D6.1

An integrated platform for the simulation and the assessment of traffic management procedures in Transition Areas

Project Acronym	TransAID
Project Title	Transition Areas for Infrastructure-Assisted Driving
Project Number	Horizon 2020 ART-05-2016 – GA Nr 723390
Work Package	WP6 System Integration and Evaluation
Lead Beneficiary	DLR
Editor / Main Author	Leonhard Lücken DLR
Reviewer	Sven Maerivoet TML
Dissemination Level	PU
Contractual Delivery Date	31/10/2018 (M14)
Actual Delivery Date	31/10/2018 (M14)
Version	v1.00

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 723390.

Document revision history

Version	Date	Comments
v0.0	2018-08-01	Initial draft structure
v0.1	2018-08-28	Added first content to Sections 2 and 3
v0.2	2018-08-29	Added input on Sec. 2.2.1 - Traffic Simulation from CERTH, and content in Sec. 3.3.1 and 3.3.2.
v0.3	2018-08-30	Updated Sec 3.3 (mainly figures)
v0.4	2018-09-10	Started Section 0 on requirements and integrated first draft for Sec. 2.2.3
v0.5	2018-09-10	Reviewed draft for Sec. 2.2.3
v0.6	2018-09-10	Added content for Section 3.4.2
v0.7	2018-09-10	First content for Sec. 4.4
v0.8	2018-09-12	Added 2.2.3.2, still needs review
v0.9	2018-09-12	Integrated and reviewed 2.2.3
v0.10	2018-09-18	First draft for Sections 5.1 and 5.2
v0.11	2018-09-20	Integrated Sections 4.1 and 4.2.1-4
v0.12	2018-09-20	Integrated Section 2.1
v0.13	2018-09-20	Integrated extended Section 2.2.1
v0.14	2018-09-21	Added Section 0
v0.15	2018-09-24	Incorporated 2.2.2 and 3.3.3, updated 3.2.4
v0.16	2018-09-27	Some modifications to 2.1, integrated UMH input: 3.4.3 and 4.2.5, communication KPIs in 5.1 and some responses to comments.
v0.17	2018-09-27	Included Sec. 5.3, reordered 5.2 and 5.3; Modified 0, added 1.3, moved 4.4
v0.18	2018-10-01	First version for Figure 5.7
v0.19	2018-10-01	Integrated 4.3
v0.20	2018-10-02	Added Sec. 6, cleaned comments

v0.21	2018-10-05	Added specification of SSM thresholds in 5.1
v0.22	2018-10-17	Removed requirement “safe spot reservation”, integrated further input from UMH in Section 2.2.2, added reference in 0
v0.23	2018-10-30	Reviewed by TML
v1.00	2018-10-31	Final changes, incorporation of review comments, cleaned formatting

Editor / Main author

Leonhard Lücken (DLR)

List of contributors

Evangelos Mintsis (CRT), Dimitrios Koutras (CRT), Alejandro Correa (UMH), Sven Maerivoet (TML), Lars Akkermans (TML), Kristof Carlier (TML), Inge Mayeres (TML), Yun-Pang Flötteröd (DLR), Michael Behrisch (DLR), Julian Schindler (DLR)

List of reviewers

Sven Maerivoet (TML)

Dissemination level:

■ PU : Public

□ RE : Restricted to a group specified by the consortium (including the Commission Services)

□ CO : Confidential, only for members of the consortium (including the Commission Services)

Table of contents

List of Figures

List of Tables

1. Introduction

1.1 About TransAID

As the introduction of automated vehicles (AV) becomes feasible, even in urban areas, it will be necessary to investigate their impacts on traffic safety and efficiency. This is particularly true during the early stages of market introduction, when automated vehicles of different SAE levels, connected vehicles (able to communicate via V2X) and conventional vehicles will share the same roads with varying penetration rates.

There will be areas and situations on the roads where high automation can be granted, and others where it is not allowed or not possible due to missing sensor inputs, high complexity situations, etc. At these areas many automated vehicles will change their level of automation. We refer to these areas as “Transition Areas”.

TransAID develops and demonstrates traffic management procedures and protocols to enable smooth coexistence of automated, connected, and conventional vehicles, especially at Transition Areas. A hierarchical approach is followed where control actions are implemented at different layers including centralised traffic management, infrastructure, and vehicles.

First, simulations are performed to examine efficient infrastructure-assisted management solutions to control connected, automated, and conventional vehicles at Transition Areas, taking into account traffic safety and efficiency metrics. Then, communication protocols for the cooperation between connected/automated vehicles and the road infrastructure are developed. Measures to detect and inform conventional vehicles are also addressed. The most promising solutions are then implemented as real world prototypes and demonstrated at a test track and during the second iteration possibly under real urban conditions. Finally, guidelines for advanced infrastructure-assisted driving are formulated. These guidelines also include a roadmap defining activities and needed upgrades of road infrastructure in the upcoming fifteen years in order to guarantee a smooth coexistence of conventional, connected, and automated vehicles.

Iterative project approach

TransAID will perform its development and testing in two project iterations. Each project iteration lasts half of the total project duration. During the first project iteration, the focus is placed on studying Transitions-of-Control (ToCs) and Minimum Risk Manoeuvres (MRMs) using simplified scenarios. To this end, models for automated driving and ToC/MRM are adopted and developed. The simplified scenarios are used for conducting several simulation experiments to analyse the impacts of ToCs at TAs, and the effects of the corresponding mitigating measures.

During the second project iteration, the experience accumulated during the first project iteration is used to refine/tune the driver models and enhance/extend the proposed mitigating measures. Moreover, the complexity and realism of the tested scenarios will be increased and the possibility of combining multiple simplified scenarios into one new more complex use case will be considered.

1.2 Purpose of this document

The TransAID project conducted first baseline simulations without including traffic management procedures, nor explicitly taking into account V2X communications. As soon as traffic management is included into the simulations, V2X communication processes are to be included as well. This may be done in a simplistic fashion to obtain a prototype and a first indication towards the efficacy of specific approaches. For a more realistic simulation, the detailed representation especially of the transmission errors occurring for wireless communications between road side units and connected vehicles is necessary. Therefore, the TransAID project will conduct simulations of traffic management procedures involving the coupling between a traffic simulation and a communication simulation.

This report gives an overview of the iTETRIS software framework, which will be employed for the integrated simulation. The basic framework consists of the microscopic traffic simulator SUMO, the network communications simulator ns-3, and the middleware iCS, which all are open-source programs. In addition we describe the program structure of the iCS and the planned extensions to these programs, which will be implemented within TransAID. In particular, this includes the description of technical interfaces, which the traffic management application modules corresponding to the different TransAID services (see deliverable D2.2) can use. That is, the document should enable developers in the context of WP4 to implement the application modules. Finally, we also outline the tools to assess the simulation results by statistical and visual analysis.

1.3 Structure of this document

The present report first gives a general introduction (Section 2) to the use of integrated simulations of traffic dynamics and V2X communication, and reviews some of the existing software with a focus on the iTETRIS framework employed in TransAID. In Section 3, we report on the current state of the iTETRIS platform and present its technical side giving some more detail in Sections 3.1 to 3.3. Section 3 concludes with Subsection 3.4, which lists functionalities required in the context of the TransAID project and identifies implementation load. Section 4 is concerned with the practical side of the usage of the simulation platform. It describes installation and configuration of the software, and gives a guidance for the implementation of own applications in Section 4.3 and for setting up tests (Section 4.4). For the assessment of simulation results, the TransAID project will develop an evaluation tool chain, which is described in Section 5. Finally, a summary and outlook on the upcoming work concludes this document in Section 6.

1.4 Glossary

Abbreviation/Term	Definition
ACC	Adaptive Cruise Control
AD	Automated Driving
ADAS	Advanced Driver Assistance Systems
API	Application Programming Interface
AV	Automated Vehicles
C-ITS	Cooperative Intelligent Transport Systems
C2C-CC	Car2Car Communication Consortium
CAM	Cooperative Awareness Message
CAV	Cooperative Automated Vehicle
CPM	Collective Perception Message
CV	Cooperative Vehicle
DENM	Decentralised Environmental Notification Message
DX.X	Deliverable X.X
ERTRAC	European Road Transport Research Advisory Council
ETSI	European Telecommunications Standards Institute
GUI	Graphical User Interface
HMI	Human Machine Interface
iCS	iTETRIS Control System
ITS	Intelligent Transport System
ITS-G5	Access technology to be used in frequency bands dedicated for European ITS
LDM	Local Dynamic Map
LOS	Level Of Service (from Highway Capacity Manual)
LV	Legacy Vehicle

MCM	Manoeuvre Coordination Message
MRM	Minimum-Risk Manoeuvre
OMNeT	Objective Modular Network Testbed
RAT	Radio Access Technology
RSI	Road-Side Infrastructure
RSU	Road-Side Unit
SAE	Society of Automotive Engineers
SUMO	Simulation of Urban MObility
TA	Transition area
TCI	Task Capability Interface
TM	Traffic Management
ToC	Transition of Control
TraCI	Traffic Control Interface
TransAID	Transition Areas for Infrastructure-Assisted Driving
V2I	Vehicle-to-infrastructure
V2V	Vehicle-to-vehicle
V2X	Vehicle-to-anything
VMS	Variable Message Signs
WP	Work Package
XML	Extensible Markup Language

2 Simulation Platforms for V2X

2.1 The Need for an Integrated Simulation Platform

2.1.1 Context

One of the main objectives of WP6 is to integrate the various simulation components that have been developed in the other work packages. For example, work package WP3 is concerned with creating a realistic simulation environment that mimics driver behaviour, and work package WP5 deals with creating a digital infrastructure for realistic communications. In between, work package WP4 describes various recipes for performing traffic management for the various use cases and scenarios. As of now, these simulations work in a standalone setting, whereby they are not yet depending on each other. That means for example that both WP3 and WP4 currently assume perfect communication capabilities, instead of more realistic ones with limited bandwidths et cetera. They are, so to speak, each operating in their own, ideal world [1], [2].

In order to assess the impact of WP4's TransAID traffic management measures within the various use cases, we need an integrated simulation platform that encompasses different models of realistic vehicle behaviour and V2X communication capabilities [3], [4]. Without such integration, it is difficult to evaluate such measures realistically. Indeed, having a realistic simulation of driver and vehicle behaviour is one thing, but when traffic management measures are to be simulated reliably, finite bandwidth, realistic sensor capabilities, and finite ranges of interaction have to be taken into account. This kind of interaction between the various components is needed when for example the communication aspects influence the dynamics of the traffic flows.

As such, we need a mechanism that ties the different components together, in a uniform way, guiding the communications between modules and allowing us to shape and analyse various scenarios, each to a custom level of detail. In essence, there are various simulated processes that work both independently and in a more cooperative fashion, whereby they are at times mutually dependent on each other.

The aforementioned mechanism resembles an *integrated* simulation environment. *Integrated*, because it allows better orchestration of the various interactions between its different components, and allowing us to have a tighter control over the different simulations. This is necessary as some of the components run at different time scales, e.g. communication occurring on the level of millisecond versus more high-level traffic demand strategies that operate on a second- to minute-basis. Thus there is the need to (centrally) synchronise the entire simulation of the real world via a suitable environment [4], [5].

Aside from reasons of having a (centralised) control and a system to pass messages between various simulation components, an integrated simulation platform also serves another important purpose. Extracting measurements that correspond between the various components allow us to more straightforward test and validate the simulation, as well as calculating the necessary performance statistics. This is done using, e.g., built-in KPIs, or using derived statistics that themselves are based on more rudimentary data such as vehicle trajectories, time series, et cetera. Because an integrated simulation platform allows us to have full control, we can setup controlled experiments, greatly enhancing the scientific reproducibility of the experimental outcomes.

Some of these experiments and their output can be considered in an off-line fashion, whereby one component generates a set of data (e.g., trajectories) which can then be fed into one or more other components [6]–[8]. However, if a significant bidirectional interaction between the communication processes and the traffic dynamics must be assumed, a stepwise procedure consisting of trajectory generation followed by an offline evaluation of communication performance will not reflect the behaviour of the real system.

Having an adequate integrated simulation platform is thus a keystone in order to achieve the required level of realistic detail within the simulation environments. In addition, there are some requirements that we impose on the selection and use of such a platform within TransAID:

- The platform should be open-source (see also Section 2.1.2 for a more elaborated explanation); the main benefits are that we can build upon available dedicated research within the sector, and it allows us to feedback to the scientific community. The latter is especially necessary in order to gain 'trust' in the modelling approach and, consequentially, its results. In addition, when new insights or developments become available from outside the TransAID project, these can then be directly incorporated within the simulation platform and if needed be externally verified by other parties (take new message sets, other vehicle models, et cetera as examples).

- The platform should be able to deal with large-scale simulations, in that a large-enough geographical scope can be simulated, together with a high amount of traffic demand and various types of vehicles. This in turn leads to lots of dynamic interactions within the different traffic flows in the simulation, as well as a high degree of (albeit mostly local) communications. As the simulations are to represent a mirror of the real world as closely as possible, the integrated simulation platform should be able to replicate these setups.

2.1.2 The benefits of software development under an open-source flag

Software developed in the open-source domain has the great incentive that it can be propelled by a large community of developers. Where there is usually financing available for the development of proprietary software within a company, this is not always the case for the development of free software. The fact that a larger group of dedicated people can contribute is a very important benefit in terms of economies of scale. Additional advantages are that the maintenance costs, which typically represent a fairly large share of the software price, go down (i.e., the maintenance is effectively outsourced to volunteers), and that more eyes can look at the same source code, which hugely benefits debugging (the same holds true for testing the software). In the case of proprietary software, a design of a number of software architectures is usually implemented, while in the open-source context this is more decentralised and self-organised eco-system (that can still be controlled and coordinated).

A critical remark in the previous is that for such (large) projects there is only real success when there is sufficient critical mass that the project picks up. There are many software projects that get 'stranded' because there are not enough developers available. For small projects this is not such a disaster as they are often started and maintained by individuals, but for the larger projects this can cause problems.

Despite the fact that most of the software models tend to be developed in an accessible academic or private setting, the traditional approach towards the creation of the majority of ready-to-use software is mainly oriented towards its commercialisation. In many cases, the main stream company policy is aimed towards the non-disclosure of the models' internals, effectively reducing these

commercial packages to advanced versions of black-box models. When such software starts to grow more mature and complex, it becomes increasingly difficulty to answer the question 'What is really under the hood?' The importance of this statement should not be underestimated, as it can be vital for researchers to be acquainted with a model's inner workings, features, and limitations, when interpreting results for, e.g., policy decisions.

This lack of openness can be remedied by developing the software under an open-source flag. From this point on, the complete underlying model structure remains revealed at all times, as it is now possible for many programmers to read, redistribute, and modify the source code. When a company exhibits this sagacity, the unlocked potential of open-source can be fully brought into play. One of the main benefits of this paradigm is that there are effectively 'many eyes looking at one single problem'. As a direct result, the debugging, maintenance, and support life cycles of such software become more transparent, as opposed to the monolithic approach typically encountered in propriety software. If such an open-source project is properly managed (which implicitly requires skilled people), it can receive an increased gain from the feedback of its user base. Already, several successful examples of this type of software development can be found in real life, e.g., the Linux operating system, the Netscape and Mozilla web browsers, the StarOffice suite and OpenOffice.org project.

When releasing open-source software, there literally exists a myriad of licences that regulate the commercial and non-commercial use of this type of software, as well as its incorporation in third-party software. Archetypical examples are the Free Software Foundation's GNU General Public Licence (GPL) with the popular catch phrase 'free as in free speech, not as in free beer', the Open-Source Initiative (OSI) which provides a marketing vessel for 'selling' free software, the Creative Commons Licences (CCL) that offer a flexible copyright for creative work, . . .

Finally, note that in our discussion, we did not state anything about legal issues such as the management of intellectual property rights, issues related to the patenting of ideas, inventions, and algorithms, et cetera. Indeed, most licences undoubtedly steer clear of these topics, allowing their interpretation to remain up to the developer and/or the company. However, the central core that forms the business model for open-source software is to freely share the software, whilst selling support. With respect to academic institutions and their management of intellectual property, dissemination of algorithms by means of publications in journals might be discouraged. In these cases, we still deem it appropriate to publish the results, as we believe that the money remains in the selling of the software. Another less-commercial track that can be followed is to release the software as a web service, thus effectively hiding the underlying code of an algorithm's implementation when confidentiality issues and ownership of intellectual property rights are at stake.

2.2 Overview of Available Software

This Section gives a brief overview of the software that is available for the coupled simulation of vehicle movements and wireless communications.

2.2.1 Traffic Simulation

Simulation is the imitation of the operation of a real-world system or process, while computer simulation is the replication of the real-world system or process on a computer. In the transportation domain, computer simulation has been widely used to facilitate the planning, design, and operation of transportation systems (roads, airports, rail, ports, et cetera).

Road traffic (more specifically, the physical propagation of traffic flows as opposed to more strategic traffic planning models) has been simulated macroscopically, mesoscopically, and microscopically [9]. These approaches adopt different mathematical models to describe the dynamics of traffic flows and are selected based on the scope and scale of the simulation analysis [10]. Macroscopic models have the highest level of aggregation, lowest level of detail, and are based on continuum mechanics, typically entailing compressible fluid-dynamic models [11]. The models assume a deterministic relationship between the flow, speed, and density of the traffic stream (i.e., the fundamental diagram). Mesoscopic models also have a high level of aggregation, low level of detail, and are typically based on a gas-kinetic analogy in which driver behaviour is explicitly considered [12]. Both macroscopic and mesoscopic models are modelled using partial differential equations, with both first- and higher-order versions. Microscopic models have a low level of aggregation, high level of detail, and are typically based on models that describe the detailed interactions between vehicles in a traffic stream, using car-following, lane-changing, and gap-acceptance concepts [13]. For a more exhaustive and comprehensive overview of traffic flow modelling we refer the reader to [].

Most popular microscopic traffic simulators (commercial or open-source) are time-based (the temporal propagation of traffic is tracked every time step), stochastic (the model parameters are subject to randomness), and can be either static (model inputs are not affected by the evolution of time) or dynamic (model inputs evolve in time as a function of various elements). A list of the most popular microscopic traffic simulators detailing their attributes is presented in Table 1.

For the purposes of the TransAID project, a microscopic simulation has to be employed because the single vehicles' positions and states have to be known for the design of individually tailored traffic management advices and a realistic simulation of the corresponding wireless communication processes. For the integration of a traffic simulation into a coupled framework, an interface is an essential component since it allows other applications to affect a running simulation, e.g., by imposing vehicle responses to received V2X messages.

The SUMO microscopic traffic simulator is selected for the investigation of vehicular communications within the context of TransAID. The justification for the selection of SUMO is manifold. SUMO was developed by the Institute of Transportation Systems at the German Aerospace Center (DLR), which is a project partner in TransAID. Thus, there is significant expertise with respect to SUMO within the consortium. SUMO was used in the context of previous EU co-funded projects for the investigation of vehicular communications, such as iTETRIS [14], PRE-DRIVE C2X [15], DRIVE C2X, and COLOMBO [16]. Therefore, it is compatible with the iTETRIS platform that will be also used in TransAID. Moreover, some TransAID partners were also involved in the latter EU co-funded projects, thus bringing their relevant expertise into the consortium. Finally, SUMO is open-source (licensed under the Eclipse Public License (EPL) 2.0)

and thoroughly documented regarding its extensibility. Hence, new driver models reflecting existing vehicle automation prototypes which rely on vehicular communications can be integrated. Further it provides the TraCI API (short for Traffic Control Interface), which allows a dynamical interaction with a running simulation, allowing a very flexible interaction with peripheral modules, which is also highly advantageous for research that strives to model novel phenomena and processes like transitions of control as it is the case for the TransAID project.

SUMO is space-continuous, time-discrete, and can simulate large-scale road networks encompassing multiple transport modes. A network topology including road-side infrastructure information (e.g., intersection control) and traffic demand are required inputs for SUMO simulations and have to be provided as XML files. Each vehicle moves individually in SUMO and has its own route. Longitudinal and lateral movements are dictated by car-following, lane changing, and gap-acceptance models as described in [17]. SUMO was initially developed and oriented for academic use to improve the validation of different traffic management applications. However, recently it is increasingly being used by the industry mainly for the simulation of surrounding traffic to test vehicles automations (such as ADAS and higher automation systems) considering V2X as well.

Table 1: Attributes and scope of selected commercial and open-source microscopic traffic simulation software.

Simulation Software	Developer	Attributes	Scope
Aimsun Next (commercial)	Aimsun, Barcelona, Spain (https://www.aimsun.com/)	• Based on the Gipps car-following model [18] • Based on the Gipps lane-changing model [19]	• Traffic operations assessments of any scale and complexity • Environmental impact analysis, safety analysis, work zone management • Toll and road pricing, Highway Capacity Manual (HCM) analysis • Evaluation of travel demand management (TDM) strategies and intelligent transportation systems (ITS), signal control plan optimisation • Customised through API
Paramics (commercial)	Quadstone, UK (http://www.paramics-online.com/)	• Based on the psycho-physical car-following model by Fritzsche [20]	• Freeway and highways, traffic engineering, junctions and roundabouts • Public transport, ITS and toll plazas • Environment and emissions, pedestrian modelling • Customised through API
SUMO – Simulation of Urban Mobility (open-source)	DLR, Germany (http://sumo.dlr.de)	• Uses the Krauss car-following model [21]	• Microscopic simulation - vehicles, pedestrians and public transport are modelled explicitly • Online interaction – control the simulation with TraCI API • Simulation of multimodal traffic, e.g., vehicles, public transport, and pedestrians • Time schedules of traffic lights can be imported or generated automatically • No artificial limitations in network size and number of simulated vehicles • Supported import formats: OpenStreetMap, VISUM, VISSIM, NavTeq • SUMO is implemented in C++ and uses only portable libraries
TSIS-CORSIM™ (commercial)	McTrans at the University of Florida	• Uses the PITT car-following model [22]	• Freeway and surface street interchanges, signal timing and signal coordination • Land use traffic impact studies and access management studies • Emergency vehicles and signal pre-emption, toll plazas and truck weigh stations • Ramp metering, HOV lanes and HOT lanes • Two-lane highways with passing and no-passing zones • Incident detection and management, run-time extensions (RTE) for researchers
Vissim (commercial)	PTV Group (http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/)	• Uses the psycho-physical car-following model by Wiedemann [23]	• Motorway and arterial traffic (any node geometry) • Multimodal systems, active traffic management • Public transport, emissions modelling, pedestrian modelling • Virtual testing of autonomous vehicles

2.2.2 Communication Simulation

A widespread methodology for performance analysis in the field of communications is the use of network simulators. A network simulator is software that allows to model arbitrary communication networks by specifying the behaviour of the nodes and the communications channels [24]. Currently available network simulators are mostly based on the paradigm of discrete event simulators. This type of simulators is based on the successive execution of events triggered by the nodes in the network. For example, a node will schedule an event whenever it needs to transmit a packet to another node. A list of events is maintained by the simulator and the events are executed based on the defined execution time. The simulation itself is performed by successively processing the events in the queue. Nowadays, the most employed networks simulators are: ns-2, ns-3 and OMNeT++.

ns-2 (Network Simulator version 2) is an open-source (licensed for use under General Public License, GNU) discrete event simulator written in C++ and the Object-Oriented Tool Command Language (OTcL)[1]. The internal mechanism for the node simulation is written in C++ while OTcL is employed by the user for the configuration of the simulations. It provides support for the simulation of TCP, UDP, routing, and multicast protocols over wired and wireless networks. In terms of scalability to large networks scenarios, ns-2 presented poor performance due to the high memory usage [24]. The development of ns-2 was stopped in 2009 and the simulator is no longer maintained.

ns-3 (Network Simulator version 3)[2] is an open-source (licensed under the GNU GPLv2 license) discrete event simulator created with the purpose to overcome the limitation of ns-2 in terms of performance [25], [26]. The core of ns-3 was completely redesigned and there is no compatibility between ns-2 and ns-3. ns-3 is written in C++ and it provides an optional Python script interface. ns-3 architecture has been defined to support real testbed integration and network virtualisation. This allows the coupling in real time of simulated nodes with real nodes to perform combined simulations. ns-3 supports an increasing amount of models for vehicular communications. For example, it provides support for link layer protocols, such as IEEE 802.11p, IEEE 1609.4 WAVE geobroadcasting routing protocols and the ITS-G5 protocol stack [4]. Additionally, ns-3 supports multichannel operations for the ITS-G5 stack.

OMNeT++ (Objective Modular Network Testbed in C++)[3] is a discrete event simulator tool with an Eclipsed-based integrated development environment [13]. OMNeT++ provides a component-based hierarchical and modular architecture where components and modules are written in C++ and then assembled into larger components and models using a high-level language (NED) [27]. Unlike ns-3, OMNeT++ provides the kernel of the network simulator while the implementation of real communications protocols is provided by external frameworks developed by third parties. Therefore, the compatibility between different available frameworks is not always guaranteed. OMNeT++ includes modules for vehicular communications such as, IEEE 802.11p DSRC, IEEE 1609.4 WAVE, and the ITS-G5 protocol stack [4].

[1] The Network Simulator – NS-2, http://www.isi.edu/nsnam/ns/

[2] The Network Simulator – NS-3, https://www.nsnam.org/

[3] OMNeT++ Discrete Event Simulator, https://www.omnetpp.org/

For the purpose of the TransAID project, the selection of a network simulator must be based on the number of available models for the simulation of vehicular communications and on the performance of the simulator. Both, ns-3 and OMNeT++, provide models to support vehicular communications while models in ns-2 are outdated. In terms of performance, there are different works in the literature that compare the performance of networks simulators showing that the performance of ns-3 is slightly better than the performance of OMNeT++ and far better than the performance of ns-2 for large networks [24], [28]. Besides ns-3 is a full open-source program, unlike OMNeT++, which is not free for commercial purposes (the participation of a company in an EU-funded project requires a commercial license). Thus, employing OMNeT++ will reduce the potential number of users that will benefit from the extensions of the network simulator developed at the TransAID project.

2.2.3 Frameworks

Cooperative wireless communication simulations have been extensively applied for simulative tests of message exchange between vehicles and infrastructure (V2X), see Section 2.1. If a dynamic interaction of the communication processes and the traffic dynamics has to be simulated, a coupled execution of traffic and communication simulation is required. The most common approach to achieve this coupled simulation is to use a middleware to connect a traffic simulation program and a communication simulation program [4].

Several respective traffic simulation platforms have been developed for evaluating traffic-related performances, such as safety and efficiency, of the proposed V2X-based cooperative traffic management strategies and applications. An important component of a coupled simulation is the synchronising and coordinating middleware module, which conciliates the fixed simulation step size of the traffic simulation with the usually event-based progress of the communication simulation. The architectures and functionalities of three popular frameworks are briefly introduced in the following Sections.

2.2.3.1 iTETRIS

The iTETRIS platform is an open-source simulation platform and was established within the iTETRIS project of the 7th Framework Programme for Research and Technological Development by the European Commission. Its development has been continued beyond the project period within different other projects [4] for the study of various subjects as traffic estimation [3], traffic management for low penetration rates [29], smartphone sensing [30], dynamic routing [31], traffic light control [32], etc.

This platform aims at accurately simulating large scale scenarios within a cooperative ITS communication system [14]. It couples the traffic simulator SUMO [33] and the wireless communication simulator ns-3 [26]. The combination of these two simulators does not only allow performing large-scale simulations, but also provides the means for accessing the effects of vehicular information exchange on traffic mobility, network performance, and the influences of vehicle movement on wireless communication and the respective protocols.

[4] Most notably, the EU fp7-project COLOMBO (http://www.colombo-fp7.eu/), but also the MOTO (http://www.fp7moto.eu/) and HIGHTS (http://www.hights.eu/) projects, and the French collaborative research project SINETIC (https://team.inria.fr/rits/projet/sinetic/), for instance.

There are three actors in the iTETRIS platform (see also Figure 1), i.e. central, vehicle, and roadside subsystems. The Access Technologies layer follows the *Continuous Air interface for Long and Medium distance* (CALM) objectives and ensures seamless communication over six coexistent communication technologies. The Transport & Network layer provides different protocols for a complete communication within a given scenario. The Facilities layer provides the common functionalities, shared by different applications, such as data structures for storing and processing, while the Application layer contains different user applications, adopting a structure resembling the ETSI specifications for ITS applications [34]. Three categories, i.e. Road Safety, Traffic Efficiency, and Other Applications, are defined in the application layer. The Management deals with the information exchanges across the horizontal layers and the Security layer is to implement security services for the horizontal layers and the Management layer, see Figure 2.1. More detailed information can be found in [35], [36], and in Section 3.

The central module of the iTETRIS platform is the iCS (iTETRIS Control System), which is a middleware and serves as a control component to synchronise the connected simulators and applications. The iCS starts the different processes, sets up the given simulation environment, schedules control messages to be sent to ns-3, and synchronises and controls the simulation execution. The different components are connected via TCP/IP Sockets. The iCS is therefore capable to couple SUMO, ns-3, and applications, which run on various remote machines given with their IP addresses and ports. Regarding the coupling between iCS and ns-3, the communication layers, indicated in Figure 2.1, are implemented and separated from the rest of the platform, so that there is no need to query other blocks or layers. The functionalities related to communication, transport and network, access technologies and management, are implemented in ns-3, while iCS contains the functionalities for the connected applications and simulation-related controls. The traffic simulation SUMO is controlled by the Traffic Control Interface (TraCI) in favour of the principle of a modular approach. Offering an API accessible via a socket interface, TraCI serves to control the traffic simulation process (start, pause, and stop), modify the simulation environment, and retrieve data from the simulated objects.

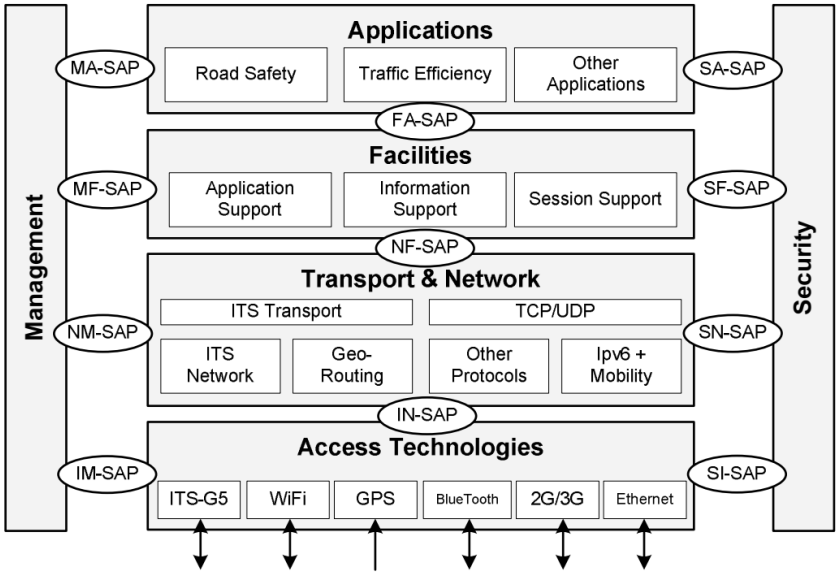

Figure 2.1: The architecture of the iTETRIS platform, adopted from the iTETRIS website[5].

[5] http://ict-itetris.eu/simulator/test_beds.htm, last accessed on 2018-09-06.

2.2.3.2 Veins

Veins is an open-source vehicular network simulation framework where both, road traffic and vehicle communication, can be simulated [37]. It contains a number of radio propagation models to make simulations as realistic as possible. Furthermore, Veins is coupled with the traffic simulator SUMO and the event-based network simulator OMNeT++ [25] such that the interactions between network and traffic can be precisely simulated online. Both simulators are connected via a TCP socket and the communication protocol is based on SUMO's TraCI so that SUMO and OMNeT++ can be coupled bi-directionally. The other components in the Veins framework serve to set up, run, and monitor the investigated simulation. Furthermore, a physical layer with the modelling toolkit MiXiM is also used for applying models for radio interference and shadows casted by static and moving objects [38], [39]. Veins has been applied for various problems involving VANETs, ADAS testing [40], [41], message authentication [42], dynamic traffic assignment [43], congestion detection [44], and others.

In addition, there is an extension of Veins, called Plexe [45]. Plexe allows for example to simulate platooning systems (i.e. automated car-following) for autonomous vehicles and cruise control models. It is also able to analyse vehicle dynamics, vehicular control systems, large-scale scenarios, networking protocols, and cooperative manoeuvres.

2.2.3.3 VSimRTI

VSimRTI is developed by the Daimler Center for Automotive Information Technology Innovations (DCAITI) and stands for V2X Simulation Runtime Infrastructure [15]. This framework allows to assess different solutions of cooperative intelligent transportation systems, as for instance benchmarking CAM message transmission [46], securing communication privacy [47], platoon management [48], cooperative speed harmonisation [49], and more. VSimRTI is designed to couple more different aspects of the studied system, such as user behaviour, vehicle traffic, wireless communication, smart grid, and applications, and couples with different communication and traffic simulators

The VSimRTI handles all simulator-related management tasks and allows simulators to be exchanged or integrated easily within the framework. Instead of a fixed simulation coupling, the most suitable simulator can therefore be adopted (e.g., SUMO, PHABMACS, VISSIM, …).

Furthermore, the architecture of the contained application simulator VSIMRTI_AppNT is based on the ETSI ITS standard and contains two layers: one is the sandboxed application layer and the other one is the facility layer, which is to provide different supports for an application (e.g., station positioning and message management), information (LDM database and station type/capacities) and communication (addressing mode and Geonet support). Besides supporting OMNeT++ and ns-3, an own communication simulator VSimRTI_Cell is provided.

Table 2 gives a concise comparison of the different functionalities of iTETRIS, Veins, and VSimRTI.

Table 2: Comparison of functionality for iTETRIS, Veins and VSimRTI

Item	iTETRIS	Veins	VSimRTI
Open-source	x	except OMNeT++	
Coupled traffic simulators	SUMO	SUMO	SUMO, PHABMACS, VISSIM
Coupled network communication simulators	ns-3	OMNeT++	ns-3, OMNeT++, SNS, VSimRTI_Cell
Implementation language			
Protocol interface	iCS/TraCI	TraCI	
Flexibility to couple with other simulators			x
Custom applications	x	x	x

2.2.4 iTetris as a Basis for TransAID

TransAID employs the iTETRIS framework because of two main reasons. First of all, in the consortium there is a dedicated expertise regarding SUMO as it is actively developed at DLR, and different partners have been involved in the development of the iCS during the iTETRIS and COLOMBO projects.

Secondly, the iTETRIS framework is presently the only framework, whose components (traffic simulation, communication simulation, and middleware) are all open-source; see the previous parts of Section 2. This has the major benefit of giving us full control over extending the framework for the specific requirements of the project's simulation cases (new message sets, vehicle types, and behaviours, etc.).

A related aspect is the academic return of an open-source project. As already elaborated upon in Section 2.1, it leads to an elevated transparency and better reproducibility of our results, as well as it lets other researchers make use of improvements to the platform resulting out of the TransAID project.

iTETRIS is highly modular, giving the user the possibility to plug components easily in and out during testing and evaluation, or to combine the required functionality from building blocks in the form of different applications. The modularity makes it possible to extend the basic functionality of the platform (i.e. extensions to the iCS, SUMO, ns-3, and the application interface) in a way that others may profit from it as well, while keeping proprietary or unpublished components (e.g., novel algorithms) undisclosed. Further advantages of the iCS are ETSI conformance [14] and the architectural orientation on large scale simulations, without sacrificing accurate radio propagation effects [50].

3 Technical Aspects of the iTETRIS Platform

This section provides an overview on the most important components and information flows within the coupled framework consisting of the traffic simulation (SUMO), the wireless communication simulation (ns-3), the middleware (iCS), and iCS application modules (TransAID services).

It also discusses in detail, which interfaces are required for the purposes of the TransAID project and which of these have to be newly included into the different components.

3.1 General Introduction to iTETRIS and its components

iTETRIS denotes a modular collection of coupled programs, which are in charge of different functionalities required for a coupled simulation of interacting traffic management procedures, wireless communications, and vehicle movements. For each of these three components, there is at least one program taking on the corresponding part. Vehicle movements are simulated by the microscopic traffic simulator SUMO, wireless communications are simulated by the network simulator ns-3, and (possibly several) applications simulating traffic management procedures have to be provided by the user. Further, the iTETRIS Control System (iCS), as a fourth component, is responsible for the synchronisation of the different simulators, which may run at different temporal resolutions (see Figure 3.1 for a schematic overview).

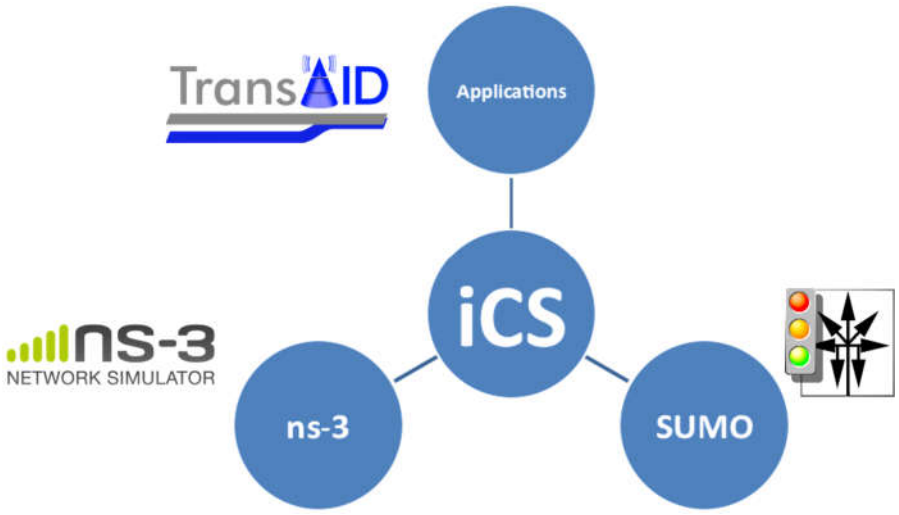

Figure 3.1: Schematic structure of the iTETRIS simulation framework. The iCS middleware manages the coupling of traffic simulation (SUMO), communications simulation (ns-3), and traffic management applications.

This Section is dealing with the description of the inner structure of the central iCS-component of the iTETRIS framework, allowing users and developers to attain an understanding of the various possibilities to use and extend this tool.

3.2 iCS Program Structure

3.2.1 General overview

The iTETRIS Control System (iCS) is a middleware connecting traffic simulation, wireless communication simulation, and ITS application simulations via sockets. It offers a flexible API regarding the connected simulators and can in principle work with different traffic and communication simulators, if these support basic server functionality.Up to this day however, clients have been integrated only for SUMO (TraCI) and ns-3 (iNCI). Furthermore, the iCS defines its own protocol for the communication with iTETRIS apps, which have to be developed including an interface compliant with the iCS-API.

To provide a synchronisation of object identities and temporal succession between all connected modules, the iCS provides a number of data structures and methods which are important for the program flow. To guide developers, who wish to take advantage of the open-source by realising their own extensions or modifications to the software, we give an overview of the structural (classes) and dynamical (program flow) aspects of the iCS in the following Sections.

Relative to the base folder of the iTETRIS repository, the code base of the iCS is contained within the `iCS/src/` subfolder, which in turn contains several folders:

- `foreign` contains foreign libraries for socket communication (`tcpip`) and random number generation (`mersenne`)
- `ics` contains the core program code of the iCS, which will be discussed further in the following sections
- `unittest` contains a single test for loading a digital map in SUMO's net.xml format
- `utils` contains various utility classes and methods, e.g., logging, geometry, program configuration, xml parsing, and others

3.2.2 iCS Classes

The top level container of the iCS is the C++ class `ics::ICS`, which contains little essential functionality aside from parsing the configuration files and launching SUMO, ns-3, and the applications. See Figure 3.2 for a schematic overview of the class structure within the iCS.

The code, which coordinates the coupled simulation process, is located within the class `ics::SyncManager`. This class contains members, which are responsible for the management of different components of the simulation (see Table 3 and Figure 3.2).

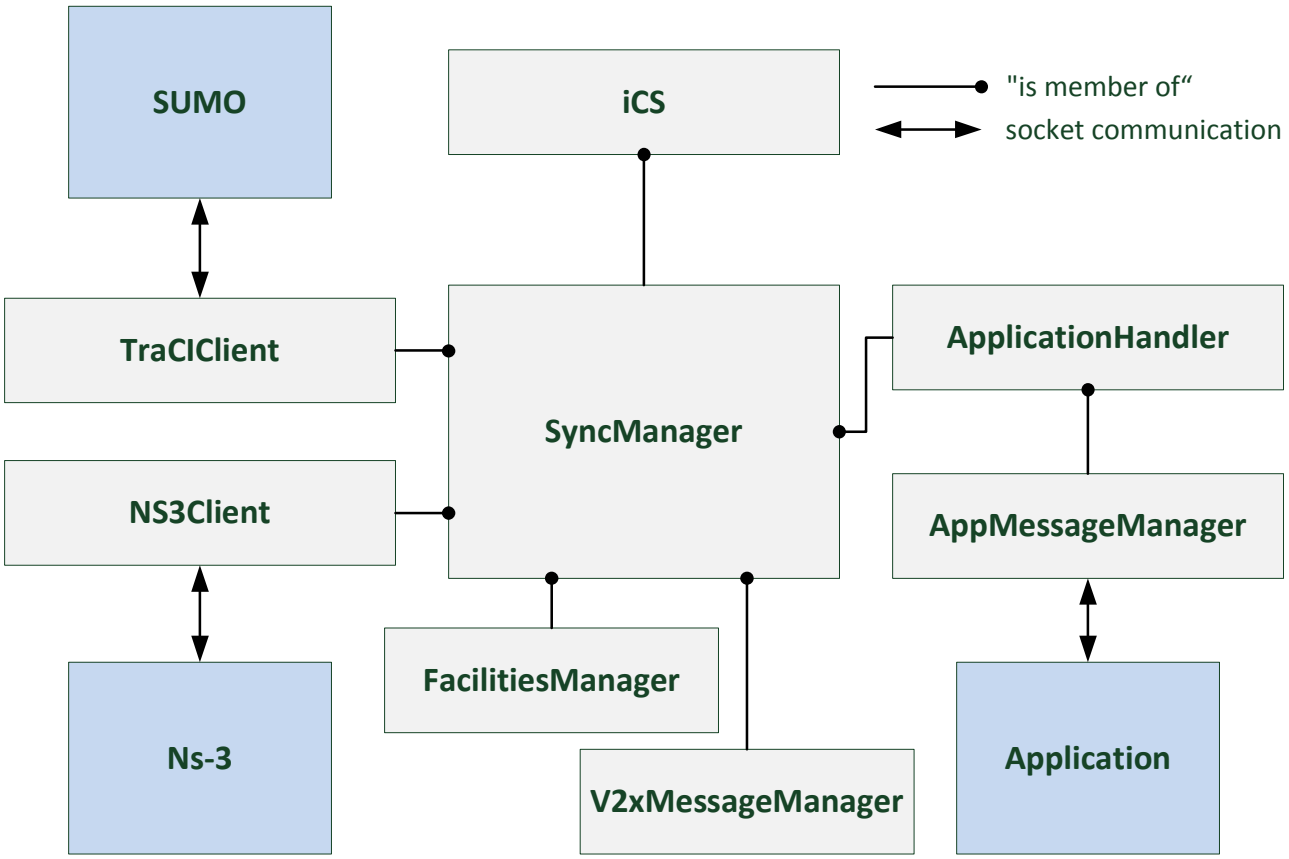

Figure 3.2: Main classes and interfaces of the iCS.

Table 3: Overview of the main SyncManager members

Class / Member's name	**Notes**
`TraCIClient / m_trafficSimCommunicator`	Manages the connection with SUMO.
`Ns3Client / m_wirelessSimCommunicator`	Manages the connection with ns-3.
`ApplicationHandler / M_applicationHandlerCollection`	Manages the connection of the applications.
`V2xMessageManager / m_v2xMessageTracker`	Keeps track of the messages and the information attached scheduled in the Wireless Communications Simulator.
`FacilitiesManager / m_facilitiesManager`	Interface between the iCS Facilities blocks and the rest of the iCS.

3.2.3 iCS Program Flow

At start up, iCS processes the command line options and loads the main configuration file, which also references the configuration of its components, namely SUMO, ns-3, and the user applications. If these files are parsed correctly, the integrated system is initialised. SUMO, ns-3, and the applications are launched as servers and connected to the respective iCS clients. After the setup is completed, the simulation loop within `ics::SyncManager::Run()` is started. Figure 3.3 shows these steps schematically.

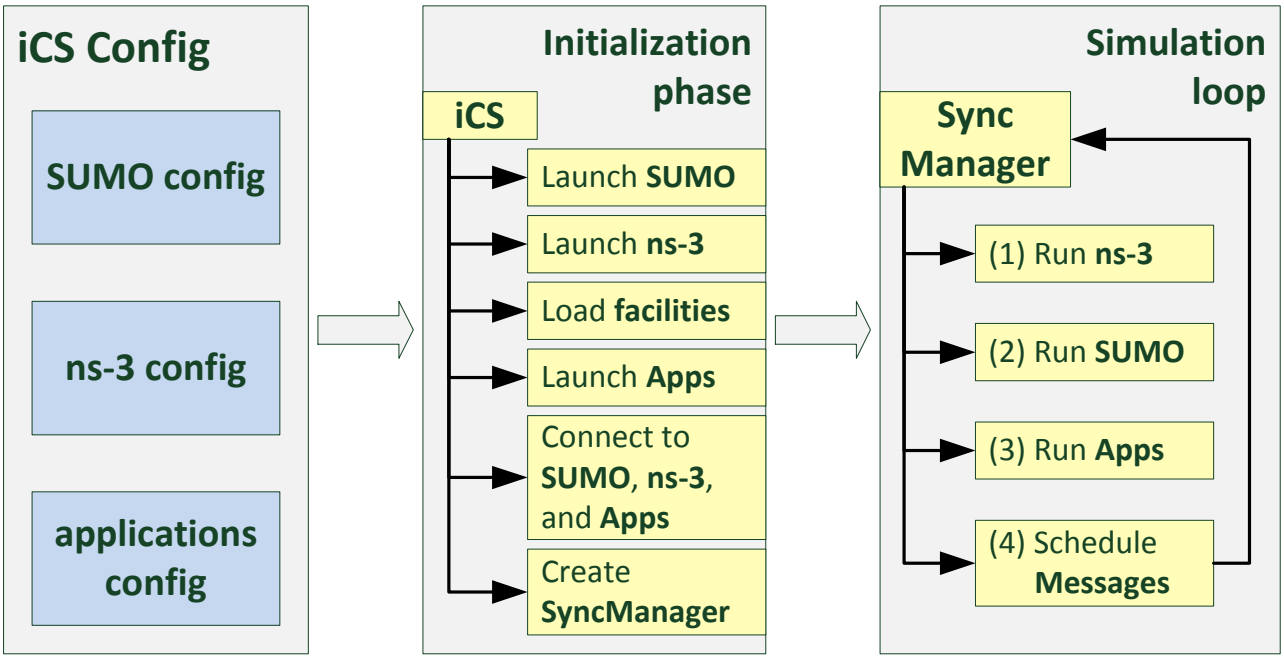

Figure 3.3: Overview of the iCS program flow.

3.2.4 The Simulation Loop

The simulation loop contains the main parts of the iCS functionality, see Figure 3.3. It ensures the synchronised execution and manages the data exchange between the different connected simulators. To this end, it also regularly updates the centralised information about the status of V2X messages and the movement of mobile stations.

It is organised into four subsequent steps, which we will briefly present in the following paragraphs.

Step 1: Run ns-3

In this first step of each simulation step, a corresponding step in ns-3 is triggered via the Ns3Client, where the simulation of message transmissions is taken out. When the simulation is completed, the Ns3Client obtains information about all received messages, and updates the corresponding data structures according to the message types, i.e., message tables and subscriptions to received messages are updated, see Figure 3.4.

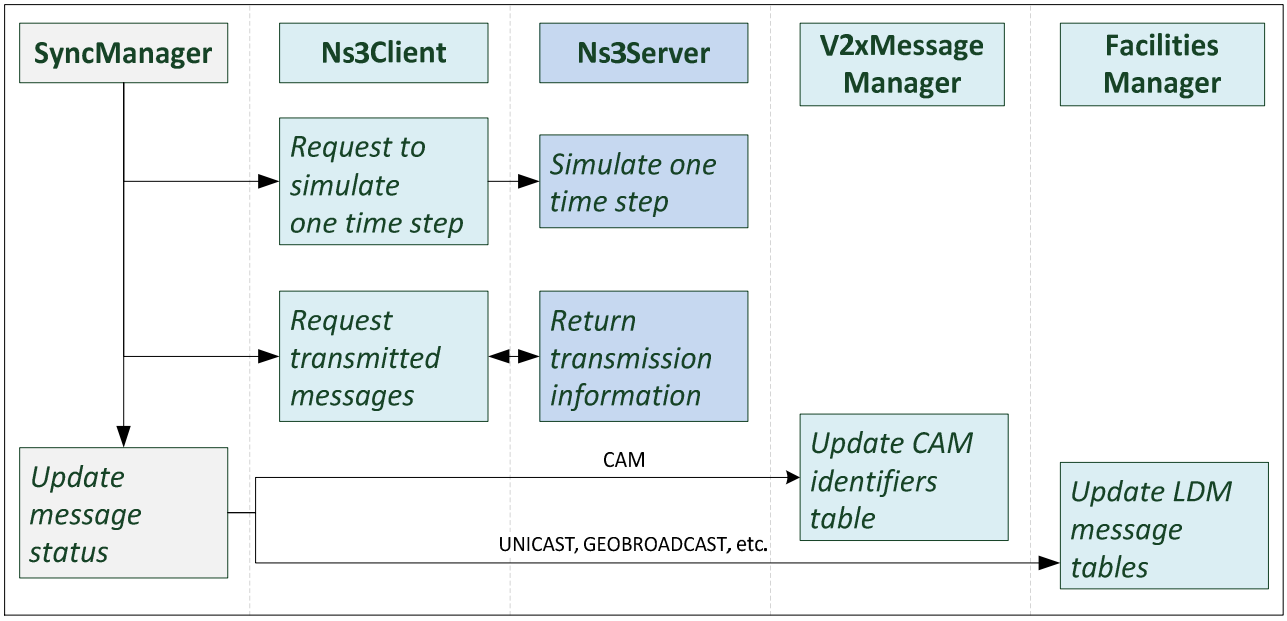

Figure 3.4: Schematic sequence diagram for Step 1 of the simulation loop.

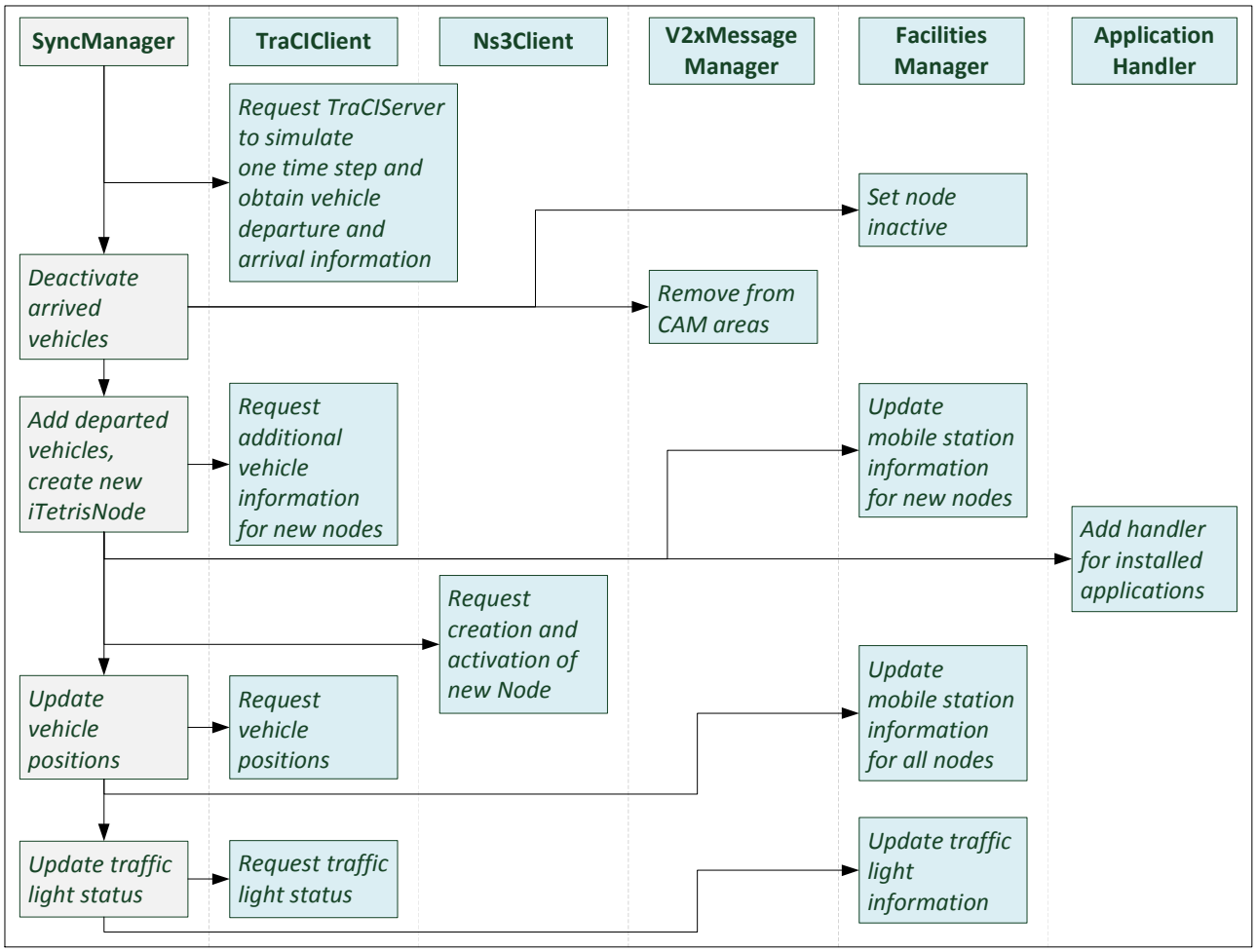

Figure 3.5: Schematic sequence diagram for Step 2 of the simulation loop.

Step 2: Run SUMO

Following the communication simulation, a simulation step of the traffic simulator is requested if the current time is a multiple of the traffic simulator's time step length, which may be larger than the step length used within the iCS. SUMO reports two lists of vehicles, i.e. those that have been inserted and those that have been removed within the last simulation step (since the iCS subscribes to this information when connecting to SUMO). Nodes are removed or inserted within the iCS data structures according to this list, and other modules (ns-3 and connected applications) are informed (see also Figure 3.5).

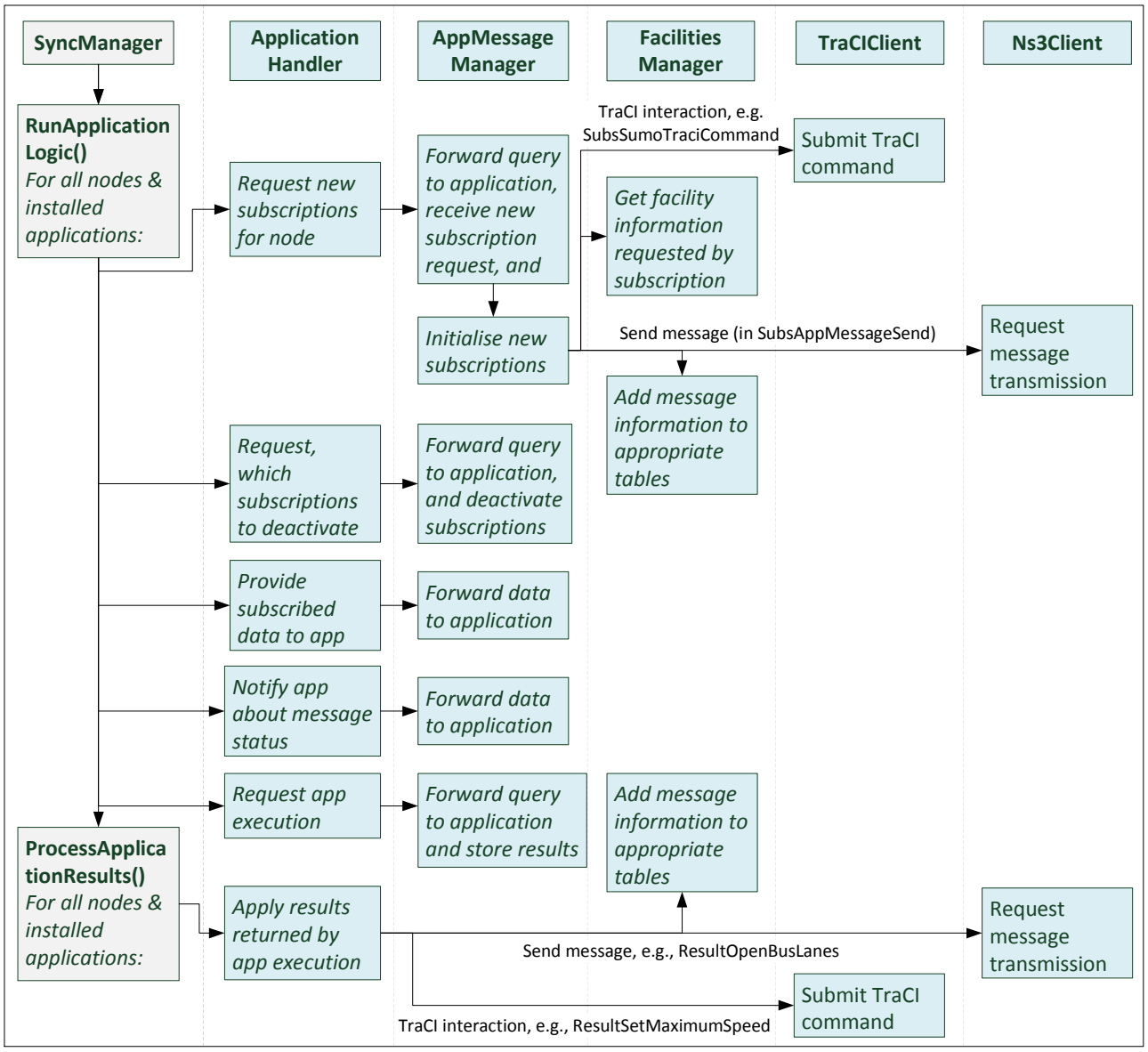

Figure 3.6: Schematic sequence diagram for Step 3 of the simulation loop.

Step 3: Run Apps

In this step the interaction with the applications takes place. The iCS queries for all nodes the installed applications and whether a new node-specific iCS-subscription should be created. Subscriptions are the mechanism by which applications can address services offered by the iCS. This may be by request of specific data as, e.g., messages received by the corresponding node, but also an arbitrary TraCI command to be forwarded to SUMO or a request to simulate the transmission of a specific message via ns-3. Similarly, the application is queried for all nodes to known whether old subscriptions should be dropped or remain active for a repeated request.

After updating the subscriptions, the data corresponding to its active subscriptions are sent to each application. Additional information about received messages is transmitted to applications whose *ResultContainer* class indicates that this is requested.

Next, all installed applications' main logic is executed, returning potential results of the execution within a *ResultContainer* making them accessible to other applications.

Furthermore, a *ResultContainer* provides an alternative mechanism to subscriptions to apply custom interactions with the iCS by means of its method `applyResult()`, which is called subsequently.

Step 4: Schedule messages and update vehicle positions in ns-3

The last phase of one iCS simulation step prepares the ns-3 for the execution of the next simulation step. The complete process is shown in Figure 3.7. First, the V2X messages are scheduled. In particular, the complete list of iTETRIS nodes is looped and for every node, the parameters of the CAM management are updated. This includes the parameters that control the transmission of CAMs but also the tables that stores the CAMs and their subscriptions. Then, the table that stores the message sent is updated and the expired messages are deleted. Finally, the ns-3 nodes are updated based on the output of SUMO. By this way, the nodes that have left the simulation in SUMO are deleted from ns-3 and the new nodes created in the SUMO simulation are created in the ns-3 environment.

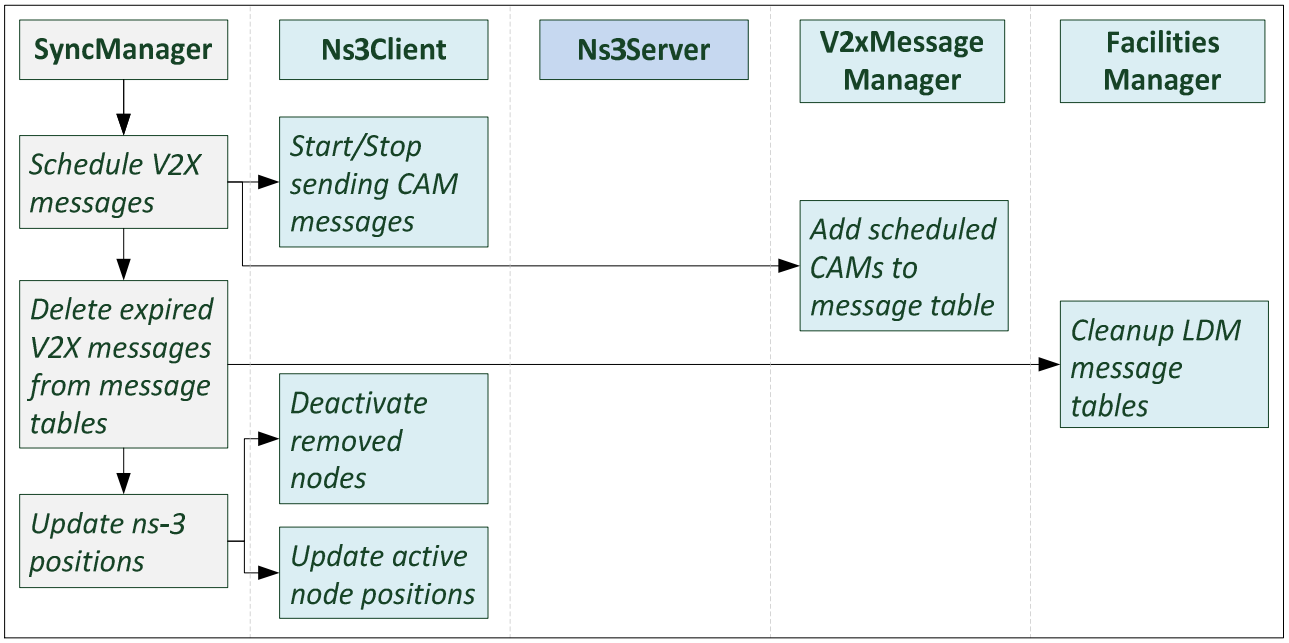

Figure 3.7: Schematic sequence diagram for Step 4 of the simulation loop.

3.3 Interaction with Applications

The application API of the iCS currently offers two ways of interaction for applications connected to one of its TCP/IP sockets. The first is to issue a *subscription* and the second is to include information in the application's *results*. The latter mechanism was originally introduced to make data available across connected applications, and developers are encouraged to use it only for this purpose, see [36]. In the following we give an overview about the two functionalities, which must be taken into account for the development of iCS applications.

3.3.1 Subscriptions

Within each simulation step, the iCS queries for all nodes and all applications installed on that node whether an interaction with or via the considered node is desired. If this is the case, an according *Subscription request* is sent by the application in response. Figure 3.8 shows details of the iCS-processes involving subscriptions.

Currently, 18 different *Subscription types* are included with the iCS, of which many concern very specific interactions. Since the extensions taken out in the context of the COLOMBO project, it is recommended to use only a limited number of Subscription types [36]. The following list contains the subscriptions, which TransAID will adopt[6]:

- `SubsGetFacilitiesInfo`:
  To obtain topological information about objects in the facility layer.
- `SubsSumoTraciCommand`:
  To pipe a TraCI command through the iCS directly to TraCI. The TraCI-response is piped back to the application at the corresponding station.
- `SubsGetMobilityInfo`:
  To obtain information on the node's movement from the information stored in the iCS facilities.
- `SubsAppMessageSend`:
  To schedule a V2X message transmission by the corresponding station.
- `SubsAppMessageReceive`:
  To be informed about successfully received V2X messages at the corresponding station.
- `SubsXApplicationData`:
  To access shared data from another connected application.
- `SubsSetCamArea:`
  Make vehicles in a given area periodically send CAMs.
- `SubsGetReceivedCamInfo:`
  To be informed about CAM messages received by the subscribed station.

The restriction to these subscriptions is on the one hand intended to keep the iCS API simple, and on the other hand to prevent the use of other existing subscription types that are deprecated and may be removed in the future.

[6] Note that [36] lists `SubsAppPullTraci` and `SubsAppPushTraci` instead of `SubsSumoTraciCommand`. The latter is used preferably as it covers the functionality of both.

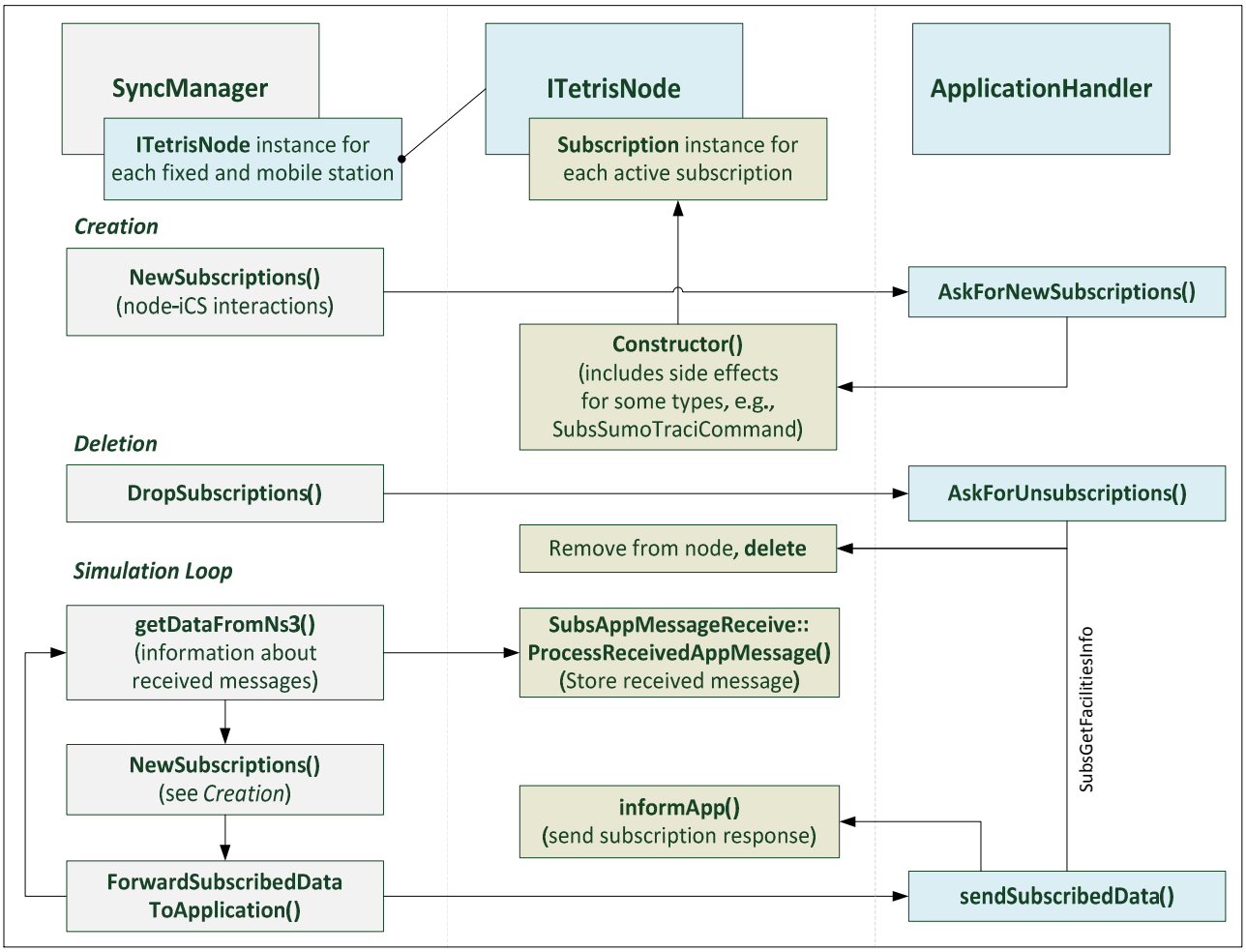

Figure 3.8: Lifecycle of a Subscription instance.

3.3.2 Data Exchange between Applications

The configuration of an application (see Section 4.2) must specify a result type for the application which determines the structure of the applications result after executing its main logic. This result is sent back to the iCS after the application's main logic has been executed and stored within a corresponding *ResultContainer*. It is recommended to use either the generic *ResultContainer* type *ResultGeneric*, or *ResultVoid*, if no data needs to be returned by an application. All other types are deprecated and might be removed in the future.

Albeit its declared role of acting as a storage place for data exchange, the *ResultContainer* class includes a legacy mechanism to issue state changes within other modules of the iCS by including corresponding code into the method `ResultContainer::applyResult()`. We emphasise that this is not considered as good practise. Preferably, only subscriptions should be used for the interaction with other iCS modules.

Figure 3.9 shows the interactions, in which a *ResultContainer* instance can be involved.

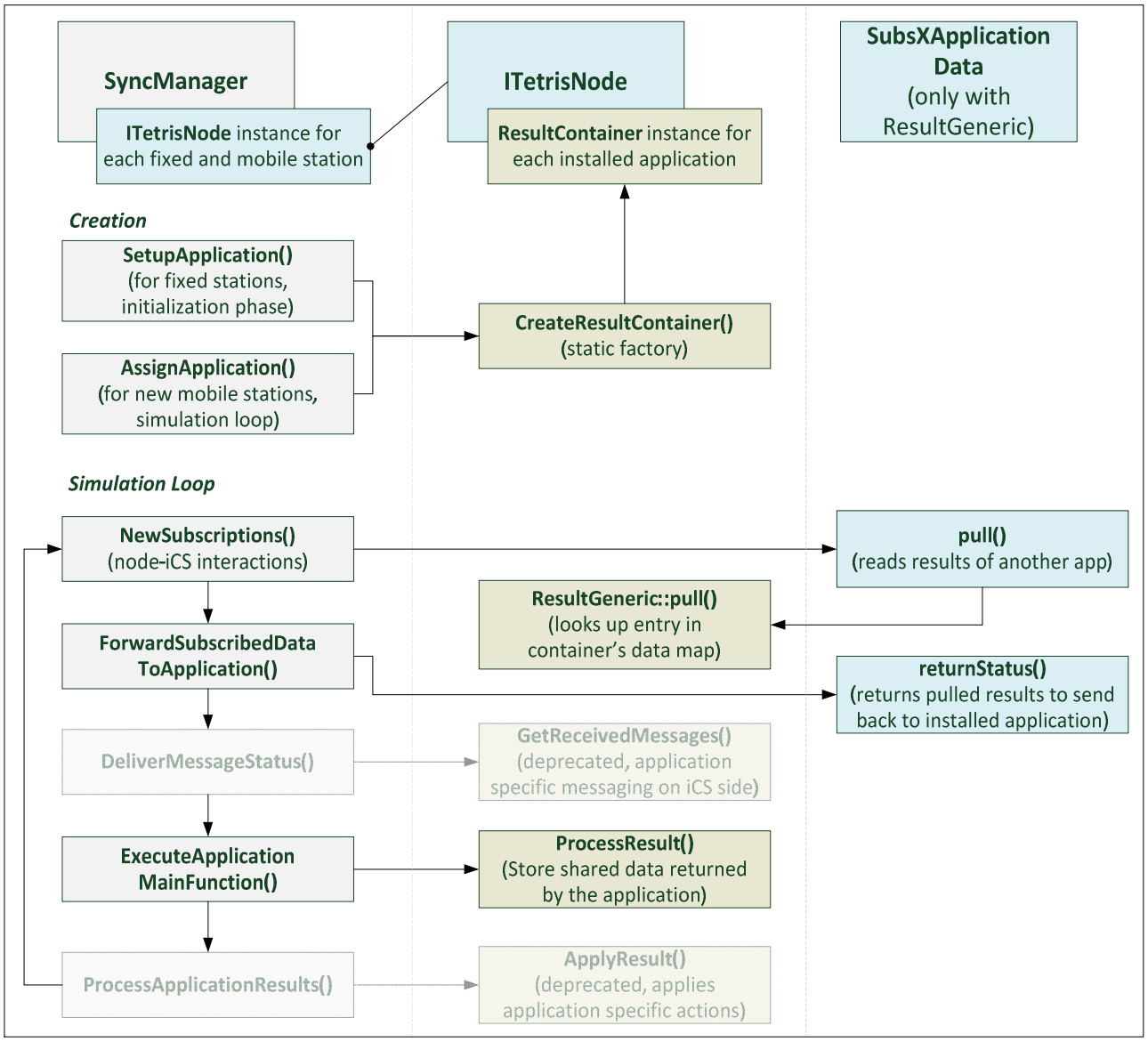

Figure 3.9: Interactions of a *ResultContainer* instance.

3.3.3 Sending and Receiving V2X Messages

The transmission and reception of messages in the iTETRIS framework starts in the application module. The sequence for the transmission of a generic message is shown in Figure 3.10. The application logic decides when a message must be sent and sends a subscription to the iCS calling the appropriate method of the *SubscriptionHolder* instance belonging to the sender. The *SubscriptionHolder* provides different methods for configuring the corresponding subscription instance of type *SubsAppMessageSend* with the appropriate transmission mode (e.g, *sendGeobroadcast()* for the geobroadcast mode or *sendUnicast()* for the unicast mode). Similarly, for the reception of messages the application uses the *SubscriptionHolder* instance to configure the *SubsAppMessageReceive* subscription. The corresponding message is scheduled for transmission when the iCS receives the subscription. To do so, it first generates the message header and the message payload. The message header is stored in the *messageMap* while the payload is stored in the *iFPT* table of the facilities layer in the iCS. The rationale behind this is to reduce the amount of information exchanges between the iCS and the ns-3. Only the length of the payload is transmitted

to ns-3 as only the size and not the specific values of the payload are relevant for the simulation of the communications. Then, the iCS commands ns-3 to send the message. In each simulation step, the iCS retrieves the information of received messages from ns-3. All successfully transmitted messages are processed and the headers of the messages are used to identify the corresponding message in the *messageMap*. Then, the payload is recovered from the *iFPT* table and stored in the table of received messages, the *iFMT*. This table contains an entry for each received message with a list of the nodes that have received the message. Then, the message is sent to the application module where it is processed.

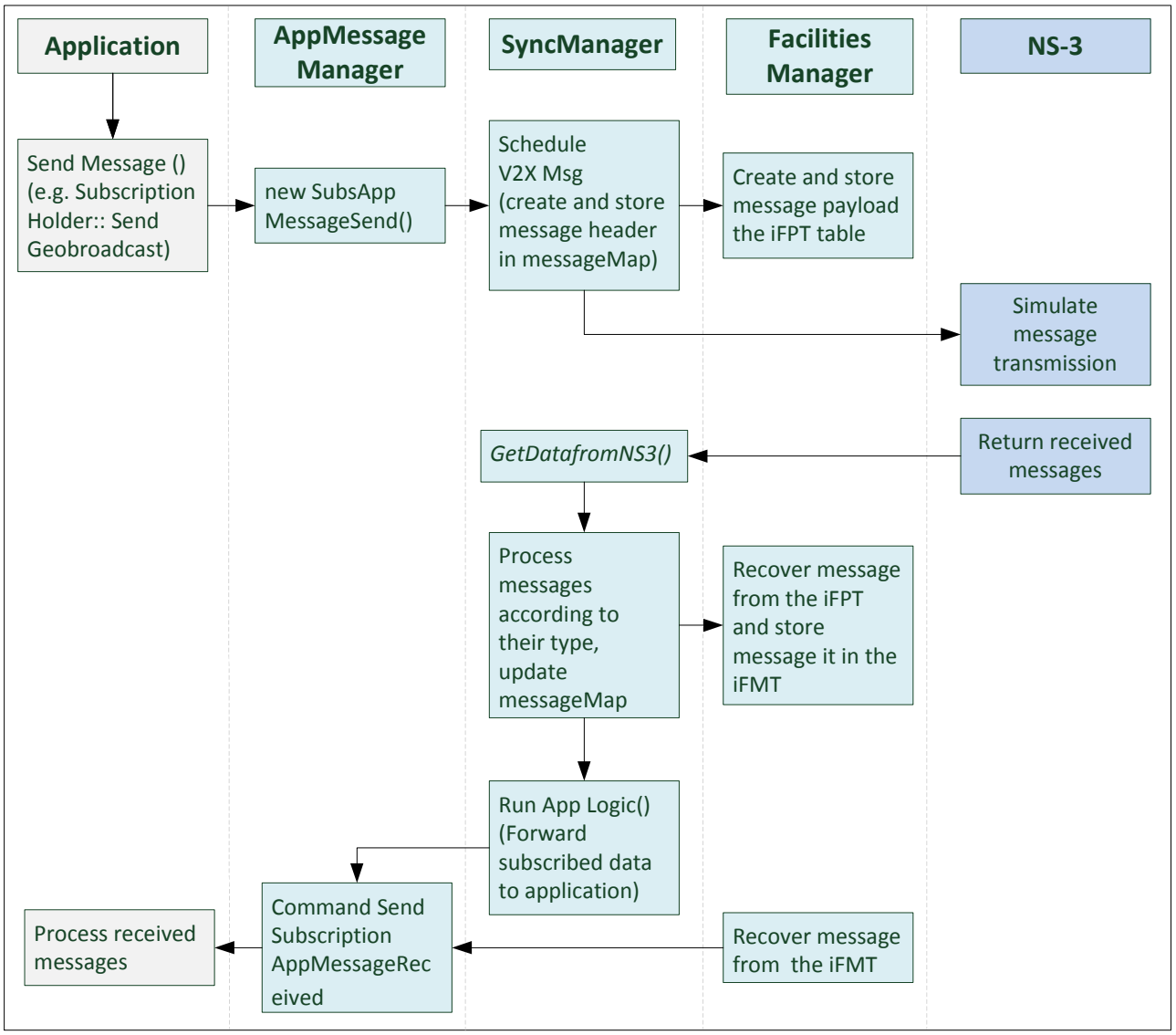

Figure 3.10: Sequence for the transmission and reception of a message.

The transmission of CAM messages does not use the generic message transmission procedures although it follows a similar structure. The sequence for the transmission and reception of CAM messages is shown in Figure 3.11. The triggering of the periodic transmission of CAM is done by creating the so called *CAMarea*, a *CAMarea* is a geographic area assigned to a node which represents its maximum communication range. The rationale behind this is to reduce the simulation time of the network because only the nodes inside the area will be considered as potential receivers

and therefore the simulation of the message transmission will be limited to the receivers inside the *CAMarea*. Note that if this method is not employed, the network simulator will check every node as a potential receiver, which can be a very time consuming process if the number of nodes is large. The *CAMareas* are also employed to modify the transmission frequency as a function of the number of nodes in the area, i.e. stopping the transmission if there are no nodes in the area. The *CAMareas* are updated in every simulation loop and if necessary the parameters of the CAM transmission are updated too. The rest of the sequence is similar to the generic transmission of messages with the exception of the *messageMap* which is substituted by a specific table for CAM, the *ScheduledCAMmessageTable*.

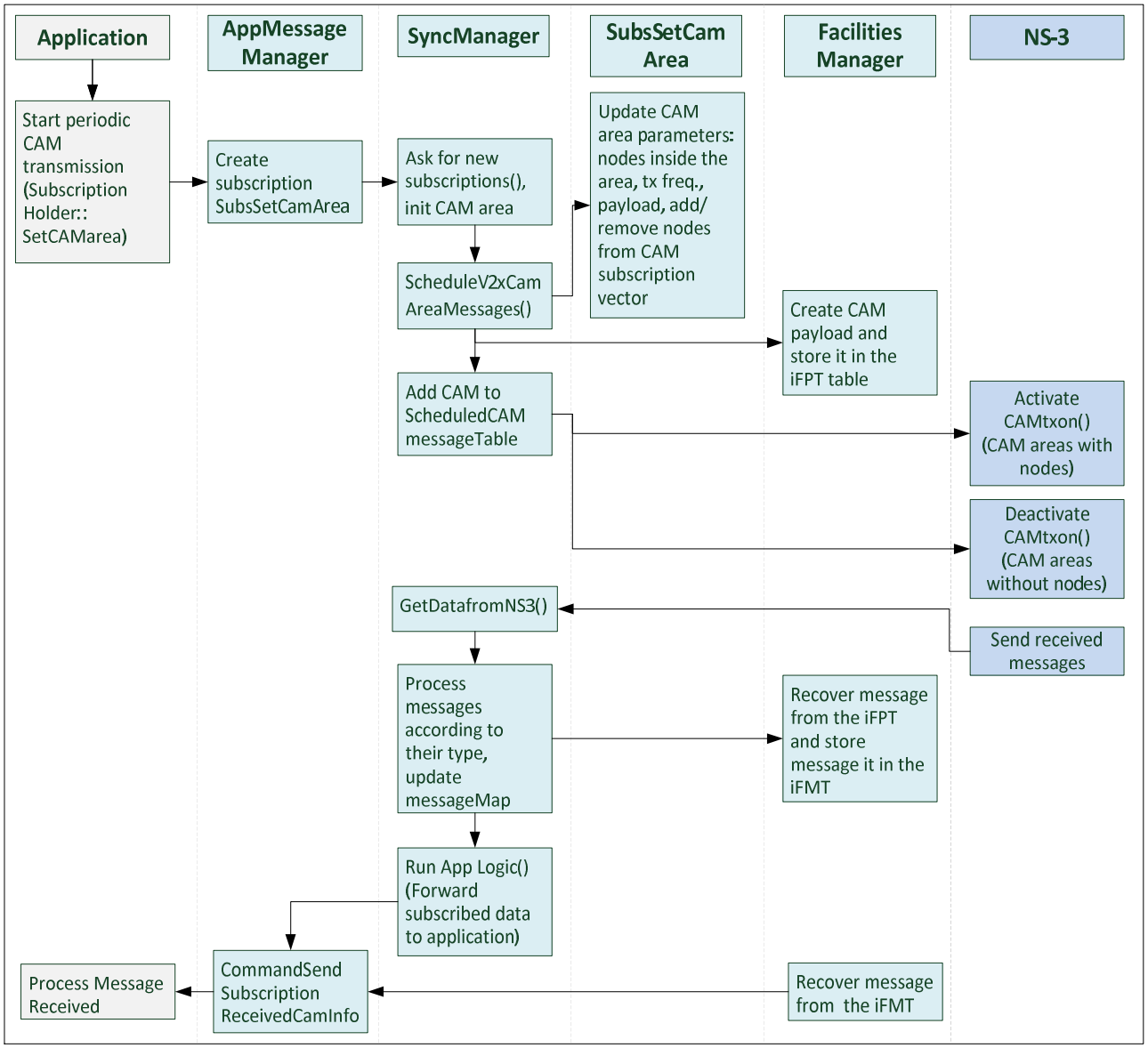

Figure 3.11: Sequence for the transmission and reception of CAM messages.

3.4 Interface requirements

In the following sections requirements for the interfaces provided by the simulation framework derived from the envisioned applications within the TransAID project are listed. For unmet items, the interface extensions to be defined within the project are specified.

Within the iTETRIS framework three types of interfaces exist, through which the different involved programs communicate (see also Figure 3.12):

- SUMO ↔ iCS
- ns3 ↔ iCS
- Application ↔ iCS

All three interfaces are built upon TCP/IP sockets which are connected in the iCS initialisation phase. Here, the iCS acts as a client to all other modules in the sense that it initialises the communication and orchestrates the execution of tasks by the other programs.

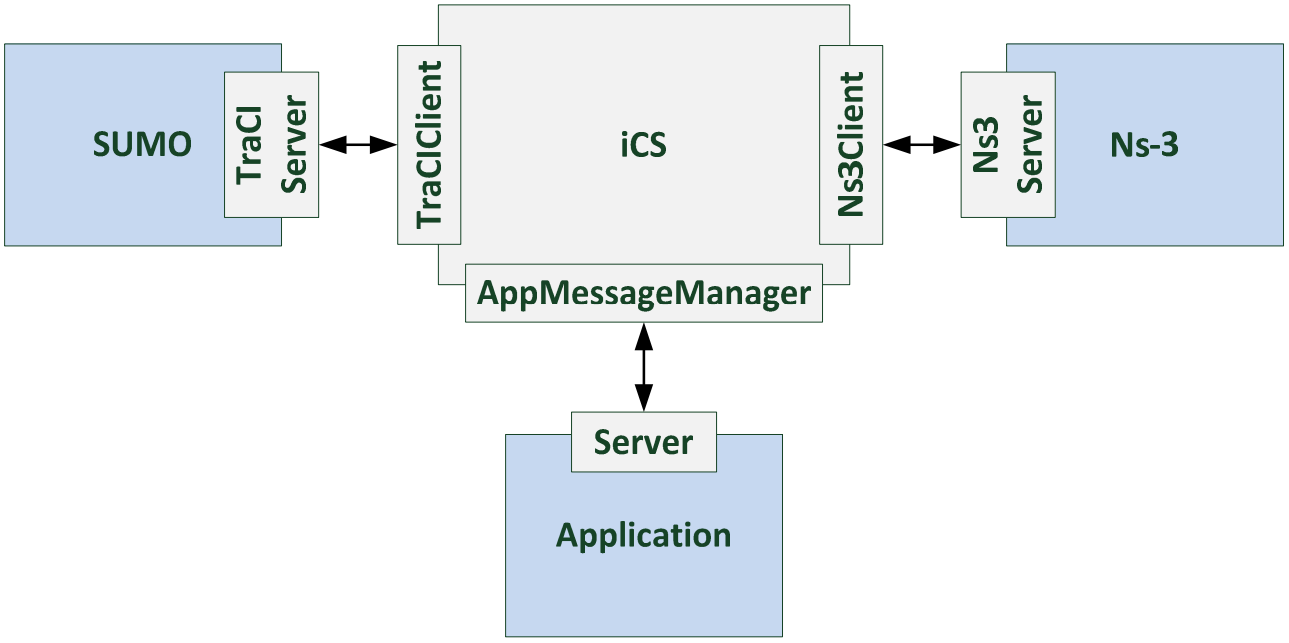

Figure 3.12: Interface types within the iTETRIS framework. Attached to the box representing the main modules, smaller boxes represent the corresponding low-level classes responsible for implementing the interfaces.

For the TransAID simulations, an application has to be able to interact with the traffic simulation SUMO and the communication simulation ns-3 in various ways. These determine the requirements on the interface between the application and iCS. All further requirements on the interfaces between the iCS and SUMO and ns-3, respectively, may be derived from the former requirements on the application interfaces in that the corresponding requests stemming from the applications have to be conveyed appropriately to SUMO and ns-3.

3.4.1 Application Interface

We classify the requirements on the application interface according to the type of interaction (modifying/observant) and the medium of interaction (wireless/wired). Modifying interactions are all requests, which possibly influence the state of the traffic system, i.e. the simulation state in SUMO. For instance, imposing speed advices on vehicles aims at actively influencing the traffic dynamics. Observant interactions merely request information about the state of the traffic system and do not impose a direct state change. We note that observant interactions in this sense may still affect the traffic state when they are transmitted via V2X communications because the corresponding messages may interfere with messages pertaining to modifying interactions. However, if only observant interactions are applied, there should be no difference in simulation results when these interactions are skipped. For such a purpose, a coupled simulation framework would not be needed, as all observations could be collected offline.

Further, we distinguish between wired and wireless interactions, where for wired interactions all simulations of the communication system are neglected and a successful transmission is presupposed for all requests. For wireless interactions a message transmission is simulated prior to simulating its effect. Only successfully transmitted messages will lead to an effective modification of the traffic system's state or to a response containing the requested information.

The general pattern for simulating this chain of scheduling a message, recognising its transmission and applying its effect is depicted in Figure 3.13, see also Section 4.3. Table 4 to Table 6 list the requirements for the application interface derived from the TransAID use cases [51].

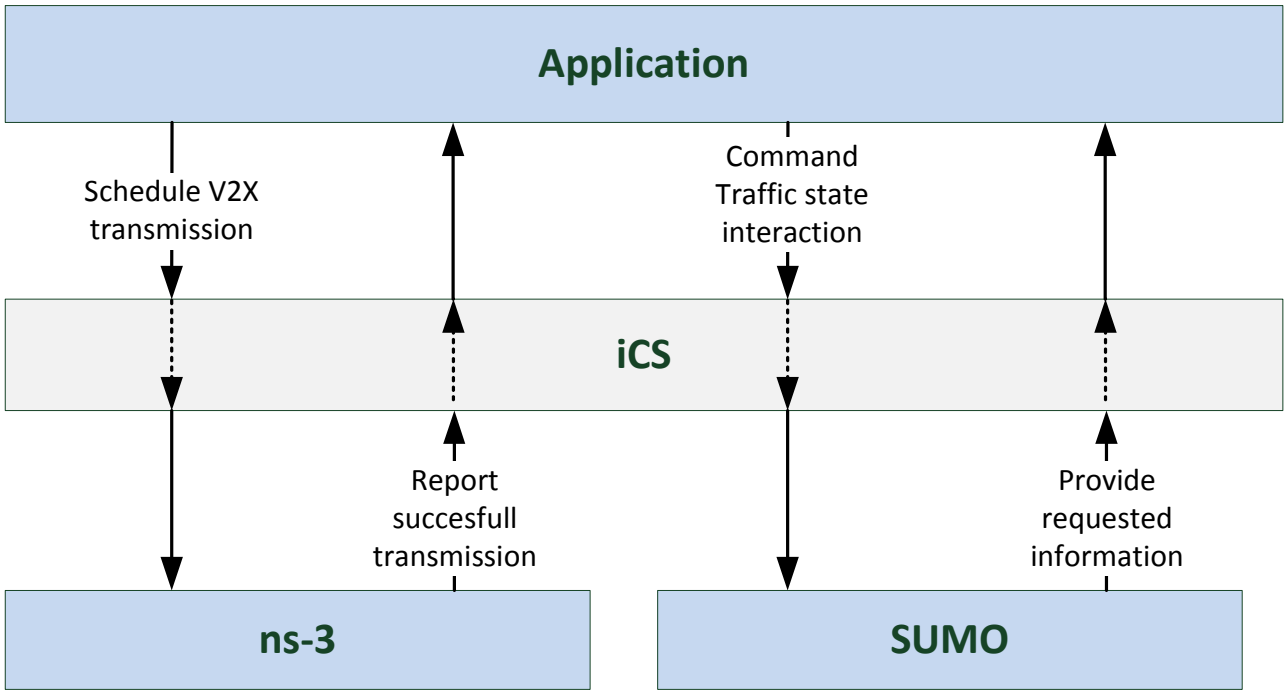

Figure 3.13: Simulation pattern for wireless interactions. The application firstly requests the simulation of the transmission of a corresponding message and at successful message retrieval it commands the appropriate effects to take place in the traffic simulation or retrieves the desired information from it.

Table 4: Required modifying interactions

Name	Description	Use cases	Wire-less	Available at project start
Maximum speed	Impose a maximal speed on a vehicle	UC3	Yes	Yes
Send safe spot location	Broadcast safe spot locations	UC4	Yes	No
Take over request	Request a take over	All	Yes	No
Take over request at specific location	Schedule ToC at specific location	UC5	Yes	No
Advise lane change	Initiate a lane change of a vehicle	UC1, UC2, UC3	Yes	No
Advise speed	`SetSpeed()` is available through generic traciCommand, `slowDown()` not interfaced from app	UC1, UC2, UC3	Yes	(Yes)
Gap creation	Decide whether this should be a SUMO function or an application component.	UC1, UC2, UC3	Yes	No
Control VMS	Directly allow switching of VMS state (might be modelled within application)	UC1	No	No

Table 5: Required observant interactions

Name	Description	Use cases	Wire-less	Available at project start
ToC state	Obtain vehicle ToC-state	UC4	Yes	No
Vehicle state	Obtain vehicle position and speed	All	Yes	Yes
Traffic information	Obtain flow, density, avg. speed	UC1, UC2, UC3, UC5	No	No
Safe spot availability	Obtain occupation state of safe spots	UC4	No	No
Detector information	Obtain current values from detectors to analyse the traffic flow composition	All	No	No

Table 6: Communication requirements

Name	Description	Available at project start
Send V2X message	Request to simulate a V2X message transmission.	(Yes)
Receive V2X message	Be informed about successful V2X message receptions.	(Yes)

3.4.2 SUMO/TraCI

Several requirements on the interaction with the traffic simulation SUMO are foreseeable for the envisioned algorithmic input and control actions taken out by TransAID traffic management procedures. A large proportion of the required functionality is already present in the well-established TraCI interface to SUMO. Only for a few manoeuvres specific to TransAID, additional interfaces have to be implemented, see Table 7. Furthermore, Table 8 lists the derived new requirements on SUMO's vehicle models as additional internal interfaces between the program core and the TraCI module (libsumo) may have to be developed.

Table 7: Requirements on TraCI

Name	Description	Use cases	Available at project start
Provide current flow, density and average speed	Provide the corresponding information to the traffic management simulation	UC1, UC2, UC3, UC4	Yes
Set vehicle class	To induce changed lane usage emulating received path info	UC1	Yes
Provide vehicle types	To access flow composition	UC3, UC5	Yes
Set maximal speed	Limit a vehicle's maximal speed	UC3	Yes
Schedule stop	Allow to command a vehicle to stop at a specific location	UC4	(Yes)
Set maximal deceleration rate	To simulate an MRM while guiding a vehicle to a specific stopping place	UC4	Yes
ToC State	Provide ToC state information for individual vehicles	All	No
Safe spot availability	Detect the occupation state of safe spots, e.g., by means of E2 detectors.	UC4	(Yes)
Schedule ToC	Issue a takeover request for specific vehicles within a specified timeframe	All	No
Create gap	Command a vehicle to create a spacing of a given length between itself and its leader	UC1, UC2, UC3, UC4	(No)

Table 8: Induced new requirements on vehicle models. For more details see [1].

Name	Description	Use cases	Available at project start
Perform ToC	Perform a transition of control (switching driver behaviour, short term reduced driving performance)	All	No
Perform simple MRM	Execute simple MRM (moderate braking until stop on current lane)	All	No
Perform advanced MRM	Execute MRM with additional parameters, stop on a given lane or at a given spot	UC2, UC3, UC4, UC5	No
Create gap	Model response to a request for establishing a gap	UC1, UC2, UC3, UC4	No

3.4.3 ns3/iNCI

The simulation of the wireless communications required by the TransAID traffic management measures is based on the interaction with the network simulator ns-3 through the iNCI interface. On the one hand, it is necessary to initiate the simulation of the V2X message transmissions in ns-3. On the other hand, it is also necessary to update the information about the vehicles in the simulation (i.e. update the position of the vehicles or create/delete vehicles in the simulation) based on the output of the traffic simulator SUMO. The majority of the interface requirements are already present in the current version of iTETRIS and only minor updates are required, see Table 9.

Table 9: Requirements on iNCI.

Name	Description	Use cases	Available at project start
Schedule V2X geobroadcast transmission	Schedule the simulation of the transmission of a geobroadcast message. The transmission of periodic messages is available for the CAM, but needs to be updated to allow other types of periodic messages (i.e. CPM, MCM, etc.)	All	Partially
Schedule V2X unicast transmission	Schedule the simulation of the transmission of a unicast message	All	Yes
Get received messages	Get the messages that have been successfully received from ns-3	All	Yes
Create node in NS-3	Add new nodes created in iTETRIS to ns-3	All	Yes
Remove node in NS-3	Remove deleted nodes in iTETRIS from ns-3	All	Yes
Update node position	Update the position of a mobile node in the ns-3 simulation	All	Yes

4 Setting up a simulation

This section first gives some basic summary for the configuration of a coupled simulation using the iTETRIS platform. These follow in some parts the iTETRIS Guidelines[7] and update them where necessary.

4.1 Installation

The installation procedure described hereafter has been tested successfully under Ubuntu 16.04 and 18.04. To install the iTETRIS simulation platform the user has to download the dependencies, the iTETRIS repository, and afterwards build and install its different components. We give a step-by-step guideline for the installation in the following. All these steps have to be executed from the systems command line interface with the working directory set to the parent directory of the desired installation directory. Root-privileges for the user performing the installation are required only to complete the first step.

Step 1 - Install the required libraries

The different modules of the iTETRIS platform depend on various other packages, which must be obtained before installing them. To download and install these dependencies, execute the following commands:

```
$ sudo apt-get update
$ sudo apt-get install git libxerces-c-dev autoconf automake
libtool libfox-1.6-dev libgl1-mesa-dev libglu1-mesa-dev libgdal-
dev libproj-dev libgeographic-dev python-pip
$ sudo pip install texttest
```

Step 2 - Get the source files from the git repository

The iTETRIS version used in TransAID will be released after the project's end in a git repository. It builds upon the iTETRIS version released by EURECOM on 5 April 2018[9]. The url of the repository containing the TransAID contributions should be inserted in place of `<git-repository-url>`. Note that downloading the repository may take a while.

```
$ git clone --recursive <git-repository-url>
$ cd transaid
$ git checkout transaid-dev
```

[7] www.ict-itetris.eu/10-10-10-community/wp-content/uploads/code/iTETRIS-Guidelines0.3.0.pdf, last accessed on September 19th, 2018.

[9] https://gitlab.eurecom.fr/iTETRIS/iTETRIS-release, last accessed on September 19th, 2018.

```
$ git submodule update --init
```

Step 3 - Build SUMO

The next step configures and builds the traffic simulation SUMO. If difficulties are encountered here, we recommend consulting the SUMO wiki pages[10].

```
$ cd sumo
$ autoreconf -i
$ ./configure
$ make -j
$ export SUMO_HOME=$PWD
$ cd ..
```

Step 4 - Build NS-3.20

The commands used to build ns-3 are as follows:

```
$ cd ns-3.20
$ ./waf configure --prefix=$PWD/..
$ ./waf -j8
$ ./waf install
$ cd ..
```

Note that the version of ns-3 used in the iTETRIS framework has been extended by a socket interface, which is not contained in the official release at the time of writing.

Step 5 - Build iCS

These commands are used to build the iCS:

```
$ cd iCS
$ autoreconf -i
$ ./configure
$ make -j
$ make install prefix=$PWD/..
```

[10] http://sumo.dlr.de/wiki/, last accessed on September 19th, 2018.

```
$ cd ..
```

Step 6 - Build applications

The following commands exemplify this for the test application developed in TransAID. See Section 4.3 on how to implement your own application for the framework.

```
$ cd iTETRIS-Applications/testApp
$ autoreconf -i
$ ./configure
$ make -j
$ make install prefix=$PWD/../..
$ cd ../..
```

After completing the previous steps the corresponding executable binary files should be located in the project's `bin/` directory. As a test whether the installation was successful the test scenario may be started manually.

```
$ cd iTETRIS-Applications/testApp/testScenario
$ iCS -c itetris-config-file-gui-ns3.20.xml
```

4.2 iTETRIS configuration files

Several configuration files have to be provided to the different components in order to execute a simulation with the iTETRIS platform (see also Figure 3.1). All configuration files described in this subsection are xml-files. In general, the configuration parameters are stored within the attributes of the xml-elements. In the following, we provide overview tables, which describe the structure of the configuration files.

4.2.1 The main configuration file

The configuration of iCS is hierarchically organised with a single main configuration file, which configures the communication between the modules and general scenario parameters. Furthermore it configures logging and references the more specific configuration files for the connected applications, the ETSI facilities [34], and for SUMO and ns-3, see also [52], [53] and the example files for the test scenarios contained in the repository (e.g., in the directory `iTETRIS-Applications/testApp/testScenario`).

The main configuration file is structured in several sections:

```
<configuration>
   <scenario> … </scenario>
```

```
    <trafficsim> … </ trafficsim>
    <communicationsim> … </communicationsim>
    <applications> … </applications>
    <logs> … </logs>
</configuration>
```

The content of these sections is described in the subsequent tables.

Table 10: Section `<scenario>` of the main iCS configuration file

Tag	Value
begin	Time (in seconds) at which the connected applications and ns-3 are started.
end	Time (in seconds) at which the simulation ends.
penetration-rate	Percentage of vehicles equipped with a communication device. This takes a random selection among all vehicles, determining which vehicles are known to the iCS.
facilities-config-file	Path to file in which the ETSI facilities are specified, see below.
message-reception-window	Time span (in seconds) for which scheduled messages are kept within the message storage of the iCS after being emitted.
interactive	May be used to switch on the stepping mode for the simulation, which waits for an ENTER key after each simulation step (mainly for debugging purposes).

Table 11: Section `<trafficsim>` of the main iCS configuration file

Tag	Value
traffic-executable	String containing the system command that starts the SUMO instance.
traffic-file	Path to the SUMO configuration file.
traffic-host	IP address of the SUMO server.
traffic-port	Port where the SUMO server is listening for commands.

Table 12: Section `<communicationsim>` of the main iCS configuration file

Tag	Value
communication-executable	String containing the system command that starts the ns-3 instance.
communication-general-params-file	Path to a text file (.txt) providing general simulation attributes. This file can be created through the ns-3 functionality ConfigStore.
communication-config-technologies-file	Path to the ns-3 file (.xml) where the communication module installers to be used in the simulation are specified. An installer allows installing a given communication module (i.e. WAVE, WiMAX, etc.) in a node in ns-3 based on iCS requests.
communication-host	IP address of the ns-3 server.
communication-port	Port where the ns-3 server is listening for commands.

Table 13: Section `<applications>` of the main iCS configuration file

Tag	Value
app-config-file	Path to the XML file containing the detailed configuration of applications that are active in the scenario.

Table 14: Section `<logs>` of the main iCS configuration file

Tag	Value
ics-log-path	Location of the log file generated by the iCS
ics-log-level	Determines the logging verbosity. Can take three values (listed by ascending verbosity): ERROR, WARNING, and INFO
ics-log-time-size	Duration or the temporal interval (in seconds) at which a new log file is started to limit the single file's size. (Log file names are suffixed by a counter.)
ics-log-begin	Time (in milliseconds) at which logging should start.
ics-log-end	Time (in milliseconds) at which logging should end.

ics-log-omit-systime	Whether a time stamp before each logging message shall be omitted.
ns3-log-path	Location of the log file generated by ns-3.

4.2.2 The facilities configuration

The iCS manages data structures resembling the facility layer of the standard communication architecture for ITS defined by the ETSI [34]. This layer is configured by the facilities configuration file, whose location is specified in the main iCS configuration file as seen previously. The facilities main configuration file consists of a single section contained in the top-level XML-element `<facilities>`.

Table 15: The facilities main configuration file

Tag	Attribute	Value
localCoordinates	latitude, longitude, altitude	Geodetic coordinate of the map origin.
mapConfig	mapConFilename	The location of the road network file (a SUMO network, i.e. .net.xml file). This file must coincide with the network file supplied to SUMO in the traffic simulation configuration file. For details, see the SUMO documentation[11].
stationsConfig	mapConFilename	The location of the ITS station description file.
LDMrulesConfig	LDMrulesConFilename	The location of the LDM rules description file.

The stations configuration file contains the penetrations rates of the radio access technologies (RATs) installed on vehicles in the scenario, and the declaration of road side units, i.e., fixed stations.

All XML elements are contained in two nested parent elements:
`<stations><default>…</default></stations>`

The RATs are declared within elements of type `<mobileSta>` in a parent element `<mobileStas>` and similarly the fixed stations are declared within elements of type `<fixedSta>` in a parent element `<fixedStas>`.

[11] http://sumo.dlr.de/wiki/Networks/SUMO_Road_Networks, last accessed on September 20th, 2018.

Table 16: The stations configuration file

Tag	Attribute	Value
RATseed	value	Random generator seed for assigning RAT technologies to mobile nodes in the simulation.
mobileSta	RAT-type	Identifier for the declared RAT technology.
	penetration-rate	Percentage of vehicles equipped randomly with the corresponding RAT in the simulation.
	communication-profile	Identifier for the corresponding ns-3 communication profile. Currently available: WaveVehicle, UmtsVehicle, WimaxVehicle, DvbVehicle, LteVehicle
fixedSta	id	Identifier for the RSU
	x, y	Coordinates of the RSU in local coordinates of the road map.
	RAT-type	Identifier for the RAT technology of the fixed station. (Each fixed station has a unique RAT.)
	enabledRAT	Whether the RAT is enabled at simulation begin.
	communication-profile	Identifier for the corresponding ns-3 communication profile. Currently available: WaveRsu, UmtsBs, WimaxBs, DvbhBs, LteBs

The LDM configuration file contains information regarding the configuration of the LDM logic implemented within the iCS facilities. On the one hand it must specify the messages' lifetime interval and on the other hand it allows defining a collection of relevance filters for the communication processes. All its elements are contained in a root element `<LDMrules>`. The various filters that allow describing the relevant areas for message retrieval are contained in an element of type `<relevantArea>`.

Table 17: The LDM rules configuration file

Tag	Attribute	Value
defaultMessageLifeInterval	value	Number of simulation steps until messages are removed from the iCS facilities message table.
relevantStationTypes	fixed, mobile	Whether received messages are relevant (“1”) or not (“0”) for each type of node.
relevantMessages	cam, denm	Whether the message types are relevant (“1”) or not (“0”)
relevantArea		Container for specification of the area of relevance, see [53]. If empty or not present, the complete area will be taken into account.

4.2.3 Configuration of connected applications

To make the iCS of the connected application modules, the user has to provide the corresponding information about the applications in a configuration file, which has to be referenced from the main configuration file in turn (see previously). This file contains a list of `<Application>` elements inside a root level `<Applications>` element. The structure of these elements is described in Table 18.

Table 18: The `<application>` element of the applications configuration file

Tag	Attribute/Text	Value
name	Text	Name of the application.
executable	Text	String containing the system command that starts the application instance.
ip	Text	IP address of the application’s server.
port	Text	Port where the application’s server is listening for commands.
seed	Text	Seed value for the random number generator that is responsible for random assignment of the application to vehicles
rate	Text	Percentage of communication equipped vehicle’s, which have the application installed.

result-container	Text	ID string for the result container type to be used by the application. Though some legacy result types are still available, it is recommended to restrict this to either "OUTPUT_VOID" or "OUTPUT_GENERIC", see Section 3.3.2.
serviceId	unicast, multicast, broadcast, geobroadcast, topobroadcast	The provided attribute/value pairs define a mapping between the specified transmission mode and the given ns-3 identifier for a corresponding transmission mode. These may be one out of serviceIdUnicast, serviceIdMulticast, serviceIdBroadcast, serviceIdGeobroadcast, serviceIdTopobroadcast
stations		This element contains a list of `<id>` elements, which may be used to explicitly require the installation of the application on the stations with the id given in the text node of the `<id>` element. It is necessary to list all fixed stations on which the application should be installed here.

4.2.4 Configuration of SUMO

A comprehensive elaboration on the possibilities to configure SUMO is out of scope for this document. We advise the reader to consult the SUMO documentation for getting started with SUMO, as well as for specific questions. Though one important point for the configuration of SUMO is worth emphasising here. The road network file given to SUMO in the `<net-file>` element of the sumo configuration file must coincide with the file specified in the `<mapConfig>` element of the facilities configuration file.

4.2.5 Configuration of ns-3

The technologies implemented in iTETRIS must be configured in ns-3 in order to allow the simulation of the communications. iTETRIS provides a master configuration file, within the "communication-config-technologies-file" tag, that defines the installers for the different types of communications technologies available in iTETRIS and defines the file where the configuration parameters for each type of technology are defined. For each type of technology, two configuration files (one for the vehicle case and one for the RSU case) must be provided defining the parameters shown in Table 18. The main difference between the configuration of the files for the vehicles and the RSU are physical parameters as the transmission power or the height of the antenna employed.

Table 19: The NS-3 technology configuration file

Tag	Description
yansWifiChannel	Defines the propagation model and the parameters of the model.
yansWifiPhy	Defines the specific parameters for the physical layer, i.e. minimum and maximum transmission power, transmission/reception gains, etc.
qosWifiMac	Defines the specific parameters for the MAC layer, i.e. the fragmentation threshold or the RTS/CTS threshold.
MobilityModel	Defines the antenna height.
application	Defines the parameters of the transmission modes (unicast, geobroadcast, etc.) employed by the iTETRIS applications. It defines the ports employed and also the frequency and packet size.

4.3 Writing your own TM-application

The TransAID project will provide iTETRIS application developers with a simpler framework to start developing their own traffic management applications. In essence, the developer will create a folder containing a small number of C++ files (main, node behaviours, and behaviour factory), which follow given template structures. Having this set up, the desired traffic management procedures can be written into the behaviour classes for the different node types. This procedure is described in more detail in the following paragraphs.

A basic traffic management application is about defining how the different elements which participate in the traffic flow react on each other and send messages about the current status and their actions. In order to do so, all elements (called nodes in the iCS context) implement behaviour classes reacting to events. These behaviour classes adhere to some basic interface (defined in the abstract base class `application::Behaviour`) which allows them to receive messages (custom and CAM messages in the methods `Receive` and `processCAMmessagesReceived`) and perform their task (including sending messages) in every simulated time step (method `Execute`).

To implement a custom behaviour the most important part is to define the message flow which is to be implemented. This also includes the decision whether the behaviour will be completely message-triggered (for instance whether the traffic light only reacts to vehicles sending a request) and can thus be implemented in the `Receive` methods alone, or if there is regular activity in the `Execute` part as well. It is also possible to model time-triggered activity without checking every time step by using scheduled (local) events, which call predefined methods of the behaviour at a fixed time. The method to use here is `Scheduler::Schedule`. The current setup allows sending arbitrary TraCI messages to the traffic simulation giving it the full flexibility of the underlying SUMO simulation.

In addition to the regular (time driven) and the event- or message-triggered activity, every behaviour might also choose to have an initialisation and a clean-up procedure (called `Start` and `Stop`) which will be executed when the application starts or finishes, respectively.

It is currently not possible to implement different behaviours for the same node type in different classes. For instance all vehicles will share the same behaviour class and the specific differences will need to be resolved in the methods of the class itself either by deriving it from the vehicle type or the ID or by implementing parameters for the underlying vehicles, which are then questioned using the communication facilities. Nevertheless, all nodes get a single instance of the behaviour object and if a behaviour needs to store data, this storage can be done locally to the node and is not shared with all the other nodes. The most common use case here is currently to store a reference to scheduled events to allow cancellation of the event if necessary, but other applications are conceivable.

The process for implementing your own application is now:

1. Define the message flow between the different nodes
2. Define the time-triggered dependencies
3. Extract external parameters for the different parts of the message flow
4. Implement behaviour classes for the different nodes which emit and respond to the corresponding messages, and schedule the timed events
5. Implement a behaviour factory class which generates the new behaviour classes on demand
6. Implement a new main method for your application which
    a. parses the parameters (preferably from a new section in the general configuration file)
    b. Instantiates the behaviour factory
    c. Passes it to the Server on creation

This will result in the server process using the factory methods to create new behaviours for the nodes which implement the desired functionality.

4.4 Test Suite

Regular testing is an integral part of sustainable software development. Software that is not continuously tested is prone to regression and deterioration due to inadvertently introduced failures. Therefore we consider it of the highest priority to establish a comprehensive collection of tests for the iCS and the functionality of its applications. Tests may also serve as examples for the usage of specific program features.

As a convenient approach for testing a complex interplay between the several components comprising the iTETRIS platform, we firstly develop a suit of integrative tests, which evaluate outputs of simple simulation scenarios each testing a simple functionality, where an application interacts with SUMO and/or ns3 by means of the iCS interface. As a testing framework realising the required functionality, *TextTest*[12] was identified as suitable.

[12] http://texttest.sourceforge.net/

4.4.1 Running tests

To start the TextTest GUI and load the iCS tests, a script `runTests.sh` located in the directory `iCS/tests/` should be executed, i.e. the command `iCS/tests/runTests.sh` launches TextTest's static GUI with the iCS tests (see also Figure 4.1). The left pane of the GUI shows all tests for the iCS. These are organised in several test suites containing different test categories. To add a new test suite or a new test, the parent suite or the application name, i.e., *iCS (icsapp)*, is selected and the corresponding item, *Add Test* or *Add Suite*, from the *Edit* menu (or from the context menu) is clicked. A test may also be created by duplicating an existent one. To achieve this, the test is selected and copy-pasted (see menu items *Copy* and *Paste* from the *Edit* or context menu). The right pane of the static GUI initially shows information about the selected test(s) or suite in the tab *Test*, but also provides other configuration and filter utilities. For more details see the TextTest documentation.

One or several tests, or an entire suite are executed by selecting the corresponding item in the left pane of the static GUI and clicking on the *Run selected tests* button in the tool bar above (or by pressing `Ctrl-r`). This opens another window, the TextTest dynamic GUI, which shows the test results for the executed tests. As for the static GUI, the left pane displays the individual tests, where successfully completed tests (green colour) are hidden by default, and only failed tests (red colour) are shown. The right pane initially shows the *Status* tab, which provides an overview of the number of successful and failed tests. If a test is selected by clicking on it in the left pane, the right view switches to the *Test* tab displaying detailed information on the selected test's results.

The testing concept employed by TextTest is to compare expected output of an entire program run with actual output (output files or `stdout` and `stderr`). This approach of monitoring the results of the complex interaction of different program components, i.e. the different modules in the iTETRIS framework, is also termed integrative testing, a term, which is used in contrast to unit testing, where small pieces of code are tested in isolation. If changes in the test results are encountered in the test results after modifying the program, these are highlighted in red by the TextTest GUI. If changes are present it has to be judged whether they are expected and acceptable in the light of the program modification, or indicate an unintended, erroneous behaviour. In the latter case, the program modifications cannot be accepted in their current state since they impair the program functionality. In case that the changing test results correspond to the desired behaviour, they are approved. This is done by selecting the tests to be approved in the left pane and select the item *Approve* in the *File* menu (or pressing `Ctrl-s`). It is also possible to accept only specific output files and not the complete test results by selecting and approving the corresponding files in the right pane under the *Test* tab. To examine the changes it is recommended to install a program that allows to view file differences (a diff-viewer, e.g., Meld[13]) and add a corresponding entry, e.g., `diff_proram:meld`, in the TextTest configuration file (usually `$HOME/.texttest/config`).

[13] http://meldmerge.org/

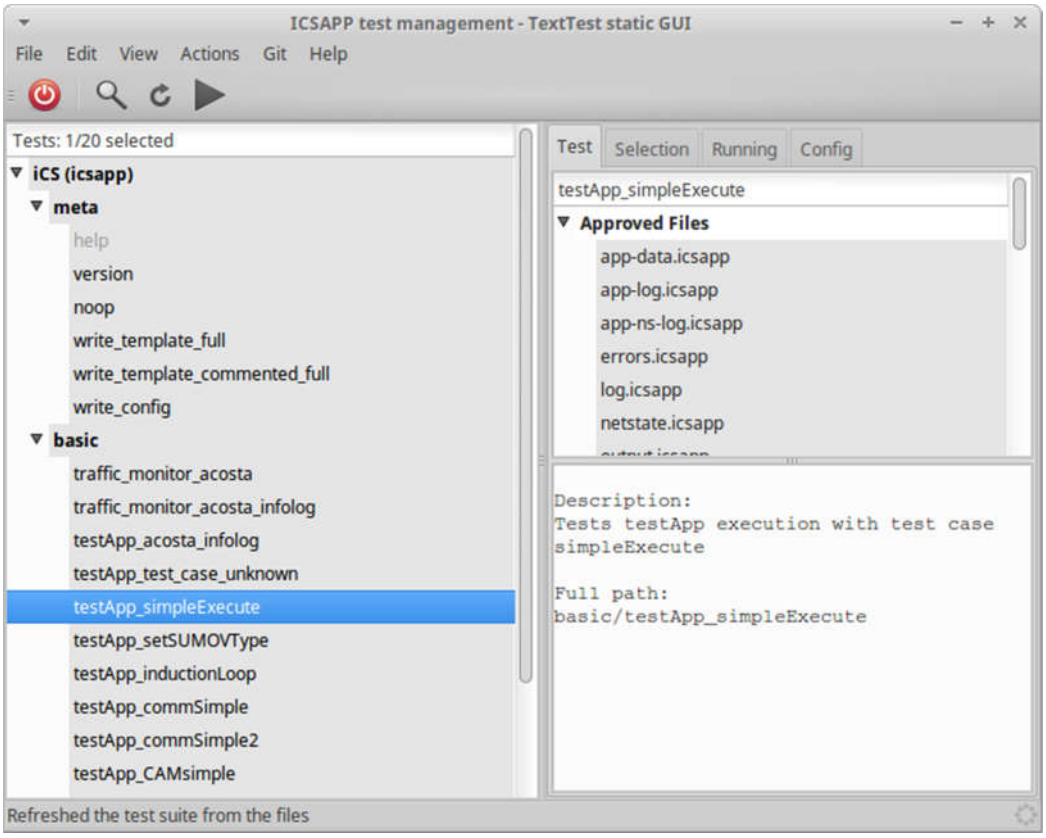

Figure 4.1: Screenshot of an initial version of the iCS test suite in the TextTest static GUI.

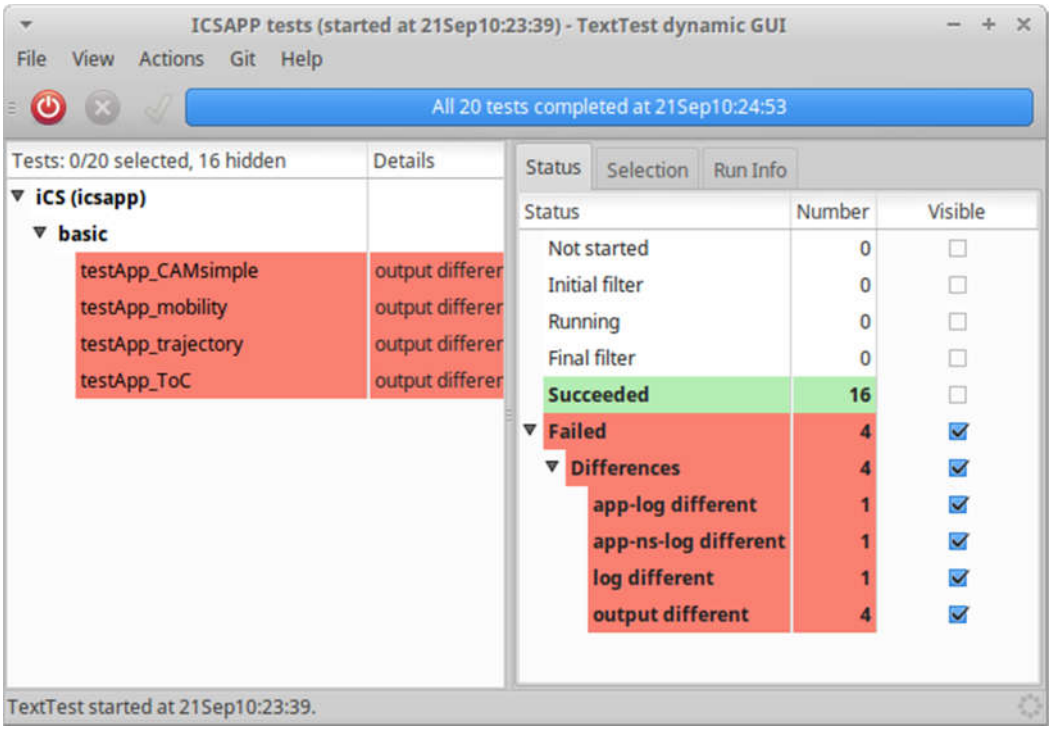

Figure 4.2: Screenshot of the test results summary in the TextTest dynamic GUI.

4.4.2 Adding tests

After a version of the iTETRIS framework, which contains the test application (in the subdirectory `iTETRIS-Applications/testApp`) has been installed on a computer (see Section 4.1), the typical workflow for adding a new test for an iCS or application interface functionality is as follows:

1. Start the iCS test suite by executing `iCS/tests/runTests.sh`.
2. Duplicate the test case `basic/testApp_simpleExecute` and rename it.
3. Edit the file `data/option.txt` in the test directory (see the right panel in the *TextTest* static GUI) and replace the test's name appropriately (no white spaces or special characters are permitted for the test name). Let us assume that the new test case is called 'myTest' for the next steps.
4. In the file `iTETRIS-Applications/testApp/src/program-configuration.h`, add a constant for the new test case to the `TestCase` enum, say `TEST_CASE_MYTEST`.
5. In the file `iTETRIS-Applications/testApp/src/app-main.cpp` add an if-case in the method `loadOptionFile()` for your test case:

```
if (key == ”test”) {
   …
   if (value == “myTest”) {
      ProgramConfiguration::SetTestCase(TEST_CASE_MYTEST);
   }
   …
}
```

6. Everywhere, where testcase-specific output or interaction with the simulation is executed, guard the code with:

```
if (ProgramConfiguration::GetTestCase() == TEST_CASE_MYTEST){
   …
}
```

Usually, this code should be located in the file in the file `iTETRIS-Applications/testApp/src/application/model/behaviour-test-node.cpp`

When the changes are implemented correctly, the *testApp* has to be rebuilt and installed as described in Section 4.1. Running the test case without any new code should at least change the application log output to recognise the new test name. The effect of the code testing additional iCS or application functionalities may require adding additional output to the application. This may, for instance, be achieved by using one of the macros `NS_LOG_DEBUG` and `NS_LOG_INFO` (see the existing code in the previous cpp-file).

5 Assessment of simulation results

The TransAID project aims at estimating the potential impacts of traffic management procedures in future scenarios, which cannot be forecasted in every detail. Therefore some assumptions have to be made. This concerns the parameters of the vehicle automations and the driver behaviour in scenarios involving transitions of control, which are not well-studied up to this day. The general strategy of the simulation is to cover a broad range of possible parametrisations and identify the preconditions for the observed impacts of the transition area scenarios and the applied traffic management, see [1]. To facilitate the simulation, assessment, and comparison of the outcomes of simulations for many different parameter combinations, a *user interface* is developed. It provides the means to specify the simulation scenarios and parameters within the range of interest, as well as the evaluations, which are to be generated automatically for the simulated cases. In the background an evaluation toolchain will take over the tasks of setting up the simulations and processing the generated data. Section 5.1 gives an account for the acquisition of the raw data from the simulation modules and its aggregation to generate the KPI distributions of relevance. Section 5.2 gives a more detailed concept for the user interface and section 5.3 presents the code developed so far for the evaluation of the baseline.

5.1 Data sources

The assessment of the simulation scenarios is based on raw data obtained from the different simulation modules. The following tables summarise the KPIs and the corresponding data sources, which serve as a basis for the scenario evaluation.

Table 20: Traffic KPIs

KPI	Units	Description	Data source
Average speed	km/h	The average speed of vehicles, calculated as the total distance travelled divided by the total travel time.	SUMO: trip information output
Mean speed for selected cross sections	km/h	Time average of the observed speed at given locations in the road network.	SUMO: virtual induction loop
Mean flow for selected cross sections	#Vehicles/h	Time average of the observed flow at given locations in the road network.	SUMO: virtual induction loop
Standard deviation of the speed at selected cross sections	km/h	Square root of the observed variance of the short interval average speed at given locations in the road network.	SUMO: virtual induction loop
Headway distribution	s	Distribution of the observed temporal headways between successively detected vehicles	SUMO: instantaneous induction loops

Number of lane changes	#LCs/km	Number of lane changes per kilometre travelled	SUMO: lane change output
Number of critical events (for different thresholds)	#Events/km	Number of observed episodes with a time-to-collision value below a given threshold $\theta \in \{1,2,3,4,5\}$s divided by the total kilometres travelled.	SUMO: trip information output and SSM device output
CO_2 emissions	g/km	Carbon dioxide emissions per kilometre travelled	SUMO: trip information output and emissions output (PHEMlight model)
Proportion of MRMs	-	Number of initiated minimum risk manoeuvres divided by the number of AVs.	SUMO: ToC device output
Duration of MRMs	s	Average duration of initiated minimum risk manoeuvres.	SUMO: ToC device output

Table 21: Communication KPIs

KPI	Units	Description	Data source
Neighbourhood Awareness Ratio	-	The proportion of vehicles in a specific range from which a message was received in a defined time interval	Ns-3 tracing system
Neighbourhood Interference Ratio	-	The ratio between the number of vehicles outside the specified range from which the given vehicle received a message, and the total number of vehicles from which the given vehicle has received a message	Ns-3 tracing system
Latency	s	The time difference between the transmission and reception time of a packet	Ns-3 tracing system
Data age	s	The time interval between the instant when the data is generated in the source vehicle and the actual time.	Ns-3 tracing system
Packet Delivery	-	The ratio of packets successfully received over the total number of	Ns-3 tracing system

Ratio		packets transmitted.	
Footprint	-	The total channel resources consumed by the radio of a single vehicle in time and space.	Ns-3 tracing system
Channel Busy Ratio	-	The percentage of time that the channel is perceived as busy for a given time interval.	Ns-3 tracing system
Messages received per vehicle	Messages / s	The number of messages of a specific type received by a vehicle in a determined time interval.	Ns-3 tracing system
Inter Package Reception Time	s	The interval of time elapsed between two successful receptions of packets of the same type.	Ns-3 tracing system

5.2 Concept of a user interface

To assess and process simulation results of single scenarios and compare results of different scenarios, TransAID develops a tool chain consisting of the following steps:

1. Configuration
2. Simulation
3. Output processing
4. Data analysis
5. Visualisation

Figure 5.1 shows an overview of the tool chain, which supports simulation experiments to be taken out within the TransAID project. The interaction of the user consists firstly in the specification of appropriate experiment configuration files, which contain information on:

- Scenario IDs (the use cases to be simulated, which induces loading the corresponding scenario for the simulation platform)
- Traffic mixes (the shares of different vehicle types)
- Demand levels (the total vehicle flows present in the scenario)
- Parameter schemes for the vehicle models[14] (PE, PS, MSE, OE, OS, see [1])
- Traffic management algorithm's parameters (see [2])
- Output and processing parameters (for selecting output and processing options, e.g., specifying working directory, selecting output, skipping steps, selection of statistical comparisons, etc.)

[14] Pessimistic Safety (PS), Pessimistic Efficiency (PE), Moderate Safety and Efficiency (MSE), Optimistic Efficiency (OE), and Optimistic Safety (OS)

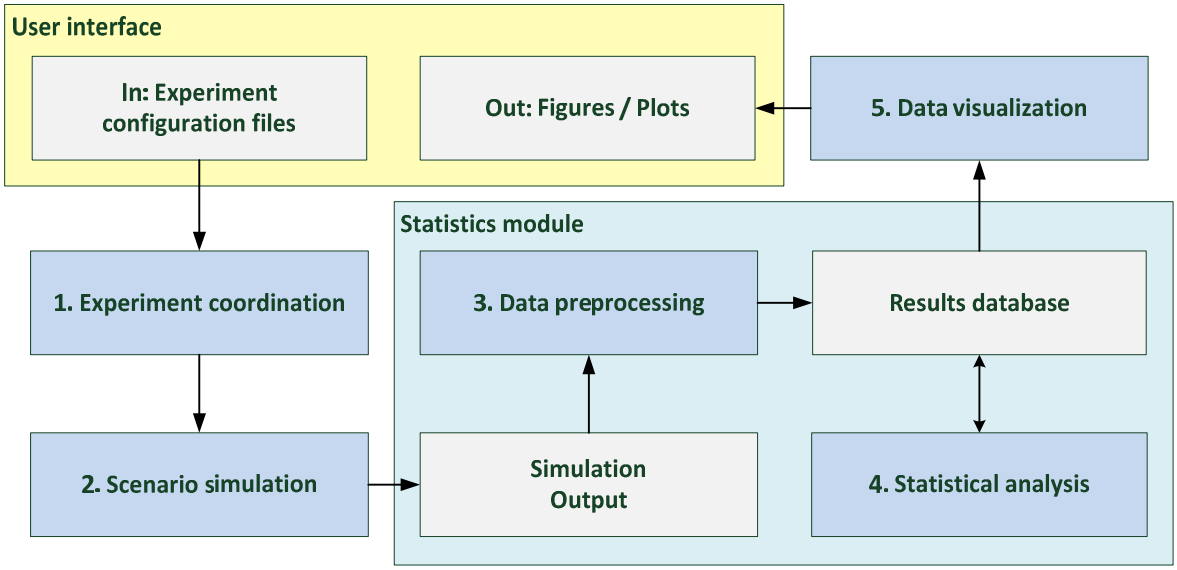

Figure 5.1: Overview of the simulation toolchain

The script which interprets the experiment configuration files also coordinates the execution of simulation runs, creates a temporary directory structure for the raw output obtained from the simulation in form of disaggregated data and sequentially triggers the simulation and processing steps.

The scenario simulation step consists of a parallel execution of various instances of the simulation platform, each writing its output into a separate temporary directory corresponding to the simulated scenario. In the next step, the data is processed for further analysis as the simulation outputs have to be aggregated over different simulation runs. As a result, distributions of KPIs are made available in a database for further processing and analysis, which is performed by a module for the statistical analysis, which assesses the individual scenario's statistics and offers sanity checks as well as the pairwise or collective comparison of different scenarios. That is, on the one hand, the statistical analysis module will extend the entries of the results database by estimates for the parameters of the corresponding distribution. On the other hand, it will provide the user with the possibility of cross-scenario comparisons of statistics, quantifying, and testing for their difference. Finally the obtained results can be used for the automatic generation of plots, which allow a visual assessment by the user.

Here, we plan to use various types of plots. A first type summarises the results for an individual KPI as a colour plot in a plane of two varying parameter dimensions, as shown in Figure 5.2(a). The colour of the corresponding coordinate encodes the value of the depicted KPI. For instance, the variable parameters could be chosen as the share of AVs (i.e. vehicle mix) and the total traffic demand (level of service), cf. [1]. Another type of visualisation is devised to compare the results of two specific scenarios in some more detail. Here the observed distributions (histograms) and the estimated probability density functions of the KPI are plotted side by side on a single axis for both scenarios, as shown in Figure 5.2(b). The confidence intervals for the expectation of the distribution for a specified confidence level α are indicated and the α-significant difference $c = c(\alpha)$ is given, that is, the largest constant c with $P(\Delta\mu \geq c) \geq 1 - \alpha$, where $\Delta\mu = \mu_2 - \mu_1$ and μ_1 is the smaller and μ_2 is the larger of the two estimated expectation values.

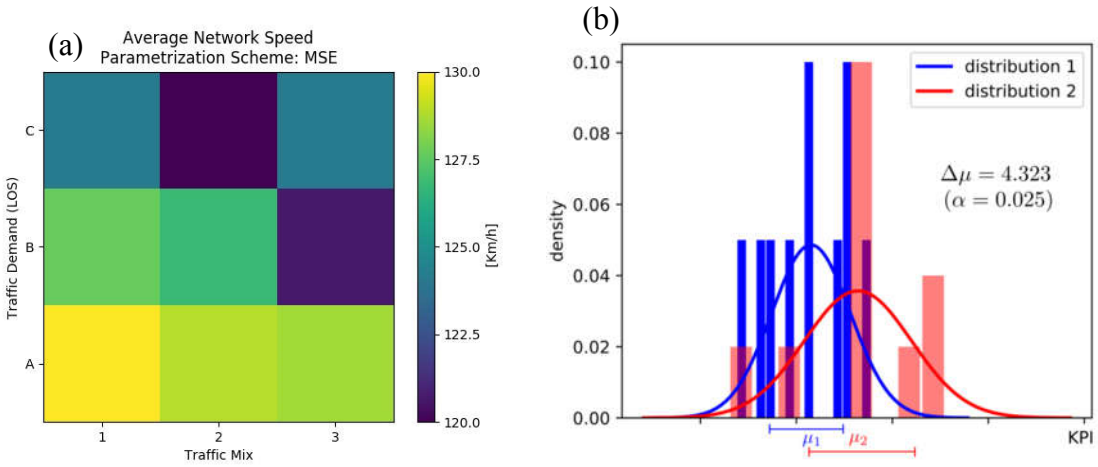

Figure 5.2: Example layouts for the plot types provided by the user interface.

Regarding the spatio-temporal dynamics of a traffic stream, we can also consider the collective movements of vehicular traffic by means of time-space (tempo-spatial) diagrams of (single-lane) road traffic. In such a diagram, the progression of a vehicle is denoted by a continuous line, which in reality represents a ribbon because of the vehicle's **length** l_i. Referencing an arbitrary point on the road, vehicles have a longitudinal position x_i, and follow upstream vehicles at a distance called the **space gap** g_{s_i}. If the vehicle's length is included, the quantity is called the **space headway** h_{s_i}. Similarly, these quantities have counterparts on the time axis, thereby defining the **time gap** and **time headway** g_{t_i} and h_{t_i}, respectively. The 'duration' of a vehicle at certain location in space is called its **occupancy** (the temporal counterpart to its spatial length), denoted by ρ_i. Vehicles' **speeds** and **accelerations** at time t are denoted by $v_i(t) = \dot{x} = \frac{dx_i(t)}{dt}$ and $a_i(t) = \ddot{x} = \frac{dv_i(t)}{dt} = \frac{d^2x_i(t)}{dt^2}$, respectively. Higher-order derivatives are also possible[15], althoughly for vehicular traffic only the concept of **jerk** (also called jolt or surge) $j_i(t) = \dddot{x} = \frac{da_i(t)}{dt} = \frac{d^3x_i(t)}{dt^3}$ is practically used. An overview of all quantities in a time-space diagram is given in Figure 5.3. The diagram contains two trajectories (with vehicles travelling at the same constant speed). As the time direction is horizontal and the space direction is vertical, the vehicles' respective speeds can be derived by taking the tangents of the trajectories. Hence, accelerating vehicles have steep inclining trajectories, whereas those of stopped vehicles are horizontal. Crossing trajectories stem from overtaking vehicles. Note that a vehicle's speed can also be derived as follows:

$$v_i(t) = \frac{h_{s_i}(t)}{h_{t_i}(t)} = \frac{g_{s_i}(t)}{g_{t_i}(t)} = \frac{l_i}{\rho_i(t)}$$

[15] These higher-order derivatives are also known jounce/snap (4th), crackle (5th), pop/dork (6th), lock (7th), drop (8th), shot (9th), and put (10th).

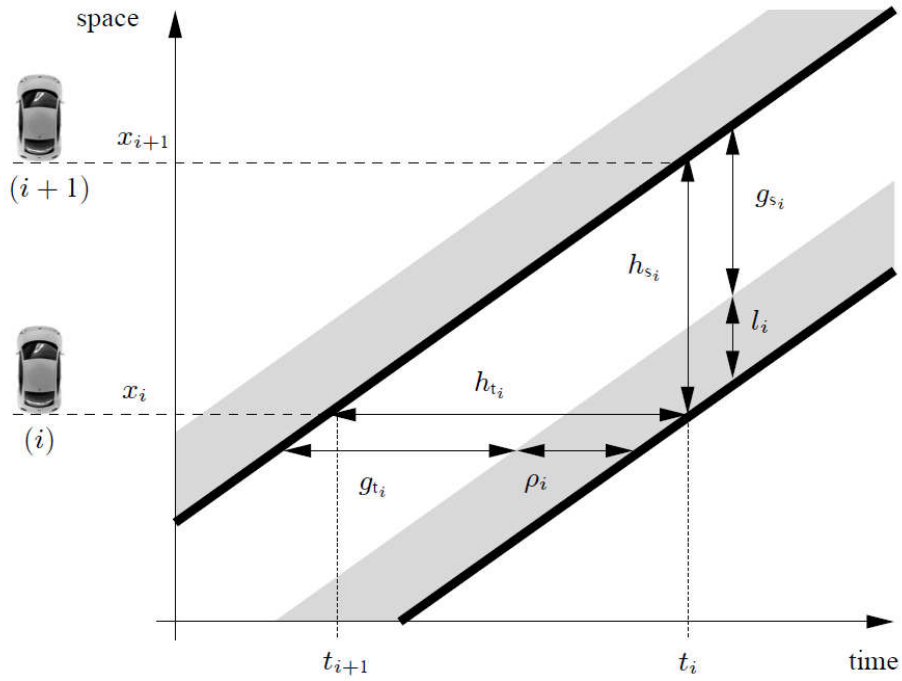

Figure 5.3: A time-space diagram showing two vehicle trajectories, as well as the space and time headways. Both headways are composed of the space gaps and the vehicle lengths, and the time gap and the occupancy time, respectively. The time headway can be seen as the difference in time instants between the passing of both vehicles.

The dynamics and interactions of multiple vehicles can be shown in such diagrams; an example of this is given in Figure 5.4, which contains many simulated vehicles travelling on circular, closed road (the vehicles are coloured individually by shades of yellow).

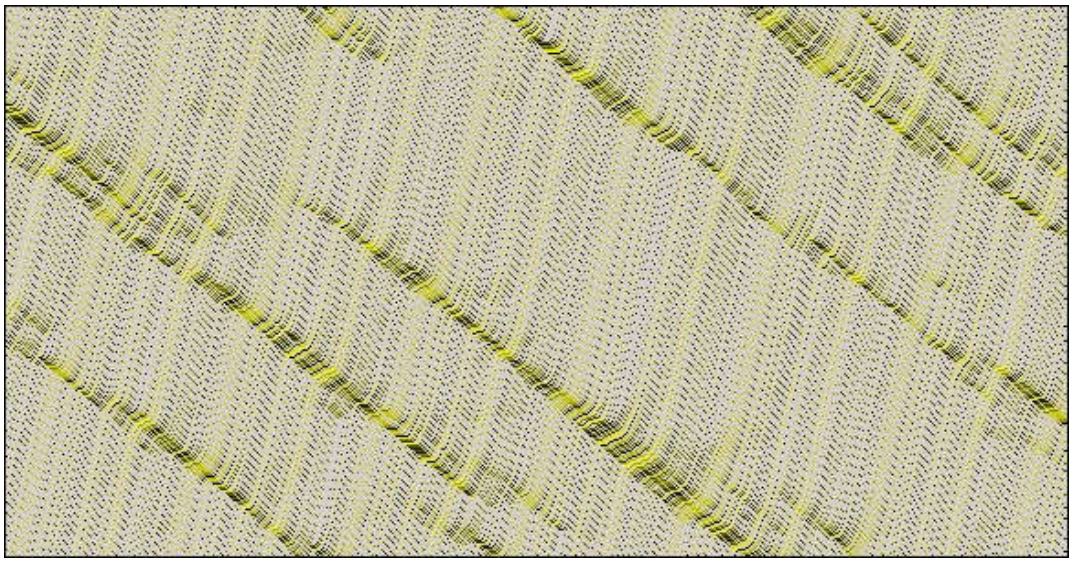

Figure 5.4: An example of a time-space diagram containing multiple vehicles driving around in a simulated circular, closed road (the vehicles are coloured individually by shades of yellow) .

Time-space diagrams can also contain macroscopic measurements, such as traffic flows, densities, vehicle speeds, … An example is shown in Figure 5.5, where average speeds on the Brussels ring road are shown for a median Monday in 2003. Note that in this case, it is useful and highly recommended to smooth the time-space diagrams by means of a **Treiber-Helbing filter**[16], which is a far better alternative than regular smoothing operators as it also takes into account the direction of information depending on the traffic regime, i.e. free-flowing (downstream information flow) or congested (upstream information flow). This filter is particularly useful if there are few detector measurement locations available, and/or if they are spaced unevenly along the road.

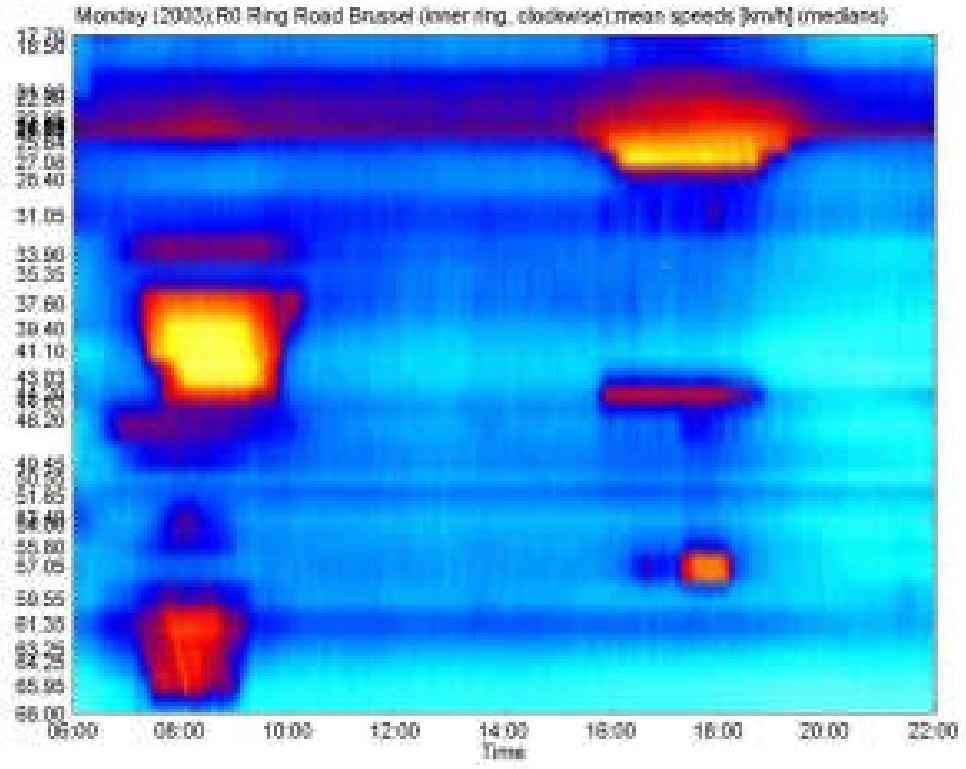

Figure 5.5: A Treiber-Helbing filtered time-space diagram representing the average speed on the Brussels R0 ring road on an median Monday in 2003.

From a more concrete statistical point of view we can also compare (distributions of) measured quantities, for example, the average speed in a section with and without (i.e. the baseline simulation) traffic management measures.

[16] Treiber, M. and D. Helbing, Reconstructing the Spatio-Temporal Traffic Dynamics from Stationary Detector Data. Cooper@tive Tr@nsport@tion Dyn@mics, 2002. 1: p. 3.1-3.24

There are various statistical tests at our disposal, e.g., to compare the means of distributions with each other, to compare their variances, to compare higher-order moments or even to compare the distributions full density functions with each other. A classic approach is to use Student's *t*-test for comparison. However, in most cases, these test require the data (amongst other requirements) to be normally distributed. This can by itself be tested using the following approaches:

- Quantile-quantile (Q-Q) plots
  This is the quantile-quantile plot, whereby the quantiles of two distributions are plotted against each other. The more similar both distributions are, the more the points in the Q-Q plot will lie on the identity function (y = x); in case they are only linearly related, the points will still lie on a straight line. The Q-Q plot is robust with respect to differences in location and scale. If the general trend of the Q–Q plot is flatter than the identity function, the distribution plotted on the horizontal axis is more dispersed (heavier tail) than the distribution plotted on the vertical axis, and vice versa.

- Normal probability plots (rankit plots)
  The normal probability plot is a graphical technique to identify substantive departures from normality. A normal probability plot plots the empirical cumulative distribution of the sample data versus the theoretical cumulative distribution function of a normal distribution. The horizontal axis plots the sorted sample data. The vertical axis plots the normal order statistic medians, calculated using the uniform order statistic medians and the inverse cumulative distribution function (CDF^{-1}) of the normal distribution. If the data are consistent with a sample from a normal distribution, the points should lie close to a straight line. As a reference, a straight line can be fit to the points. The further the points vary from this line, the greater the indication of departure from normality.

- The Shapiro–Wilk test, which assesses if a sample comes from a normally distributed population (that is the null hypothesis). If the *p*-value is less than the chosen alpha level, then the null hypothesis is rejected and there is evidence that the data tested are not from a normally distributed population. If it is greater than the chosen alpha level, then the null hypothesis that the data came from a normally distributed population cannot be rejected (e.g., for an alpha level of 0.05, a data set with a *p*-value of 0.05 rejects the null hypothesis that the data are from a normally distributed population). Note that the test does not work well in samples with many identical values.

- Another possible test statistic is the Jarque–Bera test. This test checks the goodness-of-fit of whether the distribution's skewness and kurtosis match that of the normal distribution. The test result should be compared to the values of the χ^2 distribution with 2 degrees of freedom (DoF), because the null hypothesis is a joint hypothesis of the skewness being zero and the excess kurtosis being zero. In its simplest form, the test statistic is given as:

$$\mathrm{JB} = \frac{n\,b^2}{6} + \frac{g^2}{24}$$

  with *b* the sample skewness and *g* the sample kurtosis. Note that the test is only reliable for larger sample sizes, which is fine for TransAID given the abundance of measurements that are available at each simulation run.

As opposed to the previously described tests, we can also use the Kolmogorov–Smirnov (KS) test that allows for testing the data set against an arbitrary reference distribution, other than the normal distribution. It is a non-parametric test that can be used to compare a sample with a reference probability distribution (one-sample KS test), or to test whether two underlying one-dimensional probability distributions differ (two-sample KS test). The Kolmogorov–Smirnov statistic quantifies a distance between the empirical distribution function of the sample and the cumulative distribution function of the reference distribution, or between the empirical distribution functions of two samples, as shown in Figure 5.6. The null distribution of this statistic is calculated under the null hypothesis that the sample is drawn from the reference distribution (in the one-sample case) or that the samples are drawn from the same distribution (in the two-sample case). The power of this test is that we can compare distributions in full, irrespective of them being normally-distributed or not.

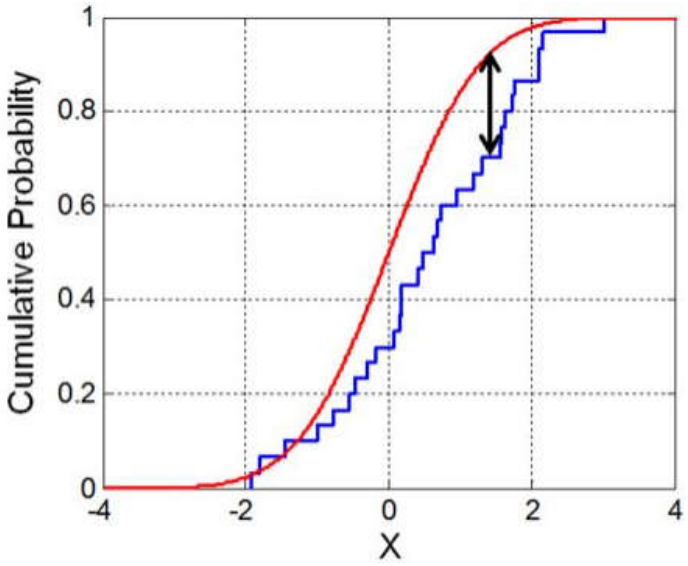

Figure 5.6: An example of comparing two distributions (red and blue curves) by assessing the differences in their cumulative distribution functions using the Kolmogorov-Smirnov (KS) test.

The goal is now to test whether there are significant differences between (1) quantities measured during a certain scenario with traffic management and those measured in the corresponding baseline, and (2) quantities measured from one scenario to another given different input/setup parameters. These are then reported as such, and will allow us to assess whether or not a certain traffic management scheme is more performant/safe/… (based on the chosen set of KPIs to compare), and if so, to what degree.

5.3 TransAID evaluation scripts

TransAID's evaluation scripts handle tasks 3, 4, and 5 of the simulation toolchain. Initially the raw simulation output (XML format) is saved in a temporary directory at the end of the simulation runs. The output of all simulation runs for a single scenario (all parameters except the fixed random seed) is then processed (e.g., filtered, cleaned, and aggregated) and included in a data cell (a Python dictionary), which is compressed and pickled[17] to the hard drive to be used both for the statistical analysis and visualisation purposes in the consecutive processing steps, as shown in Figure 5.7. The map contains all information needed to compute the statistics of the KPIs, i.e. it contains lists of KPI values per executed run for the corresponding scenario, cf. Section 5.1. The statistical analysis then traverses the stored data and computes statistics aggregated over the single runs and confidence measures (see Section 5.2), which are added to the data cell obtained from the first processing step. The visualisation module finally assesses the results and creates the selected evaluation plots.

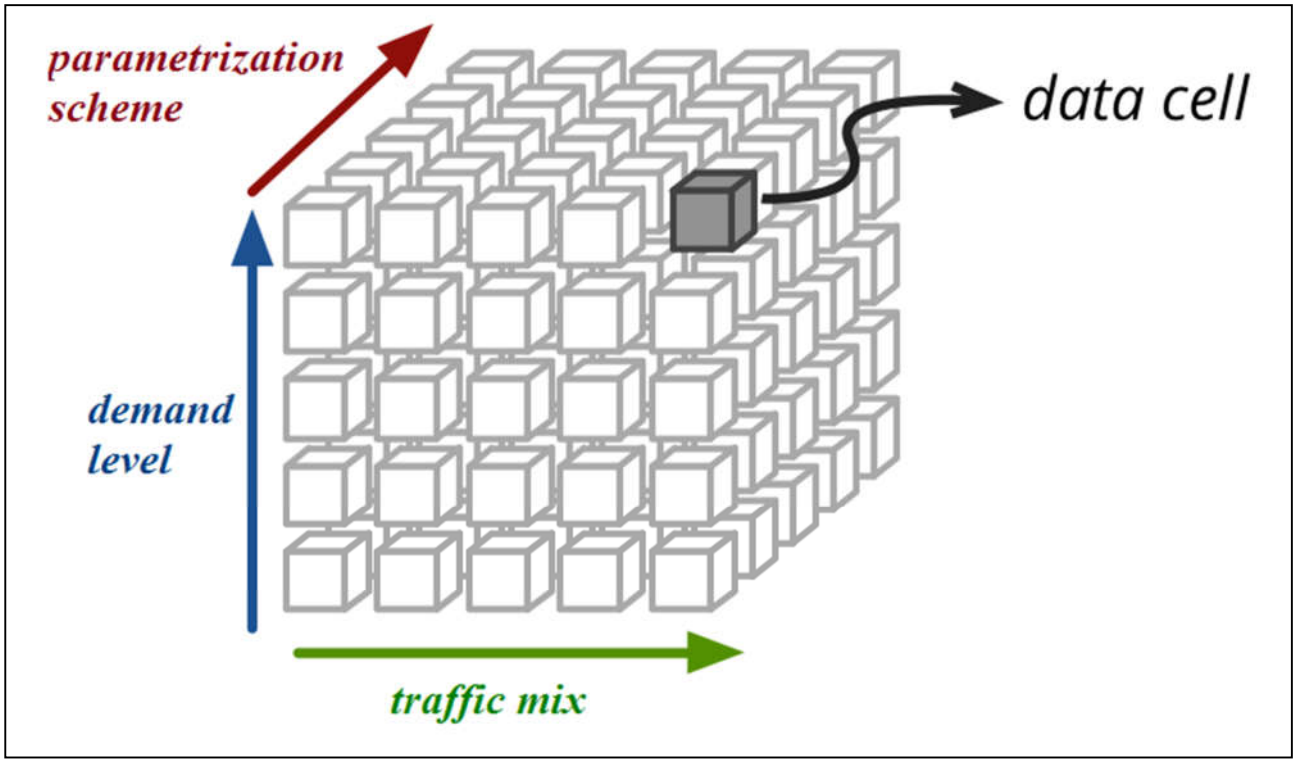

Figure 5.7: Scenario data cells in several parameter dimensions.

Raw output processing differs per KPI. For example, the average network speed KPI pertaining to a specific scenario is estimated as the mean of ten different simulation runs (10 seeds) which are provided directly by SUMO. On the contrary, retrieving the number of critical events within a specific scenario requires another processing step, as we initially have to traverse the SSM output per simulation run to collect all relevant events. Additional processing and formatting of raw simulation output might be required depending on the type and complexity of statistical tests to be conducted for the comparison between different scenarios.

[17] 'Picking' means Python object serialisation:

6 Summary and Outlook

In this report we provided a technical overview of the program flow of the iCS and discussed components, which are important for the development of iCS application modules (see Section 3). As a result we also identified some potential for architectural enhancements, which may facilitate the application developer's work. Firstly, we recommend restricting the use of subscriptions and result exchanges to the general mechanisms described in Section 3.3. Perhaps developers would benefit from a deprecation and removal of all application-specific components from the iCS, which obscure the essential iCS functionality to some degree. Secondly, TransAID will provide a clear-cut starting point to application developers in form a separate application base, which may be extended within a few simple steps, see Section 4.3. This will allow the developer to concentrate on implementing the algorithms and message interchange required for the present use case. Thirdly, as was long overdue, TransAID will establish a test suite for the iCS, whose absence was a major hindrance to a controlled development progress, see Section 4.4.

Moreover, we identified several requirements derived from the TransAID use cases, which require the incorporation of interface extensions to the iTETRIS framework, see Section 0. The most important extensions concern:

- TORs, ToCs, and MRMs
- Safe spot guidance
- Manoeuvres related to cooperative lane changes
- Periodic message transmission

In Section 5 we devised a concept for the evaluation toolchain to assess the TransAID simulations. Its structure will also provide a guidance to simulations performed in WP3 and WP4.

The previous notes already mentioned the most important upcoming steps in the WP6 development work, which are:

- Separation of a `BaseApp` (application base)
- Implementing required extensions and continuously adding corresponding tests
- Support the development of applications in WP4 by interface implementations
- Setting up simulations for the TransAID service applications
- Establishing an operational version of the toolchain
  Configuration → Simulation → Data-Processing → Evaluation

References

[1] Evangelos Mintsis *et al.*, "TransAID Deliverable 3.1 - Modelling, simulation and assessment of vehicle automations and automated vehicles' driver behaviour in mixed traffic," Aug. 2018.

[2] Sven Maerivoet, "TransAID Deliverable 4.2 - Preliminary simulation and assessment of enhanced traffic management measures," in preparation.

[3] P. Bellavista, F. Caselli, and L. Foschini, "Implementing and evaluating V2X protocols over iTETRIS: traffic estimation in the COLOMBO project," in *Proceedings of the fourth ACM international symposium on Development and analysis of intelligent vehicular networks and applications*, 2014, pp. 25–32.

[4] C. Sommer, J. Härri, F. Hrizi, B. Schünemann, and F. Dressler, "Simulation tools and techniques for vehicular communications and applications," in *Vehicular ad hoc Networks*, Springer, 2015, pp. 365–392.

[5] U. Kelin, T. Schulze, and S. Stra\s sburger, "Traffic simulation based on the high level architecture," in *Proceedings of the 30th conference on Winter simulation*, 1998, pp. 1095–1104.

[6] M. J. Booysen, S. Zeadally, and G.-J. Van Rooyen, "Survey of media access control protocols for vehicular ad hoc networks," *IET communications*, vol. 5, no. 11, pp. 1619–1631, 2011.

[7] Z. He, J. Cao, and T. Li, "Mice: A real-time traffic estimation based vehicular path planning solution using vanets," in *Connected Vehicles and Expo (ICCVE), 2012 International Conference on*, 2012, pp. 172–178.

[8] L. Codeca, R. Frank, and T. Engel, "Luxembourg sumo traffic (lust) scenario: 24 hours of mobility for vehicular networking research," in *Proceedings of the 7th IEEE Vehicular Networking Conference*, 2015, pp. 1–8.

[9] Maerivoet S., *The Physics of Road Traffic and Transportation*, in Modelling Traffic on Motorways: State-of-the-Art, Numerical Data Analysis, and Dynamic Traffic Assignment, Katholieke Universiteit Leuven, June 2006.

[10] L. Elefteriadou, *An Introduction to Traffic Flow Theory*, vol. 84. New York, NY: Springer New York, 2014.

[11] R. M. Velasco and P. Saavedra, "Macroscopic Models in Traffic Flow," *Qual. Theory Dyn. Syst.*, vol. 7, no. 1, pp. 237–252, Aug. 2008.

[12] S. P. Hoogendoorn and P. H. L. Bovy, "State-of-the-art of vehicular traffic flow modelling," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 215, no. 4, pp. 283–303, Jun. 2001.

[13] K. Nagel, P. Wagner, and R. Woesler, “Still Flowing: Approaches to Traffic Flow and Traffic Jam Modeling,” *Operations Research*, vol. 51, no. 5, pp. 681–710, Oct. 2003.

[14] M. Rondinone *et al.*, “iTETRIS: a modular simulation platform for the large scale evaluation of cooperative ITS applications,” *Simulation Modelling Practice and Theory*, vol. 34, pp. 99–125, 2013.

[15] B. Schünemann, “V2X simulation runtime infrastructure VSimRTI: An assessment tool to design smart traffic management systems,” *Computer Networks*, vol. 55, no. 14, pp. 3189–3198, 2011.

[16] D. Krajzewicz, A. Leich, R. Blokpoel, M. Milano, and T. Stützle, “COLOMBO: Exploiting Vehicular Communications at Low Equipment Rates for Traffic Management Purposes,” in *Advanced Microsystems for Automotive Applications 2015*, T. Schulze, B. Müller, and G. Meyer, Eds. Cham: Springer International Publishing, 2016, pp. 117–130.

[17] D. Krajzewicz, “Traffic Simulation with SUMO – Simulation of Urban Mobility,” in *Fundamentals of Traffic Simulation*, Springer, New York, NY, 2010, pp. 269–293.

[18] P. G. Gipps, “A behavioural car-following model for computer simulation,” *Transportation Research Part B: Methodological*, vol. 15, no. 2, pp. 105–111, Apr. 1981.

[19] P. G. Gipps, “A model for the structure of lane-changing decisions,” *Transportation Research Part B: Methodological*, vol. 20, no. 5, pp. 403–414, Oct. 1986.

[20] H.-T. Fritzsche, “A model for traffic simulation,” *Traffic Engineering + Control*, vol. 35, no. 5, pp. 317–321, 1994.

[21] S. Krauß, “Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics,” Doctoral Thesis, DLR-Forschungsbericht, 1998.

[22] A. Halati, H. Lieu, and S. Walker, “CORSIM - Corridor Traffic Simulation Model,” presented at the 76th Transportation Research Board Meeting, Washington, DC, 1997.

[23] R. Wiedemann, “Simulation des Straßenverkehrsflusses,” Tech. Rep. Institut für Verkehrswesen, Universität Karlsruhe, Heft 8 der Schriftenreihe des IfV (in German), 1974.

[24] E. Weingartner, H. Vom Lehn, and K. Wehrle, “A performance comparison of recent network simulators,” in *Communications, 2009. ICC’09. IEEE International Conference on*, 2009, pp. 1–5.

[25] A. Varga and R. Hornig, “An overview of the OMNeT++ simulation environment,” in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2008, p. 60.

[26] G. F. Riley and T. R. Henderson, “The ns-3 network simulator,” in *Modeling and tools for network simulation*, Springer, 2010, pp. 15–34.

[27] M. H. Kabir, S. Islam, M. J. Hossain, and S. Hossain, "Detail comparison of network simulators," *International Journal of Scientific & Engineering Research*, vol. 5, no. 10, pp. 203–218, 2014.

[28] A. R. Khan, S. M. Bilal, and M. Othman, "A performance comparison of open source network simulators for wireless networks," in *Control System, Computing and Engineering (ICCSCE), 2012 IEEE International Conference on*, 2012, pp. 34–38.

[29] D. Krajzewicz *et al.*, "COLOMBO: investigating the potential of V2X for traffic management purposes assuming low penetration rates," *ITS Europe*, 2013.

[30] J. Härri, P. Bellavista, L. Foschini, and R. Blokpoel, "Extending the iTETRIS platform for Smartphone Sensing and Communication Simulation," in *Proc. of the 5th European Conference on Transport Research Arena*, 2014.

[31] R. Doolan and G.-M. Muntean, "Time-ants: an innovative temporal and spatial ant-based vehicular routing mechanism," 2014.

[32] F. Caselli, A. Bonfietti, and M. Milano, "Swarm-based controller for traffic lights management," in *Congress of the Italian Association for Artificial Intelligence*, 2015, pp. 17–30.

[33] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent Development and Applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, Dec. 2012.

[34] "ETSI TS 102 894-1 - Intelligent Transport Systems (ITS); Users and applications requirements; Part 1: Facility layer structure, functional requirements and specifications," v1.1.1, Aug. 2013.

[35] J. Maneros, M. Rondinone, A. Gonzalez, R. Bauza, and D. Krajzewicz, "iTETRIS Platform Architecture for the Integration of Cooperative Traffic and Wireless Simulations," in *Proceedings of the 9th international conference on ITS telecommunications*, 2009.

[36] P. Bellavista *et al.*, "COLOMBO Deliverable 5.1 - Prototype of overall System Architecture and Definition of Interfaces," COLOMBO Consortium, 2014.

[37] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, Jan. 2011.

[38] A. Köpke *et al.*, "Simulating wireless and mobile networks in OMNeT++ the MiXiM vision," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2008, p. 71.

[39] K. Wessel, M. Swigulski, A. Köpke, and D. Willkomm, "Mixim: the physical layer an architecture overview," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, 2009, p. 78.

[40] D. S. Buse, M. Schettler, N. Kothe, P. Reinold, C. Sommer, and F. Dressler, "Bridging worlds: Integrating hardware-in-the-loop testing with large-scale VANET simulation," in *Wireless On-demand Network Systems and Services (WONS), 2018 14th Annual Conference on*, 2018, pp. 33–36.

[41] R. Riebl, H. Günther, C. Facchi, and L. Wolf, "Artery: Extending Veins for VANET applications," in *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, 2015, pp. 450–456.

[42] A. Al-Momani, F. Kargl, C. Waldschmidt, S. Moser, and F. Slomka, "Wireless channel-based message authentication," in *2015 IEEE Vehicular Networking Conference (VNC)*, 2015, pp. 271–274.

[43] W. Arellano, I. Mahgoub, and M. Ilyas, "Veins extensions to implement a message based algorithm for Dynamic Traffic Assignment in VANETs simulations," in *High-capacity Optical Networks and Emerging/Enabling Technologies (HONET), 2014 11th Annual*, 2014, pp. 29–35.

[44] M. Milojevic and V. Rakocevic, "Distributed road traffic congestion quantification using cooperative VANETs," in *13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET)*, 2014, pp. 203–210.

[45] M. Segata, S. Joerer, B. Bloessl, C. Sommer, F. Dressler, and R. L. Cigno, "Plexe: A platooning extension for Veins," in *Vehicular Networking Conference (VNC), 2014 IEEE*, 2014, pp. 53–60.

[46] J. Breu, A. Brakemeier, and M. Menth, "A quantitative study of Cooperative Awareness Messages in production VANETs," *EURASIP Journal on Wireless Communications and Networking*, vol. 2014, no. 1, p. 98, Jun. 2014.

[47] C. Büttner and S. A. Huss, "A novel anonymous authenticated key agreement protocol for vehicular ad hoc networks," in *Information Systems Security and Privacy (ICISSP), 2015 International Conference on*, 2015, pp. 259–269.

[48] B. Ribeiro *et al.*, "Simulation and Testing of a Platooning Management Protocol Implementation," in *Wired/Wireless Internet Communications*, Cham, 2017, pp. 174–185.

[49] C. Englund, L. Chen, and A. Voronov, "Cooperative speed harmonization for efficient road utilization," in *Communication Technologies for Vehicles (Nets4Cars-Fall), 2014 7th International Workshop on*, 2014, pp. 19–23.

[50] E. Egea López and others, "Simulation scalability issues in wireless sensor networks.," 2006.

[51] A. Wijbenga *et al.*, "TransAID Deliverable 2.2 - Scenario definitions and modelling requirements," Feb. 2018.

[52] "iTETRIS Building, Installation and Configuration Guidelines," iTETRIS consortium, 2010.

[53] Robbin Blokpoel, Federico Caselli, Jérôme Härri, Wolfgang Niebel, and Andreas Leich, "COLOMBO Deliverable 6.4 - Educational Kit," COLOMBO Consortium, Nov. 2015.