Vehicle-2-X Simulation Runtime Infrastructure (VSimRTI)

User Documentation

Tobias Queck
Bjoern Schuenemann
Nico Naumann
David Rieck
Stefan Reichel
Robert Protzmann
Franz Kage
Erik Schleiff
Jiajun Hu

Contents

Chapter 1

Introduction to VSimRTI

1.1 Overview

This documentation comes along with the current release of VSimRTI and aims to support users in their first steps of working with it.

VSimRTI at a glance

To provide the flexibility to exchange simulators, we have developed the Vehicle-2-X Simulation Runtime Infrastructure (VSimRTI). Our simulation infrastructure offers interfaces for the integration of simulators, e.g. for network, traffic, and environment simulations. It provides the flexibility to exchange them without changing the infrastructure. For the synchronization of and the communication among all components, the implemented infrastructure uses concepts defined in the 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 Vehicle-2-X simulations. Based on the requirements of a concrete scenario, arbitrary simulators can be plugged onto the VSimRTI and are executed together.

To get familiar with VSimRTI, a package was set up, that already contains dedicated simulators and an example scenario, where V2X communication is used to avoid dangerous situations occuring due to a slippery road segment. The scenario is located in the area of the motorway junction Autobahndreieck Schwanebeck in the north-east of Berlin. In the scenario an application ensures the notification of the slippery road and the broadcast of warning messages to other vehicles. As a result following vehicles reduce their speed and vehicles at greater distance try to reroute to circumnavigate the dangerous location when they receive the warning.

To simulate this V2X communication scenario the well known network simulator JiST/SWANS and the traffic simulator SUMO are applied. For real roadmap

data and environment events eWorld is used.

Content Of Package

The most important part of the package is VSimRTI itself. VSimRTI is fully written in Java. It requires Java 1.5 and is tested with 1.5 and 1.6. Based on the fact that it is written in plain Java it is independent of the underlying operating system.

As VSimRTI couples different simulators, VSimRTI can't be run alone. Therefore it requires pre-installed simulators. We already added the JiST/SWANS communication simulator including extensions created by Ulm University and the environment simulator eWorld from Hasso Plattner Institute Potsdam, since they are also written in Java and OS independent. The binaries for the traffic simulator SUMO are not included here, but can be obtained from the SUMO page on sourceforge.net for different operating systems. The current SUMO version, which is supported by VSimRTI is 0.12. Older versions will not work, since there were major changes within SUMO in the current release. It cannot be guaranteed, that VSimRTI runs with newer versions of SUMO. To run other simulators than the provided, an additional component has to be written, which couples the simulator to the VSimRTI. Please contact the VSimRTI team (vsimrti@fokus.fraunhofer.de) for detailed information on this. Further information on the supported simulators are available here:

Simulator	URL	Comment
SUMO	http://sumo.sourceforge.net	
JiST/SWANS	http://jist.ece.cornell.edu/	(original distribution)
	http://vanet.info/node/11	(VANET extension of Ulm University)
eWorld:	http://eworld.sourceforge.net/	

For first runs of a simulation, this archive includes additional files for the example scenario "Schwanebeck", which is referenced throughout this document.

VSimRTI Folder Structure

With the release of VSimRTI V0.9.9, the folder structure of the project has changed significantly. To maintain clarity and usability we tried to separate binary files, general VSimRTI configuration files and simulation scenario specific

configuration files in different folders.
General VSimRTI configurations which stay constant in different scenarios can be found in the /etc folder. 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 /scenarios folder you can find the scenario specific configuration files, separated by component, e.g. traffic simulator or communication simulator. To configure scenario specific VSimRTI options, e.g. which simulators should be active in a simulation scenario you can adjust the vsimrti_config.xml file located in scenarios/<scenarioName>/vsimrti.

The overall folder structure is as follows:

Directory/File	**Description**
vsimrti	
|-bin	Binary files for VSimRTI
|-etc	General configuration files for VSimRTI
|-logs	All log files go here
|-scenarios	Scenario specific configuration files
—|-Schwanebeck-Default	
|-tmp	Folder to store temporary files during a simulation

1.2 Run A Simulation

To run your first simulation, follow these steps:

1. Since many components are written in Java, be sure that a JRE (at least 1.5) is installed.
2. The package vsimrti-bin-AllInOne.zip has to be extracted to an arbitrary path. The installation path is referenced as <vsimrti-allinone> throughout this document. In this package nearly all needed parts are available, except for the SUMO traffic simulator.
3. Install SUMO. For Microsoft Windows operating systems simply download and extract sumo_winbin.zip. Please refer to inary SUMO Wiki [1] for further information. Note, that at least SUMO release 0.12 has to be installed, since there were major changes compared to older versions, which will not work together with VSim TI anymore. Future versions might work, too, but might cause unexpected problems
4. Make the SUMO binaries available everywhere by adding the SUMO binary folder to your PATH variable
5. Before starting a simulation make sure, that all configurations are properly done. More details about the configuration can be found in the next chapter.

[1] http://apps.sourceforge.net/mediawiki/sumo/index.php?title=Install

6. VSimRTI can be started from an arbitrary directory <run-dir> by calling <vsimrti-allinone>/vsimrti/vsimrti <config-file>. If <run-dir> equals <vsimrti-allinone>/vsimrti, this would be the following command:

That's it

```
vsimrti ../scenarios/Schwanebeck-Default/vsimrti/vsimrti\_config.xml
```

Listing 1.1: VSimRTI start command.

Gather Results

To get results from a simulation, the combined simulators have to be properly configured or a federate has to be integrated that generates global results. See chapter Results for detailed information and visualization possibilities, e.g.the FileVisualizer for detailed simulation data or the eWorld VSimRTI Visualizer Plugin for online visualization.

Chapter 2

Configuration

2.1 VSimRTI Scenario Configuration

Since version 0.9.9 the VSimRTI configuration is scenario specific. Each sub directory in /scenarios represents one scenario. Each of them contains a directory for every used simulator e.g. one directory for the traffic simulator SUMO, one the network simulator SWANS etc. The following chapters will show how to configure those simulators. Nevertheless at first you should configure VSimRTI itself. Its scenario configuration can be found in the subdirectory "vsimrti" and should look like below.

```
<?xml version='1.0' encoding='UTF-8'?>

<configuration>

    <simulation>
        <id>Schwanebeck-Default</id>
        <starttime>0</starttime>
        <endtime>1000</endtime>
        <projection class="com.dcaiti.vsimrti.rti.objects.UniversalProjection
            ">
            <parameter type="java.lang.String">+proj=utm +zone=32 +ellps=
                bessel +units=m</parameter>
            <parameter type="double">-802951.38</parameter>
            <parameter type="double">-5832768.50</parameter>
        </projection>
        <threads>1</threads>
    </simulation>

    <federates>
        <-- Network simulator federates -->
        <federate id="swans" active="true" />
        <federate id="ns2" active="false" />

        <-- Traffic simulator federates -->
        <federate id="sumo" active="true" />
        <federate id="vissim" active="false" />

        <-- Application simulator federates -->
        <federate id="application" active="true" />

        <-- Environment simulator federates -->
        <federate id="eworld" active="true" />

        <-- Mapping federates -->
        <federate id="mapping" active="true" />
```

```
        <federate id="navigation" active="false" />

        <-- Visualization federates -->
        <federate id="eworld-vis" active="false" />
        <federate id="file-vis" active="false" />

    </federates>

</configuration>
```

Listing 2.1: VSimRTI Configuration (vsimrtiConfig.xml)

To configure the whole simulation, at first an identifier is necessary. This identifier is used to distinguish between different simulations running in parallel.
IMPORTANT: Your scenario identifier and the name of the /scenarios subdirectory have to match.
Furthermore, the time frame of the simulation has to be defined. The start-time (in seconds) is used to define the earliest time at which simulators are synchronized with each other. The end-time defines the latest valid simulation time. Finally, a projection is necessary, which projects geographic coordinates to Cartesian coordinates. The sub-tags of the projection tag are XML specifications of the input parameters for the constructor of a class implementing the projection interface. Included is a universal projection class, which takes several arguments to create a projection. These parameters are as follows:

Parameters	Description
int zone, double xOffset, double yOffset	creates a projection based on the given UTM zone with a given offset.
String projectionString, double xOffset, double yOffset	creates a projection based on the given projection string with a given offset. This string contains additional information for the used ellipsoid and projection parameters.
double centerLatitude, double centerLongitude, double xOffset, double yOffset	creates a projection based on the given center coordinates with a given offset. The zone is calculated automagically based on the center coordinates of the scenario.

Last but not least you can specify how many threads VSimRTI should use, we currently recommend to stay with one thread.
The last part of this configuration determines which simulators you want to use. If you want to use a special simulator just set its active parameter to "true". You can deactivate simulators by using "false".

2.2 Network simulators

2.2.1 SWANS

SWANS is the Scalable Wireless Ad hoc Network Simulator built atop the JiST

(Java In Simulation Time) platform.
It was originally released in 2004 from the Cornell University and is free for academic use. Since it is intented mainly for simulation of MANETs it comes along with the most important models for communication according to IEEE 802.11 on the one hand and exhibits a reasonable performance in large scale scenarios on the other hand. In the meantime several extensions from other departments came up where the VANET extension from University Ulm is most of interest. This version includes a new Routing Protocol for Geographic Routing, several bugfixes and enhanced mobility models and is the basis of V2X communication research connected with VSimRTI.
In the release 0.9.9 of VSimRTI, SWANS is extended with a new propagation model Three Log Distance Path Loss which is currently in experimental state.

The SWANS configuration file, presented in *Listing 2.2*, is located in the folder */scenarios/myScenario/swans* and contains settings of the used model parameters especially for the radio channel, the physical layer and the routing protocol on the network layer. It consists of one section for common parameters and two sections for node specific parameters to support different configurations for vehicles and RSUs.
Common

- Dimensions of the simulated area (NOTE: dimX and dimY must not be smaller than spatial expansions which are configured for the traffic simulator e.g. SUMO)
- Propagation models for path loss and fast fading
  - Path loss models Free Space, Two Ray Ground and Table do not require further parameters
  - Three Log Distance Model defines parameters d0, d1, d2 for reference distances and n0, n1, n2 for path loss exponents (NOTE: this model is still experimental)
  - Fast fading models None and Rayleigh Fading do not require further parameters
  - Fast fading model for Rice Fading defines the k-factor for the relation of the dominant LOS path over the equal distributed NLOS paths
- Common parameters for physical layer (NOTE: default values of SWANS assume the standard IEEE 802.11b, even if IEEE 802.11p is the reference standard for vehicular communication)
- Routing models
  - None means no routing is applied and V2X messages are sent using single-hop broadcasting
  - CGGC (Cached Greedy GeoCast) configures the georouting protocol implemented by Ulm University

Node Specific (Vehicle and RSU)

- Specific physical layer parameters for vehicles and RSUs (e.g. different antenna heights)

```
<configuration>

    <common>

        <--spatial expansion of simulated playground in m-->
        <dimX>12000</dimX>
        <dimY>22000</dimY>

        <pathloss>
            <model>freespace</model>        <--val: freespace, twoarayground,
                  table and threelogdistance-->
            <--models Free Space, Two Ray Ground and Table-driven do not
                  support further parameters-->
            <--Three Log Distance Path Loss Model specific parameters-->
            <d0>1.0</d0>                    <--unit: m, d0 SHOULD ALWAYS
                  EQUAL 1m-->
            <d1>200.0</d1>                  <--unit: m-->
            <d2>500.0</d2>                  <--unit: m-->
            <n0>1.9</n0>                    <--unit: 1-->
            <n1>3.8</n1>                    <--unit: 1-->
            <n2>3.8</n2>                    <--unit: 1-->
        </pathloss>

        <fastfading>
            <model>none</model>             <--val: none, rayleigh, rice-->
            <--models None and Rayleigh Fading do not support further
                  parameters-->
            <--Rice Fading specific parameters-->
            <kfactor>1</kfactor>            <--unit: dB-->
        </fastfading>

        <--common radio parameters-->
        <frequency>2400000000</frequency>   <--unit: hz-->
        <bandwidth>10000000</bandwidth>     <--unit: b/s-->
        <temperature>290</temperature>      <--unit: kelvin-->
        <tempfactor>10.0</tempfactor>       <--unit: 1-->
        <ambientnoise>0</ambientnoise>      <--unit: mw-->

        <-- network layer parameters -->
        <routing>
            <protocol>none</protocol>       <--val: none, cggc-->
        </routing>

    </common>

    <vehicle>

        <--vehicle specific radio parameters-->
        <antennaheight>1.5</antennaheight>  <--unit: m-->
        <txpower>15</txpower>               <--unit: dbm-->
        <txantennagain>0</txantennagain>    <--unit: dbm-->
        <rxsensitivity>-91</rxsensitivity>  <--unit: dbm-->
        <rxthreshold>-81</rxthreshold>      <--unit: dbm-->

    </vehicle>

    <rsu>

        <--rsu specific radio parameters-->
        <antennaheight>10.0</antennaheight> <--unit: m-->
        <txpower>15</txpower>               <--unit: dbm-->
        <txantennagain>0</txantennagain>    <--unit: dbm-->
```

```
        <rxsensitivity>-91</rxsensitivity>  <--unit: dbm-->
        <rxthreshold>-81</rxthreshold>      <--unit: dbm-->

    </rsu>

</configuration>
```

Listing 2.2: SWANS config file

2.3 Traffic simulators

2.3.1 SUMO

SUMO is the acronym for 'Simulation of Urban MObility' and the name of an open source microscopic, multi-modal traffic simulation package which is developed by the Institute of Transportation Research at the German Aerospace Centre. It is designed to handle large road networks faster than real-time. Each vehicle has an own route and is simulated individually.
The SUMO configuration is quite simple. At first make sure that the SUMO binary e.g. sumo.exe is part of your PATH variable. Afterwards just put your SUMO configuration files into your scenario subdirectory e.g. */scenario/myScenario/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.

2.3.2 VISSIM

The traffic simulator VISSIM can be used in conjunction with VSimRTI, too. To run VISSIM with VSimRTI, additional configuration and modification of VISSIM is necessary. For more information, please contact the VSimRTI team at vsimrti@fokus.fraunhofer.de .

2.4 Application simulator

The application simulator of VSimRTI is one of the most important simulators of VSimRTI. It makes it possible to create your own V2X application and simulate its effects. The configuration is situated in */scenarios/myScenario/application* . It contains the main configuration file *applications_ config.xml* and three subdirectories Rsu, TrafficLight and Vehicle. To deploy your developed application, you have to put it into the appropriate subdirectory for the simulated unit.

```
<configuration>
    <general>
        <stochastic>true</stochastic>
    </general>
    <vehicle>
        <cam>
            <mode>default</mode>
```

```
        </cam>
        <mapping>
            <application>
                <filename>weather-warning-app-1.0.1.jar</filename>
                <portion>80</portion>
            </application>
            <application>
                <filename>cam-example-app-1.0.jar</filename>
                <portion>20</portion>
            </application>
        </mapping>
    </vehicle>
    <rsu>
        <cam>
            <mode>user</mode>
            <interval>2</interval>
        </cam>
        <mapping></mapping>
    </rsu>
    <traffic-light>
        <cam>
            <mode>user</mode>
            <interval>0</interval>
        </cam>
        <mapping></mapping>
    </traffic-light>
</configuration>
```

Listing 2.3: Application simulator configuration

The first part of the configuration shown in *Listing 2.3* specifies the application mapping, that means how often which application is found on an equipped unit. In this example 80% of the equipped vehicles run the weather-warning application, the rest uses the cam-example application. If a detailed specification is not needed you can leave out the application definitions in the mapping section. This was done for the road side units and the traffic lights of this example. In this case every found application will also run on every equipped unit. The stochastic, defines how the mapping is done, if true it uses a random based algorithm to do so. If the value is false, it is deterministic, so that in every simulation run the same units will run the same applications.
It is also possible to configure the CAM sending behavior. Currently three different modes for timing requirements are supported.

- Default: calculates the CAM interval depending on the equation $tc = max(0.5s,$ *5s - vmax_ objects*
- *0.1s / (km/h))*
- User: configures a fix interval for CAMs, if the interval is set to 0 no CAMs are sent
- Etsi: generates CAM messages depending on the CAM generation rules specified in the ETSI proposal ETSI TS 102 637-2

In this example vehicles will use the default velocity depending sending rate. The sending rate for traffic lights and other RSUs were defined by the user, so RSUs will send a CAM every two seconds and traffic light CAMs have been turned off.

2.5 Navigation simulator

The navigation ambassador is a new way to decouple traffic simulation and vehicle navigation systems. This approach makes it possible to retrieve much more information of the current street map than the traffic simulators SUMO or VISSIM can provide. This is achieved by using a map model based on OpenStreetMap data, which is transformed before the simulation starts.

```
<?xml version='1.0' encoding='UTF-8'?>

<configuration>

<map>
    <filename>highway.db</filename>
    <maxpaths>101</maxpaths>
</map>

<starts>
    <position lat="50.35948" lon="8.93596" />
</starts>

<ends>
    <position lat="50.49226" lon="8.68490" />
</ends>

</configuration>
```

Listing 2.4: Navigation ambassador example configuration file

The structure of the navigation ambassador is quite simple. You have to specify the map file, which should be used. Keep in mind that this file must be created in advance out of a OSM file. Scripts are available, which perform this task, please contact the VSimRTI team for further information on this.
You can also specify how many paths should be generated. The next two parts are the starts and ends paragraphs. In this sections you can define multiple points where your vehicles will start and finish their route. It is recommend to do so, if not the navigation ambassador will generate a path for each combination of border nodes. A border node is a node at the outer most edge of the map. Be aware that, this task has an exponential complexity. Finally, you have to put the specified map file into the configuration folder e.g. /scenarios/myScenario/navigation .

2.6 Event server eWorld

The configuration file for eWorld Event Server is located in folder */scenarios/myScenario/eworld* . The following listing shows an example configuration that runs the eWorld Event Server based on a database:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
    <configuration>
        <database>
            <url>localhost</url>
            <port>5432</port>
            <dbName>eWorld</dbName>
```

```
            <username>user</username>
            <password>password</password>
        </database>
    </configuration>
```

Listing 2.5: eWorld Eventserver Configuration using Database

If you do not wish to use a database, but want the Event Server to load the map based on an ewd-file, please use the following configuration style. Afterwards just put the eWorld (.ewd) file in the configuration directory mentioned above. This configuration method is the typical way in which the eWorld event server is configured in connection with VSimRTI.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
    <configuration>
        <file>
            <name>schwanebeck.ewd</name>
            <path>.</path>
        </file>
    </configuration>
```

Listing 2.6: eWorld Eventserver Configuration using ewd-File

2.7 Mapping

In *Listing 2.7*, the configuration for the mapping component is outlined. The main option to be set is the ratio that defines the relationship between classic and application supported units. Furthermore, a flag is used to distinguish between deterministic (true) and stochastic (false) mapping. A deterministic mapping produces the same sequence of mapped vehicles in different simulation runs whereas a stochastic mapping results in a random order of mapped vehicles. The mapping of applications to vehicles, traffic lights or road side units can be configured in the application simulator. The values used to specify the different ratios are proportional, i.e. a distribution of (classic = 70, app-supported = 30) is equal to (classic = 7, app-supported = 3)

```
<configuration>
<-- A deterministic mapping (true) produces the same sequence of
     mapped vehicles in different simulation runs whereas a stochastic
     mapping (false) results in a random order of mapped vehicles. -->
<deterministic>true</deterministic>

<-- classic:         ratio of objects with no v2x equipment -->
<-- app-supported:   ratio of objects with full application support and
      forwarding capabilities, see Application Simulator for configuration of
       applications  -->
<-- Note: All values are proportional, i.e. a distribution of (classic = 70,
        app-supported = 30) is equal to (classic = 7, app-supported = 3) -->

<vehicle>
    <classic>70</classic>
    <app-supported>30</app-supported>
</vehicle>

<rsu>
    <classic>0</classic>
    <app-supported>100</app-supported>
```

```
    <rsu-positions>rsu positions configuration file</rsu-positions>
</rsu>

<trafficlight>
    <classic>0</classic>
    <app-supported>100</app-supported>
</trafficlight>
</configuration>
```

Listing 2.7: Mapping Configuration

The specified RSU configuration in the example scenario Schwanebeck rsu_positions.xml contains the GPS coordinates (latitude and longitude) of the RSUs. Its layout is depicted in the following *Listing 2.8*.

```
<configuration>

    <rsu-positions>
        <position lat="52.3243" lon="13.543">
    </rsu-positions>

</configuration>
```

Listing 2.8: RSU Position Configuration

2.8 Visualizer

VSimRTI provides several ways of data evaluation by so called visualizers. On the one hand the File Visualizer aims to collect and format data for further processing after simulation. Furthermore it is possible to visualize the results in eWorld while running a simulation with the eWorld VSimRTI Visualizer plugin.

The Visualizers are integrated into the simulation with a visualization ambassador. So the common visualization configuration (mentioned in Configuration) has to be adapted as follows:

```
<federate class="com.dcaiti.vsimrti.start.fed.VisualizationConfigLoader">
        <id>visualizer</id>
        <update>1</update>

    <file id="file-visu" update="10">
        <-- file visualizer's config -->
    </file>

    <eworld id="eworld-visu" enabled="false">
        <-- eworld visualizer's config -->
    </eworld>
</federate>
```

Listing 2.9: Specific Configuration for Visualizer

About visualizer

- Only two types of visualizer are currently supported: *file* and *eworld* .
- More than one visualizer of the same type can be defined.
- The attribute *id* of each visualizer should be globally unique, even different types of visualizer should have different ids. If more than one enabled visualizer have the same id, the first will be taken and the others will be ignored.

About update

The element *update* of visualization federate defines a default update interval for its visualizers.
However, some visualizers, such as eWorld-Visualizer, need a shorter update interval, while others don't want to update so frequently for the efficiency reason, e.g. it's much more efficient, if File-Visualizer visualizes many messages at one time, rather than one message each time.
Thus, the visualizer can overwrite its own update interval by using the attribute *update* . In the example, the "file-visu" has a update interval of 10.

About Default value

The following two configurations are equal:

```
<eworld id="eworld-visu">
<eworld id="eworld-visu" update="GLOBAL\_UPDATE" enabled="true">
```

Listing 2.10: Configurations with default values

2.8.1 File-Visualizer

The file visualizer is configured under the visualization federeate. The configuration is as follows:

```
<file id="file-visu" update="10">
    <filename>file.csv</filename>
    <directory>visualization</directory>
    <append>false</append>
    <-- The delimiter "," can be escaped by a backslash -->
    <separator>\,</separator>
    <messages>
        <message id="VehicleMovements">
            <entries>
                <-- entries config -->
            </entries>
        </message>

        <message id="AddedVehiclesMapping" enabled="false"/>
    </messages>
</file>
```

Listing 2.11: Specific Configuration for File Visualizer

About file

- The file for visualization has such a path: *logDirectory* /directory/filename. *logDirectory* is a property defined in etc/lockback.xml.
- The element *append* defines, whether the content will be appended to file.
- There is also an element *write* , which is not used in the example. Its value can be file or log, which means using normal file writer or logback to visualize the message.

About message record

- Each message record is derived from a certain message type and composed of several entries, which are separated by Element *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 *id* indicates the message type, namely the class name of the message.
- Message has also an attribute *enabled* .
- The element *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.

About default value

- *append* : true
- *write* : log
- *message.enabled* : true

About entry

Basically, there are totally three types of entry: constant, basic method and extended method. Just look at an example:

```
<message id="VehicleMovements">
    <entries>
        <entry>"MOVE\_VEHICLE"</entry>
        <entry>TimeInSec</entry>
        <entry>Updated:Id</entry>
        <entry>Updated:Type</entry>
        <entry>Updated:Position.Latitude</entry>
```

```
        <entry>Updated:Position.Longitude</entry>
    </entries>
</message>
```

Listing 2.12: 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.

```
<entry>"MOVE\_VEHICLE"</entry>
```

Listing 2.13: 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

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

Listing 2.14: 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 for each of the *VehicleInfo* objects in the returned list invoking *getId* method.
- Cascading

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

Listing 2.15: 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 are also some functionality, which we can't 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

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

Listing 2.16: An example for simple extended method type entry

With existing methods of *VehicleMovements* and its super class *Message* , we can't 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

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

Listing 2.17: 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.

more about Iteration

Basically, the method type of entry definition supports cascaded iteration as follows:

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

Listing 2.18: An example for cascaded iteration

What we haven't met yet, is that, if in the definition of entries, several different iterating operations exists, for example:

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

Listing 2.19: An example for multi-level iteration

getSenders() and *getReceivers()* are two different iterations. In this case, a product of Ids in both list will be generated. The result may look like this:

```
sender1,receiver1
sender1,receiver2
sender2,receiver1
sender2,receiver2
```

Listing 2.20: 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.

Extended Method Set

```
Message.getTimeInSec()           - long
Message.getTimeInMS()            - long
ReceiveV2Xmessage.getType()      - String
SendV2Xmessage.getType()         - String
VehicleInfo.getType()            - UnitType
VehicleInfo.getTypeInOrdinal()   - int
```

Listing 2.21: A list of all extended methods

Basic Method Set

```
VehicleMovements.getUpdated()    - List<VehicleInfo>
VehicleInfo.getId()              - String
VehicleInfo.getPosition()        - GlobalPosition
GlobalPosition.getLatitude()     - double
Message.getTime()                - long
VehicleInfo.getType()            - UnitType
```

Listing 2.22: A list of some basic methods

TODO: a reference to the method list of these classes are necessary for users.

2.8.2 eWorld-Visualizer

The eWorld visualizer is configured under the visualization federate. The configuration is as outlined in the following listing:

```
<eworld id="eworld-visu" enabled="false">
    <synchronized>true</synchronized>
    <host>local</host>
    <port>50500</port>
    <messages>
        <message id="VehicleMovements"/>
        <message id="ReceiveV2XMessage"/>
        <message id="SendV2XMessage"/>
```

```
        <message id="AddedVehiclesMapping"/>
    </messages>
</eworld>
```

Listing 2.23: eWorld Visualizer configuration

Using the configuration above this visualizer establishes a connection to the running eWorld Visualizer Plugin. More details about the results and the actual output can be found in the Results chapter, section eWorld VSimRTI Visualizer Plugin.

Chapter 3

Results

3.1 Logfile Structure

VSimRTI generates logfiles for each simulation run. Logs are generated for the ambassadors of each coupled federate respectively simulator and for VSimRTI itself.
VSimRTI uses logback as logging framework and it is suggested to use logback for the other simulators as well. The main configuration file for log output is the logback.xml file in the /etc folder. 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.
Logback offers a lot of parameters to adapt the output to your needs. Please refer to (http://logback.qos.ch/manual/layouts.html#ClassicPatternLayout) for a detailed overview of parameters you can use in the logback.xml file.
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.
In the next table, an overview about the possible logging output is given.

Logfile	Information
apps/application-<id>	Detailed application specific logs for each vehicle
Application.log	Information about the application ambassador
Communication.log	Network simulation ambassador log
CommunicationDetails.log	Detailed output of network simulation federate
Environment.log	Traffic simulation log
Mapping.log	Mapping configuration logs
Traffic.log	Traffic simulator logs
VSimRTI.log	General VSimRTI information as startup sequence
Statistics.csv	Simulation overview in comma separated value - format

Please note, that you can adjust the output to your needs by setting different loglevels (ERROR, INFO, DEBUG etc.) for each component. This might also influence the simulation performance.

Federate specific Logging

Depending on the simulation purpose, further configuration possibilities for federate specific logging may be of interest. Especially the opportunities of Application Simulator and JiST/SWANS have to be named, since they are delivered within the VSimRTI AllInOne package.
The Application Simulator also relies on logback and its configuration file is located directly in the */bin/fed/application* folder. The Weather Warning application for the default scenario Schwanebeck is quite small with one package and thus has a short configuration that contains only the root logger. More complex applications with several packages can be comprehensively configured according to specific needs. Again, please refer to the logback manual. Generally the pursued strategy in this logger should be to log on a certain stream (e.g. STDOUT) and the main logger configuration redirects these logs to the dedicated files.
The logging concept of JiST/SWANS is based on log4j and the log4j.properties file is located in the */bin/fed/swans* folder. This file already includes a more detailed configuration for different packages i.e. different communication layers of SWANS. Again the concept assumes logging on the console and redirection of the stream to the file as configured in the VSimRTI main logger.

3.2 Visualizer Integration

3.2.1 File Visualizer

Capabilities

The file visualizer offers the possibility to gather simulation results in a preformatted way for further processing. Currently it copes with the data of internal messages MOVE_VEHICLE, RECV_MESSAGE, SEND_MESSAGE, TRAFFICLIGHT_UPDATE and RSU_UPDATE:

MOVE_VEHICLE

MOVE_VEHICLE: Is created as part of a VehicleMovements message and represents one single vehicle movement.
MOVE_VEHICLE; Timestamp; VehicleId; VehicleType; Position (Latitude); Position (Longitude); Speed; Direction

Here the parameter Timestamp means time of occurence of the message. VehicleId is the number of the involved vehicle. VehicleType can be either 0 (for classic vehicles, which are not able to take part in V2X communication) and 2 (application supported). Parameters Position(Longitude), Position(Latitude), Speed and Direction imply GPS related information and should be self explaining so far. Note the units for these physical values: Longitude and Latitude are given in decimal degrees, Speed is in meters per seconds and the unit for Direction is angular degrees relative to north.

RECV_MESSAGE

RECV_MESSAGE: Represents the reception of a V2X message by a simulated vehicle application.
RECV_MESSAGE; Timestamp; MessageId; VehicleId; MessageType
Parameters have similar meanings as mentioned above. In this specific case VehicleId stands for the number of the currently receiving vehicle and the MessageId is a self incrementing number. The MessageType is given according to the ETSI defined types and can be either CAM oder DENM.

SEND_MESSAGE

SEND_MESSAGE: Represents a message that has been sent into the simulated V2X network.
SEND_MESSAGE; Timestamp; MessageId; SourceVehicleId; DestinationPosition (Latitude); DestinationPosition (Longitude); DestinationRadius; MessageType

TRAFFIC_LIGHT

TRAFFIC_LIGHT: Represents a message which indicates that a new traffic light was found.
TRAFFIC_LIGHT; Timestamp; TrafficLightInternalID; TrafficLightExternalID; Position (Latitude); Position (Longitude)

RSU_UPDATE

RSU_UPDATE: Represents a message which indicates that a new RSU was found.
RSU_UPDATE; Timestamp; ID; Type; Position (Latitude); Position (Longitude)

3.2.2 eWorld VSimRTI Visualizer Plugin

eWorld is an open source framework to import mapping data from providers, such as OpenStreetMap.org, visualize it, edit and enrich it with events or annotational attributes and pass it to traffic simulators, such as SUMO or VanetMobiSim.

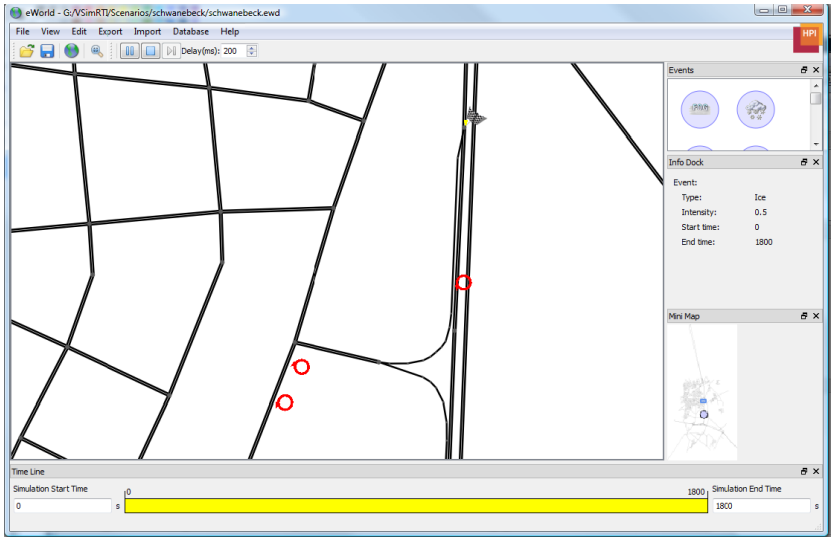

Figure 3.1: eWorld Visualization

The VSimRTI project team vitaly supports this project and has also developed a special visualization plugin for eWorld. By using this plugin you are able to view the movements and the network activity of simulated vehicles at runtime.

eWorld uses port 50500 by default. Synchronized is normally true, so each simulation step will be visualized. Otherwise VSimRTI will not wait for eWorld

to complete visualisation, therefore it will probably "stumble". These settings can be altered in the eWorld paragraph of the default configuration in etc.
After altering your configuration file start eWorld by using the eWorld.bat. Now open an eWorld file(extension *.ewd). In the default package we provide the schwanebeck.ewd. Of course the file used, must be compatible to the scenario you want to simulate. The loading process can take several minutes. When it is finished you can activate the "VSimRTI Visualizer" under "View". For now this plugin is only available internally and not included in the open source version of eWorld. After the plugin activation new buttons will appear, you can start the visualization by clicking the play button. In the new window you should select "Start listening for incoming signals", please make sure the port number is correct. eWorld is now ready and you can start VSimRTI.
In eWorld equipped vehicles a represented with a red, unequipped with a green and classic ones with a yellow color (see *Figure 3.1*). Vehicles, which are sending a message are marked with a red circle around them, while vehicles which receive a message get a gray circle.

Chapter 4

Deploy a new Scenario

Deploying a new scenario is a rather comprehensive term. A new scenario in terms of including a new application, network configuration etc. would require a detailed knowledge of VSimRTI and may include new implementations. But here new scenario should mean that the same configuration set of simulators (SUMO, JiST/SWANS, eWorld) and application (weather warning) is used with another geographic area to be simulated.
Therefore new SUMO files are needed, which can be amongst others produced with eWorld (see documentation [1]).
For VSimRTI the following adaptions to configuration.xml for SUMO and JiST/SWANS are important. When opening the new SUMO **.net.xml* file you should see the location and boundary parameters for the new area at the beginning of the <net> part, like this:

```
<net>
   <net-offset>-802951.38,-5832768.50</net-offset>
   <conv-boundary>0.00,0.00,11364.75,21763.50</conv-boundary>
   <orig-boundary>802951.38,5832768.50,814316.13,5854532.00</orig-boundary>
   <orig-proj></orig-proj>
   ...
</net>
```

Listing 4.1: Net part of the SUMO Configuration File

Copy the net-offset values to the overall simulation parameter section of the vsimrti_config.xml file (can be found in */scenarios/myScenario/vsimrti*) of your VSimRTI simulation:

```
<simulation>
  <id>Schwanebeck-Default</id>
  ...
  <projection class="com.dcaiti.vsimrti.rti.objects.UniversalProjection">
    <parameter type="java.lang.String">+proj=utm +zone=32 +ellps=bessel +
          units=m</parameter>
    <parameter type="double">-802951.38</parameter>
    <parameter type="double">-5832768.50</parameter>
  </projection>
  ...
```

[1] http://sourceforge.net/project/showfiles.php?group_id=212589&package_id=269840

```
</simulation>
```

Listing 4.2: Adapted Overall Simulation Parameter Part of the VSimRTI Config-File

Copy the geographic extent (last two values of the <conv-boundary> tag) into the <dimX> and <dimY> tags of the SWANS network simulator federate configuration in the swans_config.xml (found in */scenarios/myScenario/swans*). NOTE that larger values for the spatial expansion of the SWANS simulation area are also possible, but smaller values lead to wrong behavior and will cause exceptions:

```
<configuration>
    <common>

        <--spatial expansion of simulated playground in m-->
        <dimX>12000</dimX>
        <dimY>22000</dimY>
        ...
     </common>
     ...
</configuration>
```

Listing 4.3: Adapted common part of the Swans Config-File

Chapter 5

Develop your own application

The application simulator was developed to improve the overall simulation performance and on the other hand implement the ETSI standards for Car to Car communication. These standards contain message definitions like DENM and CAM but also a general application container design. The application simulator follows these proposals and implements a multi-layered application container with own application and facility layer and interfaces to the network layer.
To develop your own application you have to know three main concepts of the simulator. At first the interfaces which every application has to implement. Second the interfaces you can use to retrieve information or send messages. Last but not least you probably want to know how to change the content of automatically sent cooperative awareness messages.

The Application Interface

The Application Simulator builds the frame for the application and is responsible for the needs of the application e.g. time management, starting and initializing the application, etc. The application simulator calls three methods of your application, these methods must be implemented, even when they are empty.
These three methods are:

- timercall method - *timerCall()*

- initialize method - *initialize()*

- receivemessage method - *receiveV2XMessage()*

initialize method:

The initialize method is the one called first after starting a simulation. It is only called once, so it should be made sure, that all objects, which are needed in the application are referenced, e.g. reference to time component, reference to state manager, etc. It can be also used to implement an algorithm, which is used only one time.

timer call method:

The timer call method allows to get a periodical call of this method by using a call interval, which is set by the user. If one has an algorithm which needs a periodical call, the method call should be implemented in this function.

receive message method:

If an application is developed, which uses V2X messages, this method is an interesting one. It only handles V2X messages in the V2X format (CAM and DENM messages). The algorithm to handle a received CAM or DENM message has to be implemented by the developer e.g. source code of the VSimRTI example application (Weather warning).

Vehicle control, information retrieval and more

The simulator supports three different application types for RSU, traffic lights and of course vehicles. For each of them a special ComponentFactory e.g. VehicleComponentFactory is available. By calling it your application is now able to retrieve references of either a Timer, VehicleControl, CommunicationModule or (Vehicle-)StateManagement implementation. Each implementation provides different possibilities, the following lists will provide a short overview about the interface implementations for vehicles. For more information have a look at the doxygen documentation.
Timer:
This implementation is equal for every unit type

- registerApplication
- changeCallInterval
- removeApplication

VehicleControl:

- setMaximumSpeed

- slowDown
- changeRoute

CommunicationModule:
This implementation is equal for every unit type

- sendUDPMessageTopo
- sendUDPMessageGeo
- setUserTaggedValue

VehicleStateManagement:
For each unit type exists a corresponding StateManagement implementation, which are extensions of the BasicStateManagementImpl.

- getRoute
- getRoad
- getSpeed
- getHeading

BasicStateManagement:

- getSystemState
- getNodeType
- getLongitude
- getLatitude
- getLongTimeStamp
- getUnitId
- getCurrentTime
- getStateOfEnvironmentSensor

In general we also recommend to have a closer look at the example applications like the "cam-example-app" or "weather-warning-app".

CAM - Content

The following section will show how cooperative awareness messages are implemented in VSimRTI and how you can change them. The messages are divided into four parts:

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

First of all generic information like protocol version, creation time stamp are submitted. Normally this data set follows a network beacon, which already contains 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 probably changes with a new CAM specification. In the second case, we provide date like vehicle width, length, position, speed, type and heading. However, the specification is not completely implemented, especially acceleration data and light, 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 contains a tagged list, a key value map with optional data. This is fully implemented and is used for our traffic light CAM messages, which provide the traffic light status in such a list.
The CAM sending interval can be configured, more information can be found in the configuration section of the application simulator.

User defined tagged values

Despite of the already defined tagged values, it is also possible to create an own one. This can be done by implementing the UserTaggedValue interface. There are two ways of doing so.
First of all it can be done by implementing the interface methods valueToBytes and bytesToTaggedValue. The focus of these methods is the conversation from a tagged value object to bytes and vice versa. In this way every byte sent via CAM can be controlled, nevertheless this is often connected with some serious work.
If the control of every byte is not needed, the default implementation (TaggedValueDefaultImpl) of the UserTaggedValue interface can be used. The main advantage of this approach is the easy way to handle the marshalling: it is all already done. So the developer can focus on the main facts.
After the user defined tagged value was added, it will be sent with every next

CAM. The following *Listing 5.1* shows how to retrieve a user defined tagged value from a received CAM. For more information have a closer look at the example applications.

```
UserTaggedValue value = cam.getUserTagValue(new MyTaggedValue());
if(value=null){
    MyTaggedValue myValue = (MyTaggedValue)value;
    //now you can access the tagged value
}
```

Listing 5.1: Access received tagged value

Chapter 6

Further Information

Known Issues

None

6.1 Federate-specific Configuration

This chapter provides an overview on existing federates for VSimRTI and their specification. It covers details on the configuration of the specific federate as well as information about the internal communication model (sent and received messages) and the semantics of the federate. This configuration can be changed in the file */etc/defaults.xml* . 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.

JiST/SWANS

This federate integrates the network simulator JiST/SWANS
Package: *com.dcaiti.vsimrti.fed.swans.ambassador*

Messages

Message Type	Sent	Received	Comment
ReceiveV2XMessage	X		Notifies the affected vehicle and delivers the payload message when a receive-event for that simulated node happens in the simulated network
SendV2XMessage		X	Emits a new V2X message according to the applications implementation
VehicleMovements		X	Used to update the positions of the simulated network nodes
AddedVehicleMapping		X	Used to add/remove new vehicles as network nodes
RsuUpdate		X	Used to add/remove new RSUs as network nodes

Configuration

In *Listing 6.1*, a configuration for the network simulator JiST/SWANS is outlined. It starts with the necessary common parameters id, deploy, start, host, srcDir, and messages. The JiST/SWANS extension to synchronize a server requires that a port is set on which the scheduler is listening. If set to 0, the smallest free port is used. Finally, JiST/SWANS subscribes VehicleMovements and AddedVehiclesMapping messages to update positions and SendV2XMessage to insert sent messages into the communication network.

```
<federate class="com.dcaiti.vsimrti.start.fed.SwansConfigLoader">
    <-- Common parameters -->
    <config>swans config filename</config>
    <port>listening port</port>
    <messages>
        <message>VehicleMovements</message>
        <message>AddedVehiclesMapping</message>
        <message>SendV2XMessage</message>
        <message>RsuUpdate</message>
    </messages>
</federate>
```

Listing 6.1: JiST/SWANS Configuration

SUMO

This federate integrates the open source traffic simulator SUMO.
Package: *com.dcaiti.vsimrti.fed.sumo.ambassador*

Messages

Message Type	Sent	Received	Comment
VehicleMovements	X		Is sent in the defined update-interval and notifies listeners about added, updated and removed vehicles in the simulation
ChangeRoute		X	Causes Sumo to dynamically re-route the affected vehicle.
SlowDown		X	Sets the vehicles current speed to the given value
SetMaxSpeed		X	Sets the vehicle max speed to the given value

Configuration

In *Listing 6.2*, a configuration for the traffic simulator SUMO is outlined. It starts with the necessary common parameters id, deploy, start, host and messages. Furthermore, the listening port, as well as a configuration for sumo (tag=config) is needed. SUMO broadcasts VehicleMovements in fixed time steps. Using the update tag, a user is able to define the length of these time steps. For dynamic change of vehicle behavior during simulation, SUMO subscribes to the messages SetMaxSpeed, ChangeRoute and SlowDown.

```
<-- Common parameters -->
<federate class="com.dcaiti.vsimrti.start.fed.SumoConfigLoader">
    <port>listening port</port>
    <update>update interval in ms</update>
    <config>sumo configuration file ending (*.cfg)</config>
    <messages>
        <message>SetMaxSpeed</message>
        <message>ChangeRoute</message>
        <message>SlowDown</message>
    </messages>
</federate>
```

Listing 6.2: SUMO Configuration

eWorld

The eworld federate allows the integration of the eWorldEventServer which is part of the open source map data editor eWorld [eworld.sourceforge.net]. It is

capable of handling vehicle movement messages and create sensor events according to environmental data.

Messages

Message Type	Sent	Received	Comment
SensorRegistration		X	New vehicles sent this to register for environment events
VehicleMovements		X	Triggers the check if new positions are affected of environmental events
SensorData	X		Informs any vehicle that is affected of an environmental event that it has new SensorData available

Configuration

In *Listing 6.3*, a configuration for the environment generator eWorld is outlined. Additional to the necessary common parameters id, deploy, start, host, srcDir, and messages; eWorld requires the definition of the port on which the event server is listening. If eWorld is to be deployed and started by VSimRTI, furthermore the name of the jar containing the event server and its location (srcDir) are needed. In the given example (*Listing 6.3*) eWorld is registered for SensorRegistration and VehicleMovement messages.

```
<federate class="com.dcaiti.vsimrti.start.fed.EWorldServerConfigLoader">
    <-- Common parameters -->
    <jar>eWorld Eventserver jar</jar>
    <config>eworld configuration file</config>
    <port>listening port</port>
    <messages>
        <message>SensorRegistration</message>
        <message>VehicleMovements</message>
    </messages>
</federate>
```

Listing 6.3: Configuration of eWorld

Visualization

The visualization ambassador provides several ways to visualize and/or trace the activites that happen during the simulation. The here explained visualization

federate is responsible for providing data (in form of subscribed messages) to the visualization component. And the kind of visualization is completely dependent on the component itself. In chapter Results the two visualization components, currently included in VSimRTI are presented. These are the eWorld Visualizer Plugin and the FileVisualizer.
Package: *com.dcaiti.vsimrti.fed.visual*

Messages

Message Type	Sent	Received	Comment
ReceiveV2XMessage		X	Used to visualize network communication
SendV2XMessage		X	Used to visualize network communication
VehicleMovements		X	Used to traffic simulation
AddedVehicleMapping		X	Used to update mapping configuration

Configuration

In *Listing 6.4*, a configuration a visualization component is outlined. It starts with the necessary common parameters id, deploy, start, host, srcDir, and messages . Afterwards, the type of the visualization is set. Based on the type a further tag is necessary. See chapter Results for detailed configuration of current visualizer types *"eWorld"* and *"file"* . Additionally, the update tag is to use define how often the logged data should be sent to the visualization component. Finally, a visualization federate subscribes the messages that are to be logged. At the moment, ReceivedV2XMessage, SendV2XMessage and VehicleMovements messages are possible. The AddedVehicleMapping message is used by the visualization ambassador itself for state keeping of current mapping of vehicles in simulation.

```
<federate class="com.dcaiti.vsimrti.start.fed.VisualizationConfigLoader">
    <-- Common parameters -->
    <type>visualization type</type>
    <additional tag>see defined additional tags</additional tag>
    <update>update interval</update>
    <messages>
        <message>VehicleMovements</message>
        <message>ReceiveV2XMessage</message>
        <message>SendV2XMessage</message>
        <message>AddedVehiclesMapping</message>
    </messages>
</federate>
```

Listing 6.4: Visualization Configuration

VSimRTI_App

The VSimRTI Application simulator uses techniques based on the ETSI-ITS standard. Further details can be found in chapter Application Simulator.
Package: *com.dcaiti.vsimrti.fed.app*

Messages

Message Type	Sent	Received	Comment
ReceiveV2XMessage		X	Delivers the payload of the V2X Message to the underlying simulated application
SensorData		X	Notifies the underlying application that new environmental data has become available
VehicleUpdate		X	Updates the local vehicle information (e.g. position, speed, etc.)
TrafficLightUpdate		X	Updates the local traffic light information (e.g. position, program, etc.)
RsuUpdate		X	Updates the local RSU information (e.g. position)
VehicleNavigationUpdate		X	Updates the local Navigation information, like street, road length etc.
SendV2XMessage	X		Emits a new V2X message according to the applications implementation
ChangeRoute	X		
SlowDown	X		
SetMaxSpeed	X		

Configuration

The configuration of Application simulator has the following structure, outlined in *Listing 6.5.*

```
<federate class="com.dcaiti.vsimrti.start.fed.ApplicationConfigLoader">
    <port>listening port</port>
    <config>application simulator configuration file</config>
    <messages>
        <message>VehicleMovements</message>
        <message>ReceiveV2XMessage</message>
        <message>SensorData</message>
        <message>VehicleUpdate</message>
        <message>AddedVehiclesMapping</message>
        <message>TrafficLightUpdate</message>
        <message>RsuUpdate</message>
        <message>VehicleNavigationUpdate</message>
    </messages>
</federate>
```

Listing 6.5: Configuration for the Application Simulator VSimRTI-App

Mapping

The mapping federate is responsible to create instances of simulated applications for a given percentage of the total simulated units. It handles RSUs, vehicles and traffic-lights.
Package: *com.dcaiti.vsimrti.fed.map*

Messages

Message Type	Sent	Received	Comment
VehicleMovements		X	Is used to extract the added/removed information from and to apply the configured mapping accordingly.
AddedTrafficLight		X	Is used to extract the added/removed information from and to apply the configured mapping accordingly.
AddedVehicleMapping	X		Updates listeners about the new mapping configuration.
VehicleUpdate	X		Sent to each simulated vehicle federate in order to enable it to update its local vehicle information.
TrafficLightUpdate	X		Sent to the application simulator in order to enable its local traffic light information.
RsuUpdate	X		Sent to the application simulator in order to enable it to update its local rsu information.

Configuration

The general configuration of the mapping ambassador has no special attributes, the detailed configuration of the mapping can be found in the mapping configuration file.

```
<federate class="com.dcaiti.vsimrti.start.fed.MappingConfigLoader">
    <config>mapping configuration filename</config>
    <messages>
        <message>VehicleMovements</message>
        <message>AddedTrafficLight</message>
    </messages>
</federate>
```

Listing 6.6: Configuration for the Mapping Ambassador

Navigation

Te navigation ambassador is a new way to decouple traffic simulation and vehicle navigation systems. This approach makes it possible to retrieve much more information of the current street map than Sumo or Vissim provide. This is

achieved by using a map model based on OpenStreetMap data, which is transformed before the simulation starts.
Package: *com.dcaiti.vsimrti.fed.navigation*

Messages

Message Type	Sent	Received	Comment
VehicleMovements		X	Is used to react on every movement by sending a NavigationUpdate
AddedVehicleMapping		X	Used to distinguish between virtual and classic vehicles
ChangeRoute		X	Informs the navigation ambassador about edge weight/traveltime changes. In addition to that a new route will be calculated
VehiclePathInitMessage	X		This message is emitted, when the ambassador completed its path generation. The message contains all generated path and its ids.
VehicleNavigationUpdate	X		Sent to the application simulator to inform it about the new route, street length, max speed etc.
ChangeStaticRoute	X		Used to inform a traffic simulator e.g. Vissim about a new route. This route was pre-calculated in advance, therefore only the new path and vehicle id is supplied.

Configuration

In *Listing 6.7*, the default configuration section of the navigation ambassador is shown. It consists of the ordered messages and a reference to the navigation configuration.

```
<federate class="com.dcaiti.vsimrti.start.fed.NavigationConfigLoader"
    >
    <id>navigation</id>
    <config>navigation configuration file</config>
    <messages>
        <message>AddedVehiclesMapping</message>
        <message>VehicleMovements</message>
```

```
            <message>ChangeRoute</message>
        </messages>
    </federate>
```

Listing 6.7: Navigation ambassador default configuration

6.2 Host Configuration

Normally it is not needed to change this file unless you want to run your simulation on different machines at the same time.
In VSimRTI, there are two kinds of hosts: local and remote hosts. All host have to be identified with a unique string that is valid for the rest of the configuration to reference a here 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 assigned according to operating system specific notation (e.g. */vsimrti/temp* for Linux and *C:|vsimrti|temp* for Microsoft Windows). Note also relative paths are possible. To use remote hosts, SSH and SFTP is used to deploy and start federates. Therefore, address of the host as well as port, user name, and password for an SSH connection must additionally be given.

```
<?xml version='1.0' encoding='UTF-8'?>
<configuration>

<hosts>
    <host local="true">
        <id>local</id>
        <root>./tmp</root>
        <os>linux</os>
    </host>

       <host>
         <-- ... -->
       </host>
</hosts>

</configuration>
```

Listing 6.8: Host List(hosts.xml)

References

[1] Thomas Benz, Björn Schünemann, Ralf Kernchen, Moritz Killat, and Andreas Richter. A comprehensive simulation tool set for cooperative systems. *Advanced Microsystems for Automotive Applications 2010 : Smart Systems for Green Cars and Safe Mobility*, pages 411–422, May 2010.

[2] Nico Naumann, Björn Schünemann, and Ilja Radusch. Vsimrti - simulation runtime infrastructure for v2x communication scenarios. In *Proceedings of the 16th World Congress and Exhibition on Intelligent Transport Systems and Services (ITS Stockholm 2009)*. ITS Stockholm 2009, September 2009.

[3] Nico Naumann, Björn Schünemann, Ilja Radusch, and Christoph Meinel. Improving v2x simulation performance with optimistic synchronization. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 52–57, Dec. 2009.

[4] Robert Protzmann, Björn Schünemann, and Ilja Radusch. The influences of communication models on the simulated effectiveness of v2x applications. In *IEEE Vehicular Networking Conference (VNC 2010)*, Jersey City, NJ, USA, Dec. 2010. To appear.

[5] Tobias Queck, Bjöern Schünemann, and Ilja Radusch. Runtime infrastructure for simulating vehicle-2-x communication scenarios. In *VANET '08: Proceedings of the fifth ACM international workshop on VehiculAr InterNETworking*, pages 78–79, New York, NY, USA, 2008. ACM.

[6] Tobias Queck, Björn Schünemann, Ilja Radusch, and Christoph Meinel. Realistic simulation of v2x communication scenarios. In *APSCC '08: Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference*, pages 1623–1627, Washington, DC, USA, 2008. IEEE Computer Society.

[7] David Rieck, Björn Schünemann, Ilja Radusch, and Christoph Meinel. Efficient traffic simulator coupling in a distributed v2x simulation environment. In *SIMUTools '10: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, pages 1–9, ICST, Brussels, Belgium, Belgium, 2010. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[8] Björn Schünemann, Kay Massow, and Ilja Radusch. A novel approach for realistic emulation of vehicle-2-x communication applications. In *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, pages 2709–2713, May 2008.

[9] Björn Schünemann, Kay Massow, and Ilja Radusch. Realistic simulation of vehicular communication and vehicle-2-x applications. In *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–9, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[10] Björn Schünemann, Jan W. Wedel, and Ilja Radusch. V2x-based traffic congestion recognition and avoidance. *Tamkang Journal of Science and Engineering*, 13(1):63–70, March 2010.

[11] Jan W. Wedel, Björn Schünemann, and Ilja Radusch. V2x-based traffic congestion recognition and avoidance. *Parallel Architectures, Algorithms, and Networks, International Symposium on*, 0:637–641, 2009.