Small or medium-scale focused research project (STREP)

ICT Call 8

FP7-ICT-2011-8

Cooperative Self-Organizing System for low Carbon Mobility at low Penetration Rates

COLOMBO: Deliverable 5.1

Prototype of overall System Architecture and Definition of Interfaces

Document Information	
Title	Deliverable 5.1 - Prototype of overall System Architecture and Definition of Interfaces
Dissemination Level	PU (Public)
Version	2.2
Date	14.02.2014
Status	Updated Version after Annual Review (final)

Authors	Paolo Bellavista (UNIBO), Anna-Chiara Bellini (UNIBO), Laura Bieker (DLR), Robbin Blokpoel (PEEK), Jérémie Dubois-Lacoste (ULB), Luca Foschini (UNIBO), Nikolaus Furian (TUG), Stefan Hausberger (TUG), Jérôme Härri (EURECOM), Cornelia Hebenstreit (TUG), Marek Heinrich (DLR), Daniel Krajzewicz, (DLR), Michela Milano (UNIBO), Thomas Stützle (ULB)

Table of contents

1 Introduction 5
1.1 Motivation 5
1.2 Objectives 5
1.3 Structure 6
2 Overall System Concept 7
2.1 Tasks 7
2.2 State-of-the-Art 8
2.3 COLOMBO Prerequisites 9
2.4 Architecture Overview 11
2.5 Functionality and Usage Overview 11
3 COLOMBO Components 14
3.1 ns-3 14
3.2 Automatic Configuration Tool kit 17
3.3 PHEM 20
3.4 SUMO 25
3.5 Application Simulator(s) 29
3.6 iCS - iTETRIS Control System 33
4 COLOMBO Applications 39
4.1 Cooperative Traffic State Evaluation System 39
4.2 Intersection-based Emissions Monitoring System 46
4.3 Traffic Light Algorithms 48
4.4 Emission-optimal Driver Behaviour for Controlled Intersections 54
5 System Architecture Prototype 57
5.1 Software Development in COLOMBO 57
5.2 Components Status 59
5.3 Components Interaction in the Prototype 60
5.4 Needed Extensions 61
6 Abstract usage examples of the overall system 65
6.1 Coupling SUMO, a traffic light algorithm and the tuning tool kit 65
6.2 Coupling SUMO, algorithms and the tuner extended by PHEMlight's emission module.. 67
6.3 Coupling SUMO and ns-3, iCS to enable an example Application 68
7 Summary 71
Appendix A – References 72

1 Introduction

The goal of the COLOMBO project is to deliver a set of modern traffic management applications, all based on information gained from wireless communication performed between traffic participants and between traffic participants and infrastructure. Field trials allow proving the technical functionality of such applications or systems. But their effect on the traffic system, especially under changing system characteristics, such as the fraction of participants equipped with communication technology, is very expensive. For this reason and because the deployment of new technologies may raise safety issues, new technical solutions are usually evaluated first by using simulations. Simulations allow to predict the performance of a new technical system in-vitro at relatively low costs without harming any living creature.

The COLOMBO project develops applications only relying on simulations. If these applications prove to bring benefits to the traffic system within the simulation, further steps will get necessary to realize a real-world implementation of these systems.

The applications developed in COLOMBO use information obtained from wireless communication to gain information about the state of traffic, to give traffic participants further information, to control them via traffic lights, and to advise them to perform certain actions for being more environment-friendly. The simulation of such systems’ behaviour requires tools that resemble traffic as well as communication, and that allow to include the applications under development into this simulation system. Targeting environment-friendly solutions, proper models for emission computations are additionally needed. Finally, one of COLOMBO’s goals is to generate solutions that are automatically calibrating themselves to the environment they are meant to be deployed within.

These requirements show that a complex simulation system is needed, which combines models from different knowledge domains. COLOMBO’s work package WP5 is responsible for delivering such a system and this document describes its first realization, including detailed requirements as well as a presentation of the simulation system itself.

1.1 Motivation

COLOMBO is a multi-disciplinary project involving experts in traffic simulation, traffic control, emissions modelling, swarm intelligence, telecommunication, and automatic algorithm configuration and optimization. The major motivation of the COLOMBO work package 5 is to transfer and to join the knowledge located at these partners into a simulation system that allows to evaluate traffic management solutions that are developed in later project steps. It is hardly possible to consolidate the already available methods and tools into a single, completely new tool within the project’s life time. Instead, COLOMBO relies on already available solutions supported by the project partners. Work package 5 is dedicated to extending and opening these applications, named “components” or “COLOMBO components” in the following, in a way that allows them to interact for having a system that covers the needs posed by the developed applications.

The motivation behind this deliverable is to have a common view on the technical interfaces between the involved topics which the components are responsible for – communication simulation, traffic simulation, traffic light control, configuration and tuning, emissions modelling – and on how to make them work together.

1.2 Objectives

This document shall present the design and the current state of the simulation system used in COLOMBO. It shall show that the system matches the requirements given by the applications that shall be developed within the project. Where needed, extensions to the overall system and its components shall be shown.

1.3 Structure

The simulation system was already described within COLOMBO’s description of work [COLOMBO, 2012] and it is an extension of an already existing system for simulating applications based on vehicular communication. For this reason, no software development approach is followed. The system is presented and described as following: Chapter 2 outlines the tasks of the system, explains the reasons for choosing its components, and outlines its application from a user perspective. Chapter 3 gives detailed descriptions of the components, mainly targeting on revealing their interaction with the user, other components, and the developed applications by describing the needed inputs, outputs, their usage modes, and their interfaces. Chapter 4 focuses on the applications that are developed within COLOMBO, describing their dependencies to the simulation system component-wise. Chapter 5 describes the system’s prototype as-is, as well as its needed extensions concluding with a selection of basic usage examples.

2 Overall System Concept

This chapter introduces the COLOMBO architecture at a coarse level, describes its role within the project and explaining why this architecture and the specific components have been chosen. All involved applications are described in more detail in chapter 4.

2.1 Tasks

COLOMBO's major objective is to deliver a set of concepts for solutions for traffic surveillance and control. These solutions rely on information gained via wireless communication channels. Usually, functions based on exchanging messages between traffic participants or between traffic participants and the infrastructure are named "V2X-applications", where "V2X" usually denotes the exchange of messages between vehicles and the infrastructure or other vehicles using the 802.11p communication standard. COLOMBO extends the number of participant types that communicate by additionally including pedestrians and bicyclists. Additionally, other communication channels, such as WiFi-direct will be investigated in COLOMBO as well. Nonetheless, the term "V2X-applications" will be continued to be used.

A large set of such V2X-applications was standardized by the European Telecommunications Standards Institute (ETSI) as the "Basic Set of Applications" [ETSI, 2009], albeit most of them target at increasing traffic safety while COLOMBO focuses on traffic management solutions that increase traffic efficiency and decrease the amount of pollutants emitted by traffic.

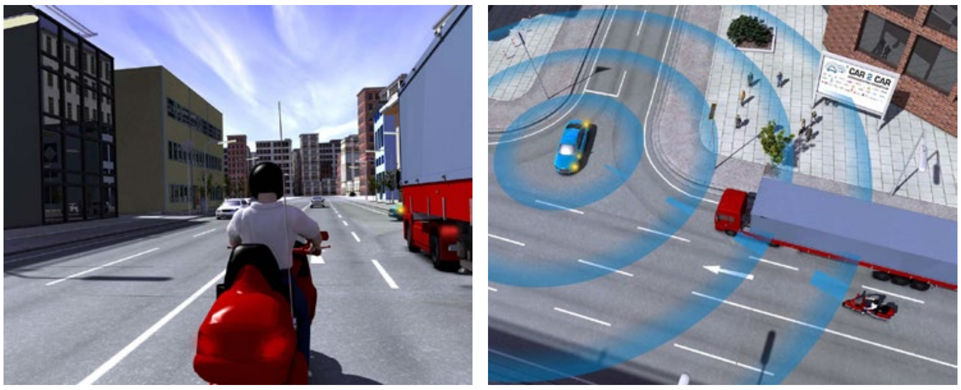


Figure 2.1: Example of a V2X-application for collision avoidance. The motorbike driver does not see the approaching blue vehicle, but the application is aware of the dangerous situation and warns him (source: Car 2 Car Communication Consortium web site)

When deployed, the real world scenario would involve traffic participants that are equipped with wireless communication hardware used to send and receive information while travelling through the road network. In addition, communication hardware (road side units, RSUs) may also be deployed stationary at the road, allowing to use the collected information for traffic management applications or to improve the dissemination of information by rebroadcasting it or passing it to a different place using wired communication. A V2X-application can use this information to trigger certain actions if a certain event is recognized or directly reported. Examples may be to warn the driver if a vehicle in front is braking hard, to control traffic lights by preferring the highest in-flow, or to warn about the approach of an emergency vehicle, see also Figure 2.1.

The use of wireless communication in order to disclose individual traffic relevant information creates a bi-directional interdependency: The positions and behaviour of traffic participants build the content of sent messages while received messages may have an effect on the participants' future behaviour.

COLOMBO relies on simulations for the development and the evaluation of V2X-applications. Such simulations need to resemble the movement of participants through the road network as well as their communication. Due to the aforementioned bi-directional influence both has to be simulated simultaneously and in an interacting way. Earlier investigations of vehicular communications used traffic simulations to obtain "trajectories" (positions and optionally other values over time) first, which were then given to a communication simulator. Such simulation set-ups allow to investigate the performance of communication, but fail in simulating V2X-applications, as their effects on traffic cannot be put back into the simulated traffic (see also [Sommer et al., 2008]).

Two further objectives shape the development of COLOMBO's V2X-applications adding further requirements for the used simulation system: the strong emphasis on developing solutions that a) reduce the amount of pollutants and emissions and b) are configured automatically.

Ecological measures can only be obtained using proper models for vehicular emissions. If the developed application has to react on the amount of emitted pollutants, these models have to be included into the simulation loop. Tools for automatic configuration / optimization use a given computer program and execute it in a loop. The behaviour of this program is controlled by a set of parameter values, computed by the configurator / optimizer and the program must return a performance measure. In the case of COLOMBO, the configurator / optimizer must be able to interact this in way with the complete V2X-application simulation system like the ones described above.

2.2 State-of-the-Art

Different software packages for the simulation of V2X-applications were developed, mainly in the past decade. They differ in license, availability, the chosen architecture, and the (re-)used traffic and communication models and/or simulators. Table 2.1 lists some of the best known packages for simulating V2X communication.

Table 2.1: Some chosen examples of V2X communication packages

Coupling software	Communication simulation	Traffic simulation	Software category	Open source	Reference
TraNS	• ns2	• SUMO	socket connection	+	[Piorkowski et al., 2008]
Veins	• OMNet++/MiXiM • INET • INETMANET • JiST/SWANS	• SUMO	socket connection	+	[Veins, 2013]
iCS	• ns-3	• SUMO	middleware	+	[iTETRIS, 2013]
VSimRTI	• ns-3 • JiST/SWANS • OMNeT++ • VSimRTI_cell	• SUMO • VISSIM	abstract simulation coupling	-	[VSimRTI, 2013]
MSIECV	• ns2	• VISSIM	socket connection	-	[Schünemann, 2011]

The architecture of these simulation packages can be coarsely classified as following:

- monolithic applications that include models for both, traffic and communication ([Gorgorin et al., 2006], [Krajzewicz et al., 2008]);
- direct socket connections between a communication simulator and a traffic simulator;
- middleware solutions that connect a communication simulator, a traffic simulator, and usually also an application simulator;
- abstract simulation coupling architectures

The main advantage of monolithic applications is that the whole V2X communication can be simulated by only one software tool and no other tools have to be installed and used. On the other hand, simulating traffic as well as the communication propagation are both very complex topics and are not trivial to simulate [Schünemann, 2011].

For the purpose of COLOMBO the following simulations have to be coupled, therefore a brief overview of the existing simulation software is given in the following. Mentioning all of the existing simulation packages would exceed the scope of this document, so only an incomplete list is presented here:

- **Traffic simulation:** The implemented models and the focus of the functions vary a lot. A comparison of 17 traffic simulation programs gives an overview over the different functionalities in [Pell, 2013]. In the research field of V2X these commercial simulators are well known e.g. PELOPS ([PELOPS, 2013]), Quadstone Paramics ([Quadstone Paramics, 2013a]), VISSIM ([VISSIM, 2013]) or TransModeler ([TransModeler, 2013]). Their open source counterparts are SUMO ([SUMO, 2013]) and FreeSim ([FreeSim, 2013]);
- **Emission simulation:** can often be directly simulated via own emission models within the traffic simulation (e.g. SUMO, VISSIM, Quadstone Paramics ([Quadstone Paramics, 2013b])). These models estimate the fuel consumption and the produced emission based on average measures from databases like HBEFA. But there are also emission simulations like PHEM or CMEM available which concentrate on the pure emission modeling and consider also other important factors like cold start effects or gear shift behavior ([Hirschmann et al., 2010; Kun et al., 2007]);
- **Communication simulation:** also in the field of communication propagation several simulators are used e.g. ns-2 ([ns-2, 2013]), ns-3 ([ns-3, 2013]), OMNet++ ([OMNet++, 2013]), JiST/SWANS ([Jist/SWANS, 2013]).

2.3 COLOMBO Prerequisites

The COLOMBO “overall simulation system” (abbreviated as COSS in the following) is a continuation of the work on tools developed by the COLOMBO partners in the past and was already defined within the project’s description of work [COLOMBO, 2012]. The simulation architecture is an extension of the system developed within the project iTETRIS[1] [Rondinone et al., 2013] that was co-funded by the European Commission under contract no. 224644 (FP7) between 2008 and 2011. It consists of the communication simulator ns-3[2], the traffic simulation SUMO[3] [Krajzewicz et al., 2012] and the V2X-application under evaluation. These parts are connected using the “iTETRIS Control System” (iCS). The overall architecture of this simulation system is depicted in Figure 2.2. All computer programs that belong to the architecture are described in more detail in chapter 3.

There are different reasons for using iCS within COLOMBO:

- Three COLOMBO project partners (PEEK, EURE, DLR) were participating in the iTETRIS project, contributing to the design and implementation of the iCS.
- The iCS system is known to be working.
- The iCS was continued to be developed beyond the scope of the iTETRIS project and is used by external partners as well.
- iCS is available as open source under the GPL.

[1] http://www.ict-itetris.eu/
[2] http://www.nsnam.org/
[3] http://sumo.sourceforge.net/

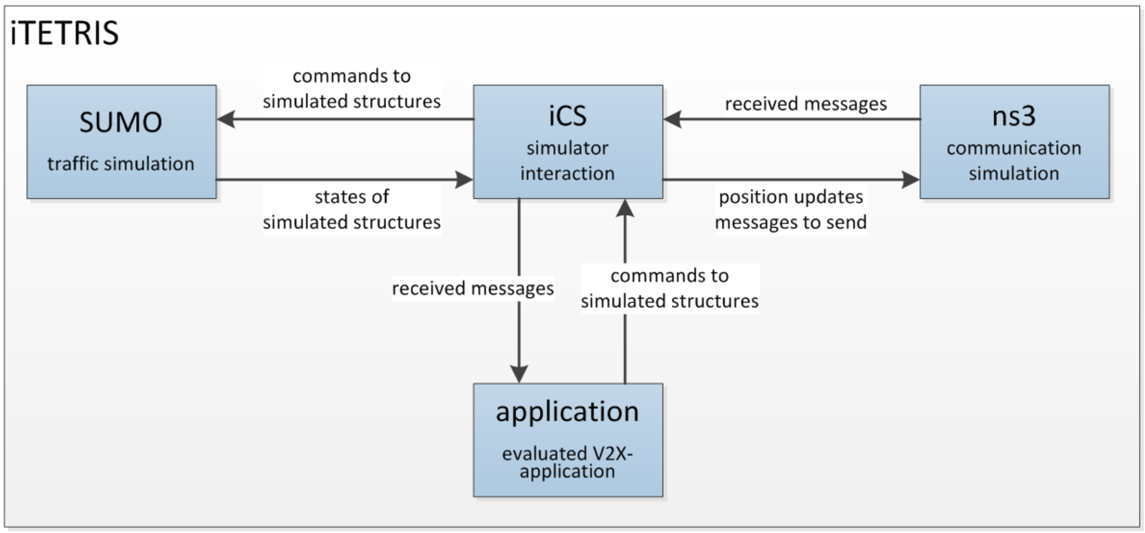

iTETRIS
SUMO
traffic simulation
commands to simulated structures
states of simulated structures
iCS
simulator interaction
received messages
position updates messages to send
ns3
communication simulation
received messages
commands to simulated structures
application
evaluated V2X-application

Figure 2.2: Schematic overview of the simulation system including the involved components' roles and the information they exchange (briefly)

Similar reasons count for the usage of the other components, namely SUMO and ns-3:

- SUMO is a development of the project partner DLR and other contributors since 2001 and is a well-established tool within the V2X-community [Krajzewicz, 2013; Joerer et al., 2012]. It is also licensed under the GPL.
- Being the successor of ns-2, ns-3 is one of the most often used tools for simulating vehicular communications [Joerer et al., 2012], albeit the according models are not directly embedded but must be given as "patches" that have to be applied to the application's sources before compiling it.

The system is extended by PHEM, a pollutant emission model developed by TUG, which replaces the one implemented directly within SUMO. To be exact, PHEM will not be used directly, but a derivative of it named PHEMlight will be directly embedded into SUMO, see section 3.3. Using PHEM/PHEMlight will improve the quality of computing the amount of emitted pollutants and adds new vehicle emission classes, such as new EURO-6 or (semi-)electrical vehicles.

Tuning and configuration tasks that arise in the project, for example, for defining traffic light control policies, will be performed by the automatic algorithm configuration tool kit and the irace package supported by the project partner ULB.

Usually, algorithm tuning and optimization is performed by searching for a minimum cost – the optimal solution – within a high dimensional parameter space. Even though optimization algorithms employ heuristics for finding the optimal solution, usually a large number of evaluations is needed. In our case, each evaluation corresponds to a simulation of the system using a specific setting of the relevant parameters being tuned; thus, this results in a high computational effort. Within COLOMBO, a Linux cluster with about 1000 computing cores located at ULB will be used for executing COSS with parameters changed by tuning/optimization algorithms. As a consequence, COSS must be executable under the Linux distribution CentOS 6.

Tuning and optimization also requires the execution of the program that computes the cost, given input values from the parameter space. The executed program must obtain the input values and in return make the resulting cost available to the tuning algorithm.

2.4 Architecture Overview

The COLOMBO overall simulation system consists of iCS, SUMO, and ns-3, extended by the PHEM/PHEMlight emissions computation module and iteratively executed using the configuration/optimization tool kit, see Figure 2.3.

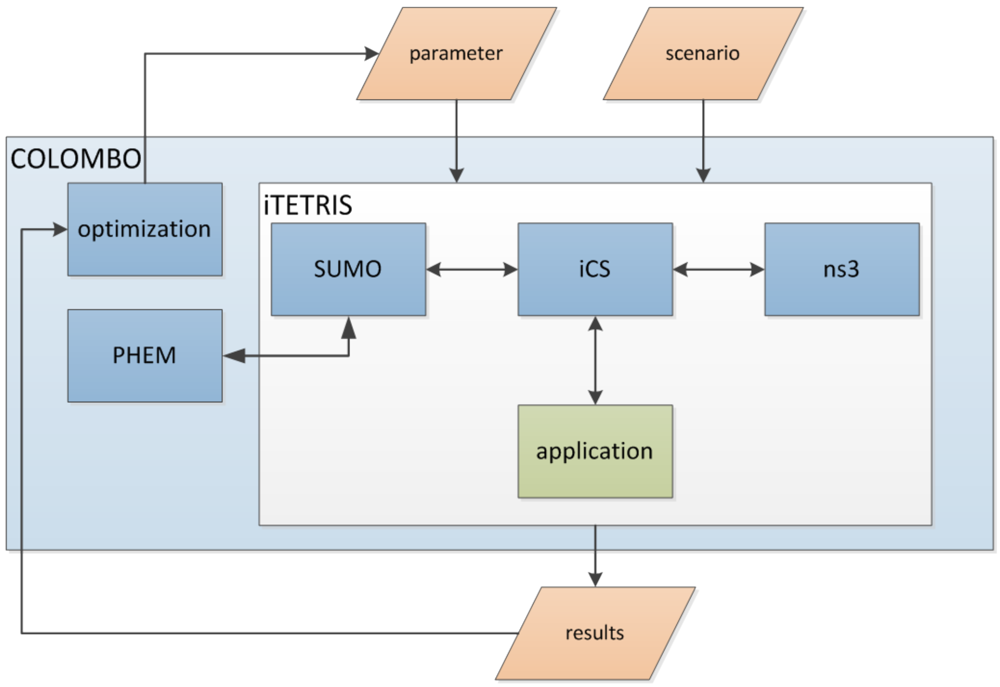


Figure 2.3: Interaction between data and components within the COLOMBO overall simulation system

The main objective of COLOMBO is the development of V2X-based traffic management applications for traffic surveillance and traffic light control and COSS is only a tool within this development. In theory, COSS is capable to simulate all types of V2X-based applications. In practice, all involved simulations have their own limits. Nowadays traffic simulations, for example, are hardly capable to resemble traffic characteristics related to traffic safety. ns-3 is capable to simulate communication between vehicles, but only if the implemented channels (see section 3.1) are used. Specific application may need additional interfaces or data storages. Therefore, a survey is given in chapter 4 that describes the requirements on the COSS due to the applications developed in COLOMBO.

Not all of the developed applications require the involvement of all COLOMBO components. Additionally, the current needs may change at different development steps, changing the requirements to the simulation architecture. Therefore, COSS is not a monolithic application itself, but rather a set of coupled tools that can be combined to meet the current scientific needs.

The components are extended during the project's life time within different work packages. Nevertheless, their interfaces are assumed to be kept as described in this document. Changes in the components shall not remove nor outdate any interfaces, albeit new ones may be added. If not explicitly mentioned, it is also not planned to replace any components during the project life time.

2.5 Functionality and Usage Overview

COSS allows to simulate an "application" that uses information exchanged between vehicles and between vehicles and infrastructure (V2X). The "application" may be a driver assistance system, such as those defined by ETSI (see Figure 2.1), but it may as well be a traffic management application that does not influence individual vehicles, but rather the road infrastructure. Traffic light adaptation to current traffic situation, as designed in COLOMBO, is one of such applications, but also other infrastructure attributes may be changed, such as closing roads or disallowing to use a certain lane by certain vehicle types. As traffic surveillance is a major step in traffic management, an “application’s” purpose may be the determination of traffic state as well.

To evaluate an application using COSS, the user has to implement a model of this application as an “iCS application”. A description how such an application has to be implemented is given in Section 6.3 and a user tutorial as well as an example application is supplied with the iCS. One of the major requirements for an “iCS application” is the interaction with the iCS. When starting COSS, iCS starts all applications defined in the according iCS configuration file and establishes a connection to them. From that point on, the simulation loop is performed (see also section 3.5). Via the socket connection, the application has the access to attributes of the simulated objects, such as vehicles, traffic lights, etc. The major source of information the application model should use are but the messages exchanged between the vehicles / between vehicles and infrastructure. Based on this information, the application performs its logic, transferring the obtained information into actions. These actions can then be sent back to the simulated object they affect (vehicle/traffic light). This way, the application (model) alters the behaviour of traffic represented in the traffic simulation. This “closed loop” – get information via V2X communication, process it, execute actions that affect traffic – is what an V2X-simulation packages should realize.

However, the evaluation of an application requires more. On one hand, a “scenario” is needed - the representation of the situation the application is executed within. In the case of V2X-applications, a scenario usually consists of inputs as used by the involved traffic simulation required to simulate the vehicular traffic and parameters that describe the communication between them - the used communication channel mainly, with additional parameters. An attribute of the scenario that belongs neither to the traffic simulation nor to the communication simulation is the equipment rate. Within COSS, the iCS is responsible for assigning simulated communication devices to vehicles following. A more detailed overview about the input files needed by COSS is given in the project’s deliverable D1.1 “Scenario Specifications and Required Modifications to Simulation Tools” and they are defined in the documentation of the respective COSS components.

The limits to the scenario size and duration are mainly dictated by the available computer power - and the quality of the respective input data. SUMO is known to be able to simulate vehicle traffic in middle-sized cities (300000 inhabitants) faster than real time. Larger scenarios have been used as well, but the effort to build a large scenario is tremendous for different reasons, such as: a) the generation of a SUMO road network requires manual adaptations due to the lack of digital representations in adequate quality, b) settings of the infrastructure existing in the reality, mainly traffic light programs, have to be imported, but are not available in a standardized format. Supporting a set of realistic scenarios was targeted within COLOMBO by Task1.1 “Scenario Generation” where a set of scenarios was generated (see D1.1). ns-3 was found to be more stable than its predecessor ns-2 when a large number (>10000) of vehicles is simulated, but is much slower than SUMO. Increasing the simulation speed is planned to be performed by COLOMBO in further steps of the work package 5.

On the other hand, an evaluation requires measures generated by the execution system that allow to determine whether and what benefits the application delivers. COSS relies mainly on the outputs generated by the involved simulators. It retrieves its communication performance metrics from ns-3. Within the context of the work performed in COLOMBO, traffic related measures are but of a bigger interest. The used traffic simulation SUMO is capable to generate a range variety of outputs including virtual sensors (different types of inductive loops, areal detectors), floating car data, aggregated performance indicators (PIs) for edges and/or lanes including pollutant and noise emissions, vehicle-based PIs, such as individual travel time, route length and others, traffic light performance in means of phase lengths and order. In many cases, the model of the evaluated application should deliver also some outputs that allow to investigate whether it is correctly implemented and/or how it reacts to given inputs.

COSS consists of a set of tools well known in the respective field of application and as shown in chapters 5 and 6, they are ready to use for being combined to simulate a large variety of “applications” based on V2X communication. Some issues that reduce their usability, such as the lack of ready-to-use scenarios, shall be solved within COLOMBO.

3 COLOMBO Components

This chapter describes the chosen COSS components in a greater detail. Besides their overall duty and an outline of their development, a strong focus is put on their usage and the components' possibilities to interact with other applications.

3.1 ns-3

General Introduction

Within COLOMBO, wireless communication for ITS application will be simulated using the Network Simulator 3 (ns-3). ns-3 is an open discrete-event simulation environment that been designed to be the successor of the popular simulator ns-2. Aiming to be more scalable and more open for extension, it significantly differs from ns-2 with its novel structural and modular implementation.

Although ns-3 already contains a large number of protocols and access technologies, notably IEEE 802.11a, it still lacks significant functionalities for the close-to-reality modelling and simulation of ITS application. Within the iTETRIS project, ns-3 was therefore extended by an ITS-specific communication stack, including mechanisms for Delay Tolerant Network (DTN) protocols. Figure 3.1 briefly describes the key available functionalities on the extended version of ns-3.

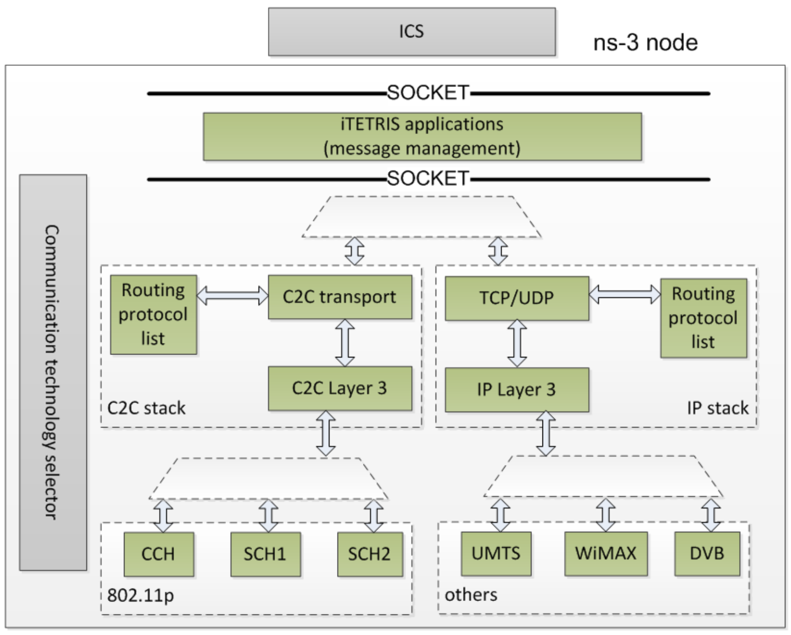


Figure 3.1: Functionalities of ns-3 as extended within the iTETRIS project

In the framework of the iTETRIS project, the ns-3 simulator has also been enhanced with V2X-specific capabilities that are briefly explained below:

- **Access Layer**: The IEEE 802.11p has been integrated as another WLAN access technology, while a channel router module controls the multi-channel operation of ns-3 between three channels: CCH, SCH1 and SCH2. Additionally, a cellular module, UMTS, WiMAX and a broadcast channel DVB-H have been implemented in ns-3 for infrastructure-type communication. The selection of the appropriate communication technology is done via a *techno-selector* module, which indicates via a Communication profile which interface to use on a per-packet basis.

- **Network Layer**: A dual stack (IPv6, ETSI ITS Geonetworking) is available. The V2X capable protocol stack ETSI ITS Geonetworking protocol stack includes geographic addressing capabilities, as well as multi-hop geographic routing.
- **Facilities Layer**: The facilities layer (as defined by the ETSI ITS Standard) has been separated into two parts: the application facilities, implemented in the iCS, and the communication facilities implemented in ns-3. The latter are composed of three major blocks: the CAM/DENM message generators, the communication technology selector, as well as the DTN module.

Another key extension of ns-3 is the addition of the INCi interface. Similar in design to the TraCI interface (API for external integration with SUMO), the INCi provides an open API for external interaction with ns-3. In the current version, the INCi connects ns-3 to the iCS. ns-3 contains and runs on demand an INCi server, to which an INCi client may connect over a socket connection. An INCi client is located at the iCS, as illustrated on Figure 3.2.

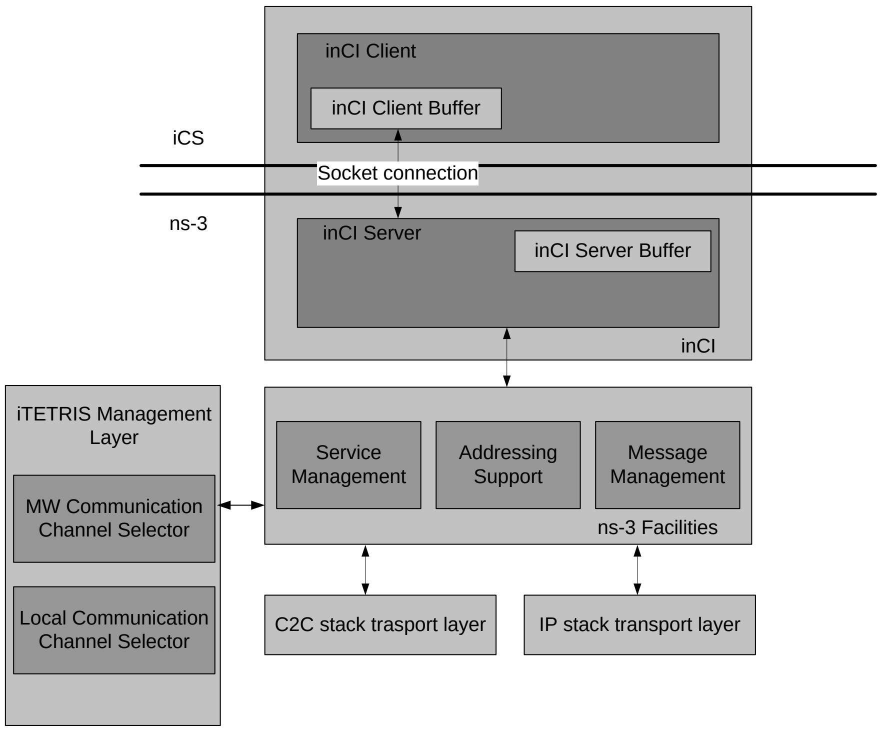


Figure 3.2: Outline of the configuration process

Implementation and Interfaces

The core architecture is object-oriented, it has been developed in C++, contrary to ns-2 (which is written in OTcl and C++), ns-3 optionally uses python scripts for simulations. Some C++ based models have been ported from ns-2 to ns-3 e.g., OLSR (Optimized Link State Routing), random variables and error models. Being open also to commercial use, its base architecture has been designed to support network virtualization and real test bed integration. The object aggregation model is the main feature of ns-3. Multiple objects could be linked together at run-time such as, nodes, applications and protocol stacks. This mechanism has been proposed as solution to the “weak base class” problem in C++ where the base class should be modified each time the programmer needs to reuse it with different configuration. Moreover, the aggregation model handles the access between the different aggregated objects and eases the automatic memory control.

Within the V2X communication system developed in iTETRIS, neither SUMO nor ns-3 are directly connected, but rather interlinked over socket connections via the iCS as depicted on Figure 3.3. The iCS consists of more than a simple interface between modules. Considering its location between modules, it also contains the application facilities and a client implementation of the TraCI and INCi.

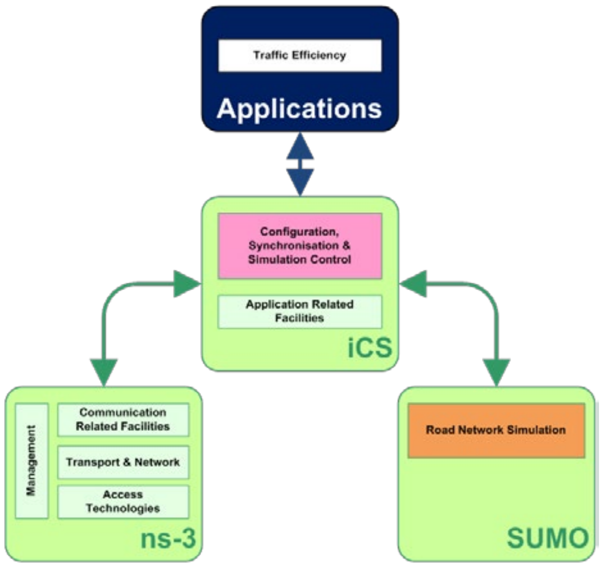

Traffic Efficiency
Applications
Configuration, Synchronisation & Simulation Control
Application Related Facilities
iCS
Management
Communication Related Facilities
Transport & Network
Access Technologies
ns-3
Road Network Simulation
SUMO

Figure 3.3: Simulators joined via the iCS

The interlinking between ns-3 and iCS is supported by the INCi. It contains open bidirectional APIs, to either trigger communication steps, change node position information in ns-3, or to return feedback of received messages to the iCS. Figure 3.4 illustrates the open APIs exposed by the INCi and allowing external interactions between iCS and ns-3.

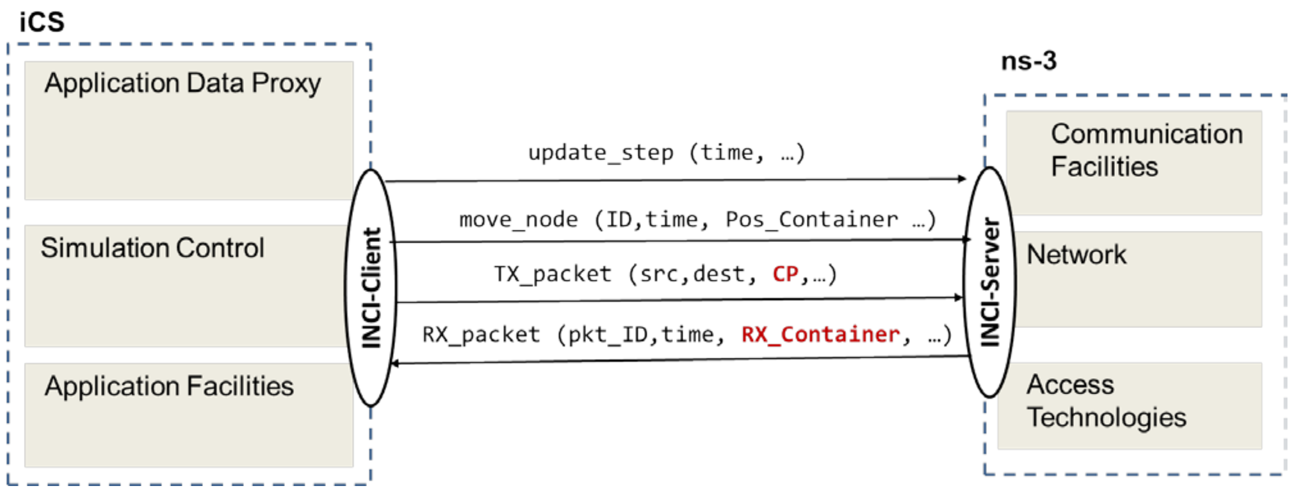


Figure 3.4: Online interaction with ns-3 via INCi

With more details, the INCi provides three major open APIs:

- **`update_step`**: runs a given number of ns-3 simulation steps, up to the indicated `time`.
- **`move_node`**: used to let ns-3 reflects the mobility changes modeled by SUMO in a position container `Pos_Container`.
- **`TX_packet`**: triggers the transmission of a given packet specified by its source, destination and a *Communication Profile (CP*.
- **`RX_packet`**: indicate to the iCS that a packet with `pkt_ID` has been received at a given `time`.

Within COLOMBO, the INCi will not need to extend the open API, but some functions will need to be extended:

- `TX_packet`: the transmission of a packet over a given interface (Direct_WiFi, Sensor) will be indicated by new values for the communication profile **`CP`**.
- `RX_packet`: when a packet will be received, RX statistics will be indicated in a new container called `RX_Container`.

ns-3 is not directly interlinked with SUMO. All data exchanges between SUMO and ns-3 are processed and synchronized by the iCS. Typical data exchanged are mobility updates of SUMO to ns-3 over the INCi interface function `move_node(node_ID, time, Pos_Container)`, where `Pos_Container` contains `(X,Y)` and `speed` information of a node `node_ID` at a given `time`.

Neither the tuning tool kit, nor the traffic light algorithms defined in an external module will directly interact with ns-3, but rather communicate through iCS which will influence ns-3 over the existing INCi interfaces.

Extensions within COLOMBO

Within COLOMBO, ns-3 will be extended to support the following capabilities:

- **Direct WiFi**: In order to support smart-phone communication, the WLAN module of ns-3 will be extended to support partial functions of Direct WiFi. The objective is not to implement the total functionalities of Direct Wifi, but the key functions that will be required to support smartphone communications in COSS.
- **Sensed Data feedback**: the INCi is capable of commanding communications to ns-3, while mobility payload are fed back to the application facilities over the INCi. Within COLOMBO, additional statistics will be required in order to support the detection, triangulation and estimation of traffic densities by COLOMBO vehicles. In particular, RSSI and SNR of each received packets will be provided to the COLOMBO application via the INCi and the iCS. For that purpose, a generic container will be specified and added to any packet reception notification that the INCi already send to the iCS. The ***RX_Container*** follows the same design principles as the data containers of the iCS subscriptions, by following a generic **`<type><length><value>`** structure, with given open API for ns-3 and the iCS to agree on data type, format and size.
- **Sensor Interface**: Class B vehicles containing only sensor nodes (Zigbee or Bluetooth) will be emulated by an C2X extension, with a simplified header with position and speed fields kept empty. Detection will be supported by the sensed RX statistics described above.

3.2 Automatic Configuration Tool kit

General Introduction

Recently, techniques to perform the automatic configuration of algorithms have been proposed, see [López-Ibáñez et al., 2011; Hutter et al., 2009; Hutter et al., 2011]. These methods are increasingly often used to assist the algorithm designer, in particular, in the area of optimization and stochastic local search algorithms. In the *offline configuration* case, the goal of automatic algorithm configuration techniques is to help support the algorithm designer in choosing the best possible settings of parameters of an optimization algorithm during a training phase. These parameter settings are then applied in a successive deployment phase, where the optimization algorithm tackles the problem under concern. Automatic algorithm configuration tools have already shown a strong impact on the way algorithms are designed. This is because a parameter in automatic algorithm configuration can also decide, for example, among different types of search strategies. Such decisions are associated rather to algorithm design than only calibrating a small subset of a fully designed or algorithm. Hence, the advent of automatic algorithm configuration tools has a major implication not only for the practice of fine-tuning algorithms, but it opens a fully new paradigm for algorithm design. This allows not only to make the design process faster, but often it can also allow to find more effective algorithms. It is important to note that these techniques can, in part, even replace the tasks traditionally done by algorithm designers: these methods can effectively compose algorithms from various alternative algorithm components. In this case, the task of the algorithm designer becomes rather to define appropriate algorithm components and their possible interactions, while the automatic algorithm configuration technique takes care of finding the best possible combination of algorithm components and of setting numerical algorithm parameters. The mathematical formulation of the configuration problem is given in [COLOMBO D3.1, 2013], section 2.1.

In this prototype of the overall system, we use the "irace" (iterated racing) software package[4], a tool that is part of the automatic configuration tool kit. irace has been applied successfully in the past to many different algorithms and problems, in particular, when the goal is to optimize the quality of algorithm / system performance. This is a situation analogous to the design of traffic light control algorithms, where some performance measure of traffic flow or emission reduction is to be optimized. The main advantages of irace are that it can handle different types of parameters, it is an effective method, and that it can evaluate different parameter settings in parallel, an important aspect that allows to perform large computational tasks in reasonable time. "irace" is described concisely also in COLOMBO project's Deliverable D3.1: "Prototype of Automatic Configuration and Tuning Tool Kit", section 2.2.1.

This offline algorithm configuration context is also the context that is relevant for the COLOMBO project. The task of the algorithm configuration techniques will be to help in the configuration and fine-tuning tasks that especially arise in the design of the traffic light control algorithms.

Tasks and Usage

Technically spoken, the configurator works similar to optimization algorithms. In fact, automatic configuration can be described as a stochastic, mixed discrete-continuous optimization problem. The configurator itself is started from the command line. As input it receives the list of parameters together with the parameter names, the type of each parameter (e.g. real-valued parameter, integer, ordinal, categorical) and the ranges of possible parameter values for each parameter. The configurator generates parameter settings that the program (or algorithm) whose performance is to be tuned, uses in its execution. The program than is executed on one or usually several instances of the problem that it needs to solve. The program returns a single performance measure (the cost), which is used by the configurator to choose a next set of parameters settings. Then the loop starts over again. The overall process is depicted in Figure 3.5.

The program to configure is – as seen from the configurator's perspective – a black box that obtains parameters and returns the resulting costs. Within COLOMBO, this program will be the complete iCS, which then starts other COLOMBO components to perform a simulation of the V2X-application that is the object to configure. The main practical use case that is envisaged in COLOMBO is the configuration of the traffic light control algorithms and the reduction of the emissions. In some cases, other combinations of COLOMBO components and the application can be used instead of the iCS.

[4] http://iridia.ulb.ac.be/irace

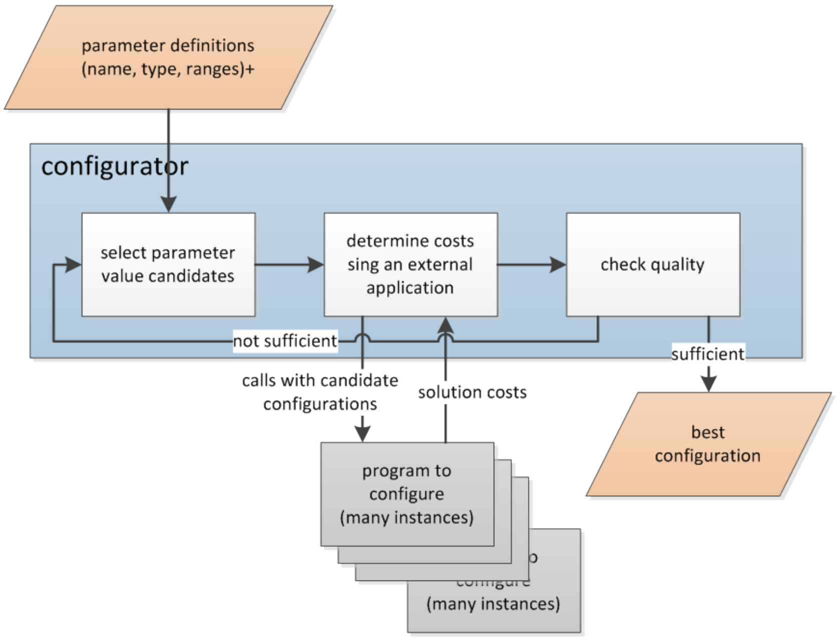

parameter definitions
(name, type, ranges)+
configurator
select parameter value candidates
determine costs sing an external application
check quality
not sufficient
sufficient
calls with candidate configurations
solution costs
program to configure (many instances)
(many instances)
best configuration

Figure 3.5: Outline of the configuration process

Implementation and Interfaces

The irace software package has to be installed independently of the other components of COLOMBO. For this prototype we rely on the version that is currently available on the COLOMBO SVN (see chapter 5). irace is a module for the statistical computing package "R-project"[5] that has to be installed, too.

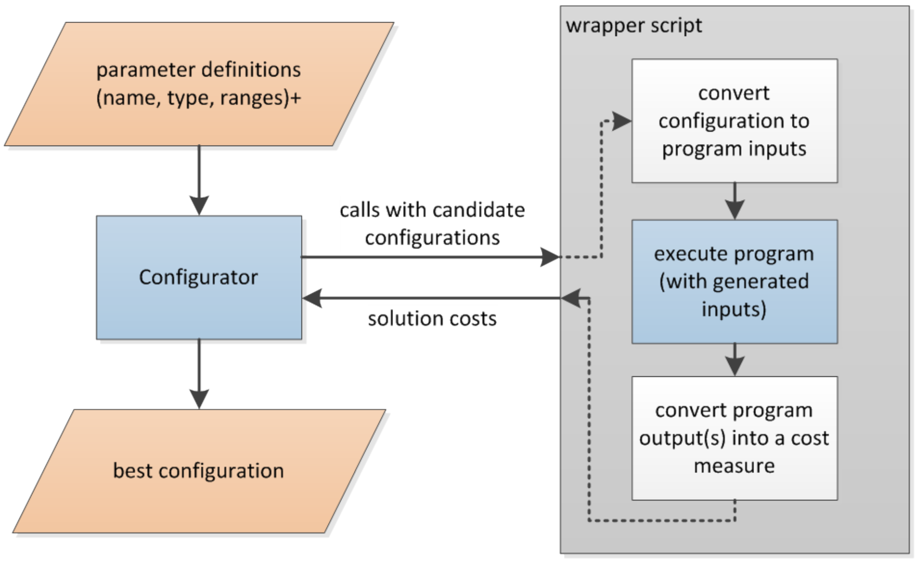

wrapper script
parameter definitions
(name, type, ranges)+
convert configuration to program inputs
calls with candidate configurations
Configurator
execute program (with generated inputs)
solution costs
convert program output(s) into a cost measure
best configuration

Figure 3.6: Wrapper script as an interface between the configurator and the program / application to configure

As outlined, the only interfaces are the parameters that are passed to the program to be configured on its start-up, and a result number given back by this program on the command line. As most of the COLOMBO components require more complex inputs such as configuration files and different input files, such as road networks, traffic demand, infrastructure descriptions, etc., the executed program will have to be "wrapped" by a script that converts the parameters set by irace into proper inputs to the program to configure, executes the program, and converts the outputs generated by the

[5] http://www.r-project.org/

program into a performance measure. This is depicted in Figure 3.6. The wrapper script has to be executable by a simple system call (like "on the command line"). This means that almost any programming language can be used for its implementation, as almost all languages supply the possibility to implement such programs.

The result from the execution of irace is then the best configuration that has been found in the configuration process.

Extensions within COLOMBO

Within the COLOMBO project, several extensions of the configurators will be considered. One extension will target the improvement of the currently available configuration tools, in particular, the irace tool that has been developed at the ULB partner. A second direction will consider the extension of the configurators to handle situations in which the performance of the system to be configured will be evaluated by multiple performance measures, that is, situations in which the configuration task actually becomes a multi-objective task; this contrasts with the fact that so far all configurations.

3.3 PHEM

General Introduction

PHEM[6] ("Passenger Car and Heavy Duty Emission Model", [Hausberger, 2003; Rexeis, 2009; Zallinger, 2010; Hausberger, 2011; Hausberger, 2012; Luz, 2013]) computes in a 1 Hz frequency the required propelling power to reach the velocity and acceleration demanded by a given driving cycle. Thereby, all driving resistances, including road gradients and losses in the drive train are taken into consideration. In addition, a gear shift model provides the respective gear used by the vehicle which completes the load point (power Pe, engine speed n) of the combustion engine for every second. By using specific emission maps for a certain vehicle, the corresponding pollutant emissions and fuel consumption for each load point can be derived. PHEM is developed at TUG since 1999.

More complex modules have been added to PHEM, e.g. thermic effects of complex exhaust gas after-treatment systems and the impact of transient operating conditions on emissions and fuel consumption. Figure 3.7 illustrates the main modules and functionality of PHEM.

PHEM is applied within COLOMBO, since it offers some advantages against other existing instantaneous models:

- PHEM is based on an extensive European set of vehicle measurements and covers passenger cars, light duty vehicles and heavy duty vehicles from city buses up to 40 ton semi-trailers. For all vehicle categories predefined model input data is available from the actual update of the HBEFA [Kirschmann et al., 2010].
- PHEM has already a validated interface to micro scale traffic models from former projects [Kirschmann et al., 2010; Tielert et al., 2010; Hausberger et al., 2009].
- PHEM also provides the emission values for the traffic situations in the HBEFA and also feeds the model COPERT with emission factors. Thus all results will be compatible with European macroscopic emission modelling approaches in terms of absolute emission levels (e.g. to evaluate emission reduction potentials).

[6] http://www.ivt.tugraz.at/de/forschung/emissionen.html

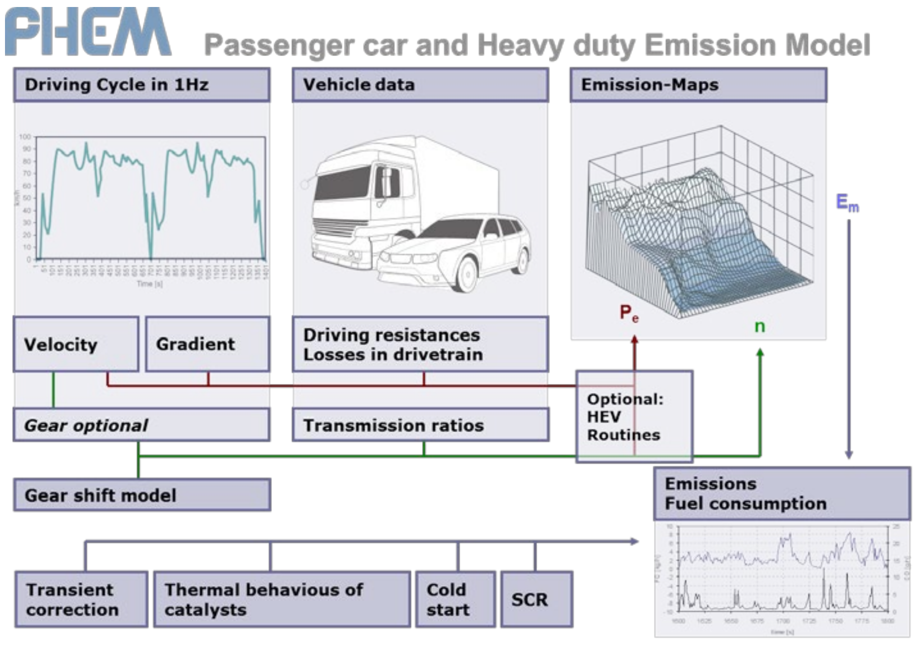


Figure 3.7: Structure of PHEM

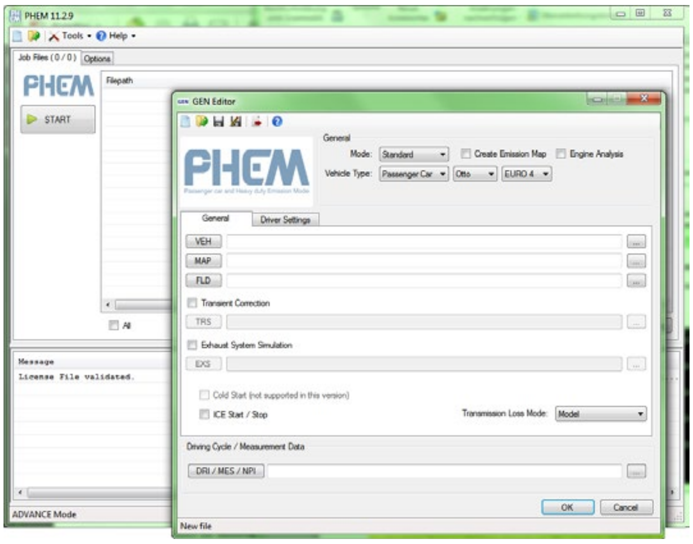
This approach allows PHEM to simulate all possible combinations of speed course, road gradients, vehicle loadings and drivers gear shift behaviour. When combined with traffic models, PHEM is used to calculate modal emissions for all vehicles in the traffic network defined in the traffic model. The vehicle speed data is used as simulated by the traffic model. The required vehicle and engine data is taken from the existing PHEM database on average vehicles automatically according to a user defined or a default fleet composition. If information about the simulated road network is available (i.e. geo-coded street networks) PHEM can assign the calculated emissions to road segments as input for emission dispersion models if required.

PHEM is the only non-open application used within the project. PHEM may be licensed from TUG, and is made available to project partners as well. A “PHEM light” version is produced within COLOMBO which is integrated into SUMO. Here, the source code as well as sample data files will be released to public source.

Tasks and Usage

Within COLOMBO, PHEM is responsible for computing the amount of pollutants emitted by the simulated vehicles. These measures are compared across different parameter settings of a developed application to determine the most environment friendly parameters.

From a user perspective, PHEM obtains the trajectories of vehicles using them to compute the number of emissions generated by each vehicle. The trajectories are given as .csv-files (“comma separated values”) that contain at least the time step, the vehicle’s velocity and the slope of the road for all time steps and all vehicles. Additionally, the vehicle engine’s number of revolutions can be given. An extended version of the input formats includes the information about the road and allows to specify the vehicle’s class. If not given, the class is chosen randomly based on the used data base, given the percentages of vehicle classes within the vehicle population stored in the data base. Usually, a graphical user interface is used to interact with PHEM, see Figure 3.8, but it is also possible to run PHEM from the command line.



Message
License File validated.

Mode: Standard
Vehicle Type: Passenger Car, Otto, EURO 4
Transmission Loss Mode: Model

Figure 3.8: PHEM's graphical user interface

PHEM can be controlled to a higher degree, describing the engine map, the full load curve, etc. These possibilities will not be exploited when evaluating COLOMBO applications, as standard vehicle classes and their distributions within the vehicle population will be used. Further information can be found in the "PHEM User Guide" that comes with the PHEM package.

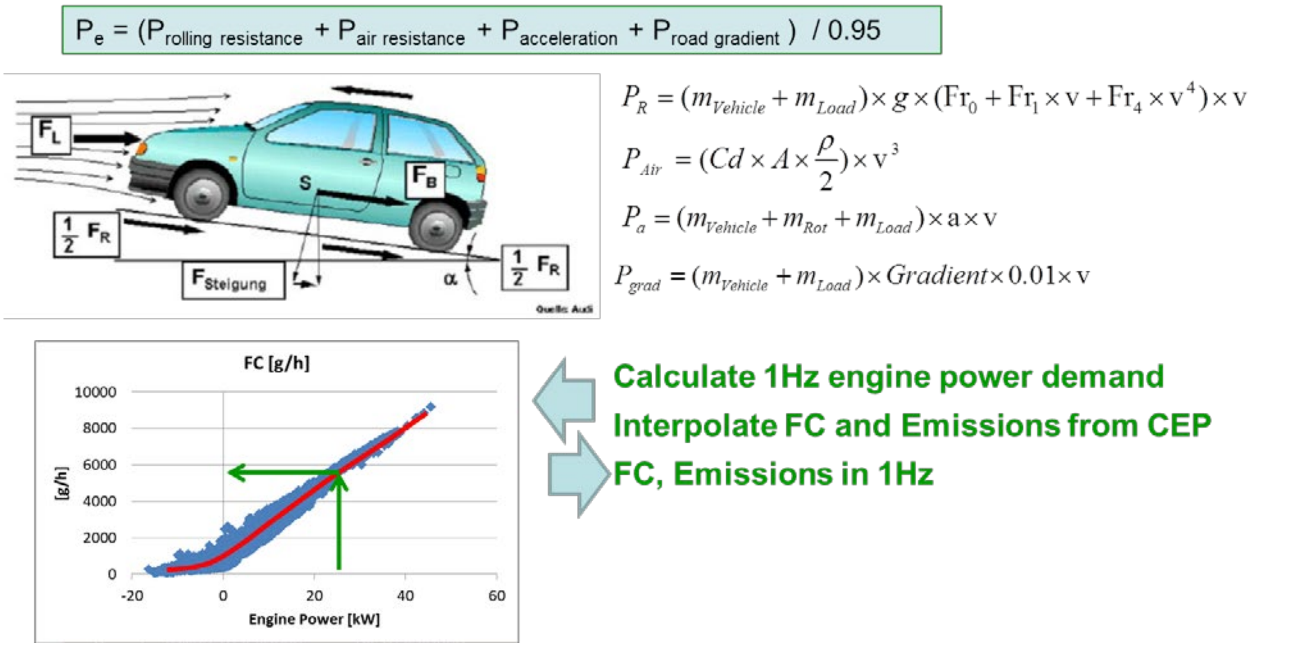
Implementation and Interfaces

In a first attempt PHEM was coupled offline to SUMO to compute emissions for a specific traffic scenario. To use PHEM's full functionality, especially the dynamic correction and the calculation of exhaust gas after-treatment system temperatures, it requires the whole driving cycle per vehicle to track transient effects over time. As SUMO computes states for all vehicles per second, this data had to be rearranged to fit PHEM. Sorting SUMO results according to single vehicles can be done only at the end of the traffic simulation. This is the main reason for the offline coupling of the full version of PHEM.

The realization of the off-line coupling was already performed as one work within COLOMBO's work package 4, task 4.2 and works as following. SUMO generates vehicular trajectories in a non-standardized format. A converter script "traceExporter.py" written in Python converts these trajectories into PHEM-inputs (.dri, .flt, .fzp, and .str). These files can be read by PHEM. A complete presentation will be given in COLOMBO Deliverable D4.1 "Draft of 'Extended Simulation Tool PHEM coupled to SUMO with User Guide'", due in October 2013.

As driving trajectories (speed and acceleration) gained from traffic simulations include uncertainties and do not exactly represent realistic driving behaviour, a simplified version of PHEM was designed, the so called PHEMlight. This version of the tool omits dynamic corrections, temperature influences for after-treatment-systems and, most importantly, the driver gear shift model to compute the engine speed during the computation of pollutant emissions. However, since emissions in SUMO shall be simulated for average vehicles and average driver behaviour and, as mentioned above, driving trajectories have an artificial origin, an acceptable level of accuracy can be reached by the light version of PHEM. The gain of the full version is assumed to be very small. A final assessment can be made after a data set for validation is produced.

Similar to PHEM also PHEMlight computes the power demand for a specific vehicle per second, using the vehicle speed, acceleration, vehicle-load and road gradient. Vehicle parameters, as e.g. rolling resistances or cross-sectional area are taken from a data-base and represent the average vehicles of a typical fleet. For each vehicle type (e.g. passenger car) and emission class (e.g. EU5) in a fleet a data-set is provided. It holds the corresponding parameters and an emission curve, CEP, which describes pollutant emissions and fuel consumption for the whole power range. The influence of the engine speed has been removed by applying probability distributions for the engine speed at a certain power level and computing the expected emission value from the original PHEM engine emission maps. Using these curves and the computed power demand, PHEMlight is able derive the emissions for a vehicle per second. Figure 3.9 describes the functionality of PHEMlight. As for the off-line coupling, the on-line coupling will be presented in draft in COLOMBO Deliverable D4.1, the final results in COLOMBO Deliverable D4.2 “Extended Simulation Tool PHEM coupled to SUMO with User Guide”, due in February 2014.



$$P_e = (P_{\text{rolling resistance}} + P_{\text{air resistance}} + P_{\text{acceleration}} + P_{\text{road gradient}}) \ / \ 0.95$$

$$P_R = (m_{Vehicle} + m_{Load}) \times g \times (\mathrm{Fr}_0 + \mathrm{Fr}_1 \times \mathrm{v} + \mathrm{Fr}_4 \times \mathrm{v}^4) \times \mathrm{v}$$

$$P_{Air} = (Cd \times A \times \frac{\rho}{2}) \times \mathrm{v}^3$$

$$P_a = (m_{Vehicle} + m_{Rot} + m_{Load}) \times \mathrm{a} \times \mathrm{v}$$

$$P_{grad} = (m_{Vehicle} + m_{Load}) \times Gradient \times 0.01 \times \mathrm{v}$$

Figure 3.9: Structure of PHEMlight

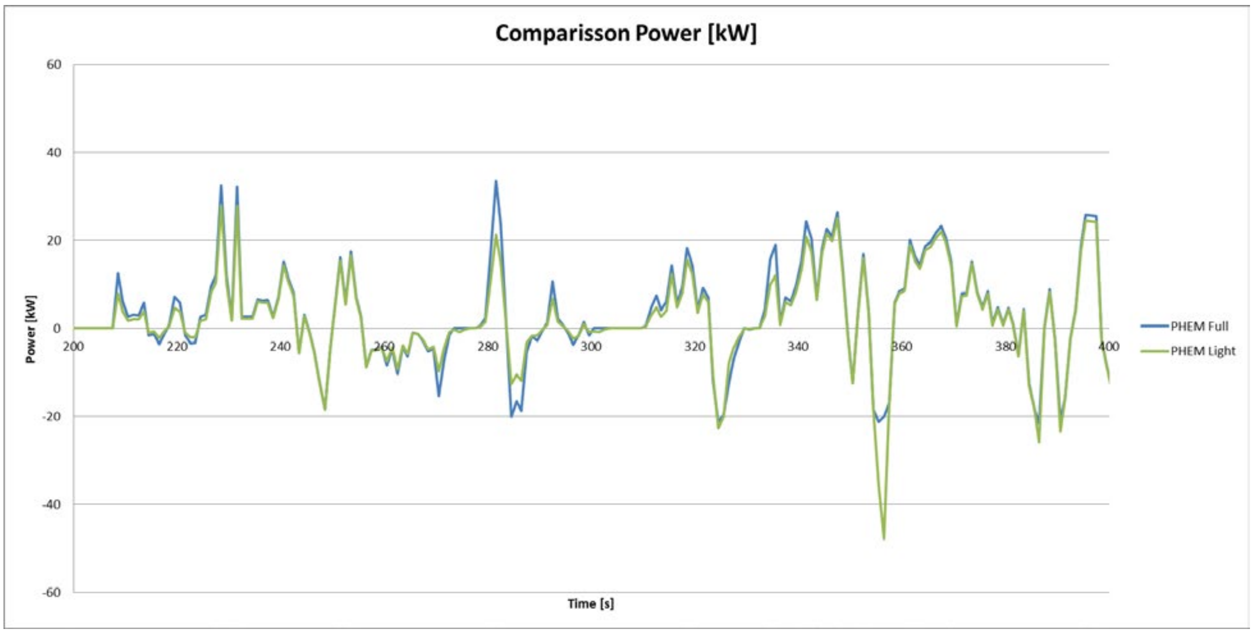
To analyse the accuracy of PHEMlight compared to the full version a representative test cycle (CADC) and average vehicle (diesel passenger car, EU4) have been chosen. Power demands and emission values have been computed with both methods and results have been compared. Figure 3.10 shows some of the findings.

Results show that the engine power can be computed quite accurately, even by neglecting engine speed and transient effects. Also the NO_x emissions have already a quite high level of accuracy, although the lack of information on engine speed has a bigger influence in this case. However, the construction of CEP data sets still leaves possibilities to enhance this behaviour and is subject of next steps.

PHEMlight is directly included into SUMO, though parameters that describe the vehicle classes’ emission behaviour are given as external files that are read on demand. Internally, the interfaces are an extension to the ones implemented for SUMO’s first emission model. They allow generating the following outputs:

- Emissions on lanes, aggregated along complete lanes, and over a user-defined time span
- Emissions on edges, same as for lanes
- Per-vehicle emissions, where emissions are collected for each vehicle over the vehicle’s complete ride

These outputs can be written to files or sent to an external application via sockets (not based on TraCI). Currently, a visualization as available for the HBEFA-based emissions is not implemented. PHEMlight is also not yet accessible via TraCI.



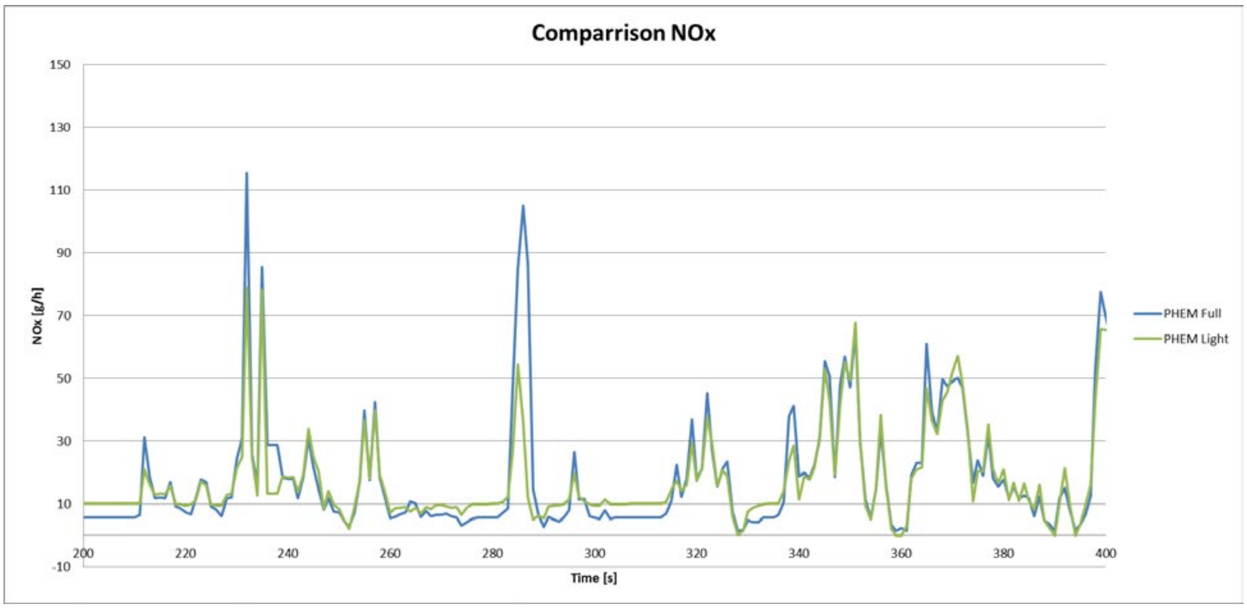


Figure 3.10: Preliminary comparisons between PHEM and PHEMlight; top: results for power, bottom: results for NOx emission

Extensions within COLOMBO

PHEM is extended by models for modern EURO-6 and for semi-electrical vehicles within COLOMBO's work package 4, task 4.1. The draft results will be presented in COLOMBO Deliverable D4.1, the final results in COLOMBO Deliverable D4.2.

These extensions will only extend the data base of the modelled vehicle types. Resulting, no changes in the APIs to PHEM are assumed. The on-line connection between PHEM and SUMO, as outlined above, is a work-in-progress of COLOMBO's work package 4, task 4.2, and will be reported in deliverables D4.1 and D4.2 as well. As explained, the work is performed to allow an on-line interaction, so no negative side-effects on using PHEM within the project should be expected.

3.4 SUMO

General Introduction

SUMO [Krajzewicz et al., 2012] is a microscopic, open source road traffic flow simulation, licensed under the GPL. SUMO is space-continuous, time-discrete, inter- and multimodal. In principle, SUMO requires a road network that includes road-side infrastructure, such as traffic lights, and a traffic demand for performing a simulation. Given both, the simulation SUMO moves the vehicles from the start position of their journey till their end positions. The simulation's duty is hereby to represent the movement as close to reality as possible. The simulation is purely microscopic – each vehicle/pedestrian is moved separately and has an own route through the road network. Besides the road network and the demand, SUMO may obtain further files that contain the definitions of other structures, such as detectors, variable speed signs, additional traffic light programs, bus stops, etc. The major models used within SUMO for describing the longitudinal and lateral movement of vehicles are described in [Krajzewicz, 2010]. Pedestrians are currently moved along streets with a pre-defined speed.

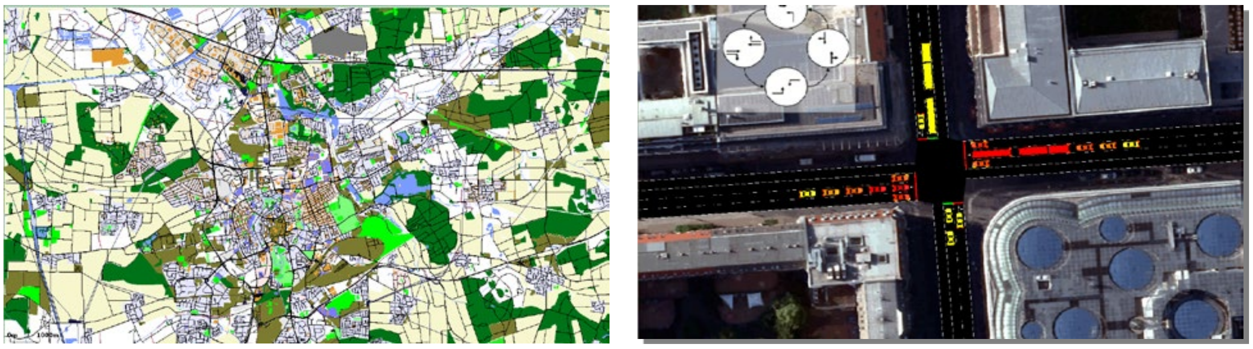


Figure 3.11: Example screenshots from SUMO (left: Brunswick area represented in the simulation, with additional visualization of areas and points of interest; right: a view at a single intersection)

All SUMO inputs are encoded as XML files. The semantics of their contents represent the simulation's current features and/or assumptions. To allow to use foreign data sources, SUMO as downloaded contains a set of additional applications besides the simulation SUMO itself. The complete suite contains:

- **netconvert**: allows to convert foreign networks into SUMO-networks; currently the following input formats can be read: VISUM, Vissim, OpenStretMap, OpenDriveArcView shapefiles, Robocup Rescue, MATsim, ITSUMO, “plain XML”, as well as SUMO networks and a DLR-internal representation of NavTeq GDF-networks;
- **netgen**: allows to generate abstract road networks; currently manhattan grids, spider-network like networks and random networks can be generated;
- **od2trips**: converts origin-destination (O/D) matrices into single vehicle “trips”, where a “trip” defines a vehicle's journey by listing the start and end edge (road) and the departure time;
- **duarouter**: reads "trips" and performs a shortest-path computation and a dynamic user assignment, the latter usually iteratively in combination with the traffic simulation [Gawron, 1998; Behrisch et al., 2008];
- **jtrrouter**: a simple route computation application based on turning percentages;
- **dfrouter**: a route computation application based on detector measures, only applicable to highway traffic, as loops in routes are not recognized;

The applications are described in a higher detail in [Krajzewicz et al., 2012]. Further documentation can be found on the project's web pages (http://sumo.sf.net).

For traffic management it is often needed to simulate huge areas or over several hours. One of the advantages of SUMO is that it is possible to simulate large-scale traffic scenarios. Even the traffic of a whole city like Bologna can be simulated. The number of vehicles and the size of the traffic network are only limited by the performance of the used computer for the simulation.

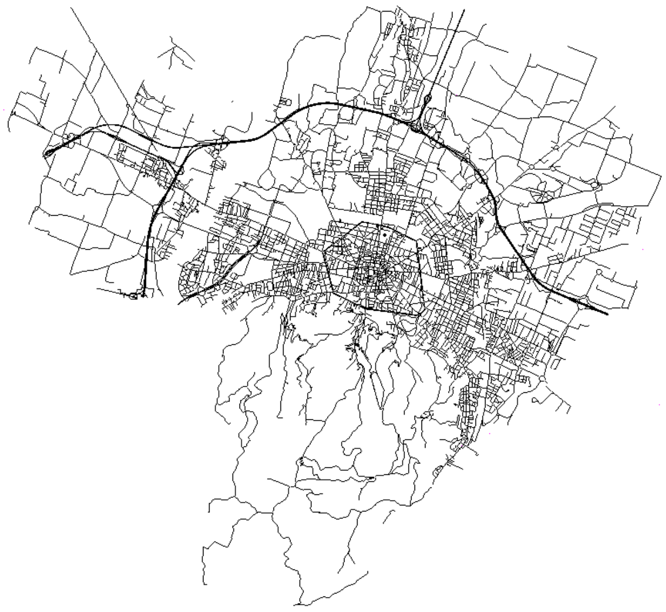


Figure 3.12: A city-wide scenario covering the city of Bologna as shown in SUMO

SUMO is a development of the Institute of Transportation Systems at the German Aerospace Center. The first concepts were developed in the year 2000, the implementation began in 2001, and the first public release was done in the year 2002. The motivation for releasing the application was to improve the validation of different traffic management applications by offering a common test bed - and the hope to get user contributions.

SUMO was used by the DLR in a large number of national and international projects. The main applications were the evaluation of new algorithms for traffic lights control (projects OIS and ORINOKO), for navigation (INVENT project), and for traffic surveillance (TrafficOnline). Since the year 2008, the DLR is also using SUMO to investigate the potential of vehicular communication, mainly in the context of projects co-funded by the European Commission, namely iTETRIS, PRE-DRIVE C2X and DRIVE C2X. SUMO is also used by other research institutes with a growing number of publications referring to it. Figure 3.13 shows some statistics on SUMO's usage taken from a report on evaluation of publications that mention SUMO [Krajzewicz, 2013].

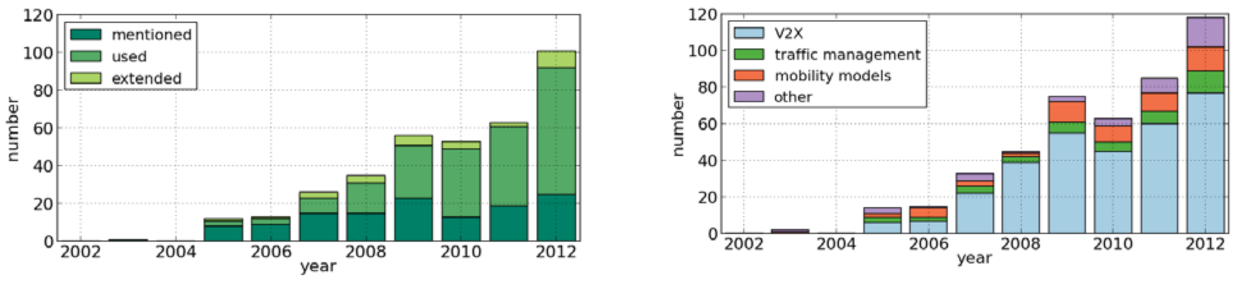


Figure 3.13: left: Number of publications mentioning SUMO over the years, divided by SUMO's role in the publication, right: the publications' topics over time (a document may be assigned to different topics)

Figure 3.14 shows the download rates for the different operating systems. Of course, all used libraries are compliant with SUMO's license.

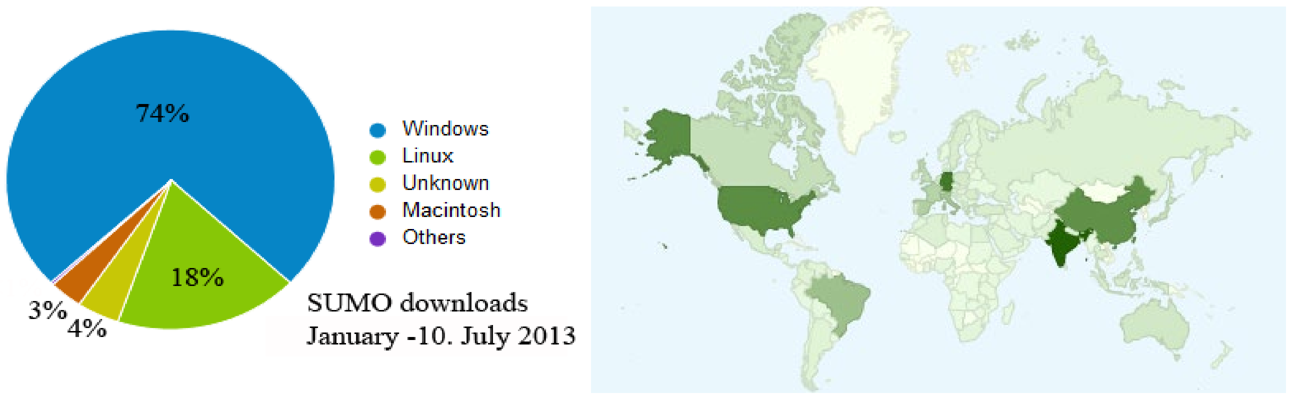


Figure 3.14: SUMO Downloads between the 1st of January 2013 and 10th of July 2013; left: by operating system, right: by country

Tasks and Usage

A usual approach in using SUMO is as follows. At first, the scenario is generated, often resembling a real-world road network, a representation of a complete city, or sometimes also smaller areas. The road network of the area to simulate is imported from available sources. When working on German cities, the DLR usually uses data from NavTeq as they assure a certain quality and because they model road networks in a fashion the DLR staff members know. The free OpenStreetMap data can also be used, but the quality of these data is differing very much across the world's regions, being sometimes more detailed than commercial road networks, but usually being quite less detailed. In any case, the imported road networks must be corrected after being imported. This is necessary, as available digital road network representations are usually used for routing purposes and lack different information a microscopic road simulation needs – may it be the correct lane number, the information which lane has to be chosen to reach a certain destination, traffic light programs, etc., see Figure 3.15 for an example. Even though netconvert includes a set of heuristics for computing some of these values, especially larger intersections have to be edited by hand. It is assumed, that this can hardly be compensated by improving the heuristics but that rather a higher detail within the input data is needed.

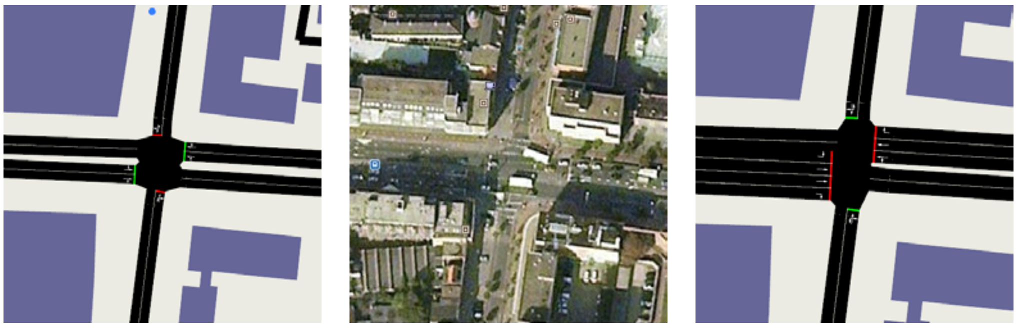


Figure 3.15: Adaptations to the OSM network; left: original OSM data, centre: Google Earth image; right: network after adaptation (from: DRIVE C2X Deliverable D22.1 "Methodology Framework available")

After editing the road network and implementing the road infrastructure – which usually comes in a proprietary format that requires own import procedures – the demand is set up. The size and type of the road network determines which demand type is the most appropriate one. For a realistic traffic simulation it is important to know how many vehicles are taking which routes in the traffic network. There are different ways to set up a realistic traffic demand. For large areas, such as complete cities, origin-destination (O/D) matrices are the best choice. With a given O/D matrix and the tool od2trips, the traffic demand and the used routes can be estimated for the scenario. Also measurements from induction loops or floating car data can help to generate a realistic traffic demand or can be used to validate the simulation.

Validation as such is usually performed by putting real-world measures against simulated measures of the same type. In some cases, one part of given real-world measures can be used to calibrate the simulation, first. Then, the remaining measures are used to validate the simulation's set-up. Figure 3.16 shows a validation result by example.

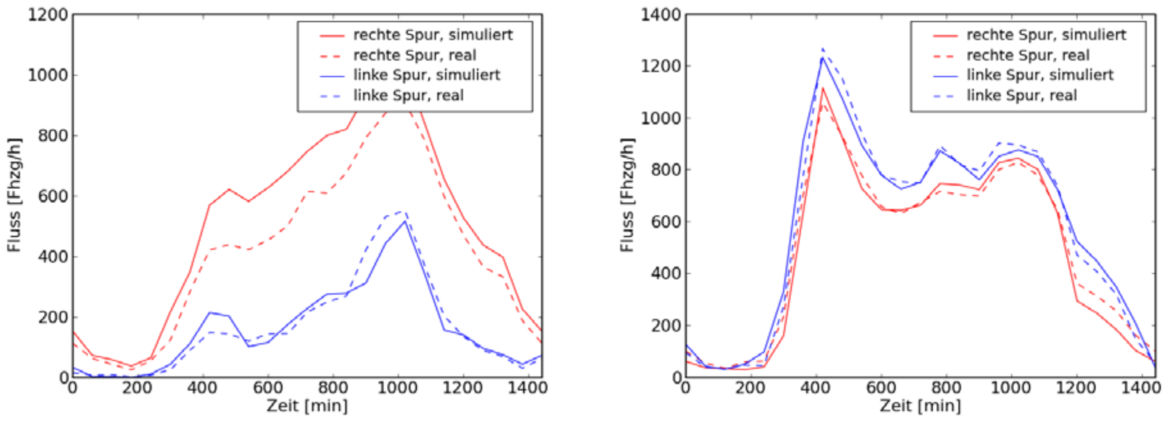


Figure 3.16: Example validation of the simulation based on comparison of real and simulated traffic flows for two detector sites (red lines: right lane, blue lines: left lane; full lines: measures from the simulation, stroke lines: measures from real-world detectors) [Flow and Time]

These steps deliver a well-prepared scenario. In most cases, the performance measures delivered by such a "normal day" or "zero case" scenario are compared to the ones of a modified scenario, where modification means the embedding of the traffic management or vehicular application that is investigated. Figure 3.17 shows an example comparison of vehicle trajectories between a "zero case" scenario (left) and the same scenario where vehicles are using the GLOSA application to progress through the road network without halting (from [Krajzewicz et al., 2012b]).

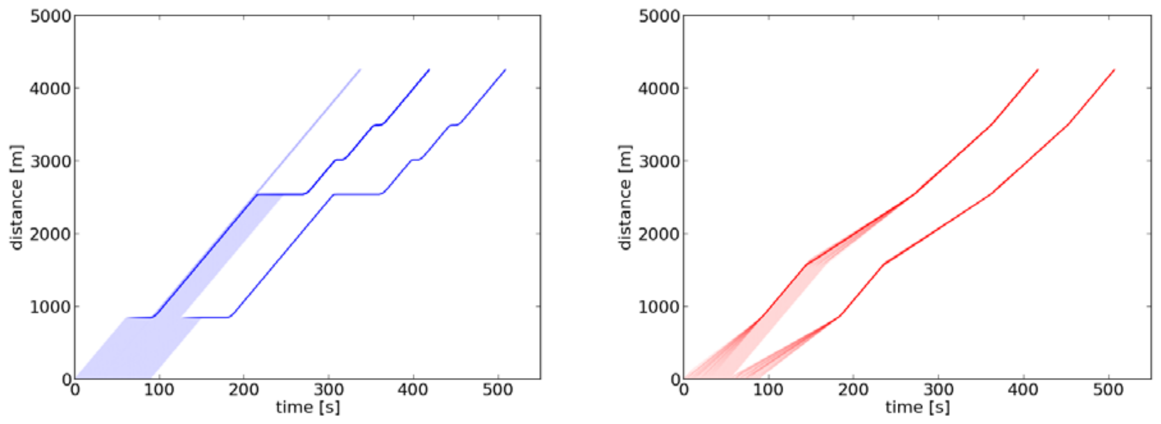


Figure 3.17: Comparison of vehicle behaviours when using the GLOSA application as an example of evaluating V2X-applications ([Krajzewicz et al., 2012b])

Implementation and Interfaces

SUMO is written in C++, and includes additional tools written in Python. Only standard C++-classes are used and a great care has been taken to assure that the software stays portable across the main available platforms. SUMO can be built and used under Microsoft Windows, Linux, and MacOS.

The usage of XML as base for all input and output files assures a high portability as the files can be read using simple text editors and because proper parsers exist for all common programming languages and all common platforms.

All applications from the SUMO suite can be run on the command line. Only “sumo-gui” has a graphical user interface common to Microsoft Windows applications. The possibility to run the applications on the command line allows to process simulation runs, including their preparation and their evaluation in a “batch” – instead of starting them interactively, a script (small, usually not a compiled but an interpreted application) is written, allowing to perform an arbitrary number of different simulation runs.

In the year 2008, the University of Lübeck supported a socket-based communication interface to the simulation. This interface named “TraCI”, an acronym for “Traffic Control Interface” allowed to interact with the simulation by reading values from vehicles, traffic lights, and the road network the simulated scenario consists of. TraCI was reworked within the iTETRIS project by modularizing it for allowing adding interaction with a larger set of simulation structures. “TraCI” plays an important role in COLOMBO as it will be used for most of the interactions performed between SUMO, other components, and applications.

3.5 Application Simulator(s)

General Introduction

The components allow to simulate the traffic and communication environments of an application, to explore the application’s environmental and efficiency benefits by generating performance measures, and to optimize the application to work at best in a given environment by re-running the simulation with parameters optimized by the optimization/tuning tool kit. But the application itself must be designed, developed, and evaluated itself as well. Application models were already included in the initial iCS architecture. Here, a computer program that mimics the behaviour of an application is included into the simulation loop, obtaining information from the communication simulator and transferring the resulting commands into orders for the traffic simulator. The application model does not communicate with these simulators directly, but via the iCS. This requires to embed the application to simulate into COSS, and opening the application to interact within other COSS components.

When started, iCS starts the involved simulators ns-3 and SUMO and the program that includes the application to evaluate as well. After starting these programs, socket connections to them are established. Being one of those programs, the application receives information about messages that were sent by communicating traffic participants. Optionally, the application may also use the connection to ask the traffic simulation about the state of certain objects, e.g. traffic lights. This information is returned indirectly – the iCS obtains the request, passes it further to the traffic simulation, and returns the result obtained from the traffic simulation to the application.

The application is capable to send action commands – changing the state of the traffic light, force a vehicle to change the lane, etc. – to the traffic simulation, again via the iCS.

Even though most of the applications developed within COLOMBO rely on vehicular communication, not all influence traffic. In such cases, it may be more appropriate not to use the complete system but only some of its components for reducing the simulation’s complexity and the execution time as well. Additionally, some applications are developed step-wise, using only a part of the COSS for an initial implementation of the algorithms, then embedding these algorithms into an iCS application. This will be discussed again in chapter 4 on COLOMBO applications.

The application simulator is a discrete event simulator which is interconnected to the iCS by exploiting the open socket APIs of the AppCI server. It contains two key functions:

- **Communication Management** – it represents the TraCI client that is in charge of interfacing with the iCS, opening a socket connection and implementing the open APIs of the iCS.
- **Data Management** – the iCS has been optimized to reduce overhead in data being exchanged between modules. Accordingly, when data needs to be exchanged between nodes, it is stored locally in the Data Management database and will be transmitted directly between the nodes in the application simulator upon confirmation by iCS that the transmissions were successful.

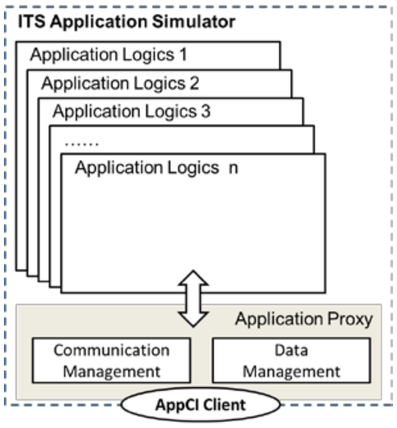


Figure 3.18: The Application Logics of the ITS Application Simulator

From a software perspective, the application simulator is composed of two major block, beside the main class:

- **an application server –** It represents actually the AppCI client, not a server. It has yet been called server as it is responsible to receive calls from the iCS related to the various required steps of the application logics
- **one or more application logics –** It represents the intelligences and the functions of the application simulator itself. It may be composed of 1 or more application logics that will be called by the server upon the reception of the appropriate API from the iCS.

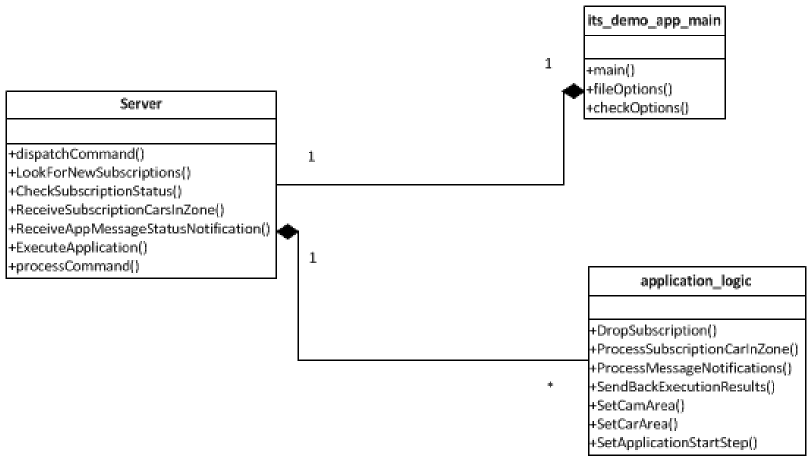

its_demo_app_main
+main()
+fileOptions()
+checkOptions()
Server
+dispatchCommand()
+LookForNewSubscriptions()
+CheckSubscriptionStatus()
+ReceiveSubscriptionCarsInZone()
+ReceiveAppMessageStatusNotification()
+ExecuteApplication()
+processCommand()
application_logic
+DropSubscription()
+ProcessSubscriptionCarInZone()
+ProcessMessageNotifications()
+SendBackExecutionResults()
+SetCamArea()
+SetCarArea()
+SetApplicationStartStep()
1
1
1
*

Figure 3.19: Object structure of the Application Logic

The Server implements the open API that is exposed by the AppCI Server hosted by the iCS, and exchange data with the iCS via a socket interface.

The iCS is capable of dealing with one or more application simulator, which leaves the design of the application logics (all in a single simulator or as independent modules) to the choice of the designer.

Implementation and Interfaces

Every application simulated in the COSS in integration with iCS (see below) underlies a typical life cycle. The application simulator is periodically called by the iCS after ns-3 and SUMO. The order in which each of the applications is called is not specified in iCS and should be extended by COLOMBO.

When the application is called, the iCS will request a sequence of actions that correspond to the Application Simulator lifecycle itself.

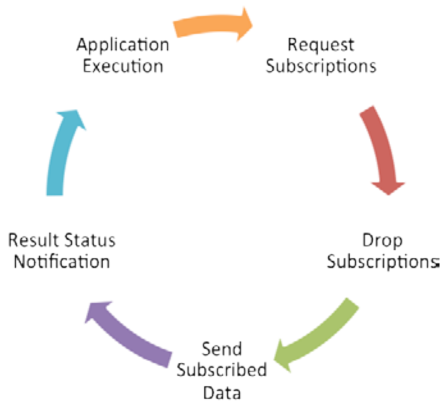


Figure 3.20: the iCS – the Application Life-cycle

The Application lifecycle is composed of five steps:

- **Request Subscription –** the iCS asks if the application requires any subscription for this round.
- **Drop Subscription –** the iCS asks the application if it needs to cancel any recurring subscriptions (like CAM transmit)
- **Send Subscribed Data –** as the name indicates, the iCS sends to the application the data it required, if it is available at that round.
- **Result Status Notification –** notification when the result container has been changed or updated by any external module
- **Application Execution –** the application simulator runs its application logics and saves the outcome in the result container.

The Application simulator therefore interacts with the iCS in a particular order that all application logic must follow. For instance, if the application requests data via a subscription, it may only get it at the next round. Also, if the outcome of the application execution is to trigger ns-3 or SUMO, the request for subscription will only be conducted at the next round, as the life-cycle is complete. Such structure allows yet a lot of flexibility in how to interact with iCS, ns-3, SUMO or other modules.

The Application Finite State Machine (FSM)

The Application logics maintain different state of process with respect of what is asked and what is completed between the application and the iCS (ns-3 and SUMO). The Application simulator defines 6 states, based on which all application logics may control the process of their interaction with the iCS. The states are described in Table 3.1.

Table 3.1: States of the Application Interaction

Name	Description
kToBeScheduled	The Application requests the iCS to schedule the result for transmission in ns-3.
kScheduled	The result was scheduled in ns-3 and it is about to be transmitted or the transmission is in progress.
kArrived	The transmission was successful and the information reached the destination.
kToBeApplied	The Application requests the iCS that the value attached to the messages should be applied in the corresponding environment.
kToBeDiscarded	The Application requests the iCS to discard the result and not to make use of the value.
kMissed	The transmission was unsuccessful and the information did not reach its destination. (Note: This status is not currently supported by the iCS)

These states are common between iCS and the application simulator. In the current implementation, it is therefore not possible to add new ones without the iCS being also modified.

Extensions within COLOMBO

Figure 3.21 describes a typical application logics that would use the previously described states integrated in the Application simulator lifecycle. All steps may be directly observed on the figure (indicated by numbers), but we may briefly mention the following indications for designers related to the COLOMBO project:

- Not all states need to be used, and we may jump to various states, for instance when the application schedules a subscription to send data by ns-3, we may implicitly directly move to the state *kScheduled*.
- COLOMBO will rely heavily on the interaction between multiple application simulators. The interaction will take place at:
  - *first lifecycle* – request data from other modules via a cross-application subscription
  - *third life-cycle stage* – requested data from other application modules will be delivered.
  - *last life-cycle* - An application module is able to share its data to other modules. After completion of its logics, the application will directly transfer the result to the result container located in the iCS.

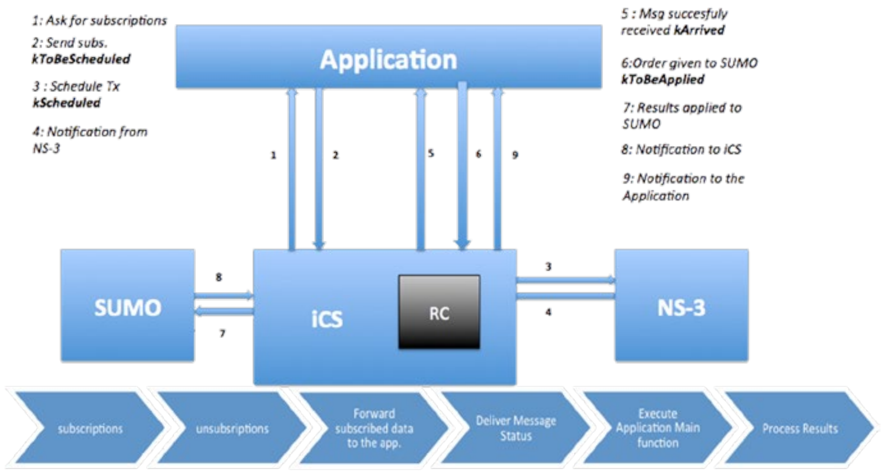

1: Ask for subscriptions
2: Send subs. kToBeScheduled
3 : Schedule Tx kScheduled
4: Notification from NS-3
Application
5 : Msg succesfuly received kArrived
6:Order given to SUMO kToBeApplied
7: Results applied to SUMO
8: Notification to iCS
9: Notification to the Application
1
2
5
6
9
SUMO
8
7
iCS
RC
3
4
NS-3
subscriptions
unsubsriptions
Forward subscribed data to the app.
Deliver Message Status
Execute Application Main function
Process Results

Figure 3.21: Typical application logics

3.6 iCS - iTETRIS Control System

Tasks and Implementation

The iCS is an interface interconnecting various modules or simulators via socket APIs. It is simulator agnostic and can be interfaced with different simulators by adapting the simulators to the open APIs. The iCS provides three APIS to interconnect to a traffic simulator, a network simulator, and an application simulator. In the current implementation, iCS interconnects SUMO, ns-3 and application modules. The iCS contains a

- **TraCI client:** to interconnect SUMO and iCS.
- **INCi client**: to interconnect ns-3 with iCS.
- **AppCI server:** to support the connection of multiple application modules to the iCS

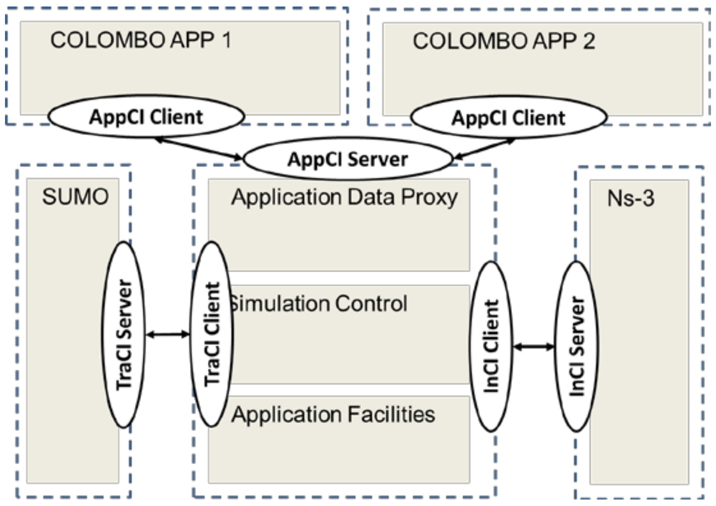


Figure 3.22: socket based interfaces provide by the iCS

The iCS in turn will then play the role of an interface between all simulators so that SUMO and ns-3 may exchange data or instructions between each other. Yet, the iCS is more than a simple interface. Figure 3.23 depicts the architecture of the iCS and shows that it contains more than synchronization functions:

- **Simulation Control:** As in all V2X simulation architectures, the iCS controls the simulation first by keeping all simulators synchronized and second by triggering actions in different modules. Moreover, in order to univocally identify the same stations on the different simulators, an identity management system is implemented.
- **Application Facilities:** The iCS contains the application-related facilities of the ITS stack, and accordingly all their functionalities.
- **Applications data proxy:** The iCS finally manages the data exchange between different applications.

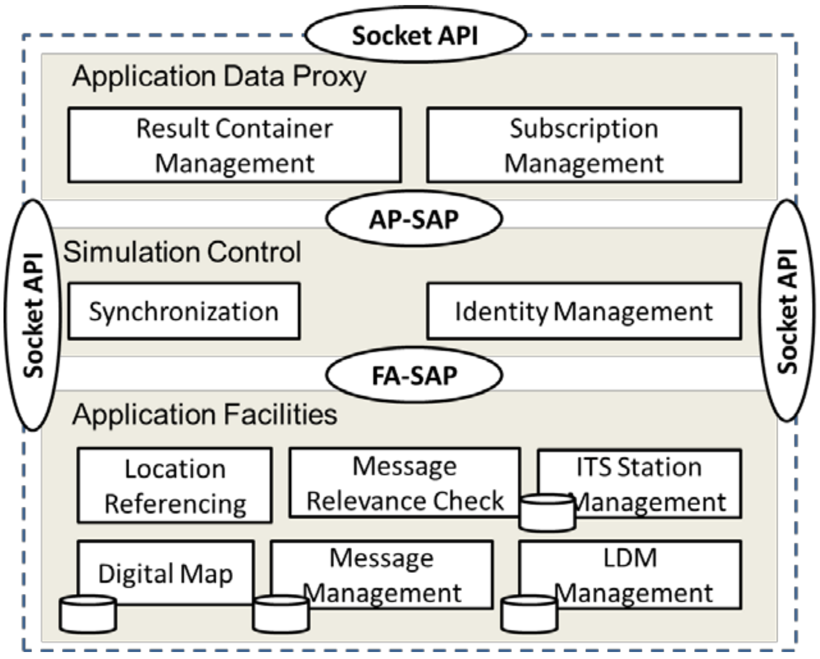


Figure 3.23: iCS Architecture

The iCS Life Cycle

The iCS plays the role of synchronizer between the various simulators, and represents the heart of COSS. It also synchronizes the various time granularity and time steps of the various modules. When starting, the iCS starts ns-3 and SUMO on separate processes. The iCS then opens a socket connection to each of the modules and proceeds to their initializations. Once completed, iCS will activate the various modules. iCS yet not activates all modules at the same time, but cycles between modules as illustrated in Figure 3.24. It first activates ns-3 and proceeds to the communication steps for one iCS time step. Then, it activates SUMO to move vehicle for one SUMO time step. It further calls each of the various Application modules connected on the AppCI server and activates their respective application logics for one application step. Finally, it runs facilities and controlling logics to prepare the next step.

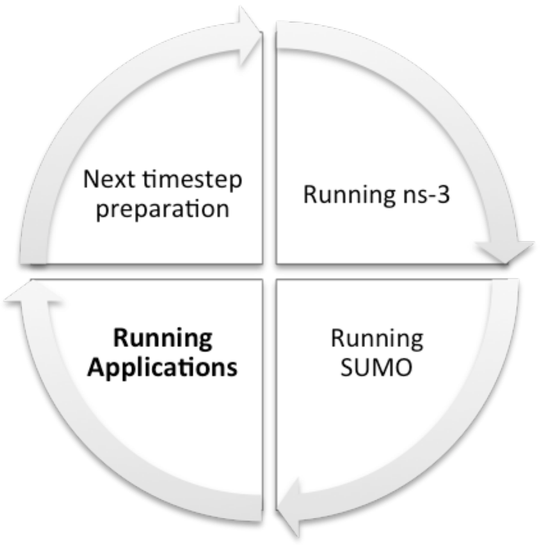


Figure 3.24: iCS Life Cycle

The key note of the interaction of the iCS with other modules, is that it first simulate communication, then apply mobility, before running the application logics, which requires joint mobility and communication updates for their internal logics.

Implementation and Interfaces

Inter-module interaction through iCS

The iCS contains mechanisms for applications to interact with each other as well as with other modules (SUMO, ns-3). This mechanism is called "**subscription**". Conceptually speaking, an application logic will subscribe to a service of the iCS for a special purpose, such as retrieving position from SUMO, commanding the transmission of messages to ns-3, or even getting data from other application logics or modules. The iCS has two types of subscriptions, called hereafter 'legacy' or 'generic'. In practice, these subscriptions may be jointly used, but the best practice would suggest to rely as much as possible on the 'generic' type of subscription.

Legacy Subscriptions

For the purpose of the different ITS applications developed for the ICS, various subscriptions have been developed. The two major drawback of the legacy subscriptions are first that they have been developed for a specific service (like sending a specific type of message or retrieving a specific type of information from SUMO), and an extension of their purpose would need to modify their code in the iCS. Second, they are tightly bound to the application logics, and require partial application logics to be implemented in the iCS. Nonetheless, some legacy subscriptions remain very crucial for any ITS applications and could be used in conjunction with the new generic type of subscriptions. Table 3.2 lists the some generic subscriptions that could also be used by COLOMBO.

Table 3.2 Subscriptions available to COLOMBO

Defined Name	Description
SUB_RETURNS_CARS_IN_ZONE	This Subscription returns the mobile stations that are currently present over a configurable area according to the information stored in the iCS Facilities.
SUB_SET_CAM_AREA	The Applications can set a CAM transmission area with this Subscription. The area defines potential reception coverage of CAM messages from a particular station (the subscriber) by the stations inside of the area. They are used by the iCS to minimize the simulation time and the computational resources needed by ns-3. This Subscription does not return any kind of data.
SUB_RECEIVED_CAM_INFO	This subscription is set to retrieve the data contained in received CAM messages.

Generic Subscriptions

In order to add more flexibility to the application modules to use iCS subscriptions, the generic type of subscriptions have been developed. The major difference between legacy and generic subscriptions are that the latter are providing a generic link between the application and a module, where the precise task is only known between the application and the module itself. For instance, a generic subscription between SUMO and an application would require to indicate in its parameters what kind of data you need (speed, position, traffic light status, etc..) and as long that the target module understand your command, the information will be transferred, without the iCS being aware of the content. This approach decouples totally the application logics from the iCS. We illustrate the generic services provided by the iCS to applications to interact with various modules in Figure 3.25.

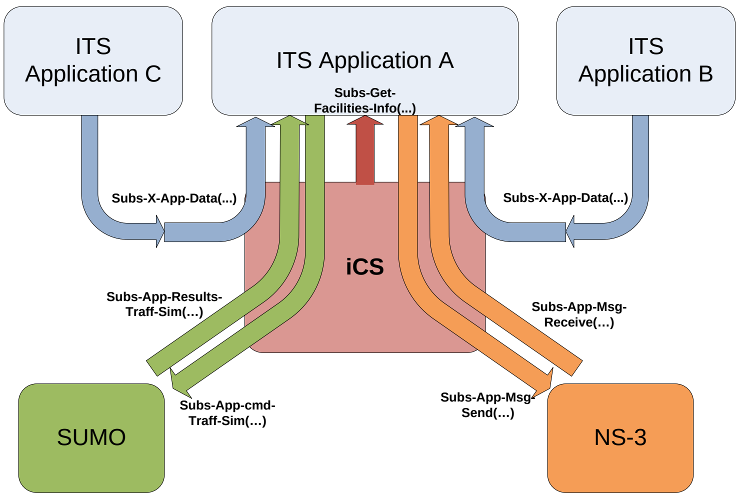


Figure 3.25: iCS's generic subscription scheme

As it may be observed, the iCS provides two types of generic services per module (Push, Pull). One exception is for the Facilities information service, where only a Pull service is provided, as the current implementation of the iCS assumes that only SUMO or ns-3 may feed the application facilities with data. Another iCS service is to interlink various application modules, where one application may provide data required by other applications. This subscription will be used in COLOMBO, as it will provide a mean for the traffic surveillance system to provide consolidated traffic information to the traffic light applications. This subscription is based on another service of the iCS, which is called **Result Container**, which locally stores in the iCS result data from applications. In order to avoid designers to have to know the functionality of the result containers, they have a generic storage capability, where a '*key*' will link to a given '*data*'. The combination between a 'key' and the 'data' format is only known to the two applications.

We list in Table 3.3 the available generic subscriptions that will be used for COLOMBO.

Table 3.3: Generic Subscriptions

Defined Name	Description
SUBS_GET_FACILITIES_INFO	This Subscription returns requested data available at the facilities layer. Currently supported data types are: Awareness data from either CAM or from topological map.
SUBS_APP_CMD-TRAFF-SIM	The Applications can command the traffic simulator to conduct a particular action as specified in the subscription's parameter field. Currently supported commands are re-routing, changing speed, assigning road weights.
SUBS_APP_RESULT-TRAFF-SIM	The Applications can request data from the traffic simulator for its internal logic as specified in the subscription's parameter field. Currently supported are current route and road weights.
SUBS_APP_MSG_SEND	This is a generic subscription for Applications to command the communication simulator to send messages. The type and configuration of the message is specified in the subscription's parameter field. Currently supported transmissions are UNICAST, GEO_UNICAST, GEO_BROACAST and TOPO_BROADCAST.

SUBS_APP_MSG_RECEIVE	This is a generic subscription for applications to be notified about received messages. Depending on the subscription's parameter field, it can select to only receive message from or to a particular source/destination or be ubiquitous. Note that for receiving messages transmitted by the SUBS_APP_MSG_SEND subscription, a SUBS_APP_MSG_ RECEIVE MUST be scheduled.
SUBS_X_APP_DATA	This subscription allows the exchange of data between different applications using the target application's result container. The type of data is specified in the subscription's parameter field.

Extensions within COLOMBO

COLOMBO Module Interconnection

Within COLOMBO, several external modules will be developed: traffic surveillance, traffic light, algorithm tuning, etc.. We describe here how they will be interconnected with the iCS and the COSS, as well as the various interfaces required to exchange data between the modules and COSS.

Figure 3.26 depicts how the modules will be integrated in COSS. Each module will be considered as an application, even though some modules may not have any application logics in it.

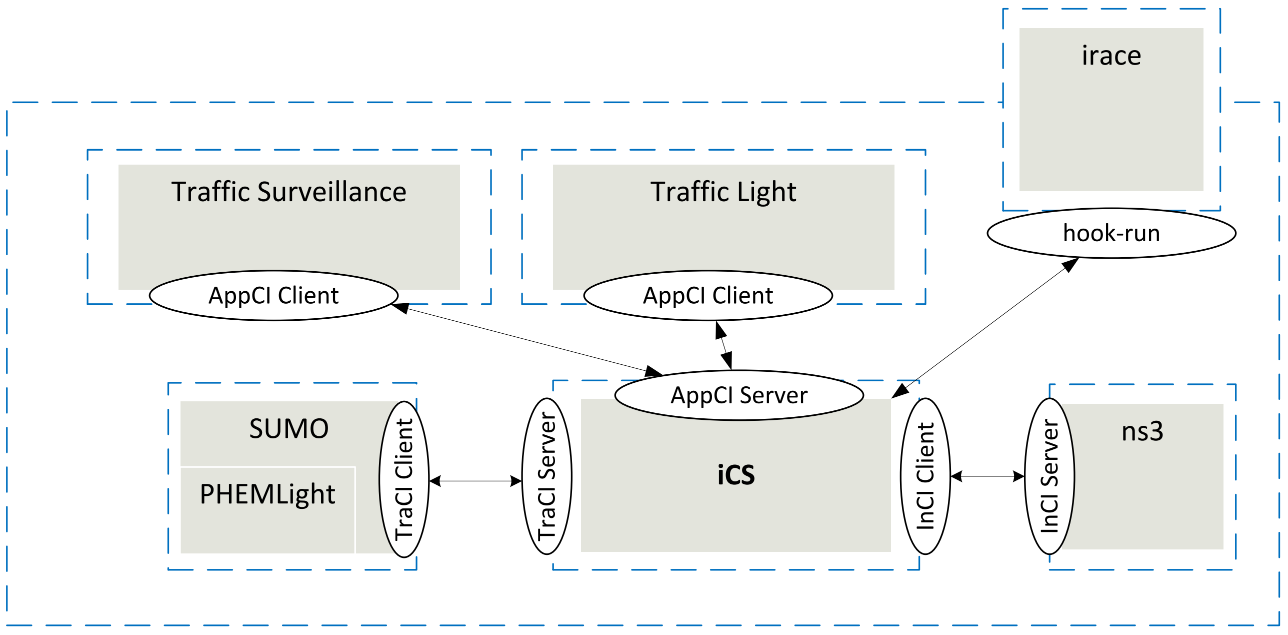


Figure 3.26: Example of COSS interfaces

Interaction to the Traffic Surveillance Application

In order for the Traffic Surveillance module to extract and consolidate traffic densities, it can interact with ns-3 via a *Pull* ns-3 subscription using `<keys>` corresponding to the required information, or may also *Pull* other modules as well as its own previously saved traffic volume to update them. The traffic surveillance will identify uniquely traffic densities in various positions along road segments by 'virtual' sensors, and a unique `virtual sensor_ID`.

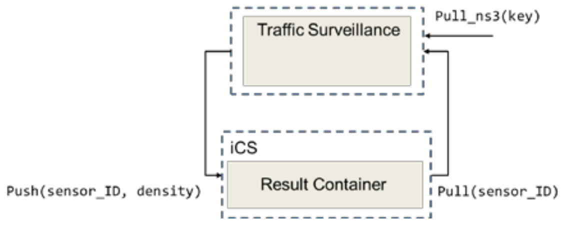


Figure 3.27: Detector Data handling of the iCS

The density indicated in the drawing represents a density structure, composing modular sub fields. To maintain flexibility, such density structure will be encoded according to a `type-length-value` structure:

```
<sensor_ID>::<density><length><position><length><xyz><volume><length><val
ue><time><length><value>…
```

Interaction to the Traffic Light Algorithms

The intelligence of the traffic light control systems is related to the knowledge of the traffic conditions. The traffic light module relies on the iCS cross-app subscriptions, where it would *Pull* data from sensor_IDs previously provided by the Traffic Surveillance module.

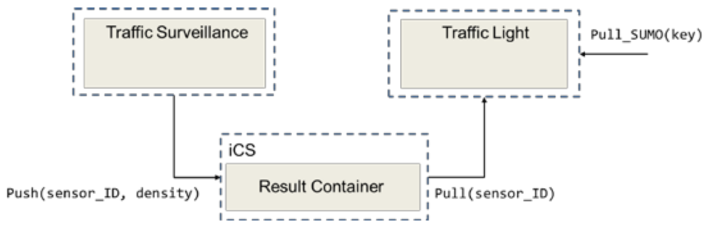


Figure 3.28: Interaction with the Traffic Light Algorithms and the Traffic Surveillance Application

The Traffic Light Module also sends to the Result Container the configuration of its Traffic Light Controllers (TLC), which could be later used by SUMO. The Traffic Light module will uniquely identify the parameters of a Traffic Light Controller (TLC) by a `TLC_ID`.

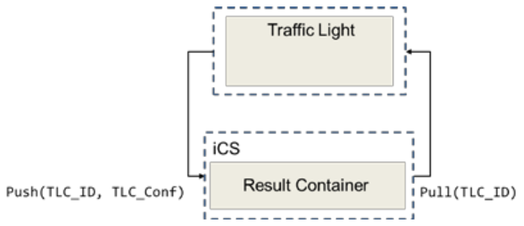


Figure 3.29: Result handling by iCS

The TLC_Conf represents a structure containing the configuration parameters of a TLC and contain various modular sub fields. To maintain flexibility, such configuration structure will be encoded according to a `type-length-value` structure:

```
<TLC_ID>::<TLC_Conf><length><flow_ID><length><phase><length><time><color>
```

The precise description of the structure will be further adjusted upon implementation and in cooperation with the partners.

4 COLOMBO Applications

COSS is used in later project steps to allow the evaluation of V2X-applications, which are the project's main focus. This chapter describes the V2X-applications planned to be developed or that are under development within COLOMBO one by one, mainly showing the needed interfaces by listing

- which data is read by the applications, how it is processed, and what data is written
- which COLOMBO components are needed and how they interact with the application

It should be noticed that the development of most COLOMBO applications starts at later times of the project. Changes to the described interactions with COSS have to be expected, but the described needs are considered to be close to the final implementations.

Two expected project results are not listed here as they are no V2X-applications even though COSS may be used during their generation: the functional education tool kit (developed in work package 6, see [COLOMBO, 2012], page 24) and the guideline on developing eco-friendly traffic light controls (developed in work package 4, see [COLOMBO, 2012], page 17).

Within the following sections, abbreviations for component names are used as following:

- iCS: the middleware iCS
- ns-3: the communication simulation ns-3
- SUMO: the traffic simulation SUMO
- TT: the tuning tool kit / configuration tool
- PHEM: either PHEM coupled offline or PHEMlight as embedded in SUMO
- TSES: Traffic State Estimation System
- EMS: Emission Monitoring System
- EDB: Emission-optimal Driver Behaviour

4.1 Cooperative Traffic State Evaluation System

COLOMBO's work package 1 [COLOMBO DoW, page 13] will develop *a cooperative traffic state evaluation system, based on traffic state extrapolations, data gathering and fusion, including cooperative data dissemination between vehicles and RSUs*. In the following, one of the approaches followed to design such a system is given. Being one of the major project goals, the development of the traffic state evaluation system will follow other approaches as well.

The popularity of wireless devices and new wireless-based positioning methods and systems, spanning from completely decentralized ones based on heterogeneous wireless technologies, such as IEEE 802.11 (WiFi) and WiFi direct to Bluetooth (BT), to more precise ones, such as low-/high-precision GPS systems, are stimulating new location-aware traffic state extrapolation scenarios. The new COLOMBO cooperative traffic state evaluation system shows its main benefits when applied in areas that are populated by a sufficient number of vehicles; providing experimental evidence of a lower bound over the density of vehicles in order to make the data gathering and fusion process effective is one of the main goals of COLOMBO. The proposed system presents two core advantages. It does not require any pre-installed localization infrastructure and is resilient to potential damages of the fixed network infrastructure: i) it automatically and continuously re-organizes Vehicular Ad-hoc NETworks (VANET) impromptu realized by wireless-equipped vehicles in the proximity of the intersection. On the other hand, it takes advantage of dense VANET population (rather than considering it a technical obstacle) to grant a high localization precision at any time: it proposes a fully decentralized location-aware technique to precisely demarcate one or more regions in the intersection approach before the actual junction, called in the following Approach Markers (AMs); ii) it exploits AMs to trigger traffic evaluation protocols over traversing vehicles, and to fuse together and store evaluated traffic metrics.

With a closer view on details, the purposes of the proposed system are threefold. First, we want to effectively cluster vehicles approaching to the intersection in a few groups: each group (as a single entity) can track its average speed, direction, composition, and size. To counteract group mobility, our proposal opportunistically chooses for each group a cluster-head, called *marker vehicle* or simply *marker*, as the centremost vehicle is likely to be reachable by all vehicles in the group; in addition, it autonomously re-organizes the group VANET topology in case of marker leaving the group. Second, it is proposed to fully decentralized, approximated, and lightweight localization techniques (based on Received Signal Strength Indication - RSSI, as well as on more precise triangulation and GPS location methods, infrastructure-less, and anchor-free) to locate the marker and to choose vehicles to join the group. Third, our solution exploits geographical-based and gradient-based routing solutions to deliver main traffic state parameters estimated for each group toward the RSU deployed at the intersection.

The needed information, collected and fused by using the solution briefly sketched in the paragraphs before and more detailed in the following, come from the RSU and three main different entities:

- RSU (Road-side unit):
  - no limit on power consumption or computational capabilities;
  - defined location;
- Class A (shadow vehicle):
  - no communication;
  - invisible by other vehicles and RSUs;
- Class B (PDA/smart-phone aboard a vehicle):
  - limited to medium communication capabilities;
  - low-to-medium location accuracy: low-precision GPS and Direct WiFi RSSI;
- Class C (vehicle on-board system):
  - full communication capabilities;
  - medium-to-high location accuracy: high-precision GPS and IEEE 802.11p RSSI as well as Direct WiFi;
  - loose constraints on power consumption.

Needed information includes:

- RSU position that comes from SUMO;
- Class B positions: precise location information that comes from SUMO and then, within ns-3, is modified adding some noise and inaccuracies to emulate low-to-medium GPS location accuracy;
- Class B RSSI estimations: needed for the execution of the grouping protocols and to estimate reciprocal positions between group members; these estimations are obtained within ns-3, by grabbing them from the low physical and data link layers and making them available to the ns-3 application layer.
- Class C positions: precise location information that comes from SUMO and is used within ns-3 to emulate high-precision GPS (some noise and inaccuracies could be added in ns-3 as for Class B entities, if needed);
- Class B RSSI estimations: needed for the execution of the grouping protocols and to estimate reciprocal positions between group members; these estimations are obtained within ns-3, by grabbing them from the low physical and data link layers and making them available to the ns-3 application layer.

In order to process that information we are prototyping a new VANET grouping solution and data fusion algorithms that follow three main design directions: we favour simplicity over precision; we exploit simple broadcast-based protocols aimed at grouping vehicles approaching to traffic lights; we propose new heuristics to evaluate traffic density.

Direction detection

First, we derive the vehicles' direction: in fact, no interest in communicating with vehicles following different paths; moreover, vehicles are grouped and organized based on their direction because:

- direction is to perform correct data fusion (only for the vehicles in the same interaction approach);
- direction allows to design and perform group communications for vehicles in the same group and following the same direction;
- direction is obtained through consecutive position sampling over time.

For instance, in Figure 4.1 yellow cars are travelling toward a different direction than green ones. On the one hand, vehicles can manage their communication using their position, speed, and direction; on the other hand, RSUs can differentiate traffic flows using their (average) direction.

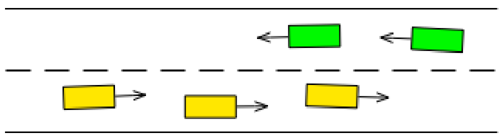


Figure 4.1: Distinguishing directions by averaging vehicle directions

Group creation

Nodes are grouped together if sufficiently close and for each group (considered as a single entity), we track its average speed, direction, composition (Class B and Class C entities), and size. Hence, there is the need for simple protocols to manage group creation, deletion, and management. We foresee the group lifecycle as consisting of the following main steps:

1. group creation when vehicles are in the approach are in proximity of an RSU;
2. traffic metrics collection, within the group and coordinated by the cluster-head, as the marker approaches to the RSU;
3. marker sends traffic flow estimations to RSU when prompted.

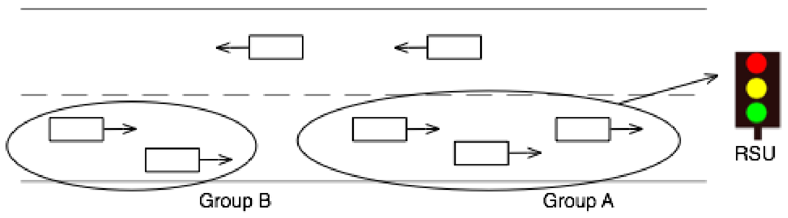


Figure 4.2: Grouping of vehicles

One of the main issues is that in order to increase the reliability of the data collection and fusion process, vehicles should start to create groups as soon as possible when they are approaching an RSU; at the same time, in order to avoid useless communication and increase communication scalability vehicles should be grouped only when there is an RSU in proximity. Possible solutions to this problem include extending RSU beaconing to support:

1. formation of groups of larger size:
- need for multi-hop communication;
- more complexity and overhead;

2. election of marker nodes.

Following our main design guidelines, we opt for the second one; the main idea is to extend RSU range by exploiting far nodes to actively anticipate group creation as shown in Figure 4.3.

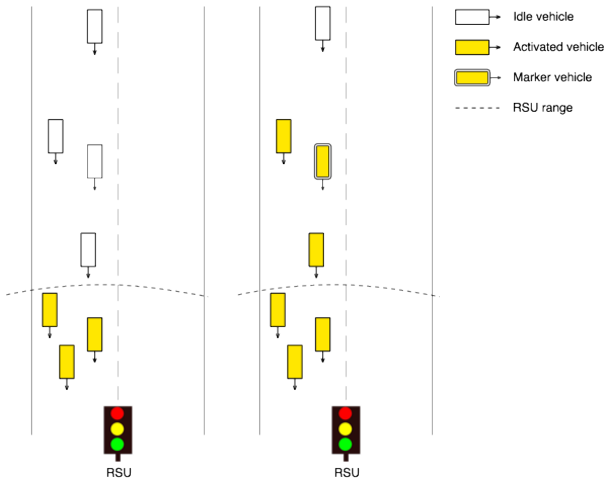


Figure 4.3: Extending the surveillance edge by incorporating vehicles

In order to do that, we are studying a protocol that works as follows by including 5 main execution steps:

1. when approaching to an RSU, a (preferably far) node is elected as marker;
2. as the marker gets near to the RSU, it prompts for a switch-off;
3. queried nodes reply;
4. the most desirable node (more central to the group) is selected and acknowledged by the old marker;
5. acknowledged node becomes new marker.

This process can be replied to obtain a *multi-level approach* that create multiple groups at increasing distances from the RSU, by electing (and re-electing) for each of them one marker.

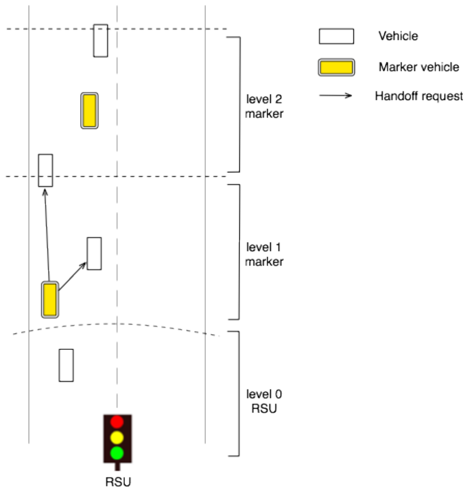


Figure 4.4: Extending the surveillance edge by incorporating vehicles

Initially, the vehicles in the RSU range elect a marker that tries to propagate its role to level 1, then 2 and so on. As the marker senses it is quitting its level, the role of marker is passed to another vehicle in the attempt to re-establish a marker for that level (we call this operation handoff). An open question is the evaluation of the minimum traffic density that allows the proposed protocol to work reasonably well. Finally, the measurement of relative distance between nodes is another critical point both for the grouping and the marker protocols. Among the research ideas we are considering to use: GPS (not very accurate), receiver SNR (Signal-to-noise ratio), inter-node triangulation, and a combination of these.

In any case, the output of the cooperative traffic state evaluation system will include, for each detected group, an estimation of the following parameters:

- group size and vehicle types;
- average speed;
- average direction;

Derived traffic density measurements for the different directions will made available to the traffic light algorithm module, via iCS.

Simulation Concept

During the first development steps, basic algorithms can be evaluated using small scripts, which use vehicle trajectories obtained from SUMO, optionally pass them to ns-3 for obtaining the exchanged messages, and try to resemble the traffic state based on these messages. In this case, the surveillance quality can be measured by comparing the outputs to direct simulation outputs of the same type – for example average travel times. This attempt is depicted in Figure 4.5.

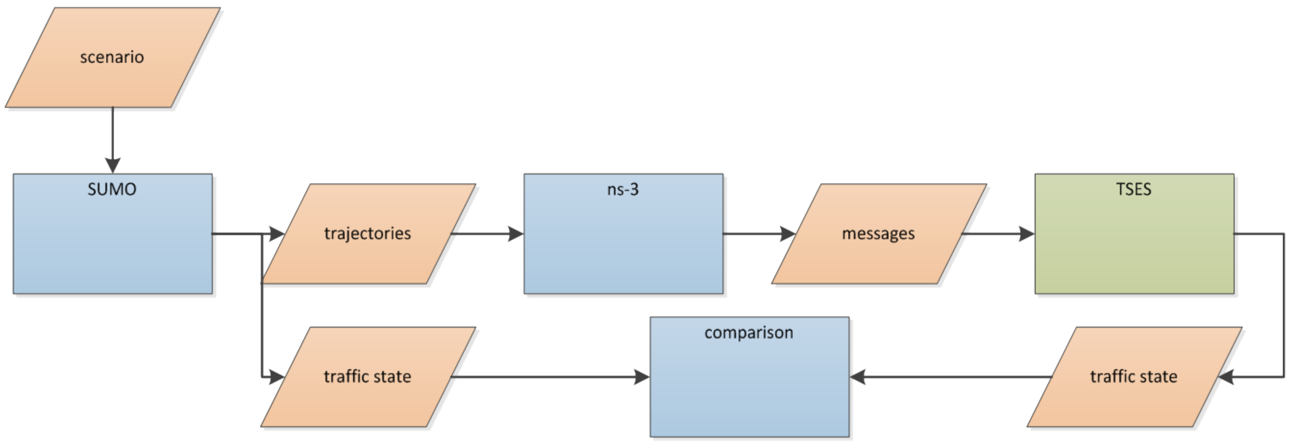


Figure 4.5: Simple, initial set-up for algorithms development

At a later time of the project, the traffic state computed by the surveillance system must be made available to the traffic light controls algorithms. For this purpose, the surveillance system must be implemented as an iCS application. This application receives information about exchanged messages from ns-3 via the iCS and uses them to compute the regarded traffic values. These values are then send back to the iCS and stored there for further use. The traffic lights applications is then able to ask for them, and performs its operations based on them.

Resulting, TSES will use all COSS components but PHEM / PHEMlight, see also Figure 4.6. PHEM / PHEMlight is not needed as TSES does not influence the traffic so that no changes in the emissions will take place and also no emission measurements are needed for traffic state estimation.

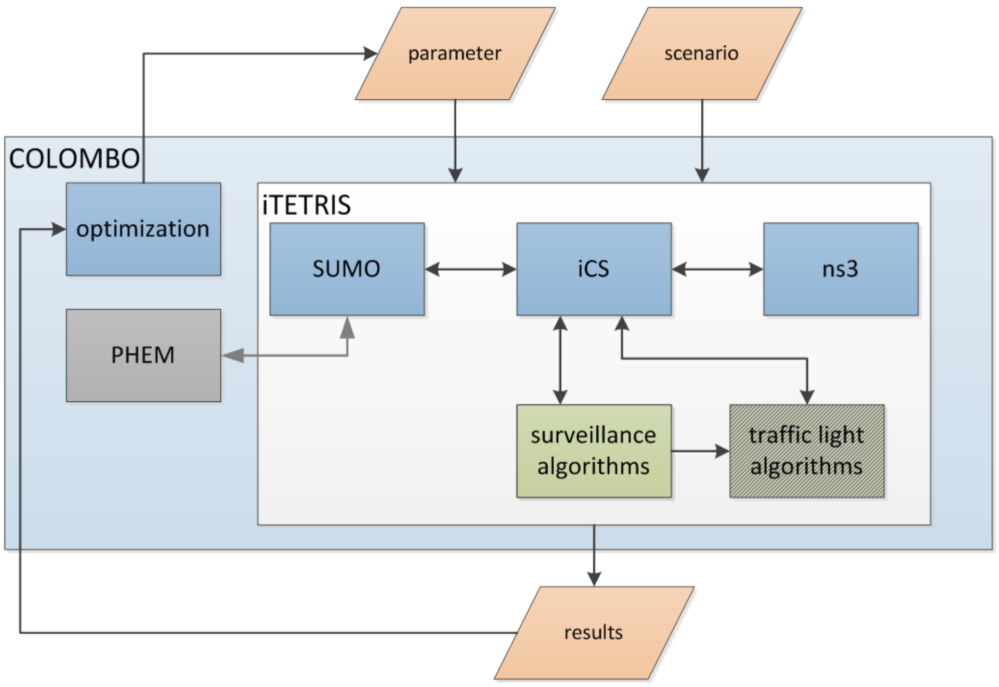


Figure 4.6: COSS Components used by the Traffic State Evaluation System

Requirements to COSS

The requirements to the COSS components are summarized in Table 4.1. Here, we use the requirements of the system's final implementation, where it runs in combination with the traffic light logics. The abbreviation "TSES" is used to denote the Traffic State Evaluation System. Please note that for keeping the table at honest length, we do not list the requirements that form the main responsibility of iCS – selecting vehicles equipped with V2X technology, passing vehicle positions to ns-3, triggering the ns-3 to simulate V2X messages, being informed by ns-3 that a simulated V2X message was received, etc.

Table 4.1: Mandatory Requirements from the Traffic State Evaluation System (grey: already available)

Component / Application	Requirement ID	Description
SUMO	TSES.SUMO.1	SUMO must be able to deliver the information about a vehicle's speed
	TSES.SUMO.2	SUMO must be able to deliver the information about a vehicle's driving direction (angle)
	TSES.SUMO.3	SUMO must be able to deliver the information about a vehicle's position
	TSES.SUMO.3	SUMO must be able to deliver the information about a vehicle's signal status (blinker, brake lights)
	TSES.SUMO.4	SUMO must be able to save the aforementioned information into a file
iCS	TSES.iCS.1	iCS must be able to read information about a vehicle's speed from SUMO
	TSES. iCS.2	iCS must be able to read information about a vehicle's driving direction from SUMO
	TSES. iCS.3	iCS must be able to read information about a vehicle's position from SUMO

	TSES. iCS.4	iCS must be able to read information about a vehicle's signal status from SUMO
	TSES.iCS.5	iCS must be able to send information about a vehicle's speed to TSES
	TSES. iCS.6	iCS must be able to send information about a vehicle's driving direction to TSES
	TSES. iCS.7	iCS must be able to send information about a vehicle's position to TSES
	TSES. iCS.8	iCS must be able to send information about a vehicle's signal status to TSES
	TSES. iCS.9	iCS must be able to retrieve information about the traffic states from TSES
	TSES. iCS.10	iCS must be able to store information about traffic states
	TSES. iCS.11	iCS must be able trigger a node's communication via WiFi-direct within ns-3
	TSES. iCS.12	iCS must be able to retrieve information about simulated WiFi-direct message reception from ns-3
	TSES. iCS.13	iCS must be able to send retrieved information about simulated WiFi-direct messages to TSES
	TSES. iCS.14	iCS must be able to select vehicles for being equipped with a V2X device or a WiFi-device, more than one of the latter can be found in a simulated node
ns-3	TSES.ns3.1	ns-3 must be able to simulate WiFi-direct communication
	TSES.ns3.2	ns-3 must be able to obtain commands from iCS that start WiFi-direct communication for a given node
	TSES.ns3.3	ns-3 must be able to inform the iCS about the simulated reception of WiFi-direct messages
TSES	TSES.1	TSES must be able to retrieve information about retrieved simulated V2X messages
	TSES.2	TSES must be able to retrieve information about retrieved simulated WiFi-direct messages
	TSES.3	TSES must be able to submit computed traffic states to the iCS

Needed Scenarios

TSES is developed for being deployed at single intersections as an application running on a road-side unit (RSU). Both controlled and uncontrolled intersections should be used. For determining how the system behaves in larger contexts and for having the possibility to compute travel times between intersections, scenarios covering more than one intersection are needed.

The scenarios must cover different traffic demands, different in the amount of vehicles, their types, and their rate of being equipped with V2X and with WiFi-direct devices. The scenarios must include pedestrians and bicycles.

4.2 Intersection-based Emissions Monitoring System

Within COLOMBO's work package 1 [COLOMBO DoW, page 13] *a local (intersection-based) emissions monitoring system* will be developed. The system is meant to be deployed on an intersection and to compute the amount of currently emitted pollutants, where "currently" means a defined time span. In the following, the local emissions monitoring system will be abbreviated by "EMS".

The basic technological idea is to combine single vehicle trajectories obtained from vehicular communication with additional information about the traffic flow collected via conventional detectors (e.g. inductive loops). Assuming a low number of communicating vehicles, both information sources are needed. The obtained trajectories support the system with knowledge about common acceleration and deceleration profiles at the intersection the system is deployed at. Information about acceleration profiles is needed as it influences the amount of emissions more than the speed. Conventional detectors are not capable to measure accelerations, but they are needed to determine the number of passing vehicles, which can be hardly obtained via vehicular communication if a low penetration rate is assumed. By now, it is planned to use V2X CAM ("Cooperative Awareness Messages") as the only vehicular communication source of information about trajectories. The major technical problem is the correct fusion of both data sources.

Simulation Concept

EMS can be evaluated using a scenario with just one intersection. Albeit V2X communication is used as an input, no feedback to the traffic simulation is needed, as the EMS is only evaluating but not influencing traffic. As a result, EMS does not require the usage of iCS. It is assumed, that during the development of the EMS, no algorithm tuning and no optimization is needed, albeit this is the most probable extension of the simulations to be performed within the EMS development.

This means that for simulating the EMS, only the following components are needed (see also Figure 4.7):

- SUMO for simulating traffic at the intersection
- PHEM/PHEMlight for computing the emissions
- optionally the tuning tool kit for changing algorithms, e.g. for data fusion

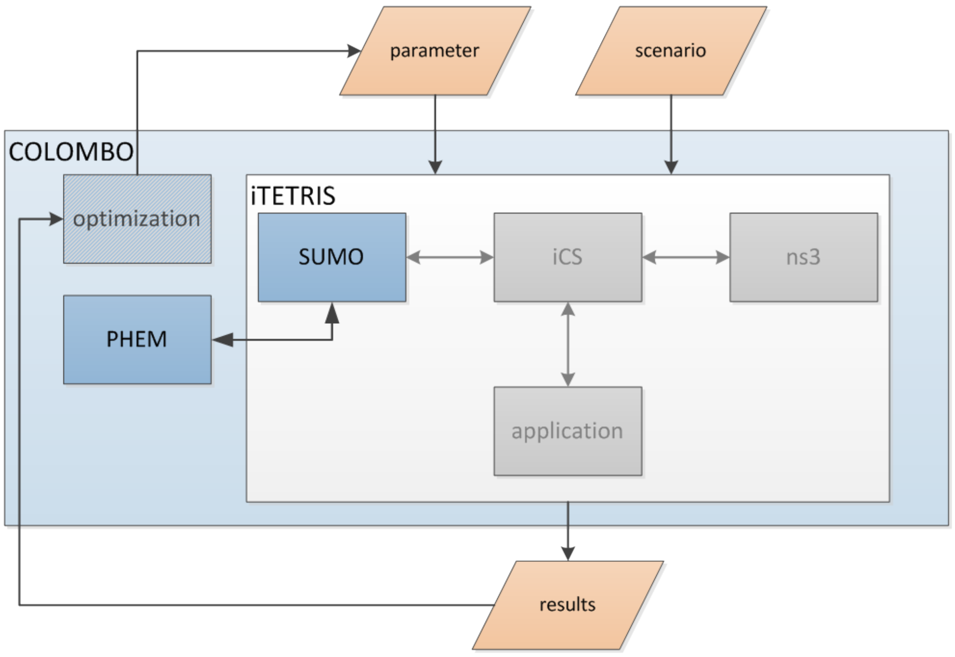


Figure 4.7: COSS Components used for EMS development

Figure 4.8 depicts the data flow across the applications used to simulate EMS. It can be summarized as following: the traffic simulation delivers measures from simulated conventional detectors as well as vehicle trajectories to the EMS. In parallel, it computes the "correct" emissions using PHEM/PHEMlight and saves them for a later validation of the results obtained from EMS.

EMS obtains the conventional detector measures and trajectories and applies filters that resemble the equipment rate of vehicles equipped with V2X technology and optionally other information quality reductions, such as GPS location noise that would be present in real life CAM, to resemble V2X communications properly to the latter. Given these inputs, the EMS performs data fusion algorithms on the filtered trajectories and the conventional detector measures, first. This may also be done beforehand by an additional program. Given these inputs, the EMS computes the amounts of emitted pollutants, probably using PHEM/PHEMlight again.

The amounts of emitted pollutants computed by EMS using incomplete knowledge is stored or kept in the memory and compared to the "correct" emission amounts that were previously generated by the traffic simulation/PHEM using a complete knowledge.

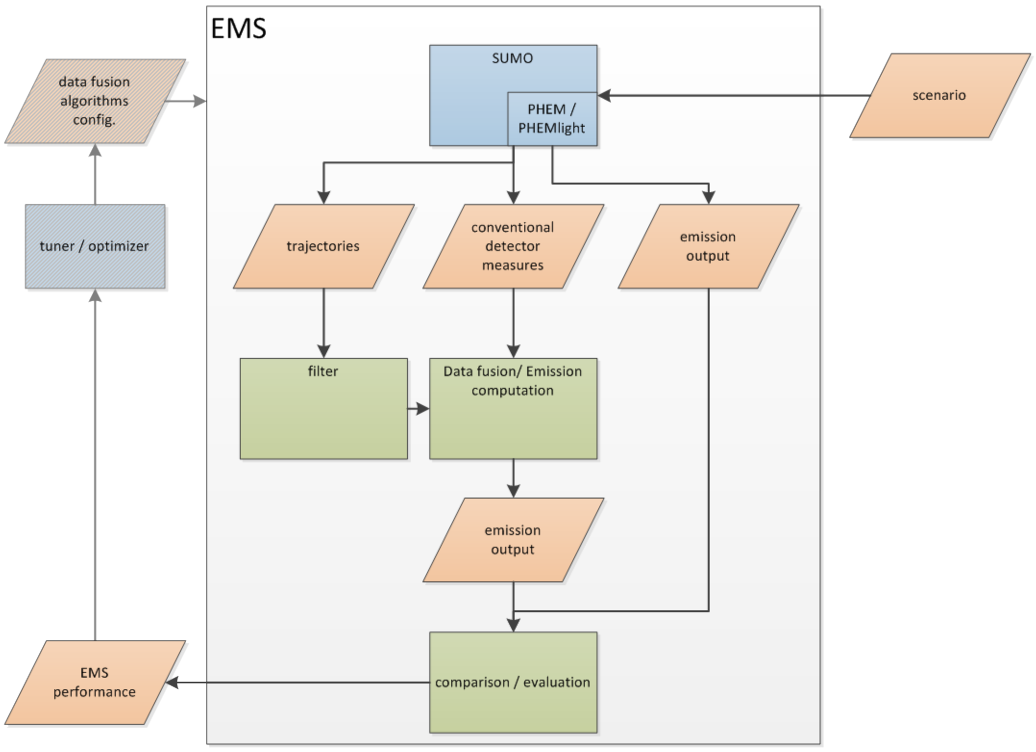


Figure 4.8: Data Flow of the Emissions Monitoring System Simulation

Requirements to COSS

The requirements to the COSS components are summarized in Table 4.2.

Table 4.2: Mandatory Requirements from the Intersection-based Emissions Monitoring System (grey: already available)

Component / Application	Requirement ID	Description
SUMO	EMS.SUMO.1	SUMO must be able to simulate (controlled) intersections
	EMS.SUMO.2	SUMO must be able to represent different vehicle types
	EMS.SUMO.3	SUMO must generate vehicle trajectories
	EMS.SUMO.4	SUMO must simulate conventional detectors (no errors are assumed) and write their values
PHEM	EMS.PHEM.1	PHEM / PHEMlight must be able to process trajectories obtained from SUMO and to save computed pollutant emissions
EMS	EMS.1	EMS must be able to read SUMO trajectories
	EMS.2	Filters that resemble the amount of equipped vehicles and information inaccuracy must be applicable to the read trajectories
	EMS.3	EMS must be able to read SUMO's representation of conventional detector measures
	EMS.4	EMS must compute and save emission measures
	EMS.5	EMS emission measures must be able to be compared against the ones obtained from SUMO/PHEM directly (EMS.SUMO/PHEM.2)

Needed Scenarios

EMS is developed for being deployed at single intersections as an application running on a road-side unit (RSU). Simulation scenarios should cover an intersection and incoming lanes should be long enough for tracking vehicles within the RSU's communication range. Different demands – changing in the amount of vehicles and the rate of being equipped with V2X technology should be investigated.

4.3 Traffic Light Algorithms

COLOMBO's work package 2 [COLOMBO DoW, page 17] will develop *a self-organizing, adaptive, distributed and monitoring-aware approach to traffic light control*. The main goal of the traffic light algorithms is to implement a control system that is efficient, highly scalable and that is able to react in real time to the sensed traffic conditions. These goals are achieved by means of a distributed system, where each traffic light controller, controlling a junction, is independent of the others and gathers only local traffic information. Based on this local knowledge, the traffic light algorithm regulates the duration and succession of the different signal cycles of the controlled junction, reacting rapidly to changes in the traffic conditions.

In particular, a high-level policy reacts to the overall measured level of traffic at the macroscopic level, and selects a low-level policy. This microscopic level policy reacts to more rapid changes, such as vehicles waiting on a lane or even absence of vehicles from a direction. This approach allows the automatic adaptation to events that change local traffic conditions, such as road works that limit or otherwise change the capacity of a road and increase the flow on another path. Automatic adaptation ensures handling of the mutated conditions even for short periods, such as the temporary changes in traffic due to an accident or any other unpredictable event, all without requiring any reprogramming of the traffic light control.

This approach views traffic as a complex system and each traffic light controller as an autonomous agent. Studies on this kind of systems show that synchronization and overall coordination emerge as global properties of the network, even without any explicit communication between the controllers/agents.

Operation of these algorithms requires information from the infrastructure regarding the current traffic conditions and the state of the controlled traffic lights. Each time where the algorithm is invoked to decide what the next state of the traffic light should be, it needs a measure of the vehicles on each incoming lane or group of lanes and a measure of the congestion level on outgoing lanes, that can be for instance estimated as the difference between the maximum allowed lane speed and the average measured lane speed. Currently, the information is gathered directly from SUMO and perfect knowledge is assumed.

Similarly, the algorithm needs an estimate of the presence of pedestrians at each regulated pedestrian crossing that can be obtained from direct pedestrian requests or through sensing. Knowledge of the current traffic light state is assumed to be perfect and its change instantaneous.

The estimates of the presence of vehicles and pedestrians are processed both by the macroscopic-level policy and by the currently selected microscopic-level policy. The macroscopic level policy uses the aggregate information on the incoming traffic and the congestion of the outgoing traffic in order to switch between the low-level policies. In order to avoid too rapid reaction to short changes in the conditions, a pheromone-based measure of traffic is used: each car is considered to release a virtual pheromone trail on the lanes that it travels. This trail accumulates with each car and more if the car is moving slowly or if it is stopped at the traffic light. At each time step, a portion of this pheromone evaporates, so that after a while, if no more cars are passing, there will be no trace left. This approach allows a memory of the traffic level that follows smoothly the overall change in traffic volume but not the contingent changes due, for instance, to a group of lanes being empty right after the green light has ended. The pheromone levels are then used to probabilistically select one of the microscopic policies, where the policy that is most suitable for the current traffic level has a higher probability of being selected.

The same information, i.e. the estimates of vehicles and pedestrians, is used by the currently active microscopic policy to determine the duration of the green light time and the order in which the green light is given to the different lanes.

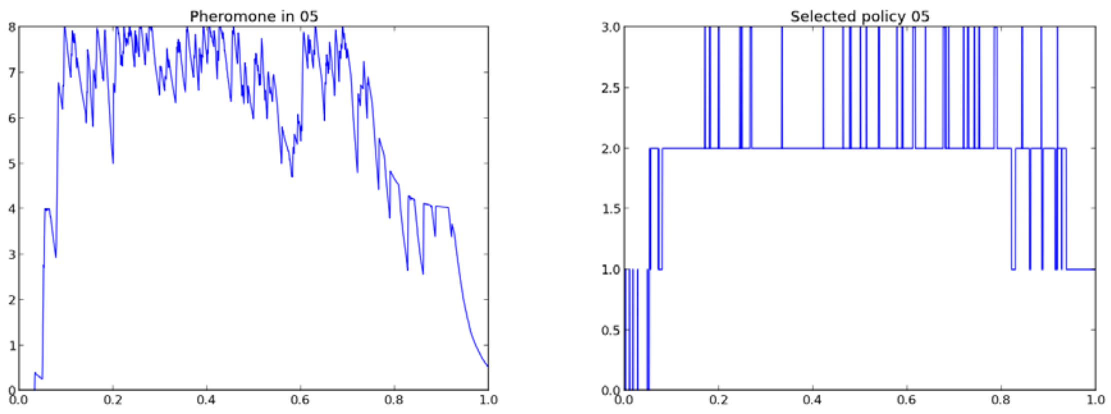


Figure 4.9: left: pheromone level, right: accordingly selected policy: 0, 1, 2 or 3

From a configuration point of view, the algorithm needs to know the possible traffic light states that can be actuated and the relative safety times, such as minimum green duration or inter-green times.

The effect of the macroscopic policy is internal to the traffic light algorithm, while the output of the microscopic policy is the next state of the controlled traffic lights. The current status of the high level policy can be monitored through a stream where the policy writes information about the current pheromone levels of each lane or group of lanes, the currently active microscopic policy, and other status variables.

Figure 4.9 shows plots of the pheromone level and the active policy during a short simulation. It can be seen that, as the pheromone level varies, different microscopic level policies are selected. There is no precise ordering of the policies, but in this simulation policies with lower numbers (0, 1) are suitable for lower levels of traffic while policies with higher numbers (2, 3) are suitable for higher levels of traffic. It can also be seen that the selection of the policy is nondeterministic but probabilistic, with a higher chance of switching to a different one when in uncertain areas.

Simulation Concept

In this first development stage, the traffic light algorithm is embedded into SUMO; therefore all inputs and outputs are received and sent directly to the simulator. In particular, the traffic light algorithm gets information about the current number of vehicles and the average speed from lane detectors placed in each incoming and outgoing lane, and operates the changes to the traffic lights state directly in the simulator.

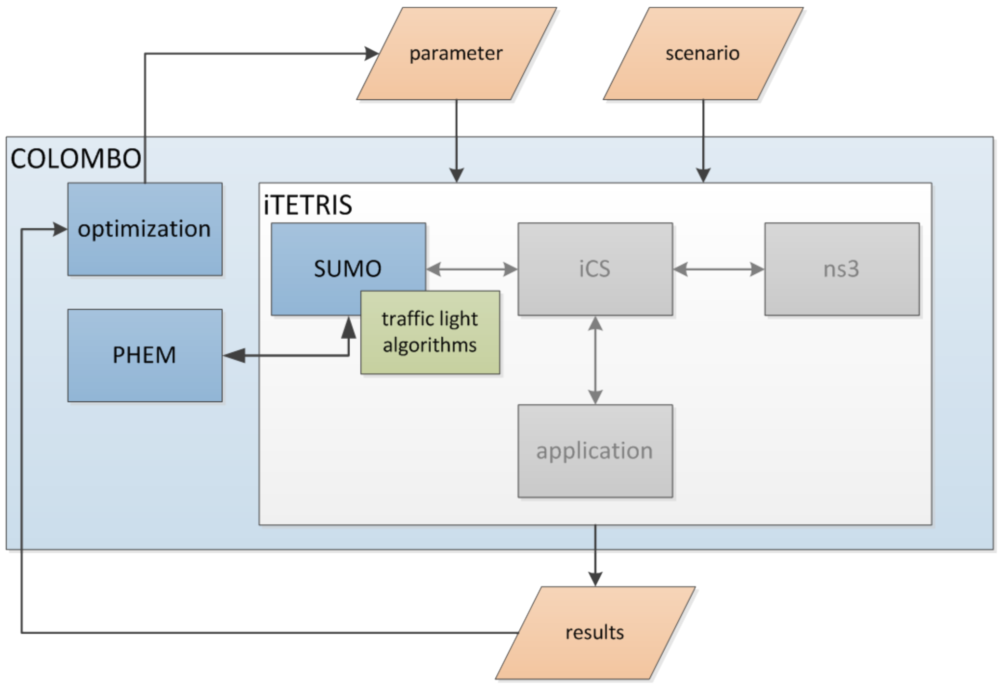


Figure 4.10: COSS Components used for the development the TLS algorithms within the first development steps

Once the integration of the overall system is complete, the traffic light algorithm will be a separate iCS application, therefore communicating with the other components only through the iCS.

The estimates on instantaneous traffic conditions will come from the traffic surveillance algorithms, and the traffic light algorithms will process them as presented above. The uncertainty in the number of vehicles due to non-perfect information will be resolved in the traffic surveillance system. However it is possible to receive an estimate with a reliability measure.

The configuration of the junction, i.e. the lanes and the possible traffic light states, will always come from the configuration of SUMO, but should be read only once, while there will be continuous communication from the Traffic Light Algorithm to SUMO through iCS regarding the traffic light state changes. For reliability reasons, the current traffic light state and the elapsed duration of each signal should be read from SUMO by the TL application, while the state to be applied is written by the TL application to SUMO.

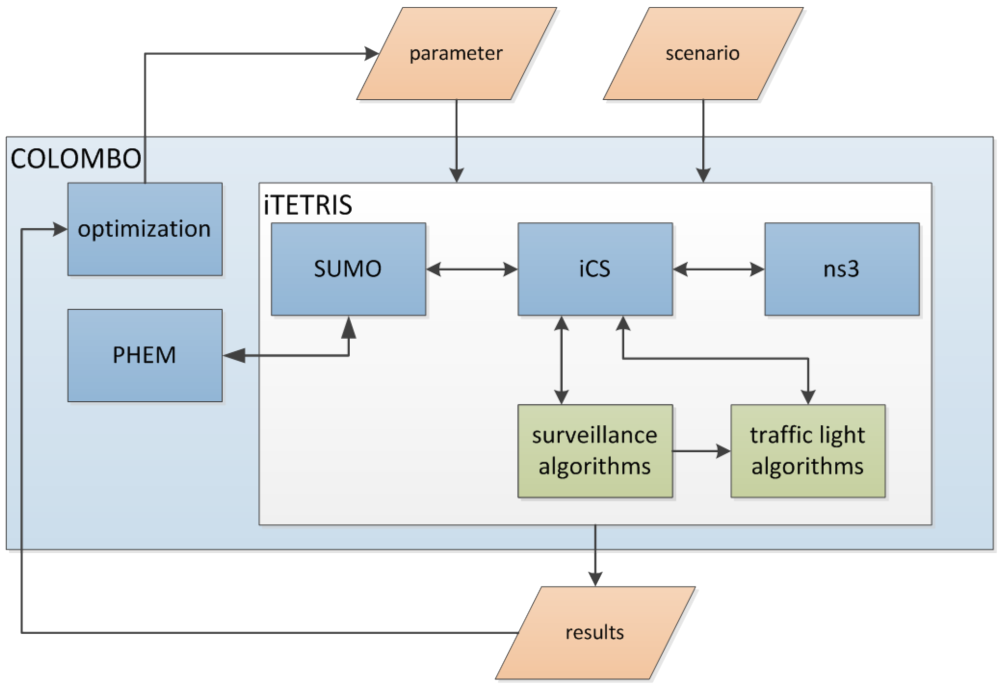


Figure 4.11: The final COSS configuration used for TLS development

As each junction will behave differently given the same traffic load, given to different geometries or other differences between the junctions, all the parameters that regulate the high-level swarm policy for each junction have to be tuned using the tuning tool kit. For instance, the same load of 1000 vehicles per hour could represent a high traffic situation in a junction with few lanes and low maximum speeds, or a medium-low traffic situation in a junction with more lanes and higher speed limits. As a consequence, different microscopic policies will be more appropriate for each junction, and the high-level policy has to be tuned accordingly, as different parameters affect the selection probabilities.

Requirements to COSS

The requirements to the COSS components posed during the first development steps of the traffic light algorithms ("TLS1") are summarized in Table 4.3.

Table 4.3: Mandatory Requirements from the Traffic Light Algorithms within their first development steps (grey: already available)

Component / Application	Requirement ID	Description
SUMO	TLS1.SUMO.1	SUMO must be able to simulate controlled intersections
	TLS2.SUMO.2	SUMO must allow to replicate different intersection settings
	TLS2.SUMO.3	SUMO must allow to replicate different demands on intersections
	TLS1.SUMO.4	SUMO must allow to retrieve measures that can be used to compute pheromone levels
	TLS1.SUMO.5	SUMO must allow control traffic lights by an embedded algorithm
TT	TLS1.TT.1	TT must be able to set a traffic light's parameter
	TLS1.TT.2	TT must be able to retrieve the performance of a traffic light algorithm setting

TLS1	TLS1.1	TLS must read values from SUMO and compute the pheromone levels
	TLS1.2	TLS must choose the policy regarding the read traffic state
	TLS1.3	TLS must adopt the policy to the read traffic state
	TLS1.4	TLS must be able to change the signalling of SUMO traffic lights

The requirements to the COSS components as posed by the final traffic light system ("TLS2") are summarized in Table 4.4. It should be mentioned, that it is not yet decided whether the traffic light controlling application shall directly communicate with the traffic surveillance system or whether the measures obtained from the latter shall be passed to the iCS and the traffic lights systems retrieves them from the iCS. For this reason, Table 4.4 lists requirements to two variants of the system, denoted with the appendices "a" and "b".

Table 4.4: Mandatory Requirements from the Traffic Light Algorithms within their final implementation (grey: already available)

Component / Application	Requirement ID	Description
SUMO	TLS2.SUMO.1	SUMO must be able to simulate controlled intersections
	TLS2.SUMO.2	SUMO must allow to replicate different intersection settings
	TLS2.SUMO.3	SUMO must allow to replicate different demands on intersections
	TLS2.SUMO.4	SUMO must be able to post traffic light status to the iCS
	TLS2.SUMO.5	SUMO must be able to read traffic light status from the iCS
	TLS2.SUMO.6	SUMO must be able to save information about vehicle trips for performance evaluation
	TLS2.SUMO.7	SUMO must be able to simulate pedestrian walks
	TLS2.SUMO.8	SUMO must be able to simulate explicit pedestrian crossing requests by push button
	TLS2.SUMO.9	SUMO must be able to post pedestrian crossing requests to iCS
	TLS2.SUMO.10	SUMO must supply values to TSS
TSES	TLS2.TSES.1	Surveillance algorithms must be able to provide incoming traffic estimates per lane or group of lanes
	TLS2.TSES.2	Surveillance algorithms must be able to provide congestion levels per lane or group of lanes, as a measure inverse to the average traffic velocity or similar metric
	TLS2.TSES.3a	TSS must deliver measurements to the iCS
	TLS2.TSES.3b	TSS must deliver measurements to the TLS
iCS	TLS2.iCS.1	iCS must be able to post traffic light status to TLS2
	TLS2.iCS.2	iCS must be able to read traffic light status from TLS2

	TLS2.iCS.7	iCS must be able to change the traffic lights state within SUMO
	TLS2.iCS.3	iCS must be able to post pedestrian crossing requests to TLS2
	TLS2.iCS.4	iCS must retrieve measurements from the TSS
	TLS2.iCS.5	iCS must pass measurements to TLS2
	TLS2.iCS.6	iCS must be able to retrieve signal change information from TLS2
TT	TLS2.TT.1	TT must be able to set a traffic light's parameter
	TLS2.TT.2	TT must be able to retrieve the performance of a traffic light algorithm setting
TLS2	TLS2.1	TLS must be able to read SUMO junction configuration from SUMO input files
	TLS2.2a	TLS must be able to read estimates of traffic from surveillance algorithms via iCS
	TLS2.2b	TLS must be able to read estimates of traffic directly from TSS
	TLS2.3	TLS must be able to read pedestrian requests from SUMO via iCS
	TLS2.4	TLS must be able to read traffic light status from iCS
	TLS2.5	TLS must be able to post traffic light status to iCS
	TLS2.6	TLS must be able to compute the traffic light next status, abiding all safety rules and according to the allowed traffic light changes
	TLS2.7	TLS2 must choose the policy regarding the read traffic state
	TLS2.8	TLS2 must adopt the policy to the read traffic state

Needed Scenarios

TLS is developed for being deployed at single intersections as an application running on a road-side unit (RSU).

A simple cross intersection scenario is needed for testing the software tools integration. A second scenario of a real intersection controlled by TLS in a small urban area with other intersections, some without traffic lights and some with standard traffic light controllers will allow to test the TLS algorithms in case of a single installation; for this scenario, some real traffic data is needed. At least a more complex scenario, of a wider area with at least five close-by TLS controlled junctions and traffic data in different situations – workdays, peak hours, weekend, football match. Summarizing, the development of traffic light algorithms requires a large set of scenarios with controlled intersections. Scenarios should cover all cases of:

- Regarded area including single intersections, corridors, network, the latter both to determine how well the system coordinates itself
- Different intersection layouts, mainly four-arms and three-arms crossings
- Different traffic demands including different amounts of vehicles and different vehicle types
- Incorporation of pedestrians and bicycles

4.4 Emission-optimal Driver Behaviour for Controlled Intersections

Within COLOMBO's work package 4 [COLOMBO DoW, page 17] *an emission-optimal driver behaviour for controlled intersections* will be developed, abbreviated by "EDB" in the following. The behaviour of the drivers on traffic signals in front may heavily influence the resulting emissions. E.g. perfectly synchronized traffic lights could switch to green directly before the first car of a vehicle group arrives. In reality the drivers however may decelerate when they see the red light in front and thus would not make use of the "ideal" traffic light coordination.

It is foreseen to make a sensitivity analysis on the influence of typical driver reactions on fuel consumption and emissions to elaborate the most important reactions of drivers. These findings will be used to elaborate which information an I2V communication system shall provide the driver to reduce energy consumption and emissions adapted to traffic control algorithms.

Driver reactions planned to be tested are (with/without I2V information):

- Distance to junction when driver starts deceleration when the traffic light is red
- Acceleration level dependency on light signal ahead
- Target speed after acceleration dependency on light signal ahead

It is assumed that different "driving strategies" exist, all build upon a set of "driving actions", such as acceleration, keeping the speed, hard braking, freewheeling. Given the vehicles surrounding the regarded vehicle, the state of the traffic light in front, and the state of the regarded vehicle itself, the actions and their parameter – duration, braking/acceleration strength, speed, etc. – have to be chosen and instantiated yielding in an emission-optimal behaviour. By now, it is assumed that only the longitudinal behaviour of vehicles will be controlled.

Simulation Concept

EDB simulations are assumed to be performed on controlled intersections. Both, single intersections as well as corridors should be evaluated in order to determine the behaviour between consecutively passed intersections as well. The behaviour of "equipped" vehicles – those that have an advice how to behave friendly to the environment – will be dictated by an external application that controls the vehicles via TraCI. Surely the first steps will use scenarios where only one equipped vehicle is simulated, comparing them to a usual vehicle's performance. Incrementally the scenarios will be filled by other vehicles – both equipped and not equipped. Different vehicle types will be investigated as well. Even though EDB when deployed in real-life will give an advice to the driver only, the simulations will assume that all drivers follow this advice, albeit not necessarily to 100 % correct.

PHEM/PHEMlight will be used to get the emissions produced by the vehicles what will be used as the measure for determining the application's performance. As the emissions are not used to control the equipped vehicles' behaviour on-line, they may be also determined using an off-line coupling to PHEM.

As discussed, the usage of different algorithmic approaches to determine the correct order and parameter of the driving actions to instantiate is assumed to prosper from the tuning tool kit. Proper interfaces have to be established that wrap the execution of EDB in order to make it executable by the tuning tool kit.

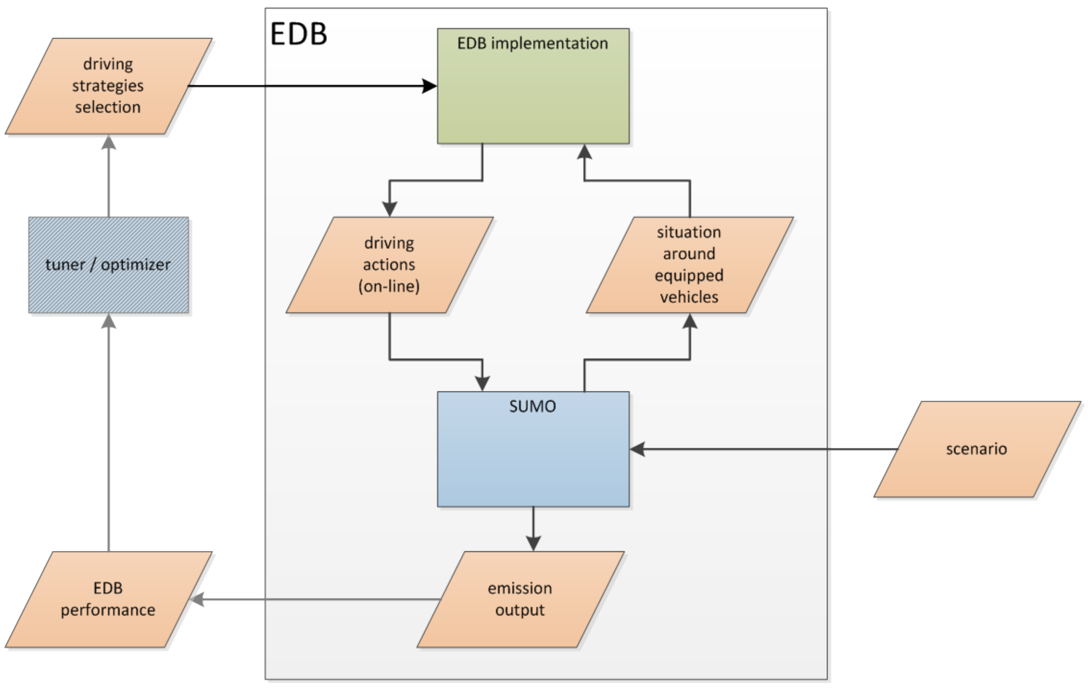

EDB
EDB implementation
driving strategies selection
tuner / optimizer
driving actions (on-line)
situation around equipped vehicles
SUMO
scenario
EDB performance
emission output

Figure 4.12: EDB data flow without involving a communication simulation

It is assumed that the communication is not a crucial part during the first development step, as the vehicle has to be informed about the state of the traffic light only. Such messages (SPAT – Signal Phase and Timing) are usually sent often enough and their content does not change often so that they can be assumed to be received in time. Simulation runs performed in DRIVE C2X that evaluated the GLOSA application, which is similar regarding the communication, have proved this, already, see [Krajzewicz et al., 2012b] This means that no communication simulation is needed, first, leaving the iCS and ns-3 out of the loop. It may be necessary to include inter-vehicle communication as soon as the influence of other vehicles has to investigate. By now, it cannot be said whether the usage of ns-3 will be necessary or whether simpler communication models will get necessary. The resulting involvement of COSS components is shown in Figure 4.13.

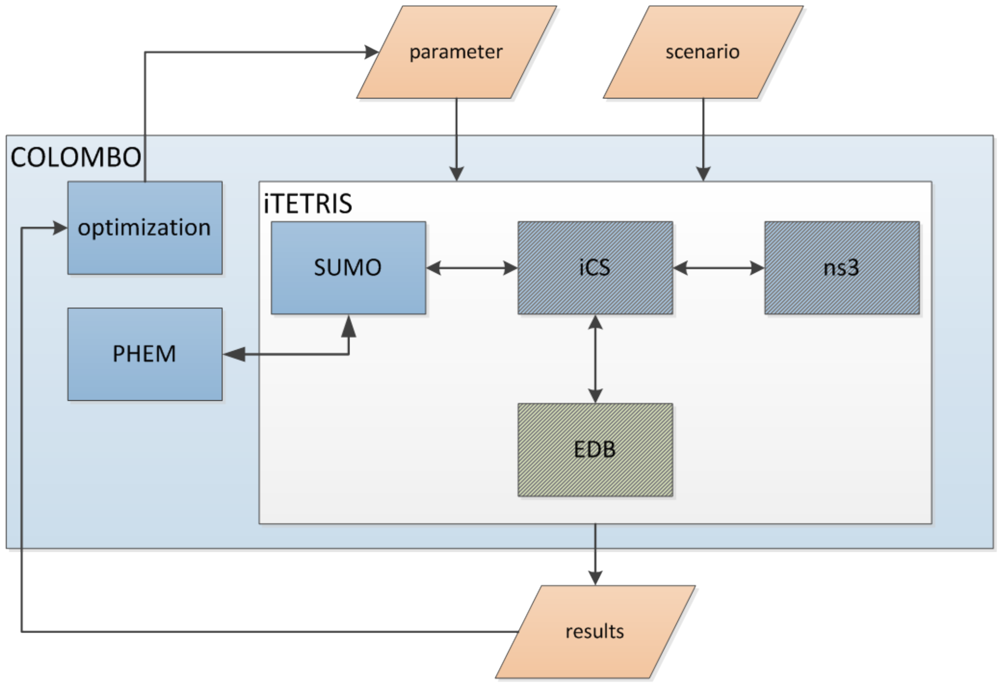


Figure 4.13: The final COSS configuration used for EDB development

Requirements to COSS

The requirements to the COSS components as posed by the EDB are summarized in Table 4.5.

Table 4.5: Mandatory Requirements from the Emission-optimal Driver Behaviour for Controlled Intersections (grey: already available)

Component / Application	Requirement ID	Description
SUMO	EDB.SUMO.1	SUMO must be able to simulate controlled intersections
	EDB.SUMO.2	SUMO must allow to retrieve the state of a traffic light, including the future state (or a prediction)
	EDB.SUMO.3	SUMO must allow to control a vehicle's longitudinal behaviour via TraCI
	EDB.SUMO.3	SUMO must be able to simulate different vehicle classes
	EDB.SUMO.4	SUMO must allow to replicate different demands on intersections
	EDB.SUMO.5	SUMO must be able to return the states (speed, positions) of vehicles in a chosen vehicle's surrounding
	EDB.SUMO.5	SUMO must be able to return the maximum acceleration of a vehicle
PHEM	EDB.PHEM.1	PHEM must be able to compute the emissions for a vehicle, distinguishing between accelerations, speeds, and the "freewheeling" mode
	EDB.PHEM.1	SUMO must be able to return the maximum acceleration of a vehicle
TT	EDB.TT.1	TT must be able to set EDB's parameter
	EDB.TT.2	TT must be able to retrieve the performance of an executed configuration of EDB
EDB	EDB.1	Vehicles must be able to be assigned to equipped/unequipped classes
	EDB.2	The EDB must be able to retrieve the positions of vehicles
	EDB.3	The EDB must be able to retrieve the state of traffic lights
	EDB.4	The EDB must be able to change the behaviour of vehicles

Needed Scenarios

Scenarios including single controlled intersections as well as ones including a corridor of controlled and uncontrolled intersections are necessary. Different demands as well as different types of equipped vehicles must be included.

5 System Architecture Prototype

The COLOMBO components differ in many ways: their age, their size, the number of organizations that develop them as well as in software engineering methods, such as code style or test coverage. In the following, the current state of the COLOMBO Overall Simulation System is presented from a technical perspective. At first, it is shown, how the software development is organized to allow a collaborative development of the whole system. Then, the status of the components – whether they can be built, how they are documented and whether they can interact – is given. At last, the extensions that need to be performed are listed and briefly discussed.

5.1 Software Development in COLOMBO

COLOMBO uses a subversion (SVN) revision control system[7] [Wikipedia SVN] for storing the source code of the developed components and applications. The SVN repository is hosted by DLR, and is accessible for project partners. In a first step, the source code of the used components was collected and inserted into the SVN. Almost all partners contributed a component.

Table 5.1 summarizes the versions used within COLOMBO as well as the components' original web presence. As shown, most of the components within the COLOMBO SVN are up-to-date or even already containing extensions that were implemented within the COLOMBO project and which cannot yet be found in the components latest release.

Table 5.1: Versions and Availability of COLOMBO components

Component	Latest official version	SVN Version	Availability
iCS	0.2.0	extended 0.2.0	At request, from the project web site, at http://ict-itetris.eu/
SUMO	0.17.0	extended 0.17.0	Direct download from the project web site, at http://sumo.sf.net
ns-3	3.17	extended	Direct download from the project web site, at http://www.nsnam.org/ (see text)
PHEM	11.2.9	N/A	Supported on demand to the project partners, official contact via http://www.ivt.tugraz.at/de/forschung/emissionen.html
irace	1.04	1.04	From the project web site, at http://iridia.ulb.ac.be/irace/
Tuning Tool kit	1.0	1.0	Not yet available for public use

COLOMBO project partners share the work on the components, e.g. extensions to SUMO are contributed by UNIBO (traffic light algorithms), TUG (emission modelling), and DLR (interface extensions, pedestrian models). Further extensions are assumed to be done by other partners as well. Nonetheless, there is at least a “historical responsibility” for most of the components. As some of the tools are developed for a large time span already – e.g. the work on SUMO started in 2001 – COLOMBO is not trying to establish one software development approach that has to be followed by all used components. Instead, it is tried to collaborate on the software modules, exchanging best practices, but keeping the components individual.

[7] http://subversion.apache.org/

Due to the diversity in the used programming languages (see also Table 5.2), the development process and differences in release policies, the connection between the original components' source code and the one stored within COLOMBO's SVN differs as well. The modules are included as following:

- ns-3: direct insertion of the code extended and used within the iTETRIS project
- iCS: direct insertion of a version newer than the one developed in iTETRIS (see also Table 5.1)
- SUMO: a link to a branch that is located in SUMO's public SVN
- tuning tool kit: direct insertion of the code (developed in COLOMBO, WP3)
- irace: direct insertion of the recent version

As one can see, besides SUMO, all components are stored directly within the COLOMBO SVN. When working on SUMO, the committed changes are stored in a "branch" of SUMO's original repository. This process was chosen to avoid influences between the developments of SUMO within different projects, which may yield in incoherent states, not being able to be built. The "reintegration" of COLOMBO changes is performed after a clean, tested state has been reached. This is done "on-demand", not following a certain time table.

Table 5.2: Programming Languages by component

Component	C++	Java	Python	VisualBasic .NET	R
iCS	X				
ns-3	X		X		
SUMO	X		X[8]		
PHEM				X	
PHEMlight	X				
irace					X
Tuning Tool kit			X		

Extensions of other modules – mainly the larger ones, iCS and ns-3 – is usually performed by "branching" the component code – moving it into a designated place within the SVN – and working on the code at this place. Again, after the wished extensions are implemented and tested, the code is moved back to the "trunk" – the place within the SVN that holds the stable version of the component.

PHEM, being the only non-open source package, is not included within the SVN, but made accessible to project partners by TUG. PHEMlight is an extension of SUMO. The source code is included in SUMO's code, additionally required data is stored within the COLOMBO SVN.

Yet unsolved is the process of versioning and releasing the complete COSS. Currently, each of the components has an own web site and/or contact point.

[8] Additional tools

5.2 Components Status

When being inserted into the SVN, the most recent versions of the COSS components were chosen. The only exception is ns-3. At the current time, the extensions of ns-3 by communication protocols and their stacks are not included in ns-3' official release. For this reason, the originally modified version of ns-3 is in use within COLOMBO. The inclusion of the stacks within the official release is currently a matter of discussions.

In a second step, the applications were compiled (if needed) and tested on different platforms. Currently both major operating systems, Windows and Linux, are targeted. MacOS is assumed to be supported "automatically" as usually Linux applications can also be compiled under MacOS (it has been proved for SUMO). "Linux" denotes here a modern Linux of any distribution. Tests are performed on openSUSE 12.1 and Ubuntu ServerLTS 13.04. It should be noticed that Linux portability is needed for executing COSS on ULB's simulation cluster. A summary of portability is given in Table 5.3, where "likely" denotes that the software is assumed to be working due to previous experiences (as running SUMO under MacOS) while "maybe" denotes that technical solutions are known to exist, but have not been proved to be working within the performed work on the components.

Table 5.3: Portability of COLOMBO components, see text for explanations

Component	Windows	Linux openSUSE 12.1	Linux Ubuntu ServerLTS 13.04	MacOS
iCS	x	x	x	likely
SUMO	x	x	x	x
ns-3	-	x	x	likely
PHEM	x	maybe	maybe	maybe
irace	x	x	x	x
Tuning Tool kit	x	x	x	x

While being an essential tool within COSS, PHEM is the only application that is not natively executable under Linux. Being written in a .NET language, one could assume that it should be able to be compiled using the MONO[9], a port of Microsoft's .NET framework to Linux. But as the work on PHEMlight is progressing in a very promising way, it is assumed that PHEMlight embedded directly into SUMO will be used instead of an off-line time consuming data conversion and passing between SUMO and PHEM.

Regarding the fact that COLOMBO components are given to the public use, one should also consider the documentation. All these applications have relatively complex duties and a good documentation is crucial for their acceptance. As summarized in Table 5.4, all components are well described, including information about the software's installation steps as well as how to use the software. All come with example files, which allow to run the software without needing to prepare the inputs, first.

[9] http://www.mono-project.com/Main_Page

It should be mentioned, that most of the components are planned to be extended within COLOMBO and hopefully beyond the project, too. For this reason no attempt to mark any as "completed" will be done.

Table 5.4: Components documentation

Component	Installation Guide	User Guide	Examples / Tutorials
iCS	X	X	X
SUMO	X	X	X
ns-3	X	X	X
PHEM	(trivial)	X	X
Irace	X	X	X
Tuning Tool kit	X	X	X

5.3 Components Interaction in the Prototype

The above described software tools are coupled to a broad framework for scientific investigations of V2X traffic management applications. The used components in COSS are designed and implemented to simulate a specific use case and are well established in their field of research as mighty tools with a huge set of features and functionalities. The simulation tools have been adjusted over the years to achieve sufficient performance. Nevertheless, coupling the components of COSS may causes serious performance drops. Hence, a lot of effort is undertaken, to keep and strengthen COSS's framework ability, inherited from iCS. This means, that at any reasonable effort, the components are deemed to be pluggable and shall not depend on each other. This allows switching off not required components or even replacing components by 'light weighted', but coarser alternatives, e.g. for tuning purposes. As a major benefit, rarely all components of the COSS need to be run together at the same time. For performance reasons, we only launch the required components, depending on the evaluation and investigation task of the experiment undertaken with COSS. Due to this, the interaction of the COSS components in the prototype will be described by typical combinations of these components.

The prototype in its current state enables interactions between:

iCS, SUMO ns-3 and external applications:

The most basic functionality which needs to be provided by the COSS is to simulate the movement and interaction of traffic entities along with their communication through electronic devices. As a major component, iCS launches the COSS components such, as SUMO, ns-3 and the applications, each in separate pThead-items and coordinates the information exchange between them, using the socket interfaces specified above.

iCS, SUMO, external applications and a coarse communication model

The interaction between SUMO and iCS is obligatory for most of COLOMBO's investigation use cases whereas ns-3 is used for scenarios only, where the most detailed simulation of communication networks is the main focus, e.g. simulation of specific communication protocols. Nevertheless ns-3 performance rates only allow a small number of experiments. When a huge amount of similar scenarios are needed to be run in a loop, e.g. for offline configuring by the tuning tool kit, ns-3 is replaced by a more coarse communication model resembling precisely ns-3's interfaces. The more coarse communication model shall only enhance performance, but at the price of a less precise simulation.

COSS components and PHEM

At multiple points of the COLOMBO project, emissions and fuel consumption are used either as input values or measurements and cost criteria. Calculating emissions in COSS is now handled by the PHEMlight extension implicitly being run with SUMO. Any emissions data which is needed outside of SUMO will be handed over by the typical iCS interfaces provided for SUMO.

COSS and the tuning tool kit

The purpose of the tuning tool kit is to pre-configure the system for optimization tasks that the COSS has to tackle while being under deployment. Therefore, modifications of configurations will be pre-evaluated on a high performing cluster, and the most successful configurations will be deployed. Beyond an automated ramp up script of the COSS, the tuning tool kit doesn't need any interaction to the COSS during the simulation run. Any data exchanged before and after will be provided file-based, either generated by iCS or additional scripts suiting the tuning tool kit.

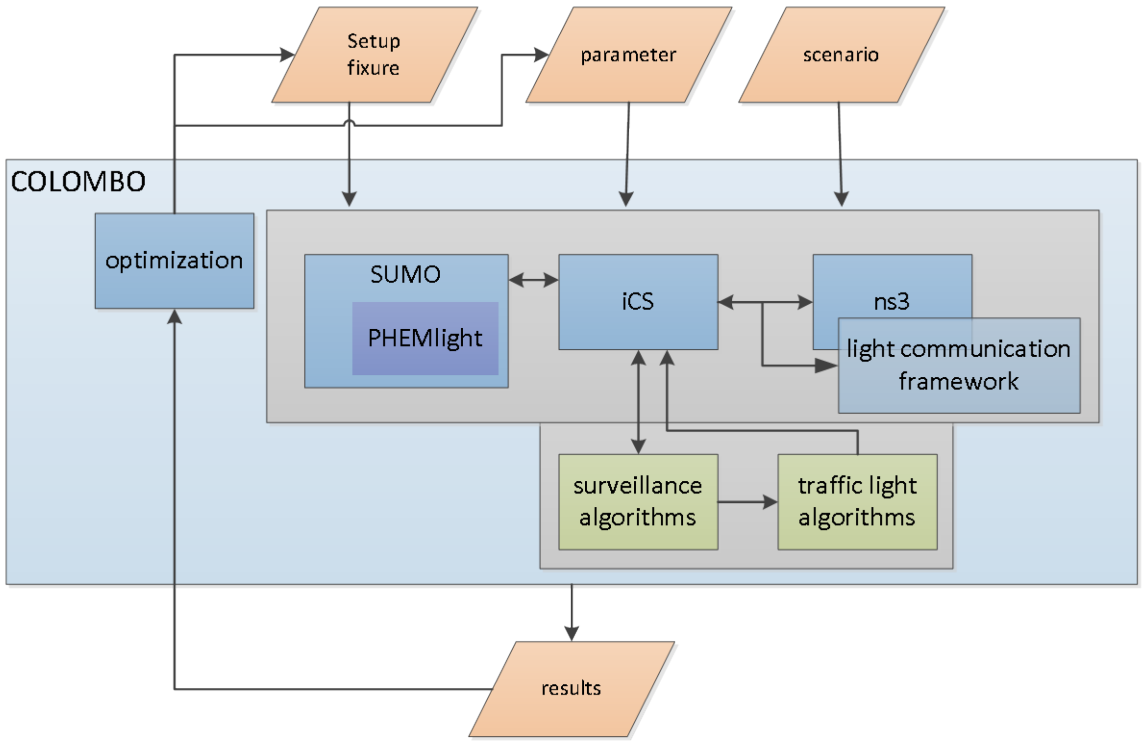


Figure 5.1: Schematic visualization of COSS-prototype outlining the tuning tool kit's interaction with the COSS as a whole, with PHEMlight being included into SUMO and the option to deploy a coarse communication model.

5.4 Needed Extensions

In the following, the requirements posed by the applications to be developed within COLOMBO presented in Chapter 4, are revisited. This is done component-wise with the target to recognize groups of necessary extensions.

Most COSS components will be extended in different ways within the project's scope. These extensions are not discussed in large detail, as they will be described in own deliverables released at later time of the project.

SUMO

SUMO already provides most of the functionality needed to set up the prototype of the system. Left open are precise adaptions for providing traffic state information through iCS to the Traffic Surveillance System or to the Traffic Light Algorithms. The open requirements mainly require extensions to the TraCI interface. A special case is EDB.SUMO.2, "SUMO must allow to retrieve the state of a traffic light, including the future state (or a prediction)", as forecasting the status of an adaptive traffic lights is usually hardly possible. This has to be discussed during the work on EBD.

A further group of required extensions is the addition of models for pedestrians that cross the streets. This is scheduled as Task 5.2, together with the implementation of bicyclists.

Table 5.5: Summary of requirements to SUMO posed by COLOMBO applications

Component / Application	Requirement ID	Description
SUMO	EMS.SUMO.1; TLS1.SUMO.1; TLS2.SUMO.1; EDB.SUMO.1	SUMO must be able to simulate (controlled) intersections
	EMS.SUMO.2	SUMO must be able to represent different vehicle types
	EMS.SUMO.3	SUMO must generate vehicle trajectories
	EMS.SUMO.4	SUMO must simulate conventional detectors (no errors are assumed) and write their values
	TLS1.0SUMO.2; TLS2.SUMO.2	SUMO must allow to replicate different intersection settings
	TLS1.SUMO.3; TLS2.SUMO.3; EDB.SUMO.4	SUMO must allow to replicate different demands on intersections
	TLS1.SUMO.4	SUMO must allow to retrieve measures that can be used to compute pheromone levels
	TLS1.SUMO.5	SUMO must allow control traffic lights by an embedded algorithm
	TLS2.SUMO.4	SUMO must be able to post traffic light status to the iCS
	TLS2.SUMO.5	SUMO must be able to read traffic light status from the iCS
	TLS2.SUMO.6	SUMO must be able to save information about vehicle trips for performance evaluation
	TLS2.SUMO.7	SUMO must be able to simulate pedestrian walks
	TLS2.SUMO.8	SUMO must be able to simulate explicit pedestrian crossing requests by push button
	TLS2.SUMO.9	SUMO must be able to post pedestrian crossing requests to iCS
	TLS2.SUMO.10	SUMO must supply values to TSS

SUMO	TSES.SUMO.1	SUMO must be able to deliver the information about a vehicle's speed
	TSES.SUMO.2	SUMO must be able to deliver the information about a vehicle's driving direction (angle)
	TSES.SUMO.3	SUMO must be able to deliver the information about a vehicle's position
	TSES.SUMO.3	SUMO must be able to deliver the information about a vehicle's signal status (blinker, brake lights)
	TSES.SUMO.4	SUMO must be able to save the aforementioned information into a file
	EDB.SUMO.2	SUMO must allow to retrieve the state of a traffic light, including the future state (or a prediction)
	EDB.SUMO.3	SUMO must allow to control a vehicle's longitudinal behaviour via TraCI
	EDB.SUMO.3	SUMO must be able to simulate different vehicle classes
	EDB.SUMO.5	SUMO must be able to return the states (speed, positions) of vehicles in a chosen vehicle's surrounding
	EDB.SUMO.5; EDB.PHEM.1	SUMO must be able to return the maximum acceleration of a vehicle

ns-3

Being used for V2X simulations, ns-3 and its ETSI extensions are available and usable already in the prototype. Nevertheless, some of the most general communication simulation enhancements, not including any sub requirement, need to be made to the prototype as well as to the COSS.

Table 5.6: Summary of requirements to ns-3 posed by COLOMBO applications

Component / Application	Requirement ID	Description
ns-3	TSES.ns3.1	ns-3 must be able to simulate WiFi-direct communication
ns-3	TSES.ns3.2	ns-3 must be able to obtain commands from iCS that start WiFi-direct communication for a given node
ns-3	TSES.ns3.3	ns-3 must be able to inform the iCS about the simulated reception of WiFi-direct messages

PHEM / PHEMlight

While working on PHEM in WP 4 of the project, it has been proved that trajectories generated by SUMO and post-processed and reformatted can be used input for PHEM. As PHEM's functionally is resembled in the PHEMlight module for SUMO, no further extensions to PHEM are advised. As PHEMlight is currently under development, the current functionality has been found to be sufficient for setting up test scenarios. The extensions that are still to be completed are subject to work already scheduled in WP4.

Table 5.7: Summary of requirements to PHEM / PHEMlight posed by COLOMBO applications

Component / Application	Requirement ID	Description
PHEM	EMS.PHEM.1	PHEM / PHEMlight must be able to process trajectories obtained from SUMO and to save computed pollutant emissions
PHEM	EDB.PHEM.1	PHEM must be able to compute the emissions for a vehicle, distinguishing between accelerations, speeds, and the “freewheeling” mode

Tuning Tool Kit

The tuning tool kit’s development is scope of the COLOMBO work package WP3; hence, essential functionality and interaction between the tuning tool kit and the COSS is available. The partly file based exchange of information must be flanked by execution and post-evaluation scripts. Their specifications are available and have been proved to fulfil their purpose. No further work on interfaces to the tuner is needed, besides the work on the execution scripts, suiting certain experiments. These script adaptions can’t be considered to be extensions of the tuning tool kit.

Table 5.8: Summary of requirements to the tuning tool kit posed by COLOMBO applications

Component / Application	Requirement ID	Description
TT	TLS1.TT.1; TLS2.TT.1	TT must be able to set a traffic light’s parameter
TT	TLS1.TT.2; TLS2.TT.2	TT must be able to retrieve the performance of a traffic light algorithm setting

6 Abstract usage examples of the overall system

This section outlines briefly and exemplifies the usage of the current prototype of the COSS. The focus is on use cases, which will become relevant in subsequent phases of the COLOMBO project.

6.1 Coupling SUMO, a traffic light algorithm and the tuning tool kit

This section describes how to launch the tuning tool kit for examining a traffic light control replicated in SUMO. The tuning tool kit will be connected to SUMO as a basic optimizer, which searches for the optimal fixed-time cycle at a single cross intersection with no turning. The optimizer deems to reduce the overall waiting time of all vehicles. The entire example can be found in the SVN.

Prerequisites

This example requires the base system, with SUMO, iCS, R, and the tuning tool kit to be installed.

Due to the fact that the tuning tool kit is designed to tune a broad variety of algorithms, it requires a rather strict but well and clearly defined infrastructure for training and testing of an algorithm.

The tuner itself needs:

- a workspace folder and an ‘`arena`’ folder
- a `hook-run` script to launch the examined algorithm
- a parameter space description file
- a set of (training) instance
- a configuration file for the tuning tool kit itself
- a temporary folder

The workspace folder is the base folder which contains all other files and subfolders listed above. As the tuner generates and relies on a number of temporary output files, these must be stored in an obligatory ‘`arena`’ folder. The information stored there is a precious source of information while debugging the tuner’s hook-run script and the system’s set up. Hence, any output which is filtered or generated by the `hook-run` script should be redirected into the ‘`arena`’ folder.

The hook-run script

The multi-purpose design of the tuner allows the tuner to launch any command line program, which itself has to prompt a cost measure back to the standard output. To launch the examined system by the tuning tool kit over and over again, wrapper scripts, also called `hook-run` scripts are needed for the proper set up of the examined system and to filter the output of the system. The `hook-run` script shall perform the following tasks:

- Launch and run the algorithm or the system to be tuned exactly as intended. By using `hook-run` scripts, the application can be launched in its native environment; precisely the intended way. When launching the examined system fixed and variable parameters need to be set. Fixed parameters are not altered by the tuning tool kit, and, hence, can be handled within the `hook-run` script itself. Variable parameters are altered by the tuner every time it re-launches the system in order to test its performance. Variable parameters have to be handed over in a proper form (for example, specific command line switches) to the system.
- For evaluation by multiple (simultaneous) runs of the system, the tool kit evaluates, the algorithms performance by a cost representing value, which is read directly from the command line. The tuning tool kit parses the standard output, hence nothing but the value under consideration (costs value) is supposed to be printed here. As this is not the typical behaviour of

our system to be tuned, the wrapper script needs to filter or to determine the cost value and to discard any other output.

- The run hook script shall provide access to the instances, e.g. as a command line files input

Defining the parameter space

A definition of the parameters to be tuned and their range must be provided to the tuning tool kit. The parameter space is declared in the `parameters.txt` – file, a text file, defining in each line one parameter, its type and the limits of its range if it's a numeric parameter or an enumeration if it's possible values if the parameter is of ordinal or categorical character. The hook-run will launch the system with varying combinations of these parameters for examination.

As the cycle time example included in the SVN only uses one parameter, which is the initial number of time segments for the green phase, the parameter file looks like this:

```
### PARAMETER FILE FOR THE SUMA SOFTWARE
# name          switch  type    values          [conditions (using R syntax)]
stepLength      ""      I       (2, 90)
```

Providing different Instances for training

Besides the launch of the same problem with different parameters, the tuner investigates the algorithms performance on multiple instances of the problem. The tuner should use multiple instances, which are similar to the instances the algorithm is supposed to tackle while deployment. The folder "`Instances`" should include all training instances.

For the provided COSS example, the instances are a set of different traffic demands, which define when and how many vehicles approach the intersection. All needed data is provided in xml files, following the SUMO route files specifications. The certain instance is accessed by handing over its location from the tuning tool kit to the examined system in the `hook-run` script.

Other conceivable instances for this example could be modifications of the intersection's geometry, its priority rules or lane assignment. The training instances can be populated with randomized traffic flows by using the `sumo-aux-bins/populateInstances.py` python script.

Configuration of the tuner

The tuner itself needs some minimal configuration as well. At least, the tuner needs the information, where to find the hook-run script and where to find its '`arena-`' and its '`Instance-`' folder. Configurations of the budgeting of single instance runs have also be made.

For the given intersection example the minimal `tuner-config` file contains the following information:

```
## Configuration for Iterated Race,
## to tune the SUMO software.
############################################################################

## File that contains the description of the parameters.
parameterFile <- "./parameters-sumo.txt"

## Directory where the programs will be run.
execDir <- "./sumo-arena"

## Folder where tuning instances are located, either absolute or
## relative to working directory.
instanceDir <- "./Instances"

## The maximum number of runs (invocations of hookRun) that will performed. It
## determines the (maximum) budget of experiments for the tuning, unless
## timeBudget is positive.
maxExperiments <- 400
```

```
## Indicates the number of decimal places to be considered for the
## real parameters.
digits <- 2

## END of configuration file
############################################################################
```

If irace is used without the tuning tool kit, a bash wrapper script, available in R's side libraries that are shipped with irace needs to be copied into the base folder in order to launch the tuner from outside the R environment. The location of the execution script can be found using the following R-commands:

```
> file.path(system.file(package="irace"), "bin")
[1] "/usr/local/lib/R/site-library/irace/bin"
```

The tuner's finding are summarized in the output file defined in the tuner's configuration or are prompted back to the user on the command line. The resulting output could look like this:

```
$ ./irace
…

# Elite candidates:
   stepLength
10          8
15         11
# 2013-07-24 14:49:04 CEST: Limit of iterations reached
# 2013-07-24 14:49:04 CEST: Stopped because there is no enough budget to sample new candidates
# remainingBudget: 6
# experimentsUsedSoFar: 394
# timeUsedSoFar: 0
# timeEstimate: 0
# currentBudget: 6
# number of elites: 2
# nbCandidates: 0
# mu: 5
# Best candidates
   stepLength
10          8
15         11
# Best candidates (as commandlines)
   command
10       8
15      11
```

6.2 Coupling SUMO, algorithms and the tuner extended by PHEMlight's emission module.

As major use case of the COSS is to evaluate the *pollutant effectiveness of traffic management measures* by generating emission and fuel consumption figures already during the simulation.

Emission Optimization Example

This section outlines how the above described example can be extended to reduce the carbon dioxide emissions instead of the waiting times, using the tuning tool kit with the algorithm and SUMO with the enabled PHEMlight.

This example requires the base system, iCS, R and the tuning tool kit to be installed. Further the COLOMBO-branch of SUMO needs to be deployed. For using the PHEM module, further related non-publicly available data sheets, which can currently only be found in the COLOMBO SVN need to be included.

The complete example can be found at SVN:

https://svn.dlr.de/COLOMBO/trunk/software/irace/iraceSumoPhemExample

Configuration of PHEMlight

Using the PHEMlight module so that SUMO computes emission output values from PHEMlight is straightforward by adding emission class values to the definition of any vehicles approaching the intersection. This is done by assigning the emission classes **(PKW_D_EU0 LNF_G_EU0)** to vehicles when populating the '`Instances`'-folder with SUMO compliant route files.

In order to enable the calculations an additional file needs to define the emission device's configuration. The needed information can be found in the example's file `emissionDevicePhem.xml` with the following content:

```
<add>
        <edgeData id="dump_900" type="phem" file="../aggregated_1.xml" excludeEmpty="true" />
</add>
```

The file could be included by the use of command line options or directly in SUMOs configuration file (e.g. `cross.sumocfg`):

```
<configuration>
…
        <additional-files value="emissionDevicePhem.xml"/>
…
</configuration>
```

When SUMO is now launched by the tuning tool kit, a file containing the emission output is generated with output lines like:

```
  <edge     id="1o"     sampledSeconds="16545.59"     CO_abs="1.856317e-310"     CO2_abs="1.856272e-310"
HC_abs="1.856359e-310"   PMx_abs="1.856482e-310"   NOx_abs="1.856440e-310"   fuel_abs="2.349873e-313"
CO_normed="1.349367e-313"            CO2_normed="1.349335e-313"            HC_normed="1.349397e-313"
PMx_normed="1.349487e-313" NOx_normed="1.349457e-313" fuel_normed="1.708136e-316" traveltime="49.49"
CO_perVeh="5.552137e-313"            CO2_perVeh="5.552005e-313"            HC_perVeh="5.552263e-313"
PMx_perVeh="5.552632e-313" NOx_perVeh="5.552508e-313" fuel_perVeh="7.028335e-316"/>
```

As this is a simplified example, the CO_2 emission is taken as criteria to be optimized. Here for minimal changes to the hook-run script and the example-optimizer, e.g. parsing for 'CO2_abs' value over all edges, have been made.

6.3 Coupling SUMO and ns-3, iCS to enable an example Application

Yet, no investigations of traffic management measures have been undertaken, as many of the components to be deployed are currently under development. Nevertheless iCS has been lifted to work with the current version of the COLOMBO/SUMO branch. Due to the fact, that iCS internally did also depend on some outdated SUMO utilities, e.g. for parsing SUMO-xml files, reorganization of the code and its dependencies has been undertaken, which changes to a certain extend the build and installation routine. We do outline the build instructions for the COSS components ns-3, SUMO, iCS in order to enable the iCS community demo app v2.

Introduction and background of the iCS community demo app v2

The iCS community demo app v2 from the iTETRIS project is well known to be a reliable and sensitive test application whenever changes to the iCS or ns-3 have been applied. Hence this chapter outlines the steps to be undertaken to set up successfully the iCS community app v2 with the current version of the engaged COSS components, as they differ from the description in the original iCS manual. Originally the iCS community demo app v2 has been designed to illustrate how the iCS coordinate the various modules (ns-3, SUMO, application) in order to influence the speed of vehicles approaching a traffic light. The speed advice in this specific case is zero to make a car come to a complete halt. The application logic applied for this demo is indicated in Figure 6.1 and can be categorized into the following interactions:

A car subscribes to the iCS of the generic subscription 'return car in zone', which will notify the application of any car arriving in a given zone.

- Upon reception of vehicles in the given zone, the application set the desired speed to zero, and requests a unicast message transmission to ns-3 to the specific vehicle just entered into the zone.
- Upon reception of the message by the car, the application is notified and move to the next step and request SUMO to effectively stop the vehicle.

Would the message not be received or if the subscription does not return the cars entering the zone where cars should be stopped, the application will not stop vehicles via SUMO.

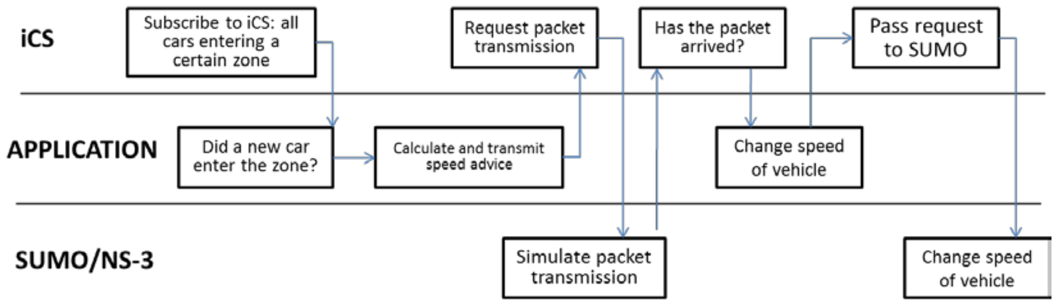

iCS
Subscribe to iCS: all cars entering a certain zone
Request packet transmission
Has the packet arrived?
Pass request to SUMO
APPLICATION
Did a new car enter the zone?
Calculate and transmit speed advice
Change speed of vehicle
SUMO/NS-3
Simulate packet transmission
Change speed of vehicle

Figure 6.1: Diagram showing the information flow in the community demo app

Prerequisites

This chapter assumes that the trunk of the COLOMBO branch is checked out, and all essential packages for building are installed. (The package list and some basic installation recommendations can be found in this file:

https://svn.dlr.de/COLOMBO/trunk/software/virtulaMachineEtc/postInstallVMScript.sh

Building COLOMBOS ns-3

Ns-3 uses the “waf” build framework, which by default exits from the build process already at gcc compiler warnings. In order to disable this behavior, ns-3 can be compiled, including some native examples by using the following commands:

```
CXXFLAGS="-Wall" ./waf configure --enable-examples
./waf -vv
```

It’s recommended to clean up before recompiling: by using “./waf clean” or even “./waf distclean”.

As a basic check two of ns-3’s native examples may be run. One example is the most simples ‘hello world’ prompting, the second example establishes a direct link between two communication nodes sending and sends packages using IPv4.

The ‘hello world’ example can be started by running: ./waf --run hello-simulator .The output should be similar to:

```
waf: Entering directory `/home/colombo/colomboSvn/COLOMBO/trunk/software/ns-3.7.0/build'
waf: Leaving directory `/home/colombo/colomboSvn/COLOMBO/trunk/software/ns-3.7.0/build'
'build' finished successfully (5.236s)
Hello Simulator
```

To enable the second example, copy the file examples/tutorial/first.cc to scratch/, recompile and run: ./waf; ./waf --run scratch/first

The output on the COLOMBO virtual machine is similar to:

```
waf: Entering directory `/home/colombo/colomboSvn/COLOMBO/trunk/software/ns-3.7.0/build'
waf: Leaving directory `/home/colombo/colomboSvn/COLOMBO/trunk/software/ns-3.7.0/build'
'build' finished successfully (8.356s)
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```

When both tests are successfully completed and all software packages are installed as mentioned above, we can consider ns-3 to be properly compiled.

Building SUMO and iCS

Building the COLOMBO branch of SUMO does not require any additional package or configuration different form the SUMO Trunk. Building instruction can be found on the project's web-page[10]. It's recommended to build SUMO with the graphical interface.

The iCS provides an autotools build environment, which follows the traditional sequence of '`./configure`' and '`make`'. In order to properly set up the build system it's recommended to run `make –f Makefile.cvs` first. For configuring the build for iCS being able to communicate with ns-3, SUMO and applications, the configure script must be run with the following parameters:

```
./configure  --enable-debug  --enable-sumo  --enable-ns3  --enable-applications  --enable-double-
precision --enable-log --with-gtest
```

For compiling the make command needs to be run.

If the building of the iCS was successful, iCS can be started from the command line by typing "./iCS". The iCS will then prompt its meta information.

Building the community demo app v2

The community demo app v2 code is located in the folder:

https://svn.dlr.de/COLOMBO/trunk/software/iTETRIS/community-demo-app-v2

In order to build the application, the simply the flowing commands need to be run:

```
make –f Makefile.cvs;  ./configure --enable-double-precision; make
```

Configure the community app v2 and the SUMO net and route files

As the original SUMO road network files were still in an old version and as the specification of the SUMO network and route files had been changed in between, these files needed to be rewritten and newly generated or converted using according scripts, available within the SUMO package. Additionally, the configuration files for the application, such as application-config-file.xml, itetris-config-file_nsTrunk.xml, LDMrules-config-file.xml, stations-config-file.xml, facilities-config-file.xml and itscoopdemoapp-config-file.xml needed specific adaptions. All files are contained now in the folder:

https://svn.dlr.de/COLOMBO/trunk/software/iTETRIS/community-demo-app-v2/example_sumo17

and enable iCS and the community demo app v2 to be run the with current version of the COLOMBO-SUMO branch by triggering the following command:

```
./iCS -c itetris-config-file_nsTrunk.xml
```

If the application works properly, the vehicles should stop at the left intersection of the road network.

[10] http://sumo.sf.net

7 Summary

The COLOMBO Overall Simulation System, built upon existing solutions for different scientific tasks, has left the stage of an experimental proto type. The operational potential of its interfaces have been proven, enabling future investigations of traffic management applications utilizing vehicular communication as well as conventional traffic detection.

The already existing COSS components have been tested for their functionalities. Even though not all functionalities are available yet, the interfaces between the components have been proved or extended to obtain an overall working system. Outcomes of other work packages, such as the tuning tool kit combining existing sub-tuners or off- and online connections between SUMO and PHEM have been integrated forming the overall simulation system. The iCS interfaces have been upgraded concerning the exchange of road network information to be suitable for current versions of SUMO. First application development using the tool kit is undertaken and the integration of traffic surveillance algorithms and controlling signalized intersections by applying swarm based policy selections is expected to be performed in the near future.

As all components are expected to change and further interfaces might be needed, maintaining the interaction between the components will be a permanent process of the software management of COSS. Yet each component is tested in its native projects only. A test system for COSS, testing larger scaled use case scenarios is to be established.

The distribution of COSS among partners in the consortium is realized either by sharing the source code via the SVN or rarely by exchanging virtual machine images. Once first findings with the COSS have been made, a distribution system will be designed, hand in hand with a user and developer documentation.

Appendix A – References

[Behrisch et al., 2008] Behrisch, Michael; Krajzewicz, Daniel and Wang, Yun-Pang (2008) Comparing performance and quality of traffic assignment techniques for microscopic road traffic simulations. In: Proceedings of DTA2008. DTA2008 International Symposium on Dynamic Traffic Assignment, 2008-06-18 - 2008-06-20, Leuven.

[COLOMBO, 2012] COLOMBO consortium. Cooperative Self-Organizing System for low Carbon Mobility at low Penetration Rates – COLOMBO, Annex I – Description of Work, 2012.

[COLOMBO D3.1, 2013] COLOMBO consortium. COLOMBO project’s Deliverable D3.1: “Prototype of Automatic Configuration and Tuning Tool Kit”, April 2013.

[ETSI, 2009] ETSI. ETSI TR 102 638: Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definition. 2009.

[FreeSim, 2013] FreeSim. http://www.freewaysimulator.com/ retrieved on 19 July 2013.

[Gawron, 1998] Gawron, C. “Simulation-based traffic assignment – computing user equilibria in large street networks,” Ph.D. Dissertation, University of Köln, Germany, 1998.

[Gorgorin et. al, 2006] Gorgorin, C.; Gradinescu, V. ; Diaconescu, R.; Cristea, V.; Iftode, L. (2006) An integrated vehicular and networking simulator for vehicular adhoc networks, in: 20th European Simulation and Modelling Conference, Toulouse, 2006.

[Hausberger, 2003] Hausberger S. (2003). Simulation of Real World Vehicle Exhaust Emissions; VKM-THD Mitteilungen; Volume 82; Verlag der Technischen Universität Graz; ISBN 3-901351-74-4; Graz 2003.

[Hausberger et al., 2009] Hausberger, S.; Rexeis, M.; Zallinger, M. and Luz, R. (2009) Emission Factors from the Model PHEM for the HBEFA Version 3. Report Nr. I-20/2009 Haus-Em 33/08/679, 2009.

[Hausberger et al., 2011] Hausberger, S.; Rexeis, M.and Luz R.(2009). PHEM das Modell der TU Graz zur Berechnung von Kfz-Emissonen und seine Datenbasis bei Euro 5 und Euro 6, Fachtagung Emissionen und Minderungspotenziale im Verkehrsbereich, Stuttgart, 21.07.2011.

[Hausberger et al., 2012] Hausberger, S.; Rexeis, M. and Luz, R.(2012). New Emission Factors for EURO 5 & 6 Vehicles, 19th International Conference „Transport and Air Pollution“, 26. – 27.11.2012, Thessaloniki.

[Hirschmann et al., 2010] Hirschmann, K.; Zallinger, M.; Fellendorf, M. and Hausberger, S. (2010). A new method to calculate emissions with simulated traffic conditions, Intelligent Transportation Systems ITSC 2010, p. 33-38.

[Hutter et al., 2011] Hutter, F.; Hoos, H.H. and Leyton-Brown, K. (2011). Sequential model-based opti- mization for general algorithm configuration. In *Learning and Intelligent Optimization, 5th International Conference, LION 5,* Lecture Notes in Computer Science. Springer, Heidelberg, Germany, 2011.

[Hutter et al., 2009] Hutter, F., Hoos, H.H., Leyton-Brown, K. and Stützle, T. (2009). ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research,* 36:267–306, October 2009.

[iTETRIS, 2013] iTETRIS consortium (2013). http://www.ict-itetris.eu retrieved on 19 July 2013.

[Jist/SWANS, 2013] Jist/SWANS (20013). http://jist.ece.cornell.edu/index.html retrieved on 19 July 2013.

[Joerer et al., 2012] Joerer, S.; Sommer, C. and Dressler, F.(2012). Towards Reproducibility and Comparability of IVC Simulation Studies--A Literature Survey 2012.

[Krajzewicz et al., 2008] Krajzewicz, D.; Boyom, Danilot Teta and Wagner, Peter (2008), Evaluation of the performance of city-wide, autonomous route choice based on Vehicle-to-Vehicle Communication, TRB 200, January 2008, Washington.

[Krajzewicz, 2010] Krajzewicz, Daniel (2010). Traffic Simulation with SUMO - Simulation of Urban Mobility. In: Fundamentals of Traffic Simulation International Series in Operations Research and Management Science. Springer. p. 269-294. ISBN 978-1-4419-6141-9. ISSN 0884-8289.

[Krajzewicz et al., 2012] Krajzewicz, Daniel; Erdmann, Jakob; Behrisch, Michael and Bieker, Laura (2012). Recent Development and Applications of SUMO - Simulation of Urban MObility. International Journal On Advances in Systems and Measurements, 5 (3&4), p. 128-138. ISSN 1942-261x.

[Krajzewicz et al., 2012b] Krajzewicz, Daniel; Bieker, Laura and Erdmann, Jakob (2012). Preparing Simulative Evaluation of the GLOSA Application. In: Proceedings CD ROM 19th ITS World Congress 2012 Wien, Österreich, Paper ID: EU-00630. ITS World Congress 2012, 22.-26. Oct. 2012, Vienna, Austria.

[Krajzewicz, 2013] Krajzewicz, D.(2013). Summary on Publications citing SUMO, 2002-2012, In: 1st SUMO User Conference, Berlin, 2013.

[Kun et al., 2007] Kun, Chen and Lei, Yu (2007). Microscopic Traffic-Emission Simulation and Case Study for Evaluation of Traffic Control Strategies. Journal of Transportation System engineering and information technology, 2007, 7(1), 93-100.

[López-Ibáñez et al., 2011] López-Ibáñez, M.; Dubois-Lacoste, J.; Stützle, T., and Birattari, M.(2011). The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Universite´ Libre de Bruxelles, Belgium, 2011.

[Luz et al., 2013], Luz R., Hausberger S.(2013). User Guide for the Model PHEM, Version 11.2; Institute for Internal Combustion Engines and Thermodynamics TU Graz, Graz, 2013.

[ns-2, 2013] ns-2 (2013). http://nsnam.isi.edu/nsnam/index.php/Main_Page retrieved on 19 July 2013.

[ns-3, 2013] ns-3 (2013). http://www.nsnam.org/ retrieved on 19 July 2013.

[OMNet++, 2013] OMNet++ (2013). http://www.omnetpp.org/ retrieved on 19 July 2013.

[Pell et al., 2013] Pell, Andreas; Meingast, Andreas and Schauer, Oliver (2013). Online traffic simulation software for heterogeneous road network – current state and future trends, 9th ITS European Congress, Dublin, Ireland, 4-7 June 2013.

[PELOPS, 2013] PELOPS (2013). http://www.ika.rwth-aachen.de/forschung/veroeffentlichung/1999/08.-09.02-3/ retrieved on 19 July 2013.

[Piorkowski et al., 2008] Piorkowski, Michal; Raya, Maxim; Lugo, Ada; Papadimitratos, Panos; Grossglauser, Matthias; Hubaux, Jean-Pierre (2008). TraNS: realistic joint traffic and network simulator for VANETs. In ACM SIGMOBILE Mobile Computing and Communications Review, vol. 12, num. 1, p. 31—33. http://infoscience.epfl.ch/record/113879?ln=en retrieved on 17 July 2013.

[Quadstone Paramics, 2013a] Quadstone Paramics (2013). http://www.paramics-online.com/ retrieved on 19 July 2013.

[Quadstone Paramics, 2013b] Quadstone Paramics (2013). http://www.paramics-online.com/insight/case-study-micro-simulation-of-a-traffic-fleet-to-predict-the-impact-of-traffic-management-on-exhaust-emissions/ retrieved on 19 July 2013.

[Rexeis, 2009] Rexeis M. (2009). Ascertainment of Real World Emissions of Heavy Duty Vehicles. Dissertation, Institute for Internal Combustion Engines and Thermodynamics, Graz University of Technology. October 2009.

[Rondinone et al., 2013] Rondinone, Michele; Maneros, Julen; Krajzewicz, Daniel; Bauza, Ramon; Cataldi, Pasquale; Hrizi, Fatma; Gozalvez, Javier; Kumar, Vineet; Röckl, Matthias; Lin, Lan; Lazaro, Oscar; Leguay, Jérémie; Haerri, Jérôme; Vaz, Sendoa; Lopez, Yoann; Sepulcre, Miguel; Wetterwald, Michelle; Blokpoel, Robbin and Cartolano, Fabio (2013) ITETRIS: a modular simulation platform for the large scale evaluation of cooperative ITS applications. Simulation Modelling Practice and Theory. Elsevier. DOI: 10.1016/j.simpat.2013.01.007. ISSN 1569-190X.

[Schünemann, 2011] Schünemann, B. (2011) V2X simulation runtime infrastructure VSimRTI: An assessment tool to design smart traffic management systems. In Computer Networks 55 (14), p. 3189-3198, http://www.sciencedirect.com/science/article/pii/S1389128611001605, retrieved 22. July 2013.

[Sommer et al., 2008] Sommer, C.; Yao, Z.; German, R. and Dressler, F.(2008). On the need for bidirectional coupling of road traffic microsimulation and network simulation Proceedings of the 1st ACM SIGMOBILE workshop on Mobility models, 2008, 41-48.

[SUMO, 2013] SUMO (2013). http://sumo-sim.org/ retrieved on 19 July 2013.

[Tielert et al., 2010] T. Tielert, M. Killat, H. Hartenstein, R. Luz, S. Hausberger, T. Benz (2010) The Impact of Traffic-Light-to-Vehicle Communication on Fuel Consumption and Emissions; Internet of Things 2010 Conference, Japan,2010.

[TransModeler, 2013] TransModeler (2013). http://www.caliper.com/transmodeler/ retrieved on 19 July 2013.

[Veins, 2013] Veins (2013). http://veins.car2x.org/ retrieved on 19 July 2013.

[Versit+, 2013] Versit+ (2013). http://www.delftdimensions.nl/versit.aspx retrieved on 19 July 2013.

[VSimRTI, 2013] VSimRTI (2013). http://www.dcaiti.tu-berlin.de/research/simulation/ retrieved on 19 July 2013.

[VISSIM, 2013] VISSIM (2013). http://vision-traffic.ptvgroup.com/de/ retrieved on 19 July 2013.

[Wikipedia SVN 2013] Wikipedia (2013). https://en.wikipedia.org/wiki/Apache_Subversion retrieved on 29 July 2013.

[Zallinger, 2010], Zallinger M. (2010). Mikroskopische Simulation der Emissionen von Personenkraftfahrzeugen. Dissertation, Institut für Verbrennungskraftmaschinen und Thermodynamik, Technische Universität Graz, April 2010.