are now ready to adapt the `vsimrti/vsimrti_config.xml` file to your needs. This usually contains of setting the scenario id (recommended to use the scenarios folder name), adapting the end time to cover everything you want to analyze.

An important setting is the transformation configuration, as this will control how geographic coordinates will be converted to cartesians and back. Having a correct setting here is crucial to get correct results that map to real world coordinates so the simulation results can be visualized in some way. The center coordinate will be used to determine the correct UTMZone, the cartesianOffset can be determined by having a look at the traffic simulators network file, e.g. SUMOs `*.net.xml` contains this information in the `netOffset` attribute of the `location` tag.

Within the `IPResolverConfig` tag the address resolution scheme is specified. The subnets for all node types are described in JSON format.

Last but not least make sure, that the correct simulators are activated and unused deactivated in the last section of the configuration.

To simulate a generated scenario, copy the now ready scenario into the `vsimrti/scenarios` folder.
**Further components**

Several components (as the battery configuration or the visualizer configuration) are generally scenario independent and can be copied from the existing tutorial scenarios Barnim and Tiergarten.

The event server component (to simulate an icy road etc.) obviously also depends on the scenario location. The position of the event should be in line with and be located in the area of the scenario.
10 Visualizers

10.1 Kinds of Visualizers

**VSimRTI** provides several ways of data evaluation by so called visualizers. To visualize a scenario, different visualizers can connect to a running simulation. Figure 10.1 gives a overview about the current coupled visualizers. More than one visualizer of the same type can be defined.

![Figure 10.1: VSimRTI visualizer structure](image)

The visualization ambassador provides several ways to visualize and/or trace the activities that happen during the simulation. The here explained visualization federate is responsible for providing data (in form of subscribed messages) to the visualization component. The kind of visualization is completely dependent on the component itself.

The **VSimRTI** File Visualizer aims to collect and format data for further processing after simulation. To get an instant impression of a simulation, the WebSocket Visualizer shows vehicle movements on a Map in the browser.

The visualizers are integrated into the simulation with a visualization ambassador.
10.2 VSimRTI WebSocket Visualizer

To get a simple and instant impression of a simulation or to get an idea of how fast it runs or where a simulation is located, the WebSocket Visualizer was created.

It shows an OpenLayers Map with markers, indicating the positions of all vehicles, updated as the simulation progresses.

![Websocket Visualizer Image](image)

Figure 10.2: Red vehicles are sending messages and green vehicles are receiving messages at that specific point of time in the simulation.

To start the visualization, simply open the `bin/tools/visualizer.html` in your browser. As soon as the page has finished loading all of its content, it starts trying to connect to the WebSocket created by the VSimRTI simulation. The WebSocket is also enabled by default for the tutorial scenario Barnim. For more details see the file `Barnim/visualizers/visualizer_config.xml`. As soon as the simulation is running, you should see vehicle markers moving around and indicating if they are sending V2X messages (green) or receiving V2X message (red).

The status bar at the bottom of the page indicates the current connection/simulation state. There are four possible states:

- **Connecting** - Trying to connect to the WebSocket opened by VSimRTI.
- **Simulating** - Periodically fetching simulation data and updating markers accordingly.
- **Simulation finished** - The simulation has finished.
- **Error** - An error occurred. Either there was a problem caused by the WebSocket itself, or a timeout occurred while trying to connect to the WebSocket.

After the simulation has finished, you can click on the reconnect button and than run VSimRTI again.
You can also start the visualization at each simulation run, using the command line parameter `-v`. Then, in the beginning of the simulation VSimRTI will automatically open the `bin/tools/visualizer.html` in your default browser.

⚠️ **Important**: By default, the WebSocket Visualizer does not work on Microsoft Edge. UWP (Universal Windows Platform) apps on Windows 10 do not have direct network access but are subject to a network isolation for security reasons, preventing localhost loopback by default. While Microsoft Edge itself does allow localhost access, it treats localhost as an Internet site, which leads to restrictions e.g. for IPC over IP. To prevent this, an exception for Edge must be added to the network isolation via the following command in an elevated command prompt:

```
CheckNetIsolation LoopbackExempt -a -n="Microsoft.MicrosoftEdge_Bvekyb3d8bbw"  
```
10.3 VSimRTI File Visualizer

The File Visualizer is a tool which gives you the opportunity to log specific VSimRTI message types.

10.3.1 Configuring the File Visualizer

- The main configuration file is located at
  \texttt{vsimrti/scenarios/\langle scenarioName\rangle/visualizer/visualizer\_config.xml}

\textbf{basic configuration}

The following listing shows a basic example for the configuration of the file visualizer:

```xml
<visualizer id="fileVisualizer" enabled="true" update="2" start="2" end="6" loader="com.dcaiti/\nsimrti\fed\visualizer\FileVisualizerConfig">
</visualizer>
```

Listing 10.1: Basic configuration parameters for file visualizer

The usage of the parameters is described in the following table:

<table>
<thead>
<tr>
<th>usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{id}</td>
<td>Sets the id for the visualizer.</td>
</tr>
<tr>
<td>enabled</td>
<td>If set to &quot;false&quot; visualizer is not used (default value is &quot;true&quot;).</td>
</tr>
<tr>
<td>update</td>
<td>Sets the update interval in seconds for the visualizer.</td>
</tr>
<tr>
<td>start</td>
<td>Sets the start time in seconds for visualization. This has nothing todo with the run time of the actual simulation.</td>
</tr>
<tr>
<td>end</td>
<td>Sets the end time in seconds for visualization. This has nothing todo with the run time of the actual simulation.</td>
</tr>
<tr>
<td>loader</td>
<td>Sets where the visualizer is loaded from using the Java-class (see listing 10.1).</td>
</tr>
</tbody>
</table>

Table 10.1: Basic Configuration of file visualizer

\textbf{message record}

- Each message record is derived from a certain message type and composed of several entries, which are separated by Element \texttt{separator}. A more detailed overview about the visualizable message types in VSimRTI is given in the next chapter Results. The configuration of the file visualizer is explained at the example of the VehicleMovements message.

- Attribute \textit{id} indicates the message type, namely the class name of the message.

- Message has also an attribute \textit{enabled}.

- The element \textit{entries} defines the format and content of the finally visualized message record.
The element entries is composed of several sub-elements entry, which correspond to columns of a message record and the sequence of the columns is the same as that of sub-elements entry.

**entry**

There are three basic types of entries. Have a look at the following example:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<message id="VehicleMovements" enabled="true">
  <entries>
    <entry>"MOVE_VEHICLE"</entry>
    <entry>Time</entry>
    <entry>Updated:Name</entry>
    <entry>Updated:Speed</entry>
    <entry>Updated:Position.Latitude</entry>
    <entry>Updated:Position.Longitude</entry>
  </entries>
</message>
```

Listing 10.2: Specific Configuration for message.

**Constant**

Every quoted entry is defined as a constant. The content inside the quotation will be directly visualized into each corresponding message record.

```xml
<entry>"MOVE_VEHICLE"</entry>
```

Listing 10.3: An example for constant type entry.

**Basic Method**

The other two types of entry originate from the getXXX() methods of a certain object. For an entry, the root object for method invoking is the corresponding message class, here VehicleMovements. If a null object is returned before the last method of cascaded methods is invoked, then "null" will be written to the corresponding field. If a null object is returned by iteration method, then all fields involving this iteration method will be set "null".

**Iteration**

```xml
<entry>Updated:Id</entry>
```

Listing 10.4: An example for method type entry with iteration.

The first part of this example is Updated, which means to invoke the getUpdated method of class VehicleMovements. Then a list of VehicleInfo objects is returned. Then :Id remains. The semicolon is an operator for iteration, which means that for each of the VehicleInfo objects in the returned list the getId method is invoked.

**Cascading**

```xml
<entry>Updated:Position.Latitude</entry>
```

Listing 10.5: An example for method type entry with iteration and cascading.
In this example, there is a dot operation. It is a cascade operation. Here `getPosition` method of `VehicleInfo` class is called and a `GlobalPosition` object is returned. Then we continuously invoke the `getLatitude` method of this `GlobalPosition` object.

**Extended Method**

All the methods involved above are the basic methods. There also is some functionality, which we cannot extract from these existing methods. So an extended method set is offered to meet these requirements and also as an extension point in the future.

**simple extended method**

```xml
<entry>TimeInSec</entry>
```

Listing 10.6: An example for simple extended method type entry.

With existing methods of `VehicleMovements` and its super class `Message`, we cannot get the timestamp of a message in second. (only `Message.getTime`, which returns a time in ns, is available). Here, `getTimeInSec` is a method extension for `Message` class. The extended method set will be listed later.

**assisting extended method**

```xml
<entry>Updated:Type</entry>
```

Listing 10.7: An example for assisting extended method type entry.

There are also methods both in the simple and extended method set. Taking `getType` method of `VehicleInfo` as an example. If the type of a `VehicleInfo` object has not been set, this method will return null. But we can take advantage of `AddedVehiclesMapping` messages to get its type. Thus, an extended `getType` will be invoked if `VehicleInfo.getType` returns null.

**Iteration**

The method type of entry definition supports cascaded iteration as follows:

```xml
<entry>List1:List2:Id</entry>
```

Listing 10.8: An example for cascaded iteration.

It is possible to handle several different iterating operations, coming from the entry definition:

```xml
<entry>Senders:Id</entry>
<entry>Receivers:Id</entry>
```

Listing 10.9: An example for multi-level iteration.

`getSenders()` and `getReceivers()` are two different iterations. In this case, a combination of both Ids from the lists will be generated. The result may look like this:

```
sender1.receiver1
sender1.receiver2
sender2.receiver1
sender2.receiver2
```

Listing 10.10: Visualising result of the above listing.
Note: the longest matched prefix will be considered as the same iterating operation, which means they are in the same level of iteration structure.

**Method Sets**

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message.getTimeInSec()</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>Message.getTimeInMS()</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>ReceiveV2Xmessage.getType()</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>SendV2Xmessage.getType()</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>VehicleInfo.getApplicationNr()</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>VehicleInfo.getType()</td>
<td>UnitType</td>
<td>@Deprecated</td>
</tr>
<tr>
<td>VehicleInfo.getTypeInOrdinal()</td>
<td>int</td>
<td>@Deprecated</td>
</tr>
</tbody>
</table>

Table 10.2: A list of all extended methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VehicleMovements.getUpdated()</td>
<td>List&lt;VehicleInfo&gt;</td>
</tr>
<tr>
<td>VehicleInfo.getId()</td>
<td>String</td>
</tr>
<tr>
<td>VehicleInfo.getPosition()</td>
<td>GlobalPosition</td>
</tr>
<tr>
<td>GlobalPosition.getLatitude()</td>
<td>double</td>
</tr>
<tr>
<td>Message.getTime()</td>
<td>long</td>
</tr>
<tr>
<td>VehicleInfo.getType()</td>
<td>UnitType</td>
</tr>
</tbody>
</table>

Table 10.3: A list of some basic methods
10.4 Integrated Test and Evaluation Framework (ITEF)

Notice: ITEF is only available with a commercial license of VSimRTI. For further information on licences, please refer to our mailing list.

The Integrated Test and Evaluation Framework (ITEF) is a webtool for planning and evaluating vehicular communication scenarios. It is suited for field operational tests as well as simulations.

ITEF also offers a variety of visualization features, making a comparison of different vehicles or between equipped and unequipped vehicles easy. It is structured into 4 screens, whereas the following 3 screens are intended for the visualization.

The Replay screen (see Figure 10.3) is intended to be used for an initial overview of the test run. The main feature is the display of the vehicle movements on a map, while the player can be used to playback the movement situation. In this manner, the ITEF allows a location and time dependent evaluation of simulation test runs.

The Evaluate screen (see Figure 10.4) allows the detailed investigation of the correlations in a test run. The main feature of this screen is to display the behavior summarized over the whole run. The structure of this screen with is similar to the Replay screen. However, the focus here is on the detailed (statistical) summary of evaluated metrics.

Finally, the Statistics screen (see Figure 10.5) provides statistical evaluations of the test and simulation run. Currently, statistics on Vehicle Speed, Travel Time, Travel Distance and several more are supported.
10 Visualizers

10.4 Integrated Test and Evaluation Framework (ITEF)

Figure 10.4: ITEF Evaluate Screen

Figure 10.5: ITEF Statistics Screen
10.5 VSimRTI 3D Visualization (PHAPMap)

Notice: The VSimRTI 3D Visualization is only available with a commercial license of VSimRTI. For further information on licences, please refer to our mailing list.

The VSimRTI 3D Visualization Tool is based on the PHABMACS vehicle simulator and uses the same 3D engine and models to visualize vehicle movements and various events which occur during the simulation. Next to the road network, which can be optionally rendered by the visualizer, the following units and events are visualized:

- **Vehicle movements** coming from the traffic simulation
- **Road Side Units** at its defined location
- **V2X-messages** sent via cellular communication (indicated as green circles)
- **V2X-messages** sent via ITS-G5 communication (indicated as blue circles)
- **V2X-messages** received by vehicles (indicated as red circles)

![Figure 10.6: The 3D visualization tool PHAPMap displaying events from the simulation](image-url)
10.6 VSimRTI Statistics Visualizer

Notice: The VSimRTI Statistics Visualizer is only available with a commercial license of VSimRTI. For further information on licences, please refer to our mailing list.

The Statistics Visualizer is another VSimRTI visualisation tool to easily measure basic simulation outcomes. With the Statistics Visualizer you will be able to obtain short or detailed results of the simulation, e.g. travel times or the average speeds of groups of vehicles, or the average flow on induction loops.

10.6.1 Configuration of Statistics Visualizer

- The main configuration file for all visualizers is located at
  vsimrti/scenarios/<scenarioName>/visualizer/visualizer_config.xml

In order to use the Statistics Visualizer, the attribute enabled of the root element visualizer must be set to "true", as shown in the following listing.

```
<visualizer id="statistics" enabled="true" loader="com.dcaiti.vsimrti.fed.visualizer."
  \ StatisticsVisualizerConfig">
  ...
</visualizer>
```

Listing 10.11: Configuration header for Statistics Visualizer.

10.6.2 Specification of the Statistics Visualizer

In this section, we take a closer look at the Statistics Visualizer by using examples and demonstrations. For each type of retrieving data we create a <statistic></statistic> block. Inside the block we define one certain data type we want to retrieve in a <source> element. If you want to retrieve different data types, just create another <statistic> block for each of them.

You can also set the wanted file name in the attribute filename of the statistic element. If the attribute hasn't been given, each <statistic> block will get the name accordingly to the order number, for example "1. StatisticsVisualizer-Block.csv".

In the output attribute two options (short|verbose) can be selected. The short option provides us a compact log file with informations about only the highest level of the retrieved data (e.g. aggregate values of grouped vehicles) in contrast to verbose option which also provides informations about every individual vehicle in each group.

For a successful start, the element source must be placed in the first position in the statistic children element list. Different options of possible data types, one of which must be specified in the source element can be seen below.

```
<statistic filename="ChooseItYourself" output="short|verbose">
  <source>VehicleSpeeds|VehicleHeadways|VehicleStops|VehicleTravelTimes|</source>
  \ VehicleTravelledDistances|VehicleFuelConsumptions|DetectorFlow</source>
</statistic>
```

For using the detector flow type, inductions loops need to be configured in the SUMO and mapping configuration files (e.g. Highway tutorial in chapter 8).

### 10.6.3 Application of the Statistics Visualizer

This section will demonstrate the basic idea and usage of the Statistics Visualizer depending on the individual requirements. Besides the retrieving raw data, the Statistics Visualizer has further features for processing of the obtained data.

1. **source**: Data to obtain, choose between:
   - **VehicleSpeeds**: Obtain the speeds of the vehicles of each simulation step.
   - **VehicleStops**: The total number of stops during the journey of each vehicle.
   - **VehicleTravelTimes**: The total travel time in s of the vehicles.
   - **VehicleDelayTimes**: The deviation of the travel time compared to the fastest travel time possible for the vehicles (in s).
   - **VehicleTravelledDistances**: The travelled distance in m of the vehicles.
   - **VehicleFuelConsumptions**: The fuel consumptions of the vehicles in l per km.
   - **VehicleHeadwayDistances**: Obtain the distance headway (from the front bumper towards the leading vehicle’s back bumper) of each vehicle for each simulation step. To obtain this value, an application has to be deployed on the vehicles which activates the front distance sensor.
   - **VehicleHeadwayTimes**: Obtain the headway time (time needed to cover the current headway distance with the current speed) of each vehicle for each simulation step. For both VehicleHeadwayDistances and VehicleHeadwayTimes it is possible to use an attribute "maxDistance" to manage the distance border, from which the headway distance values will not be taken into consideration for computing results.
   - **DetectorFlow**: The flows of each subscripted induction loop.

2. **group-by**: The vehicles will be grouped by its vehicle type name (VehicleType), group they belong to (VehicleGroup), or obtained data value (e.g. Interval:200 categorizes values into groups of size 200).

3. **aggregation**: Average | Harmonic aggregation of the obtained values. An attribute "deviation" can be set to true or false (it’s false if the attribute is not given). This attribute can be used for grouped values to get the deviation of each value from the aggregated group value or to get a standard deviation based on biased sample variance for groups (in the short output version).

4. **filter**: Filtering with the attribute filterType (possible values are keep and remove).
   a) Filtering by required value slots with two options to specify them: MoreThan:Value or LessThan:Value (e.g. MoreThan:5 to collect values which are bigger than 5 only)
   b) Filtering by vehicle type. VehicleType:Type (e.g. VehicleType:Car to collect values only of vehicles of type "Car")
c) Filtering by time. Time: From- To (e.g. Time: 0-100 to collect values only of the first 100s of simulation time)

The following example will show an example of how you can specify the Statistics Visualizer according to your desired criteria. VehicleTravelTimes are the data we want to retrieve from vehicles and we want to group vehicles by the abstract group we can define in mapping configuration file (see e.g. Barnim scenario) and then calculate the average vehicle travel time value for each of these groups.

```xml
<visualizer id="statistics" enabled="true" loader="com.dcaiti.vsimrti.fed.visualizer.
  \StatisticsVisualizerConfig">
  <statistic filename="AverageVehicleTravelTimes" output="short">
    <source>VehicleTravelTimes</source>
    <group-by>VehicleGroup</group-by>
    <aggregation>Average</aggregation>
  </statistic>
</visualizer>
```

Listing 10.13: Getting the Average time by vehicle class.

You can also combine filters if you want to get a certain interval with upper and lower boundaries. With the following input instruction, only vehicles with the obtained data values between 250 and 500 will be left after filtering.

```xml
<filter filterType="keep">LessThan:500</filter>
<filter filterType="remove">LessThan:250</filter>
```

Listing 10.14: An example for filtering.

Please notice that some sources are being not only obtained in each simulation step but also collected for further processing as separate values for each of these steps (like VehicleSpeeds, VehicleHeadways). These data types need to be aggregated to one value per vehicle if you want to group them by value or filter them.

For demonstration, the StatisticsVisualizer is configured for the scenario Barnim and calculates the average travel times of the vehicles and additionally groups them. As a result, the simulation produces the following CSV file in the log directory:

```plaintext
Group;Value;Total;
AdHoc;399.14;24;
Cellular;463.87;12;
Unequipped;459.18;84;
```

Listing 10.15: The AverageVehicleTravelTime.csv file produced by the Statistics Visualizer in the Barnim scenario
11 Run simulation series

11.1 Simulation Runner

⚠️ Notice: The simulation runner is only available with a commercial license of VSimRTI. For further information on licences, please refer to our mailing list.

The simulation runner is a tool for automatic simulation parametrization and execution. Accordingly, a comfortable way exists to configure the execution of multiple simulations, e.g. of simulation series including several runs where only few parameters are changed in each run. With the simulation runner, a simulation series can be defined, for example, where the V2X penetration rate is changed in each simulation run. As a result, VSimRTI starts all simulation runs automatically according to the defined parameters.

11.1.1 Usage

The simulation runner is started as follows:

**GNU/Linux**

```bash
./simrun.sh -c /scenarios/<scenario name>/vsimrti/simrun_config.xml -p
```

Listing 11.1: VSimRTI simulation runner start command for GNU/Linux.

**Microsoft Windows**

```bash
simrun.bat -c \scenarios\<scenario name>\vsimrti\simrun_config.xml -p
```

Listing 11.2: VSimRTI simulation runner start command for Microsoft Windows.

**Configuration file**

The configuration file `vsimrti/scenarios/<scenarioName>/vsimrti/simrun_config.xml` contains the parameterization information.
11.1.2 Configuration

The example in listing 11.3 shows a complete configuration. Using this configuration the simulation-runner would try to run a scenario called Barnim while adapting the mapping (see 3.5.15), the configuration file of SNS, and VSimRTI (see 9.3) configuration files. The actual simulation is triggered by generating an adapted scenario folder and calling the same executable the user would normally trigger himself.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.dcaiti.tu-berlin.de/research/simulation/download/~/get/scenarios/scenarioName/vsimrti/simrun_config.xsd">
  <!-- basic configuration -->
  <vsimrti location="/path/to/vsimrti_folder" executable="vsimrti.sh" username="user_id" />
  <scenario name="Barnim" config="/scenarios/Barnim/vsimrti/vsimrti_config.xml" persistent="false" repetitions="1">
    <!-- argument >-o TRACE </ argument -->
    <!-- argument >-w 0 </ argument -->
  </scenario>
  <!-- define connected values for controlled changes -->
  <dimension name="PenetrationRates">
    <parameter name="V2XVehiclePercentage" file="/mapping3/mapping_config.json" fileFormat="json" item="vehicles[0].types[0].weight" type="ValueList">
      <value>0.0</value>
      <value>0.5</value>
      <value>0.75</value>
    </parameter>
    <parameter name="ClassicVehiclePercentage" file="/mapping3/mapping_config.json" fileFormat="json" item="vehicles[0].types[1].weight" type="ValueList">
      <value>1</value>
      <value>0.5</value>
      <value>0.25</value>
    </parameter>
    <parameter name="Simulationtime" file="/vsimrti/vsimrti_config.xml" fileFormat="xml" item="/simulation/endtime" type="ValueList">
      <value>100</value>
      <value>100</value>
      <value>100</value>
    </parameter>
  </dimension>
  <!-- define values for automatically permuted simulations -->
  <parameter name="SinglehopRadius" file="/sns/sns_config.json" fileFormat="json" item="singlehop.radius" type="ValueList">
    <value>500</value>
    <value>600</value>
  </parameter>
</configuration>
```

Listing 11.3: Simulation Runner configuration example

The configuration contains three distinct parts of configuration. The system specific definition, the scenario definition and the parametrization. While the first two parts will be explained as part of this section, the parametrization will be explained in it's own section.

**system specific definition**

```
<vsimrti location="/path/to/vsimrti_folder" executable="vsimrti.sh"
username="user_id" parallelSimulations="1" />
```
The system specific part of the configuration (see listing 11.4) is the only part of the configuration that can be overwritten using a second configuration file. It contains the following values:

- **location** of the VSimRTI installation to use (can be relative or absolute).
- **executable** used to start the simulation. This value is optional and will automatically be set to the default *.bat or *.sh file when omitted.
- **username** needed to check the license against. This is the user name given to you when registering your system with us.
- **parallelSimulations** defines how many simulations are started in parallel to speed up things. This value is optional and defaults to 1. Be aware that you should only use this if you have multiple cores available. This might also coincide with the `threads` option in the VSimRTI configuration.

**scenario definition**

```xml
<scenario name="Barnim" config="scenarios/Barnim/vsimrti/vsimrti_config.xml" persistent="false" repetitions="1">
  <!-- argument >-o TRACE</argument -->
  <!-- argument >-w 0</argument -->
</scenario>
```

The scenario definition (see listing 11.5) contains everything needed to identify the scenario to execute along with parameters that needs to passed to the VSimRTI executable. It contains the following values:

- **name** of the simulation to run. This is expected to be the same as the scenarios folder name and is used to automatically generate the path pointing to the scenarios vsimrti_config.xml in a default case. It can be omitted if the config option is set.
- **config** is an optional value containing the concrete path to the scenarios vsimrti_config.xml. This can be used if the scenario is not placed in the default scenarios folder (which is discouraged) and overwrites the path generated by the name attribute. Either name or config have to be defined!
- **persistent** defines if the generated scenario folders (containing the adapted configuration files for every single simulation to be executed) will be kept after simulation (value true) or deleted (value false).
- **repetitions** is an optional value containing the number of times each simulation should be repeated. Setting this to a value other than 1 only makes sense if some part of the simulation follows stochastic models (like non deterministic vehicle spawning).

The configuration can also contain a number of additional arguments that are passed to the executable without any changes, separated by spaces.
### 11.1.3 Parametrization

The heart of this tool is the parametrization of simulations. Using this, one can define values in the default configuration that are adapted between simulation runs. How many simulation runs are performed is defined by the number of changes configured enriched with the information about simulation repetitions.

For the example in listing 11.3 it is expected that the mapping file to be changed has one `vehicles` definition spawning multiple cars with a weighted type distribution defining first the equipped and then the unequipped vehicles.

**Parameters**

```xml
<parameter name="V2XVehiclePercentage" file="mapping3/mapping_config.json"
  fileFormat="json" item="vehicles[0].types[0].weight" type="ValueList">
  <value>0</value>
  <value>50</value>
  <value>75</value>
</parameter>
```

Listing 11.6: example of a parametrization for a single value

Each value that should be changed in a run is defined by a `parameter` element identified by a `name` (see listing 11.6). The base value is the `file` which should be changed (relative to the scenario folder). Currently it is needed to define what `fileFormat` is expected from that file, which has impact on the syntax of the `item` definition which denotes what part of this file should be changed (this will be explained in a bit). The final value is the `type` which denotes how the value change behaves. The child elements depend on this definition and will also be explained in a bit.

`fileFormat` can be one of `xml` or `json`. The `item` syntax is as followed:

- **xml**: contains an XPath\(^1\) expression
- **json**: contains an array-style definition of the target value. The value in listing 11.6 would change line 13 in listing 11.7. (In the first entry of `vehicles` the attribute `weight` of the `types` first entry).

```json
{
  "prototypes": ["name": "PKW"],
  "vehicles": {
    "startingTime": 5.0,
    "targetFlow": 1200,
    "maxNumberVehicles": 250,
    "route": "1",
    "types": [
      {
        "name": "PKW",
        "weight": 0.2
      },
      {
        "name": "PKW",
        "weight": 0.8
      }
    ]
  }
}
```

type can currently only have two entries:

- **ValueList**: This expects a list of values as child elements of the parameter. Each value will be used for at least one permutation.

- **IntegerGenerator**: This automatically generates integer values to write as values. The generated numbers can be configured by adding these attributes to the parameter element:
  - `offset` denoting a minimal number where generation should start (this will be the first value), default is 0.
  - `step` denoting the number that will be added to the previous value to generate the new one, default is 1.

In contrast to ValueList, this can create an infinite number of values.

### Permutation of parameters

When multiple such parameter elements are defined, a permutation for each specific value definition is generated. Let’s say defined are parameters A and B and each parameter has values a and b. The resulting permutations would be:

1. A=a, B=a
2. A=b, B=a
3. A=a, B=b
4. A=b, B=b

### Dimensions

Sometimes it is wanted to group some value changes. This can be necessary when changed values need to sum up to a specific value or when specific (named) output files need to be defined. This can be done by enclosing the affected parameters into a dimension definition. Doing this, the values of each parameter are connected by their index. For this to work, the number of values for each parameter have to be the same. The example in listing 11.3 utilizes this function to make sure the vehicle percentages sum up to 100. The generated permutations for the dimension enclosed parameters are:

1. V2XVehiclePercentage=0, ClassicVehiclePercentage=100
2. V2XVehiclePercentage=50, ClassicVehiclePercentage=50
3. V2XVehiclePercentage=75, ClassicVehiclePercentage=25

When additionally parameters are defined which are not enclosed in the dimension tag or another dimension tag is defined, then the permutations will be extended even further. The full permutation for listing 11.3 is as follows:
11 Run simulation series

11.1 Simulation Runner

PenetrationRates (V2XVehiclePercentage=0, ClassicVehiclePercentage=100), SinglehopRadius=500
PenetrationRates (V2XVehiclePercentage=50, ClassicVehiclePercentage=50), SinglehopRadius=500
PenetrationRates (V2XVehiclePercentage=75, ClassicVehiclePercentage=25), SinglehopRadius=500
PenetrationRates (V2XVehiclePercentage=0, ClassicVehiclePercentage=100), SinglehopRadius=600
PenetrationRates (V2XVehiclePercentage=50, ClassicVehiclePercentage=50), SinglehopRadius=600
PenetrationRates (V2XVehiclePercentage=75, ClassicVehiclePercentage=25), SinglehopRadius=600

11.1.4 Additional Information

These are some side effects to remember when working with this tool.

**Ports:**  The Simulation Runner supports automatic assigning of free ports for federates. This means that all federates configured in the defaults.xml will get a free port configured by default. This enables multiple simulations to be run simultaneously as long as the federates are started by VSimRTI. If some federates are not started through VSimRTI but are already running, this will not work.

**Paths:**  Relative paths of the files to be modified will be expanded with the deployment directory of the current simulation run to an absolute one.

**Adaptations:**  All values will be modified in copies of the original scenario. The copies will be placed in the simrunner folder in the VSimRTI base directory and will be (if not deactivated by configuration) deleted upon completion of the simulation.
12 Additional tools

12.1 scenario-convert

scenario-convert is a tool used to work with a scenarios database by importing and exporting data from external sources like OpenStreetMap SUMO etc.

This database is the basis for all map-related tasks, which can be performed by VSimRTI (e.g. navigation, route calculation...).

Based on a VSimRTI database, scenario-convert can export the data to SUMO input formats, which then can be used in the simulation. To enable dynamic rerouting of vehicles, scenario-convert generates, exports and imports route data from and to SUMO. This way, one can choose, whether to use generated routes (all possible routes between a point A and B), use existing routes and import them via scenario-convert or build own routes via the route generation tools supplied with the standard SUMO installation.

Workflow

An example workflow to create a valid map database would be as follows:

- get OpenStreetMap file of the desired region
- import OpenStreetMap file and export to SUMO using scenario-convert
- create routes using existing tools or manually in sumo
- import created routes back into database using scenario-convert
- start simulation using the database, which now contains all relevant map data and route information

The example scenario-convert call for this workflow would be as follows in listing 1:

```bash
scenarioconvert --osm2sumo -i <osmFile> -d <dbFile> -n --export-traffic-lights --generate-
\ routes --route-begin-latlon 50.429997,8.6567009 --route-end-latlon 50.429024,8.6642044
```

Listing 12.1: Exemplary call of scenario-convert.jar.

(import OpenStreetMap file to database, generate routes by given coordinates, generate network with sumo-netconvert and export to SUMO configuration files).

The example of scenario-convert call for exporting the network to SUMO files and creating a SUMO *.rou file if there are some routes in the database and filling it with vehicle types and vehicles from the mapping configuration file:
12.1 scenario-convert

Listing 12.2: Another exemplary call of scenario-convert.jar.

Usage

The following listing 2 provides an overview about the scenario-convert functions.

Usage: scenario-convert [OPERATION] -d <DATABASEFILE> [OPTIONS]

Examples (options in [] are optional):
1. Import an osm file and write data into database
   scenario-convert --osm2db -i <OSMFILE> [-o] [-d <DATABASE>]
2. Export db content to sumo-readable files
   scenario-convert --db2sumo -d <DATABASE> [-a <SUMOPREFIX>] [-n]
3. Reimport a sumo routefile into a database
   scenario-convert --sumo2db -d <DATABASE> -r <ROUTEFILE>
4. Combine steps 1 and 2
   scenario-convert --osm2sumo -d <DATABASE> -i <OSMFILE> [-s <SUMOPREFIX>]
5. Export db content to Shapefile format
   scenario-convert --db2shp -d <DATABASE>
6. Import an srtm file and write elevation data to nodes of an existing database
   scenario-convert --srtm2db -i <SRTMFILE> -d <DATABASE>

--------------------------------------------------------------------------------

CONFIGURATION FILE:
-c, --config-file FILE Optional, refers to a configuration file which
contains all parameters in JSON format. Single
options can be set by the following parameters.

OPERATIONS:
--osm2db -i [-o] [-d] Imports an OSM file creating a new database.
--db2sumo -d [-s] [-n] [-l] Exports the network to SUMO node and edge files.
--sumo2db [-i] [-r] [-d] Imports a SUMO Network/Routefile. Be aware that
you have to re-export an imported network!
--osm2sumo Combination of osm2db and db2sumo.
--db2shp -d Exports the network into Shapefile format.
--srtm2shp -d Combination of srtm2db and db2shp.
--srtm2db -i [-d] Imports an SRTM file and writes elevation data
to nodes.
--db2vsimrite -d [-i] [-s] Creates directory structure for a VSimRTI scenario.
--osm2vsimrite -i [-d] Combination of osm2db and db2vsimrite.
--update -d Updates the given database to the current scheme.
-g, --generate-routes Generate routes based on the given options.
You will at least need start and end coordinates

CONFIGURATION OPTIONS:
-l, --input-file FILE Defines an input file to use in an import.
File type is dependent on the import operation.
-d, --database FILE (OSM file/Database file/SUMO net file)
When importing: file to import to. Either omit
the file extension or set it to '.db'.
For export operations: file to load from.
For update operations: file to update.
-s, --sumo-prefix SUMOPREFIX Prefix for the generated sumo files
(uses database name when not defined).
--import-zone UTM zone of location for projecting coords in
default format (e.g. 32n).
--import-lon/--import-lat center longitude/latitude of imported region to project coordinates. ATTENTION: Zone and coordinate parameters are mutually exclusive alternatives!

-`r`, `--route-file` FILE Sumo *.rou.xml file location

-`m`, `--mapping-file` FILE Export existing prototypes and vehicles data to *.rou.xml from given mapping file (only in combination with export-routes).

-`f`, `--force` Force overwrite of existing files (do not increment file names).

**INTERNAL ROUTE GENERATION**

(If you use this, a route file containing these routes will be generated too)

-`g`, `--generate-routes` Generate routes based on the given options.

Please see below for possible options,

You will at least need start and end coordinates

**Start of route:**

-`route-begin-lat` Latitude of point, needs longitude defined!

-`route-begin-lon` Longitude of point, needs latitude defined!

-`route-begin-latlon` Combined value in format [latitude],[longitude] (this is an alternative to separate definition!)

-`route-begin-node-id` OSM node id as start point (this is an alternative to the options above!)

**End of route:**

-`route-end-lat` Latitude of point, needs longitude defined!

-`route-end-lon` Longitude of point, needs latitude defined!

-`route-end-latlon` Combined value in format [latitude],[longitude] (this is an alternative to separate definition!)

-`route-end-node-id` OSM node id as start point (this is an alternative to the options above!)

-`--number-of-routes` INT Optional, limits the maximum number of generated routes. Default is 1.

**ADDITIONAL TOOLS**

-`o`, `--execute-osmosis` Execute osmosis to apply automatic filters to the given osm file (only for osm2xxx).

-`b`, `--import-buildings` Write osm building information in database

-`--ignore-turn-restrictions` Ignore all defined turn restrictions on OSM import.

-`--disable-graph-cleanup` Turns off the removal of unconnected parts from the main traffic network graph. Since several components of VSimRTI require one main graph without disconnected ways and nodes, this option should be used only if the cleanup procedure is faulty.

-`n`, `--execute-netconvert` Automatically start sumo netconvert after export to create netfile (only with xxx2sumo)

-`l`, `--export-traffic-lights` export traffic light information for nodes

-`--osm-speeds-file` Define a property file which contains speed information which are used to set the speed for OSM ways without a max speed tag. (only with osm2xxx)

-`--osm-speeds-overwrite` If set to true, the maxspeed tags of ways are ignored and replaced by either default values, or by speed information defined via the --osm-speeds-file option. (only with osm2xxx)

-`v`, `--version` Print current version number

---

**File placement:**

When a database file is given with `-d` every created file will be placed by the database file in the given path. When the file is passed without a path the path used to execute the command from will be used as a base. For import operations where `-d` is omitted all files will be placed alongside the input file given.

When a file to be created already exist the filename will be postfixed with an incrementing number. This behaviour can be overwritten
The following listing 3 shows a JSON example file which can be used with the -c option of scenario-convert.

```json
{
    "operating_mode": "osm2db",
    "input_path": "path/to/input.osm",
    "database_path": "path/to/database.db",
    "sumo_prefix": "SUMO_PREFIX",
    "execute_netconvert": true,
    "export_traffic_lights": true,
    "export_routes": true,
    "import_buildings": false,
    "force_overwrite": false,
    "disable_graph_cleanup": false,
    "ignore_turn_restrictions": false,
    "generate_routes": true,
    "number_of_routes": 2,
    "route_begin_lat": 52.5,
    "route_begin_lon": 13.4,
    "route_end_lat": 52.2,
    "route_end_lon": 13.1,
    "osm_overwrite_speeds": false,
    "osm_speeds_file": "path/to/speeds.properties"
}
```

Furthermore, in listing 4 you can find a properties file which can be used during the import of OSM data in order to define speeds for ways, which do not have a maxspeeds-tag defined. For this purpose use the option --osm-speeds-file <FILE>. In the speed properties file, for each way type a speed value can be defined, according to the OSM highway key¹.

```ini
# the unit the speed values are defined in [kmh, ms]
speed.unit = kmh

# the default speed for all way types which are not defined here
speed.default = 30

# autobahn
highway.motorway = 130
highway.motorway_link = 70

# bundesstrasse (germany)
highway.trunk = 70
highway.trunk_link = 65

# linking bigger town
highway.primary = 65
highway.primary_link = 60

# linking towns + villages
highway.secondary = 60
```

¹http://wiki.openstreetmap.org/wiki/Key:highway
We can leave only the needed options in the scenario-convert configuration file:

```json
{
  "operating_mode": "db2vsimrti",
  "database_path": "path\to\Barcelona.db"
}
```

Under Windows the paths must be configured with two backslashes.
13 VSimRTI configuration

13.1 Overview

VSimRTI can be configured by adapting the files in the /vsimrti/etc directory. The following technical aspects of VSimRTI can be configured:

- vsimrti/etc/defaults.xml - Configuration of all federates which are coupled with VSimRTI.
- vsimrti/etc/hosts.json - Configuration of the machines the simulations are executed on.
- vsimrti/etc/logback.xml - Logging configuration of VSimRTI and its federates.

A full example of the default configuration files can be found in A.2.

13.2 Federates configuration

This part of software can be configured with a configuration file. The specific path is vsimrti/etc/defaults.xml. This file is used for configuring all federates and simulators coupled with VSimRTI. For each federate you can define various aspects:

- What’s the id and class of the federate?
- On which host is the federate running?
- How is the federate started?
- What is the name of the configuration file for the federate?
- For which messages is the federate subscribing?
- The priority of the federate when it comes to message distribution.

A detailed description of the various parameters can be found in the documentation available on the DCAITI website.
13.2.1 Federate priorities

There might be cases where it is important that a certain federate receives messages with equal timestamps before or after the other ones. In the context of VSimRTI for example, it is often desirable that the Application Simulator receives the vehicle movements-message after the traffic simulator to avoid problematic behavior in the first simulator. The configuration file defaults.xml allows for a fine grained adjustment of the federate priorities.

For example, a federate with a certain priority looks like this: 
\[
<\text{federate class}="\text{com.dcaiti.vsimrti.fed.sumo.ambassador.SumoAmbassador}\text{ priority}="20">\text{. That means that the Sumo Ambassador has an priority of 20, where the following rules apply to the priorities:}
\]
- Priorities range from 0 to 100.
- 0 is the highest priority, e.g. the federate with this priority assigned will receive the message first.
- The default priority if none is given is 50.

Note that VSimRTI ships with sane default values for the priority and a modification is only necessary in rare cases.

13.3 Host configuration

This part of software can be configured with a configuration file. The specific path is vsimrti/etc/hosts.json

\[^{\text{i} \text{Notice}}\text{: The documentation for the VSimRTI specific component is freely available on the \text{DCAITI website , explaining all available options.}}\]

This file is used for configuring simulation runs using multiple machines. Under normal circumstances it is not necessary to change this file.

In VSimRTI there are two kinds of hosts: local and remote hosts. All hosts have to be identified with a unique string that is valid for the remainder of the configuration to reference a defined host.

For a local host additionally the operating system and a name of a directory that is used to deploy needed federates is necessary. Values for operating systems are "linux" or "windows".

The syntax for referenced paths has to be chosen according to operating system standards (e.g. /vsimrti/temp for GNU/Linux and C:\vsimrti\temp for Microsoft Windows). Note that also relative paths are possible.

For remote hosts SSH and SFTP is used to deploy and start federates. Therefore the address of the host as well as the port, user name, and password for an SSH connection must additionally be given.
14 Additional information

Here you find helpful information regarding VSimRTI and its configuration.

14.1 PATH configuration

PATH is an environment variable on operating systems, specifying a set of directories where executable programs are located.

**GNU/Linux**

On GNU/Linux operating system you can edit your local `.bash_profile` in your home directory with your favorite **CLI** editor e.g. Vi IMproved (VIM) or Nano's ANOther editor (NANO).

```bash
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
  . ~/.bashrc
fi

# User specific environment and startup programs
PATH=$PATH:$HOME/bin:$HOME/sumo/
export PATH
```

Listing 14.1: Add the SUMO location to PATH on GNU/Linux.

On most GNU/Linux distributions you must log out and login to refresh the PATH variable. You can check your path with the shell command `echo $PATH`.
Microsoft Windows

For Microsoft Windows please follow the instructions below.

1. Select “Computer” from the start menu. Choose “Properties” from the context menu.
2. Click “Advanced system settings” and switch to the “Advanced” tab in “System Properties”.
3. Click on “Environment Variables”.
4. Find “Path”, and click on it (choose it).
5. Click “Edit”.
6. Modify Path by adding the absolute location to the value for Path.
   (Don't forget a semicolon to separate).

Figure 14.1: Edit PATH variable in Microsoft Windows
## 15 List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCT</td>
<td>Automotive Services and Communication Technologies</td>
</tr>
<tr>
<td>CAM</td>
<td>Cooperative Awareness Messages</td>
</tr>
<tr>
<td>CLI</td>
<td>Command Line Interface</td>
</tr>
<tr>
<td>CPU</td>
<td>central processing unit</td>
</tr>
<tr>
<td>DCAITI</td>
<td>Daimler Center for Automotive Information Technology Innovations</td>
</tr>
<tr>
<td>DENM</td>
<td>Decentralized Environmental Notification Messages</td>
</tr>
<tr>
<td>ETC</td>
<td>Estimated Time to Completion</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standards Institute</td>
</tr>
<tr>
<td>EPL</td>
<td>Eclipse Public License</td>
</tr>
<tr>
<td>FOKUS</td>
<td>Fraunhofer-Institut für Offene Kommunikationssysteme</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>GNU</td>
<td>GNU’s Not Unix!</td>
</tr>
<tr>
<td>GPL</td>
<td>GNU General Public License</td>
</tr>
<tr>
<td>HLA</td>
<td>High Level Architecture</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>INET</td>
<td>OMNeT++ - Model library for wired, wireless and mobile networks</td>
</tr>
<tr>
<td>ITEF</td>
<td>Integrated Test and Evaluation Framework</td>
</tr>
<tr>
<td>ITS-G5</td>
<td>Extension of IEEE-802.11 standard for ad-hoc communication between vehicles</td>
</tr>
<tr>
<td>JAR</td>
<td>Java Archive</td>
</tr>
<tr>
<td>JRE</td>
<td>Java Runtime Environment</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>LTE</td>
<td>Long Term Evolution</td>
</tr>
<tr>
<td>LuST</td>
<td>Luxembourg SUMO Traffic</td>
</tr>
<tr>
<td>M&amp;S</td>
<td>modelling and simulation</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MBMS</td>
<td>Multimedia Broadcast Multicast Service</td>
</tr>
<tr>
<td>NANO</td>
<td>Nano’s ANOther editor</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ns-2</td>
<td>network simulator 2</td>
</tr>
<tr>
<td>ns-3</td>
<td>network simulator 3</td>
</tr>
<tr>
<td>OMNeT++</td>
<td>Objective Modular Network Testbed in C++</td>
</tr>
<tr>
<td>OSM</td>
<td>Open Street Map</td>
</tr>
<tr>
<td>RAM</td>
<td>random-access memory</td>
</tr>
<tr>
<td>RSU</td>
<td>Road Side Units</td>
</tr>
<tr>
<td>RTF</td>
<td>Real Time Factor</td>
</tr>
<tr>
<td>SNS</td>
<td>Simple Network Simulator</td>
</tr>
<tr>
<td>SUMO</td>
<td>Simulation of Urban Mobility</td>
</tr>
<tr>
<td>TMC</td>
<td>Traffic Management Center</td>
</tr>
<tr>
<td>TraCI</td>
<td>Traffic Control Interface</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunications System</td>
</tr>
<tr>
<td>UUID</td>
<td>universally unique identifier</td>
</tr>
<tr>
<td>V2V</td>
<td>Vehicle-to-Vehicle</td>
</tr>
<tr>
<td>V2X</td>
<td>Vehicle-2-X</td>
</tr>
<tr>
<td>VIM</td>
<td>Vi IMproved</td>
</tr>
<tr>
<td>VSimRTI</td>
<td>Vehicle-2-X Simulation Runtime Infrastructure</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Appendix A

VSimRTI deployment

You will find a collection of all relevant data.

A.1 VSimRTI Folder Structure

The VSimRTI all-in-one package has a clear separation of binary files, general VSimRTI configuration files and simulation scenario specific configuration files in different folders. The vsimrti/bin folder contains all binary files needed for execution of VSimRTI. Normally, you do not have to make changes here.

General VSimRTI configuration files, which are used through all scenarios can be found in the vsimrti/etc folder.

These files contain information about how simulators interact, which remote hosts exists etc.. If you only want to perform simulations using the preconfigured simulators in the all-in-one package you do not need to make changes here. In the vsimrti/scenarios folder you can find the scenario specific configuration files, separated by component, e.g. traffic simulator or communication simulator.

This configuration can be changed in the file vsimrti/etc/defaults.xml (see A.1).

For normal usage of VSimRTI, this configuration does not need to be changed. Please edit this file only, if you know what you are doing, as unwanted side effects might occur.

The configuration of VSimRTI ambassadors is done by using configuration files in the folder vsimrti/scenarios/<scenarioName>/<federateid>.

To configure scenario specific VSimRTI options, e.g. to define which simulators should be active in a simulation scenario, you can adjust the vsimrti/scenarios/<scenarioName>/vsimrti/vsimrti_config.xml file located in vsimrti/scenarios/<scenarioName>/vsimrti. The overall folder structure is as follows:
Appendix A VSimRTI deployment

A.1 VSimRTI Folder Structure

vsimrti
  __bin ........................................... path containing binary files
  __etc .............................................. path containing general configuration files
  |__defaults.xml ................................................. (see A.1)
  |__hosts.json .................................................. (see A.2)
  |__logback.xml .................................................. (see A.3)
  __lib .............................................. path containing system specific (non JAR) libraries
  __logs ...................................................... path for all log files
  __scenarios .............................................. path for scenario specific configuration files
  |__Barnim ...................................................... path containing Barnim example scenario
  |__Tiergarten ................................................... path containing Tiergarten example scenario
  __tmp ...................................................... path for temporary files during a simulation
  __simrun.sh .............................................. shell script to start a simulation series
  __simrun.bat ................................................ batch script to start a simulation series
  __systeminfo.txt .............................................. (see A.4)
  __vsimrti.sh .............................................. shell script to start a single simulation
  __vsimrti.bat ................................................ batch script to start a single simulation
  __vsimrti.dat .............................................. VSimRTI license file
  __vsimrti-license.lcs ........................................ VSimRTI license file

Figure A.1: VSimRTI folder structure
A.2 File listings

/etc/defaults.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <federates>
    <federate class="com.dcaiti.vsimrti.fed.omnetpp.ambassador.OmnetppAmbassador">
      <id>omnetpp</id>
      <deploy>true</deploy>
      <start>true</start>
      <host>local</host>
      <port>4998</port>
      <dockerImage/>
      <config>omnetpp_config.json</config>
      <messages>
        <message>AddedVehicle</message>
        <message>AddedRsu</message>
        <message>AddedTrafficLight</message>
        <message>VehicleMovements</message>
        <message>SendV2XMessage</message>
        <message>ConfigureAdHocCommunication</message>
      </messages>
    </federate>
    <federate class="com.dcaiti.vsimrti.fed.n3.ambassador.Ns3Ambassador">
      <id>ns3</id>
      <deploy>true</deploy>
      <start>true</start>
      <host>local</host>
      <port>5011</port>
      <dockerImage/>
      <config>ns3_config.json</config>
      <messages>
        <message>AddedVehicle</message>
        <message>AddedRsu</message>
        <message>AddedTrafficLight</message>
        <message>VehicleMovements</message>
        <message>SendV2XMessage</message>
        <message>ConfigureAdHocCommunication</message>
      </messages>
    </federate>
    <federate class="com.dcaiti.vsimrti.fed.sumo.ambassador.SumoAmbassador">
      <id>sumo</id>
      <deploy>true</deploy>
      <start>true</start>
      <host>local</host>
      <port>0</port>
      <dockerImage/>
      <config>sumo_config.json</config>
      <messages>
        <message>AddedVehicle</message>
        <message>AddedRsu</message>
        <message>AddedTrafficLight</message>
        <message>VehicleMovements</message>
        <message>SendV2XMessage</message>
        <message>ConfigureAdHocCommunication</message>
      </messages>
    </federate>
  </federates>
</configuration>
```

<messages>
  <message>VehicleTypesInitMessage</message>
  <message>VehiclePathsInitMessage</message>
  <message>AddedVehicle</message>
  <message>SetMaxSpeed</message>
  <message>SlowDown</message>
  <message>PropagateNewRoute</message>
  <message>ChangeStaticRoute</message>
  <message>ChangeStaticRouteByEdge</message>
  <message>ChangeLane</message>
  <message>ChangeLaneProperty</message>
  <message>ChangeTrafficLightsState</message>
  <message>Stop</message>
  <message>Resume</message>
  <message>SumoTraciByteArrayMessage</message>
  <message>EnableDistanceSensors</message>
  <message>ChangeVehicleParameters</message>
  <message>ChangeSpeed</message>
  <message>VehicleControl</message>
  <message>VehicleMovements</message>
  <message>SubscribeForInductionLoop</message>
  <message>SubscribeForLaneAreaDetector</message>
  <message>AddedRsu</message>
  <message>AddedTmc</message>
  <message>AddedChargingStation</message>
  <message>AddedTrafficLight</message>
  <message>AddedVehicle</message>
  <message>ApplicationSpecificMessage</message>
  <message>ChangeVehicleState</message>
  <message>ChargingStationUpdate</message>
  <message>ChargingRejected</message>
  <message>VehicleElectricInformationMessage</message>
  <message>PropagateNewRoute</message>
  <message>ReceiveV2XMessage</message>
  <message>ReceiveFullV2XMessage</message>
  <message>AcknowledgeV2XMessage</message>
  <message>SensorData</message>
  <message>SumoTraciByteArrayMessageResponseContainer</message>
  <message>TrafficDetectorUpdates</message>
  <message>UpdateTrafficLight</message>
  <message>VehicleMovements</message>
  <message>VehiclePathsInitMessage</message>
  <message>VehicleTypesInitMessage</message>
  <message>TrafficSignsUpdate</message>
</messages>
</federate>
<federate class="com.dcaiti.vsimrti.fed.trafficsigns.ambassador.TrafficSignAmbassador">
  <id>trafficsigns</id>
  <deploy>false</deploy>
  <start>false</start>
  <host>local</host>
  <config>trafficsigns_config.json</config>
  <messages>
    <message>VehicleMovements</message>
    <message>SightDistanceMessage</message>
  </messages>
</federate>
<message>ChangeVariableLaneAssignmentSignMessage</message>
<message>ChangeVariableSpeedSignMessage</message>
</messages>
</federate>

<federate class="com.dcaiti.vsimrti.fed.eventserver.amassador.EventserverAmbassador">
  <id>eventserver</id>
  <deploy>false</deploy>
  <start>false</start>
  <host>local</host>
  <config>eventserver_config.json</config>
  <messages>
    <message>SensorRegistration</message>
    <message>VehicleMovements</message>
  </messages>
</federate>

<federate class="com.dcaiti.vsimrti.fed.mapping3.MappingAmbassador">
  <id>mapping3</id>
  <deploy>false</deploy>
  <start>false</start>
  <host>local</host>
  <config>mapping_config.json</config>
  <messages>
    <message>VehiclePathsInitMessage</message>
    <message>ScenarioTrafficLights</message>
    <message>ScenarioAddedVehicle</message>
  </messages>
</federate>

<federate class="com.dcaiti.vsimrti.fed.cell2.amassador.Cell2Ambassador">
  <id>cell2</id>
  <deploy>false</deploy>
  <start>false</start>
  <host>local</host>
  <config>cell2_config.json</config>
  <messages>
    <message>AddedVehicle</message>
    <message>AddedRsu</message>
    <message>AddedTmc</message>
    <message>AddedTrafficLight</message>
    <message>AddedChargingStation</message>
    <message>VehicleMovements</message>
    <message>SendV2XMessage</message>
    <message>ConfigureCellCommunication</message>
  </messages>
</federate>

<federate class="com.dcaiti.vsimrti.fed.battery.amassador.BatteryAmbassador">
  <id>battery</id>
  <deploy>false</deploy>
  <start>false</start>
  <host>local</host>
  <config>battery_config.json</config>
  <messages>
    <message>AddedVehicle</message>
    <message>VehicleMovements</message>
    <message>ChargingStarted</message>
    <message>ChargingStopped</message>
  </messages>
</federate>

<federate class="com.dcaiti.vsimrti.fed.chargingstation.amassador.ChargingStationAmbassador">
  <id>chargingstation</id>
  <deploy>false</deploy>
  <start>false</start>
  <host>local</host>
  <config>chargingstation_config.json</config>
  <messages>
    <message>AddedChargingStation</message>
    <message>ChargeRequest</message>
  </messages>
</federate>
<message>StopChargeRequest</message>
<message>ChargingStationUpdate</message>
</messages>
</federate>

<federate class="com.dcaiti.vsimrti.fed.visual.VisualizationAmbassador">
<deploy>false</deploy>
=start=false</start>
$id=visualizer</id>
<htmlocal</host>
<port>5099</port>
<update>true</update>
$config=visualizer_config.xml</config>
<messages>
<message>AddedChargingStation</message>
<message>AddedRsu</message>
<message>AddedTrafficLight</message>
<message>AddedTrafficSign</message>
<message>AddedVehicle</message>
<message>ApplicationInitMessage</message>
<message>ChangeLane</message>
<message>ChangeLaneProperty</message>
<message>ChangeRoute</message>
<message>ChangeStaticRoute</message>
<message>ChangeTrafficLightsState</message>
<message>ChangeVariableSpeedSignMessage</message>
<message>ChangeVariableLaneAssignmentSignMessage</message>
<message>ChargingStationUpdate</message>
<message>ConfigureAdHocCommunication</message>
<message>ConfigureCellCommunication</message>
<message>DeleteV2XMessage</message>
<message>ReceiveV2XMessage</message>
<message>SendV2XMessage</message>
<message>SensorData</message>
<message>SetMaxSpeed</message>
<message>SlowDown</message>
<message>TrafficDetectorUpdates</message>
<message>UpdateTrafficLight</message>
<message>VehicleElectricInformationMessage</message>
<message>VehicleHandover</message>
<message>VehicleMovements</message>
<message>VehiclePathsInitMessage</message>
<message>VehicleTypesInitMessage</message>
<message>SpeedAdviceMsg</message>
<message>LaneChangeAdviceMsg</message>
<message>GapAdviceMsg</message>
<message>AccelerationAdviceMsg</message>
<message>AccelerateFromLaneMsg</message>
<message>VmsSpeedLimitMsg</message>
</messages>
</federate>

<federate class="com.dcaiti.vsimrti.fed.phabmacs.ambassador.PhabmacsAmbassador">
$id=phabmacs</id>
<deploy>true</deploy>
=start>true</start>
<htmlocal</host>
<port>50423</port>
$config=phabmacs_config.json</config>
<javamemorySizeXms>100</javamemorySizeXms>
<javamemorySizeXmx>4086</javamemorySizeXmx>
</file>

<!-- Add custom classpath entries to avoid copying jar files manually -->
<!-- Base dir is "tools/FesmrtiEmbeddedStarter/tmp/phabmacs" -->
</classpath>
</classpathEntries>
</classpath>
Listing A.1: VSimRTI file listing: etc/defaults.xml
**etc/hosts.json**

```json
{
  "localHosts": [
    {
      "id": "local",
      "workingDirectory": "/tmp",
      "address": "localhost",
      "operatingSystem": "linux"
    }
  ],
  "remoteHosts": [
    {
      "id": "remote",
      "workingDirectory": "/home/vsimrti/test/tmp",
      "address": "192.168.0.1",
      "operatingSystem": "linux",
      "user": "username",
      "password": "password",
      "port": 22
    }
  ]
}
```

**etc/logback.xml**

```xml
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:noNamespaceSchemaLocation="http://www.dcaiti.tu-berlin.de/research/veesimulation/download/get/etc/logback.xsd">
  <!-- timestamp key="timestamp" datePattern="yyyy-MM-dd HH:mm:ss" / -->
  <!-- ######################## APPENDER ############################### -->
  <!-- Console logger -->
  <if condition='isDefined("logDirectory")'>
    <property name="logDirectory" value="${logDirectory}"/>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
      <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <pattern>%date %-5level %logger{0} - %msg%n</pattern>
      </encoder>
    </appender>
    <appender name="STDOUT-TimeAdvance" class="ch.qos.logback.core.ConsoleAppender">
      <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <pattern>%date{HH:mm:ss} - %msg%n</pattern>
      </encoder>
    </appender>
    <appender name="STDOUT-DbLoading" class="ch.qos.logback.core.ConsoleAppender">
      <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <pattern>%date{HH:mm:ss.SSS} - %msg%n</pattern>
      </encoder>
    </appender>
    <appender name="STDOUT-MessageOnly" class="ch.qos.logback.core.ConsoleAppender">
      <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <pattern>%msg%n</pattern>
      </encoder>
    </appender>
  </if>
</configuration>
```

Listing A.2: VSimRTI: file listing: etc/hosts.json
Appendix A VSimRTI deployment

A.2 File listings

```xml
<encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
  <pattern>%date %-5level %logger{0} - %msg%n</pattern>
</encoder>

<!-- File Appender for different components -->
<appender name="VSimrtiLog" class="ch.qos.logback.core.FileAppender">
  <file>${logDirectory}/VSimRTI.log</file>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>%date %-5level %logger{0} - %msg%n</pattern>
  </encoder>
</appender>

<appender name="TrafficLog" class="ch.qos.logback.core.FileAppender">
  <file>${logDirectory}/Traffic.log</file>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>%date %-5level %logger{0} - %msg%n</pattern>
  </encoder>
  <append>false</append>
</appender>

<appender name="CommunicationLog" class="ch.qos.logback.core.FileAppender">
  <file>${logDirectory}/Communication.log</file>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>%date %-5level %logger{0} - %msg%n</pattern>
  </encoder>
</appender>

<appender name="CommunicationDetailsLog" class="ch.qos.logback.core.FileAppender">
  <file>${logDirectory}/CommunicationDetails.log</file>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>%msg%n</pattern>
  </encoder>
  <append>false</append>
</appender>

<appender name="ApplicationNTLog" class="ch.qos.logback.core.FileAppender">
  <file>${logDirectory}/ApplicationNT.log</file>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>%date %-5level %logger{0} - %msg%n</pattern>
  </encoder>
</appender>

<appender name="ApplicationNTLogDelegation" class="ch.qos.logback.classic.sift.SiftingAppender">
  <discriminator>
    <key>path</key>
    <defaultValue>unknown</defaultValue>
  </discriminator>
  <sift>
    <appender name="FILE-${unitId}" class="ch.qos.logback.core.FileAppender">
      <file>${logDirectory}/appsNT/${path}.log</file>
      <layout class="ch.qos.logback.classic.PatternLayout">
        <pattern>%date %level - %msg%n</pattern>
      </layout>
    </appender>
  </sift>
</appender>

<appender name="GeneralPurposeAmbassador" class="ch.qos.logback.classic.sift.SiftingAppender">
  <discriminator>
    <key>loggerId</key>
    <defaultValue>unknown</defaultValue>
  </discriminator>
  <sift>
    <appender name="FILE-${loggerId}" class="ch.qos.logback.core.FileAppender">
      <file>${logDirectory}/loggerId-${loggerId}.log</file>
      <layout class="ch.qos.logback.classic.PatternLayout">
        <pattern>%date %level %logger{0} - %msg%n</pattern>
      </layout>
    </appender>
  </sift>
</appender>
```

VSimRTI: Vehicle-2-X Simulation Runtime Infrastructure
Appendix A VSimRTI deployment

VSimRTI deployment

A.2 File listings

```xml
<appender name="StatisticsLog" class="ch.qos.logback.core.FileAppender">
  <file>${logDirectory}/statistics.csv</file>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>%date %level %logger{0} [%thread] - %msg%n</pattern>
  </encoder>
</appender>

<appender name="MappingLog" class="ch.qos.logback.core.FileAppender">
  <file>${logDirectory}/Mapping.log</file>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>%date %level %logger{0} - %msg%n</pattern>
  </encoder>
</appender>

<appender name="NavigationLog" class="ch.qos.logback.core.FileAppender">
  <file>${logDirectory}/Navigation.log</file>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>%date %level %logger{0} - %msg%n</pattern>
  </encoder>
</appender>

<appender name="EnvironmentLog" class="ch.qos.logback.core.FileAppender">
  <file>${logDirectory}/Environment.log</file>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>%date %level %logger{0} - %msg%n</pattern>
  </encoder>
</appender>

<appender name="Cell2Log" class="ch.qos.logback.core.FileAppender">
  <file>${logDirectory}/Cell2.log</file>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>%date %level %logger{0} - %msg%n</pattern>
  </encoder>
</appender>

<appender name="BatteryLog" class="ch.qos.logback.core.FileAppender">
  <file>${logDirectory}/Battery.log</file>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>%date %level %logger{0} - %msg%n</pattern>
  </encoder>
</appender>

<appender name="ChargingStationLog" class="ch.qos.logback.core.FileAppender">
  <file>${logDirectory}/ChargingStation.log</file>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>%date %level %logger{0} - %msg%n</pattern>
  </encoder>
</appender>

<!-- ######################## LOGGER ############################### -->

<logger name="VSimRTIStarter" additivity="false" level="INFO">
  <appender-ref ref="VsimrtiLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.docker" additivity="false" level="INFO">
  <appender-ref ref="VsimrtiLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.fed.visual" additivity="false" level="INFO">
  <appender-ref ref="VsimrtiLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.fed.sumo" additivity="false" level="INFO">
  <appender-ref ref="TrafficLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.fed.omnetpp" additivity="false" level="INFO">
  <appender-ref ref="CommunicationLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.fed.ns3" additivity="false" level="INFO">
</logger>
```

VSimRTI Vehicle-2-X Simulation Runtime Infrastructure

115
Appendix A VSimRTI deployment

A.2 File listings

```xml
<appender-ref ref="CommunicationLog"/>

<logger name="com.dcaiti.vsimrti.fed.sns" additivity="false" level="INFO">
  <appender-ref ref="CommunicationLog"/>
</logger>

<!-- NOTE: do not specify logging levels other than "INFO" use omnetpp.ini (OMNeT++
  \인이\) for logging configuration -->

<logger name="com.dcaiti.vsimrti.fed.ns3.ambassador.Ns3AmbassadorOutput" />
  <appender-ref ref="CommunicationDetailsLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.fed.omnetpp.ambassador.OmnetppAmbassadorOutput" />
  <appender-ref ref="CommunicationDetailsLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.fed.applicationNT.ambassador.Ns3AmbassadorOutput" />
  <appender-ref ref="NavigationLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.fed.applicationNT" additivity="false" level="INFO">
  <appender-ref ref="ApplicationNTLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.lib.routing" additivity="false" level="INFO">
  <appender-ref ref="NavigationLog"/>
</logger>

<logger name="com.graphhopper" additivity="false" level="INFO">
  <appender-ref ref="NavigationLog"/>
</logger>

<logger name="net.sf.jsi" additivity="false" level="OFF">
  <appender-ref ref="NavigationLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.fed.cell2" additivity="false" level="INFO">
  <appender-ref ref="Cell2Log"/>
</logger>

<logger name="com.dcaiti.vsimrti.fed.battery" additivity="false" level="INFO">
  <appender-ref ref="BatteryLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.fed.phabmacs" additivity="false" level="INFO">
  <appender-ref ref="TrafficLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.fed.trafficsigns" additivity="false" level="INFO">
  <appender-ref ref="TrafficLog"/>
</logger>

<logger name="com.dcaiti.vsimrti.fed.chargingstation" additivity="false" level="INFO">
  <appender-ref ref="ChargingStationLog"/>
</logger>

<logger name="statistics" additivity="false" level="OFF">
  <appender-ref ref="StatisticsLog"/>
</logger>

<logger name="omnetpp.rti.services.time" additivity="false" level="INFO">
  <appender-ref ref="STDOUT-TimeAdvance"/>
</logger>

<logger name="com.dcaiti.vsimrti.start" additivity="false" level="INFO">
  <appender-ref ref="STDOUT-MessageOnly"/>
</logger>
```

VSimRTI: Vehicle-2-X Simulation Runtime Infrastructure 116
Listing A.3: VSimRTI: file listing: etc/logback.xml

Listing A.4: VSimRTI: file listing: systeminfo.txt
Appendix B

Example scenario Barnim

You will find a collection of all relevant data.

B.1 Folder Structure

Barnim

- applicationNT
  - applications
  - Barnim.db not listed (binary content)
- battery
  - battery_config.json (see B.1)
- eventserver
  - eventserver_config.json not listed (in this release)
- cell2
  - cell2_config.json (see B.2)
  - network.json (see B.3)
- regions.json (see B.4)
- mapping3
  - mapping_config.json (see B.5)
- sns
  - sns_config.json (see B.6)
- sources
  - Barnim_fixed.osm partially listed (see B.7)
- sumo
  - Barnim.edg.xml partially listed (see B.8)
  - Barnim.net.xml partially listed (see B.9)
  - Barnim.nod.xml partially listed (see B.10)
  - Barnim.rou.xml (see B.11)
  - Barnim.sumo.cfg (see B.12)
  - sumo_config.json (see B.13)
- visualizer
  - visualizer_config.xml (see B.14)
- vsimrti
  - vsimrti_config.xml (see B.15)
  - simrun_config.xml (see B.16)

Figure B.1: Scenario Barnim: folder structure
B.2 File listings

**battery/battery_config.json**

```json
{
  "vehicleModelClass": "com.dcaiti.vsimrti.fed.battery.sim.models.vehicle.ElectricSmart",
  "vehicleModelConfig": {
    "Mass": 1060,
    "ReferenceArea": 1.95,
    "DragCoefficient": 0.375,
    "TankToWheelEfficiency": 0.7,
    "ElectricMotorOperatingVoltage": 350,
    "ConsumerOperatingVoltage": 12
  },
  "batteryModelClass": "com.dcaiti.vsimrti.fed.battery.sim.models.battery.VerySimpleBatteryModel",
  "batteryModelConfig": {
    "NumberOfCells": 93,
    "CellVoltage": 4.3,
    "CellCapacityInAh": 50,
    "ChargeEfficiency": 0.8448,
    "RechargingType": 2,
    "MinSOC": 0.30,
    "MaxSOC": 0.70
  },
  "environmentModelClass": "com.dcaiti.vsimrti.fed.battery.sim.models.environment.DefaultEnvironment",
  "environmentModelConfig": {
    "FluidDensity": 1.293,
    "RollingResistanceCoefficient": 0.01
  },
  "consumers": [
    { "Radio": 10 },
    { "HeadLight": 100 }
  ]
}
```

Listing B.1: Scenario Barnim: file listing: battery/battery_config.json

**cell2/cell2_config.json**

```json
{
  "networkConfigurationFile": "network.json",
  "regionConfigurationFile": "regions.json"
}
```

Listing B.2: Scenario Barnim: file listing: cell2/cell2_config.json

**cell2/network.json**

```json
{
  "globalRegion": {
    "uplink": {
      "delay": {
        "type": "ConstantDelay",
        "delay": 100,
        "prpl": 0,
        "retries": 2
      },
      "capacity": 28000000
    },
    "downlink": {
      "unicast": {
```

VSimRTI: Vehicle-2-X Simulation Runtime Infrastructure
Appendix B Example scenario Barnim

```
14    "delay": {
15        "type": "ConstantDelay",
16        "delay": 100,
17        "prpl": 0,
18        "retries": 2
19    }
20 },
21    "multicast": {
22        "delay": {
23            "type": "ConstantDelay",
24            "delay": 100,
25            "prpl": 0
26        },
27        "usableCapacity": 0.6
28    },
29    "capacity": 42200000
30 }
31 }
32 }
```

Listing B.3: Scenario Barnim: file listing: cell2/network.json

`cell2/regions.json`

```
1 {
2     "regions": [
3     ]
4 }
```

Listing B.4: Scenario Barnim: file listing: cell2/regions.json

`mapping3/mapping_config.json`

```
1 {
2     "prototypes": [
3         {
4             "name": "Car",
5             "accel": 2.6,
6             "decel": 4.5,
7             "emergencyDecel": 6.0,
8             "length": 5.00,
9             "maxSpeed": 70.0,
10             "minGap": 2.5,
11             "sigma": 0.6,
12             "tau": 1,
13             "speedFactor": 1.0,
14             "speedDeviation": 0,
15             "speedMode": "DEFAULT",
16             "laneChangeMode": "DEFAULT"
17         },
18         {
19             "name": "electricCar",
20             "vehicleClass": "ElectricVehicle",
21             "accel": 2.6,
22             "decel": 4.5,
23             "length": 5.00,
24             "maxSpeed": 40.0,
25             "minGap": 2.5,
26             "sigma": 0.6,
27             "tau": 1,
28             "speedFactor": 1.0,
29         }
30 }
```

Listing B.4: Scenario Barnim: file listing: mapping3/mapping_config.json
"speedDeviation": 0,
}
"rsus": [
  {
    "lat": 52.65027,
    "lon": 13.54500,
    "name": "WeatherServer",
    "applications": [ "com.dcaiti.vsimrti.app.tutorials.barnim.WeatherServer"
  ]
}
"vehicles": [
  {
    "startingTime": 5.0,
    "targetFlow": 1800,
    "maxNumberOfVehicles": 120,
    "route": "1",
    "lanes": [ 0, 1 ],
    "types": [
      "applications": [ "com.dcaiti.vsimrti.app.tutorials.barnim.WeatherWarningApp",
                        "com.dcaiti.vsimrti.app.tutorials.barnim.SlowDownApp" ],
      "name": "Car",
      "group": "Cellular",
      "weight": 0.1
    ],
    "applications": [ "com.dcaiti.vsimrti.app.tutorials.barnim.WeatherWarningApp",
                      "com.dcaiti.vsimrti.app.tutorials.barnim.SlowDownApp" ],
      "name": "Car",
      "group": "AdHoc",
      "weight": 0.2
    ],
    "applications": [ "com.dcaiti.vsimrti.app.tutorials.barnim.SlowDownApp" ],
      "name": "Car",
      "group": "Unequipped",
      "weight": 0.6
    ],
    "applications": [ "com.dcaiti.vsimrti.app.tutorials.barnim.SlowDownApp" ],
      "name": "electricCar",
      "group": "Unequipped",
      "weight": 0.1
    ]
  }
]
sns/sns_config.json

```json
{  "version": "0.18.1",  "transmissionModelType": "Flood",  "maximumNumberOfHops": 10,  "singlehop": {    "// comment": "singlehop radius in m",    "radius": 709.4,    "// comment": "delay according to vsimrti-communication",    "delay": {      "type": "SimpleRandomDelay",      "steps": 5,      "minDelay": 0.4,      "maxDelay": 2.4,      "prpl": 0,      "retries": 0    }  },  "multihop": {    "// comment": "delay according to vsimrti-communication",    "delay": {      "type": "GammaRandomDelay",      "minDelay": 10,      "expDelay": 30,      "prpl": 0,      "retries": 0    }  }
}
```

Listing B.6: Scenario Barnim: file listing: sns/sns_config.json

sources/Barnim_fixed.osm

Note: This file is partially listed.

```xml
<?xml version='1.0' encoding='UTF-8'?><osm version='0.6' upload='true' generator='JOSM'>  <bounds minlat='52.59241' minlon='13.51215' maxlat='52.65827' maxlon='13.6227' origin='/uni2926/rcurvearrowse'/><node id='9671195' timestamp='2014-03-25T09:54:31Z' uid='43566' user='anbr' visible='true' version='4' changeset='2383615' lat='52.5798854' lon='13.6374307'/><node id='9671197' timestamp='2014-03-25T09:54:31Z' uid='43566' user='anbr' visible='true' version='5' changeset='14033189' lat='52.5905574' lon='13.6219588'/><node id='9671202' timestamp='2014-03-25T09:54:31Z' uid='429794' user='SennaHB' visible='true' version='5' changeset='13985567' lat='52.6051075' lon='13.5993505'/><node id='21508739' timestamp='2014-03-25T09:54:31Z' uid='43566' user='anbr' visible='true' version='6' changeset='17739489' lat='52.6639091' lon='13.5696135'/><node id='21508746' timestamp='2014-03-25T09:54:31Z' uid='118278' user='Panketaler' visible='true' version='12' changeset='19973396' lat='52.6240762' lon='13.5434165'/><node id='21508747' timestamp='2014-03-25T09:54:31Z' uid='118278' user='Panketaler' visible='true' version='27' changeset='4746328' lat='52.6392207' lon='13.5659381'/><node id='21508748' timestamp='2014-03-25T09:54:31Z' uid='118278' user='Panketaler' visible='true' version='4' changeset='4746328' lat='52.6392207' lon='13.5659381'/><node id='21508756' timestamp='2014-03-25T09:54:31Z' uid='118278' user='Panketaler' visible='true' version='6' changeset='19973383' lat='52.6266342' lon='13.5441771'/><node id='21508757' timestamp='2014-03-25T09:54:31Z' uid='43566' user='anbr' visible='true' version='8' changeset='18626470' lat='52.614571' lon='13.5536105'/></osm>
```

VSimRTI: Vehicle-2-X Simulation Runtime Infrastructure 122
Listing B.7: Scenario Barnim: file listing: sources/Barnim_fixed.osm

Listing B.8: Scenario Barnim: file listing: sumo/Barnim.edg.xml

Note: This file is partially listed.

Listing B.9: Scenario Barnim: file listing: sumo/Barnim.net.xml

Note: This file is partially listed.
Appendix B  Example scenario Barnim

B.2  File listings

Listing B.9: Scenario Barnim: file listing: sumo/Barnim.net.xml

Note: This file is partially listed.

Listing B.10: Scenario Barnim: file listing: sumo/Barnim.nod.xml

sumo/Barnim.nod.xml

sumo/Barnim.rou.xml

VSimRTI: Vehicle-2-X Simulation Runtime Infrastructure 124
Listing B.11: Scenario Barnim: file listing: sumo/Barnim.rou.xml

```xml
<configuration>
  <input>
    <net-file value="Barnim.net.xml"/>
    <route-files value="Barnim.rou.xml"/>
  </input>
  </configuration>
```

Listing B.12: Scenario Barnim: file listing: sumo/Barnim.sumo.cfg

```json
{
  "sumoConfigurationFile": "Barnim.sumo.cfg",
  "updateInterval": 1000
}
```

Listing B.13: Scenario Barnim: file listing: sumo/sumo_config.json
visualizer/visualizer_config.xml

<?xml version="1.0" encoding="UTF-8"?>
<!-- file version: 2013-11-26 -->
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.dcaiti.tu-berlin.de/research/\simulatedownload/get/scenarios/scenarioname/visualizer/\visualizer_config.xsd">
    <visualizer id="filevisualizer" enabled="true" update="5" loader="com.dcaiti.vsimrti.fed.\visualizer.FileVisualizerConfig">
        <filename>visualizer.csv</filename>
        <directory>.</directory>
        <separator>;</separator>
        <messages>
            <message id="VehicleMovements">
                <entries>
                    <!-- Entry List for Vehicle Movements
                </entries>
            </message>
            <message id="ReceiveV2XMessage">
                <entries>
                    <!-- Entry List for Receive V2X Message
                    Time | Type | MessageId | ReceiverName | ReceiverInformation: ReceiveSignalStrength
                </entries>
            </message>
            <message id="SendV2XMessage">
                <entries>
                    <!-- Entry List for Send V2X Message
                    Time | Type | MessageId | SourceName | SourcePosition: Latitude | SourcePosition: Longitude
                </entries>
            </message>
        </messages>
    </visualizer>
</configuration>
<entry>SourcePosition.Elevation</entry>
<entry>Message.Routing.DestinationAddressContainer.Type</entry>
<entry>Message.Routing.DestinationAddressContainer.DestinationAddress.IPv4Address</entry>
<entry>Message.Routing.DestinationAddressContainer.ChannelId</entry>
<entries>
<message id="AddedVehicle" enabled="true">
<entries>
<entry>"ADDED_VEHICLE"</entry>
<entry>Time</entry>
<entry>ApplicationVehicle.Name</entry>
<entry>ApplicationVehicle.VehicleType.VehicleClass</entry>
<entry>ApplicationVehicle.VehicleType.Color</entry>
<entry>ApplicationVehicle.Group</entry>
<entry>ApplicationVehicle.VehicleType.Length</entry>
<entry>ApplicationVehicle.VehicleType.MinGap</entry>
<entry>ApplicationVehicle.VehicleType.MaxSpeed</entry>
<entry>ApplicationVehicle.VehicleType.Accel</entry>
<entry>ApplicationVehicle.VehicleType.Decel</entry>
<entry>ApplicationVehicle.VehicleType.Sigma</entry>
<entry>ApplicationVehicle.VehicleType.Tau</entry>
<entry>ApplicationVehicle.VehicleType.SpeedFactor</entry>
<entry>ApplicationVehicle.VehicleType.SpeedDeviation</entry>
<entry>RouteInfo.DepartingTime</entry>
<entry>RouteInfo.RouteId</entry>
<entry>RouteInfo.DepartureLane</entry>
<entry>RouteInfo.DeparturePos</entry>
</entries>
</message>
</message>
</message>
</entries>
<entries>
<message id="AddedTrafficLight">
<entries>
<entry>"ADDED_TRAFFICLIGHT"</entry>
<entry>Time</entry>
<entry>ApplicationTrafficLight.Name</entry>
<entry>ApplicationTrafficLight.Applications</entry>
<entry>TrafficLightGroup.FirstPosition.Latitude</entry>
<entry>TrafficLightGroup.FirstPosition.Longitude</entry>
</entries>
</message>
</entries>
<entries>
<message id="AddedTrafficSign">
<entries>
<entry>"ADDED_TRAFFICSIGN"</entry>
<entry>Time</entry>
<entry>TrafficSign.Id</entry>
<entry>TrafficSign.GeoPosition.Latitude</entry>
<entry>TrafficSign.GeoPosition.Longitude</entry>
<entry>TrafficSign.GeoPosition.Elevation</entry>
<entry>TrafficSign.Angle</entry>
<entry>TrafficSign.Edgel</entry>
<entry>TrafficSign.Lane</entry>
<entry>TrafficSign.TypeId</entry>
<entry>TrafficSign.TypeId</entry>
<entry>TrafficSign.TypeId</entry>
</entries>
</message>
</entries>
<entries>
<message id="ChangeSpeed">
<entries>
<entry>"CHANGE_SPEED"</entry>
<entry>Time</entry>
<entry>VehicleId</entry>
<entry>Type</entry>
<entry>Speed</entry>
<entry>Interval</entry>
<entry>Acceleration</entry>
</entries>
</message>
</entries>
<message id="AddedRsu">

VSimRTI: Vehicle-2-X Simulation Runtime Infrastructure
<entries>
  <entry>"ADDED_RSU"</entry>
  <entry>Time</entry>
  <entry>ApplicationRsu.Name</entry>
  <entry>ApplicationRsu.Position.Latitude</entry>
  <entry>ApplicationRsu.Position.Longitude</entry>
  <entry>ApplicationRsu.Position.Elevation</entry>
  <entry>ApplicationRsu.Group</entry>
  <entry>ApplicationRsu.Applications</entry>
</entries>
</message>

<message id="ConfigureAdHocCommunication">
<entries>
  <entry>"CONFIGURE_ADHOC"</entry>
  <entry>Time</entry>
  <entry>Configuration.NodeID</entry>
  <entry>Configuration.RadioMode</entry>
  <entry>Configuration.Conf0.NewIP</entry>
  <entry>Configuration.Conf0.NewPower</entry>
  <entry>Configuration.Conf0.Channel0</entry>
  <entry>Configuration.Conf0.Channel1</entry>
  <entry>Configuration.Conf1.NewIP</entry>
  <entry>Configuration.Conf1.NewPower</entry>
  <entry>Configuration.Conf1.Channel0</entry>
  <entry>Configuration.Conf1.Channel1</entry>
</entries>
</message>

<message id="ConfigureCellCommunication">
<entries>
  <entry>"CONFIGURE_CELL"</entry>
  <entry>Time</entry>
  <entry>Configuration.NodeID</entry>
  <entry>Configuration.Enabled</entry>
  <entry>Configuration.MaxDLBitrate</entry>
  <entry>Configuration.MaxULBitrate</entry>
</entries>
</message>

<message id="TrafficDetectorUpdates">
<entries>
  <entry>"DETECTOR_UPDATES"</entry>
  <entry>Time</entry>
  <entry>UpdatedInductionLoops:LaneId</entry>
  <entry>UpdatedInductionLoops:VehicleCount</entry>
  <entry>UpdatedInductionLoops:MeanSpeed</entry>
  <entry>UpdatedInductionLoops:TrafficFlow</entry>
  <entry>UpdatedLaneAreaDetectors:LaneId</entry>
  <entry>UpdatedLaneAreaDetectors:Length</entry>
  <entry>UpdatedLaneAreaDetectors:VehicleCount</entry>
  <entry>UpdatedLaneAreaDetectors:MeanSpeed</entry>
  <entry>UpdatedLaneAreaDetectors:Occupancy</entry>
</entries>
</message>

<message id="ChangeLaneProperty">
<entries>
  <entry>"CHANGE_LANE_STATE"</entry>
  <entry>Time</entry>
  <entry>EdgeId</entry>
  <entry>LaneIndex</entry>
  <entry>AllowedVehicleClasses</entry>
  <entry>DisallowedVehicleClasses</entry>
  <entry>MaxSpeed</entry>
</entries>
</message>
</messages>
</visualizer>
Appendix B Example scenario Barnim

```xml
<visualizer id="statistics" enabled="true" loader="com.dcaiti.vsimrti.fed.visualizer.">
  <StatisticsVisualizerConfig>
    <statistic filename="AverageVehicleTravelTime" output="short">
      <source>VehicleTravelTimes</source>
      <filter filterType="keep">Time:0,300</filter>
      <group-by>VehicleGroup</group-by>
      <aggregation>Average</aggregation>
    </statistic>
    <messages>
      <message id="AddedVehicle"/>
      <message id="VehicleMovements"/>
      <message id="TrafficDetectorUpdates"/>
    </messages>
  </StatisticsVisualizerConfig>
</visualizer>

<visualizer id="itef" enabled="false" loader="com.dcaiti.vsimrti.fed.visualizer.">
  <ITEFVisualizerConfig>
    <upload enabled="false">
      <target></target>
      <scenarioId></scenarioId>
      <user></user>
      <password></password>
    </upload>
    <messages>
      <message id="AddedVehicle" enabled="true"/>
      <message id="VehicleMovements" enabled="true"/>
    </messages>
  </ITEFVisualizerConfig>
</visualizer>

<visualizer id="websocket" enabled="true" loader="com.dcaiti.vsimrti.fed.visualizer.">
  <WebsocketVisualizerConfig>
    <synchronized>true</synchronized>
    <port>46587</port>
    <messages>
      <message id="VehicleMovements" enabled="true"/>
      <message id="ReceiveV2XMessage" enabled="true"/>
      <message id="SendV2XMessage" enabled="true"/>
      <message id="AddedVehicle" enabled="true"/>
      <message id="AddedRsu" enabled="true"/>
    </messages>
  </WebsocketVisualizerConfig>
</visualizer>
</configuration>
```

Listing B.14: Scenario Barnim: file listing: visualizer/visualizer_config.xml
vsimrti/vsimrti_config.xml

```xml
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="https://www.dcaiti.tu-berlin.de/research/\simulation/download/get/scenarios/scenarioName/vsimrti/vsimrti_config.xsd">

  <simulation>
    <!-- Scenario name -->
    <id>Barnim</id>
    <!-- Simulation time frame -->
    <endtime>1000</endtime>
    <!-- Seed for pseudo random number generator used by most of the federates -->
    <randomSeed>212323853</randomSeed>
    <!-- Projection between global and cartesian coordinates -->
    <!-- centerCoordinates: roughly the center of simulation area -->
    <!-- cartesianOffset: can be found in the generated network file for the traffic simulator, e.g. the .net.xml file for sumo -->
    <WGS84UTMTransformConfig>
      {
        "centerCoordinates": {
          "longitude": 13.56,
          "latitude": 52.63
        },
        "cartesianOffset": {
          "x": -395635.35,
          "y": -5826456.24
        }
      }
    </WGS84UTMTransformConfig>
    <IPResolverConfig>
      {
        "netMask": "255.255.0.0",
        "vehicleNet": "10.1.0.0",
        "rsuNet": "10.2.0.0",
        "tlNet": "10.3.0.0",
        "csNet": "10.4.0.0",
        "serverNet": "10.5.0.0",
        "tmcNet": "10.6.0.0"
      }
    </IPResolverConfig>
    <threads>1</threads>
  </simulation>
  <federates>
    <!-- Cellular network simulator -->
    <federate id="cell2" active="false"/>
    <federate id="cell21" active="false"/>
    <federate id="cell212" active="false"/>
    <!-- V2I (ad hoc) network simulators -->
    <federate id="omnetpp" active="false"/>
    <federate id="ns3" active="false"/>
    <federate id="sns" active="true"/>
    <!-- Traffic simulators -->
    <federate id="sumo" active="true"/>
    <federate id="applicationNT" active="true"/>
    <federate id="environment" active="true"/>
    <federate id="battery" active="true"/>
    <federate id="chargingstation" active="false"/>
    <federate id="mapping" active="true"/>
    <federate id="mapping3" active="true"/>
    <federate id="visualizer" active="true"/>
  </federates>
</configuration>
```
Listing B.15: Scenario Barnim: file listing: vsimrti/vsimrti_config.xml

vsimrti/simrun_config.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- file version: 2014-08-27 -->
<!-- This configuration induces the execution of 6 simulations (2 dimensions with 5 parameter for each dimension).
The simulations will be executed one after another (sequential mode).
following simulations would be triggered:

The parameter set (V2XVehiclePercentage, CellVehiclePercentage, ClassicVehiclePercentage and Simulationtime)
next to parameter (SinglehopRadius).

They define the dimensions of the simulations matrix.

==== 001. simulation ====
V2XVehiclePercentage=0
CellVehiclePercentage=0
ClassicVehiclePercentage=100
Simulationtime=100
SinglehopRadius=500

==== 002. simulation ====
V2XVehiclePercentage=0
CellVehiclePercentage=0
ClassicVehiclePercentage=100
Simulationtime=100
SinglehopRadius=600

==== 003. simulation ====
V2XVehiclePercentage=50
CellVehiclePercentage=0
ClassicVehiclePercentage=50
Simulationtime=100
SinglehopRadius=500

==== 004. simulation ====
V2XVehiclePercentage=50
CellVehiclePercentage=0
ClassicVehiclePercentage=50
Simulationtime=100
SinglehopRadius=600

==== 005. simulation ====
V2XVehiclePercentage=75
CellVehiclePercentage=0
ClassicVehiclePercentage=25
Simulationtime=100
SinglehopRadius=500

==== 006. simulation ====
V2XVehiclePercentage=75
CellVehiclePercentage=0
ClassicVehiclePercentage=25
Simulationtime=100
SinglehopRadius=600

==== 007. simulation ====
V2XVehiclePercentage=0
CellVehiclePercentage=50
```

VSimRTI: Vehicle-2-X Simulation Runtime Infrastructure 131
Appendix B Example scenario Barnim B.2 File listings

60 ClassicVehiclePercentage = 50
61 Simulationtime = 100
62 SinglehopRadius = 600
63
64 ===== 008. simulation =====
65 V2XVehiclePercentage = 0
66 CellVehiclePercentage = 50
67 ClassicVehiclePercentage = 50
68 Simulationtime = 100
69 SinglehopRadius = 600
70
71 ===== 009. simulation =====
72 V2XVehiclePercentage = 0
73 CellVehiclePercentage = 25
74 ClassicVehiclePercentage = 25
75 Simulationtime = 100
76 SinglehopRadius = 600
77
78 ===== 010. simulation =====
79 V2XVehiclePercentage = 0
80 CellVehiclePercentage = 75
81 ClassicVehiclePercentage = 25
82 Simulationtime = 100
83 SinglehopRadius = 600
84
85 <configuration
86 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
87 xsi:noNamespaceSchemaLocation="http://www.dcaiti.tu-berlin.de/research/simulation/download/"
88 get/scenarios/scenarioName/vsimrti/simrun_config.xsd">
89 <!-- basic configuration -->
90 <vsimrti location="/path/to/your/vsimrti_folder" executable="vsimrti.sh" username="/"
91 your_user_id" />
92 <scenario name="Barnim" config="scenarios/Barnim/vsimrti/vsimrti_config.xml" persistent="/"
93 false" repetitions="1" progressLogger="true">
94 <!-- argument -->-o TRACE</argument -->
95 </scenario>
96
97 <!-- define connected values for controlled changes -->
98 <dimension name="SimulationRuns">
99 <parameter name="V2XVehiclePercentage" file="mapping3/mapping_config.json" fileFormat="/"
100 json item="vehicles[0].types[0].weight" type="ValueList">
101 <value>0</value>
102 <value>50</value>
103 <value>75</value>
104 </parameter>
105 <parameter name="CellVehiclePercentage" file="mapping3/mapping_config.json" fileFormat="/"
106 json item="vehicles[0].types[1].weight" type="ValueList">
107 <value>0</value>
108 <value>0</value>
109 <value>50</value>
110 <value>75</value>
111 </parameter>
112 <parameter name="ClassicVehiclePercentage" file="mapping3/mapping_config.json" fileFormat="/"
113 json item="vehicles[0].types[2].weight" type="ValueList">
114 <value>100</value>
115 <value>50</value>
116 <value>25</value>
117 <value>25</value>
118 </parameter>
119 <parameter name="Simulationtime" file="vsimrti/vsimrti_config.xml" fileFormat="xml" item="/"
120 //simulation/endtime" type="ValueList">
<!-- define values for automatically permuted simulations -->
<parameter name="SinglehopRadius" file="sns/sns_config.json" fileFormat="json" item="/uni2926/curvearrowssinglehop.radius" type="ValueList">
  <value>500</value>
  <value>600</value>
</parameter>
</configuration>

Listing B.16: Scenario Barnim: file listing: vsimrti/simrun_config.xml
Appendix C

Example scenario Tiergarten

You will find a collection of all relevant data.

C.1 Folder Structure

```
Tiergarten
  applicationNT .................................................................
  applications ....................................................................
  applicationNT_config.json .............................................. not listed (in this release)
  tiergarten.db ............................................................... not listed (binary content)
  cell2 ..............................................................................
  cell2_config.json ..................................................... not listed (in this release)
  network.json ................................................................. not listed (in this release)
  regions.json ................................................................. not listed (in this release)
  mapping3 ........................................................................
  mapping_config.json ................................................... not listed (in this release)
  omnetpp ........................................................................
  omnetpp.ini ................................................................. (see C.1)
  ns3 ............................................................................
  configTechnologies.xml ........................................... not listed (in this release)
  confWifi.xml ................................................................. not listed (in this release)
  sns .............................................................................
  sns_config.json ............................................................ not listed (in this release)
  sources ........................................................................
  tiergarten.osm ............................................................... not listed (in this release)
  sumo ............................................................................
  sumo_config.json ........................................................... not listed (in this release)
  tiergarten.edg.xml ........................................................... not listed (in this release)
  tiergarten.net.xml ........................................................... not listed (in this release)
  tiergarten.nod.xml ........................................................... not listed (in this release)
  tiergarten.rou.xml ........................................................... not listed (in this release)
  tiergarten.sumo.cfg ....................................................... not listed (in this release)
  visualizer ....................................................................
  visualizer_config.xml .................................................... not listed (in this release)
  vsimrti ........................................................................
  vsimrti_config.xml ........................................................... not listed (in this release)
```

Figure C.1: Scenario Tiergarten: folder structure
Appendix C Example scenario Tiergarten

C.2 File listings

omnetpp/omnetpp.ini

```plaintext
[General]

# SimulationParameters
# -----------------
# network *.ned-file and time-scale in nanoseconds
network=omnetpp_federate.mgmt.Simulation
simtime-resolution=1ns

# EventScheduler
# ------------
# scheduler-class and debugging option for more verbose logging
scheduler-class = "omnetpp_federate::VSimRTIEventScheduler"
vsimrtieventscheduler-debug = true
# connection settings, when omnetpp_federate is started manually
vsimrtieventscheduler-host = "localhost"
vsimrtieventscheduler-port = 4998

# RecordingMode
# ------------
record-eventlog = false
cmdenv-express-mode = false
cmdenv-event-banners = false
cmdenv-log-prefix = "%t: "

# random numbers
# ------------
num-rngs = 3
Simulation.mobility.rng-0 = 1
Simulation.wlan[*].mac.rng-0 = 2

# general logging output
# ----------------
# *.mgmt.cmdenv-ev-output = false
# *.mgmt.debug = false
# *.veh[\*].**.cmdenv-ev-output = false
# *.rsu[\*].**.cmdenv-ev-output = false

# application settings
# Simulation.rsu[\*].udpApp.maxProcDelay = 1e-3
# Simulation.veh[\*].udpApp.maxProcDelay = 1e-3

# UDP Settings
# network layer settings
# mac and phy settings

**.wlan.mgmt.typename = "Ieee80211MgmtAdhoc"
**.wlan.agent.typename = ""
**.wlan.opMode = "p"
**.wlan.bitrate = 6Mbps

**.wlan.mac.typename = "Ieee80211Mac"
**.wlan.mac.qosStation = true
**.wlan.mac.*.rateSelection.Bitrate = 3Mbps
**.wlan.mac.hcf.maxQueueSize = 10
**.wlan.mac.hcf.edca.**.cuMin = 15
**.wlan.mac.dcf.recoveryProcedure.shortRetryLimit = 7
**.wlan.mac.dcf.recoveryProcedure.longRetryLimit = 7
```

VSimRTI: Vehicle-2-X Simulation Runtime Infrastructure
**.wlan*.radio.bandName = "5.9 GHz" #this string actually selects the band from (2.4; 5; 5.9)\n\n**.wlan*.radio.carrierFrequency = 5.9GHz
**.wlan*.radio.channelNumber = 4
**.wlan*.radio.bandwidth = 10MHz
**.wlan*.radio.receiver.snrThreshold = 4dB
**.wlan*.radio.receiver.sensitivity = -81dBm
**.wlan*.radio.transmitter.power = 0.05W

# ########## radio medium settings ############
Simulation.radioMedium.radioModeFilter = true #use this filter for increased performance -> \n\n\nSimulation.radioMedium.listeningFilter = true #second filter that may improve performance
Simulation.radioMedium.backgroundNoise.power = -110dBm
Simulation.radioMedium.mediumLimitCache.carrierFrequency = 5.9GHz
Simulation.radioMedium.propagation.typename = "ConstantSpeedPropagation"
Simulation.radioMedium.pathLoss.typename = "FreeSpacePathLoss"
Simulation.radioMedium.obstacleLoss.typename = ""

# ########## mobility ###################
**.mobility.constraintAreaMinX = 0m
**.mobility.constraintAreaMinY = 0m
**.mobility.constraintAreaMinZ = 0m
**.mobility.constraintAreaMaxX = 5000m
**.mobility.constraintAreaMaxY = 5000m
**.mobility.constraintAreaMaxZ = 0m

Listing C.1: Scenario Tiergarten: file listing: omnetpp/omnetpp.ini
Appendix D

Package Additional Examples
This package shows different sample applications for several application spectrum. Here you can download the `AdditionalExamples-19.1-sources.zip` file:

https://www.dcaiti.tu-berlin.de/research/simulation/download/

The following table describes these applications.
### Applications from the "Additional Examples" package

<table>
<thead>
<tr>
<th>Application Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HelloWorldApp</td>
<td>This is a simple Hello World application. It prints a hello world string in an interval.</td>
</tr>
<tr>
<td>InterconnectApp</td>
<td>This simple application demonstrates an interconnection between applications which are running on same units.</td>
</tr>
<tr>
<td>IntervalSamplingApp</td>
<td>This simple application demonstrates a sampling in a specific interval. When the simulation starts, it initialises with a random generated offset (e.g., 150 ms). Afterwards a fixed interval will be invoked (e.g., 1 second). For example, there are the following time steps: ( t = {0 \text{ ms, 150 ms, 1150 ms, 2150 ms, ...} } )</td>
</tr>
<tr>
<td>MultithreadSamplingApp</td>
<td>This simple application demonstrates a concurrency task. In a scenario, there may exist many vehicles (e.g., veh_0, veh_1, veh_2, ...). This application demonstrates the following behavior: veh_0, veh_1, veh_2, ... start a thread veh_0, veh_1, veh_2, ... do some logic parallel to each other veh_0, veh_1, veh_2, ... join the thread</td>
</tr>
<tr>
<td>MyMessage</td>
<td>This application shows how to create a message.</td>
</tr>
<tr>
<td>RandomSamplingIntervalApp</td>
<td>This application demonstrates a sampling in a random interval. One random simulation point will be chosen when the initialisation starts. For example: ( t = {0 \text{ ms, 300 ms, 967 ms, 1487 ms, ...} } )</td>
</tr>
<tr>
<td>SumoTraciInteractionApp</td>
<td>This sample shows the interaction between an application and the TraCI interface of SUMO.</td>
</tr>
<tr>
<td>OnUpdateVehicleInfoApp</td>
<td>This application reacts on a vehicle info update. Mostly a vehicle info will be triggered from a traffic simulator (e.g., SUMO).</td>
</tr>
<tr>
<td>UserTaggedValueRoundtripApp</td>
<td>Demonstrates the usage of setting a user tagged value for additional byte sending via CAM. If the control of every byte is not needed, the DefaultObjectSerialization can be used. This class converts an object into a byte field and inversely.</td>
</tr>
<tr>
<td>VSimRTIMessageHandlingApp</td>
<td>This simple application shows how to create a VSimRTI-message and sends it to all simulators. This application can also react on ApplicationSpecificMessages.</td>
</tr>
<tr>
<td>VehicleConfigurationApp</td>
<td>This simple application demonstrates a configurable application.</td>
</tr>
</tbody>
</table>

Table D.1: Applications from the "Hello world package"
Appendix E

Configuration Schemata

Some of the required JSON configuration files are automatically validated by VSimRTI and, therefore, must follow a specific schema\(^1\). If you have problems with VSimRTI accepting your configuration files, please make sure they follow the respective schema:

### JSON Schema for cell2/cell2_config.json

```json
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "type": "object",
    "properties": {
        "bandwidthMeasurementInterval": { "type": "number", "minimum": 1 },
        "bandwidthMeasurements": {
            "type": "array",
            "items": [
                { "type": "object",
                    "properties": {
                        "fromRegion": { "type": "string" },
                        "toRegion": { "type": "string" },
                        "transmissionMode": {
                            "type": "string",
                            "enum": [ "UplinkUnicast", "DownlinkUnicast", "DownlinkMulticast" ]
                        }
                    },
                    "required": [ "fromRegion", "toRegion", "transmissionMode" ]
                }
            ],
        "networkConfigurationFile": { "type": "string" },
        "regionConfigurationFile": { "type": "string" }
    },
    "required": [ "networkConfigurationFile" ]
}
```

Listing E.1: JSON Schema for cell2_config.json

\(^1\)The schemata listed here follow the specification at [http://json-schema.org/documentation.html](http://json-schema.org/documentation.html)
JSON Schema for cell2/network_config.json

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "globalRegion": {
      "type": "object",
      "properties": {
        "uplink": {
          "type": "object",
          "properties": {
            "delay": { "$ref": "#/definitions/delay"},
            "capacity": {"$ref": "#/definitions/capacity"}
          },
          "required": ["delay", "capacity"]
        },
        "downlink": {
          "type": "object",
          "properties": {"unicast": {"$ref": "#/definitions/unicast"},
          "multicast": {"$ref": "#/definitions/multicast"},
          "capacity": {"$ref": "#/definitions/capacity"}
        },
          "required": ["unicast", "multicast", "capacity"]
        }
      },
      "required": ["globalRegion"]
    },
    "definitions": {
      "delay": {
        "oneOf": [
          {
            "type": "object",
            "properties": {
              "type": { "type": "string", "enum" : ["GammaRandomDelay", "GammaSpeedDelay"]},
              "minDelay": {"type": "number", "minimum": 0},
              "expDelay": {"type": "number", "minimum": 0},
              "prpl": {"type": "number", "minimum": 0},
              "retries": {"type": "number", "minimum": 0}
            },
            "required": ["type", "minDelay", "expDelay", "prpl"]
          },
          {
            "type": "object",
            "properties": {
              "type": { "type": "string", "enum" : ["ConstantDelay"]},
              "delay": {"type": "number", "minimum": 0},
              "prpl": {"type": "number", "minimum": 0},
              "retries": {"type": "number", "minimum": 0}
            },
            "required": ["type", "delay", "prpl"]
          }
        ]
      }
    }
  }
}
```
"type": {"type": "string",
   "enum": ["SimpleRandomDelay"]},
"steps": {"type": "number",
   "minimum": 0},
"minDelay": {"type": "number",
   "minimum": 0},
"maxDelay": {"type": "number",
   "minimum": 0},
"prpl": {"type": "number",
   "minimum": 0},
"retries": {"type": "number",
   "minimum": 0}
},
"required": ["type", "steps", "minDelay", "maxDelay", "prpl"]
"
"unicast": {
"type": "object",
"properties": {
"delay": {"$ref": 
"#/definitions/delay"}
},
"required": ["delay"]
"multicast": {
"type": "object",
"properties": {
"delay": {"$ref": 
"#/definitions/delay"},
"usableCapacity": {"type": "number",
   "minimum": 0.0,
   "maximum": 1.0}
},
"required": ["delay", "usableCapacity"]
"capacity": {
"oneOf": [
   {
   "type": "integer",
   "minimum": 0
   },
   {
   "type": "string",
   "pattern": "unlimited$"
   }
]}
"required": ["type"],
"enum": ["SimpleRandomDelay"]

Listing E.2: JSON Schema for network_config.json
JSON Schema for cell2/regions_config.json

```json
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "type": "object",
    "properties": {
        "regions": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "polygon": {
                        "type": "object",
                        "coordinates": [
                            {"$ref": "#/definitions/geopoint"}
                        ],
                        "required": ["coordinates"]
                    },
                    "area": {
                        "type": "object",
                        "properties": {
                            "nw": {"$ref": "#/definitions/geopoint"},
                            "se": {"$ref": "#/definitions/geopoint"}
                        },
                        "required": ["nw", "se"]
                    },
                    "id": {"type": "string"},
                    "uplink": {
                        "type": "object",
                        "properties": {
                            "delay": {"$ref": "#/definitions/delay"},
                            "capacity": {"$ref": "#/definitions/capacity"}
                        },
                        "required": ["delay", "capacity"]
                    },
                    "downlink": {
                        "type": "object",
                        "properties": {
                            "unicast": {"$ref": "#/definitions/unicast"},
                            "multicast": {"$ref": "#/definitions/multicast"},
                            "capacity": {"$ref": "#/definitions/capacity"}
                        },
                        "required": ["unicast", "multicast", "capacity"]
                    }
                }
            }
        },
        "anyOf": [
            {"required": ["area", "id", "uplink", "downlink"]},
            {"required": ["polygon", "id", "uplink", "downlink"]}
        ]
    }
}
```

"definitions": {
    "delay": {
        "oneOf": [{
            "type": "object",
            "properties": {
                "type": "string",
                "enum": ["GammaRandomDelay", "GammaSpeedDelay"],
                "minDelay": {"type": "number"},
                "exclusiveMinimum": 0,
                "expDelay": {"type": "number"},
                "exclusiveMinimum": 0
            }
        }
    }
}
```
"prpl" : { "type" : "number", 
    "minimum" : 0},
"retries" : { "type" : "number", 
    "minimum" : 0}
},
"required" : ["type", "minDelay", "expDelay", "prpl"]
),

"type" : "object",
"properties" : {
    "type" : { "type" : "string", 
        "enum" : ["ConstantDelay"]},
    "delay" : { "type" : "number", 
        "exclusiveMinimum" : 0},
    "prpl" : { "type" : "number", 
        "minimum" : 0},
    "retries" : { "type" : "number", 
        "minimum" : 0}
},
"required" : ["type", "delay", "prpl"]
},

"type" : "object",
"properties" : {
    "type" : { "type" : "string", 
        "enum" : ["SimpleRandomDelay"]},
    "steps" : { "type" : "number", 
        "minimum" : 0},
    "minDelay" : { "type" : "number", 
        "exclusiveMinimum" : 0},
    "maxDelay" : { "type" : "number", 
        "exclusiveMinimum" : 0},
    "prpl" : { "type" : "number", 
        "minimum" : 0},
    "retries" : { "type" : "number", 
        "minimum" : 0}
},
"required" : ["type", "steps", "minDelay", "maxDelay", "prpl"]
}

"unicast" : {
    "type" : "object",
    "properties" : {
        "delay" : { "$ref" : "#/definitions/delay"}
    },
    "required" : ["delay"]
}

"multicast" : {
    "type" : "object",
    "properties" : {
        "delay" : { "$ref" : "#/definitions/delay"},
        "usableCapacity" : { "type" : "number", 
            "minimum" : 0.0,
            "maximum" : 1.0}
    },
    "required" : ["delay", "usableCapacity"]
}

"geopoint" : {
    "type" : "object",
    "properties" : {
        "lon" : { "type" : "number"},
        "lat" : { "type" : "number"}
    },
    "required" : ["lon", "lat"]
}

"capacity" : {

}
"oneOf": [
  {
    "type": "integer",
    "minimum": 0
  },
  {
    "type": "string",
    "pattern": "^unlimited$"
  }
]
# Appendix F

## Message Dictionary

The following tables show supported entries according to listing 10.2. The entries can therefore be used to configure the file visualizer accordingly. Additionally, the simulators are given which usually emit the particular message.

<table>
<thead>
<tr>
<th>Message</th>
<th>Entries</th>
<th>Simulator</th>
</tr>
</thead>
</table>

Table F.1: Traffic Messages
<table>
<thead>
<tr>
<th>Message</th>
<th>Entries</th>
<th>Simulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeLane</td>
<td>Time&lt;br&gt;VehicleId&lt;br&gt;ChangeLaneMode&lt;br&gt;TargetLaneIndex&lt;br&gt;Duration</td>
<td>applicationNT</td>
</tr>
<tr>
<td>ChangeSpeed</td>
<td>Time&lt;br&gt;VehicleId&lt;br&gt;Type&lt;br&gt;Speed&lt;br&gt;Interval&lt;br&gt;Acceleration</td>
<td>applicationNT</td>
</tr>
<tr>
<td>ChangeStaticRoute</td>
<td>Time&lt;br&gt;VehicleId&lt;br&gt;RouteId</td>
<td>applicationNT</td>
</tr>
<tr>
<td>ChangeVehicleState</td>
<td>Time&lt;br&gt;Name&lt;br&gt;State</td>
<td>applicationNT</td>
</tr>
<tr>
<td>SlowDown</td>
<td>Time&lt;br&gt;Speed&lt;br&gt;VehicleId&lt;br&gt;Interval</td>
<td>applicationNT</td>
</tr>
</tbody>
</table>

Table F.2: Vehicle Control Messages

<table>
<thead>
<tr>
<th>Message</th>
<th>Entries</th>
<th>Simulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrafficDetectorUpdates</td>
<td>Time&lt;br&gt;UpdatedInductionLoops:LaneId&lt;br&gt;UpdatedInductionLoops:VehicleCount&lt;br&gt;UpdatedInductionLoops:MeanSpeed&lt;br&gt;UpdatedInductionLoops:Occupancy&lt;br&gt;UpdatedLaneAreaDetectors:LaneId&lt;br&gt;UpdatedLaneAreaDetectors:Length&lt;br&gt;UpdatedLaneAreaDetectors:VehicleCount&lt;br&gt;UpdatedLaneAreaDetectors:MeanSpeed&lt;br&gt;UpdatedLaneAreaDetectors:Occupancy</td>
<td>SUMO</td>
</tr>
<tr>
<td>ChangeLaneProperty</td>
<td>Time&lt;br&gt;EdgeId&lt;br&gt;LaneIndex&lt;br&gt;AllowedVehicleClasses&lt;br&gt;DisallowedVehicleClasses&lt;br&gt;MaxSpeed</td>
<td>applicationNT</td>
</tr>
</tbody>
</table>

Table F.3: Traffic Management
<table>
<thead>
<tr>
<th>Message</th>
<th>Entries</th>
<th>Simulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddedTmc</td>
<td>Time&lt;br&gt;ApplicationTmc.Name&lt;br&gt;ApplicationTmc.Group&lt;br&gt;ApplicationTmc.Applications&lt;br&gt;ApplicationTmc.InductionLoops&lt;br&gt;ApplicationTmc.LaneAreaDetectors</td>
<td>Mapping3</td>
</tr>
</tbody>
</table>

Table F.4: Initialization Messages
<table>
<thead>
<tr>
<th>Message</th>
<th>Entries</th>
<th>Simulator</th>
</tr>
</thead>
</table>
| VehicleHandover                 | Time  
  Updated:NodeId  
  Updated:PreviousRegion  
  Updated:CurrentRegion    | Cell2         |
| ConfigureAdHocCommunication     | Time  
  Configuration.NodeID  
  Configuration.RadioMode  
  Configuration.Conf0.NewIP  
  Configuration.Conf0.NewPower  
  Configuration.Conf0.Channel0  
  Configuration.Conf0.Channel1  
  Configuration.Conf1.NewIP  
  Configuration.Conf1.NewPower  
  Configuration.Conf1.Channel0  
  Configuration.Conf1.Channel1 | applicationNT |
| ConfigureCellCommunication      | Time  
  Configuration.NodeID  
  Configuration.Enabled  
  Configuration.MaxDlBitrate  
  Configuration.MaxUlBitrate | applicationNT |
| SendV2XMessage                  | Time  
  Type  
  MessageId  
  SourceName  
  SourcePosition.Latitude  
  SourcePosition.Longitude  
  SourcePosition.Elevation | applicationNT |
| ReceiveV2XMessage               | Time  
  Type  
  MessageId  
  ReceiverName  
  ReceiverInformation.ReceiveSignalStrength | ns-3,  
  OMNeT++,  
  SNS,  
  Cell2         |

Table E.5: Communication Messages
<table>
<thead>
<tr>
<th>Message</th>
<th>Entries</th>
<th>Simulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentEvents</td>
<td>Time</td>
<td>eventserver</td>
</tr>
<tr>
<td></td>
<td>CurrentEvents:Type</td>
<td></td>
</tr>
<tr>
<td>SensorData</td>
<td>Time</td>
<td>eventserver</td>
</tr>
<tr>
<td></td>
<td>UnitId</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Events:Type</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Events:Strength</td>
<td></td>
</tr>
<tr>
<td>SensorRegistration</td>
<td>Time</td>
<td>eventserver</td>
</tr>
<tr>
<td></td>
<td>VehicleId</td>
<td></td>
</tr>
</tbody>
</table>

Table F.6: Environmental Events Messages
G References


