



D4.3

Suitability and Effectiveness Study of Traffic Management Strategies

Project Acronym	TransAID	
Project Title	Transition Areas for Infrastructure-Assisted Driving	
Project Number	Horizon 2020 ART-05-2016 – GA Nr 723390	
Work Package	WP4 Traffic Management Procedures for Transition Areas	
Lead Beneficiary	Transport & Mobility Leuven (TML)	
Editor / Main Author	Sven Maerivoet	TML
Reviewers	Alejandro Correa	UMH
Dissemination Level	PU	
Contractual Delivery Date	M18 (Feb19)	
Actual Delivery Date	M18 (Feb19)	
Version	v1.0	



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
v0.1	15/01/2019	Initial draft version with proposed structure
v0.2	05/02/2019	Alignment with WP6
v0.3	12/02/2019	Added application development in the iCS
v0.4	25/02/2019	Added how-to-testing of the ported applications
v0.5	26/02/2019	Added information on the iTETRIS modular system
v0.6	28/02/2019	Small corrections Added conclusions
v1.0	28/02/2019	Final peer-reviewed version

Editor / Main author

Sven Maerivoet (TML)

List of contributors

Kristof Carlier (TML)

Péter I. Pápics (TML)

Yun-Pang Flötteröd (DLR)

Leonhard Lücken (DLR)

Robert Alms (DLR)

Evangelos Mintsis (CRT)

Dimitrios Koutras (CRT)

Anton Wijbenga (MAP)

Jaap Vreeswijk (MAP)

Alejandro Correa (UMH)

Xiaoyun Zhang (DYN)

Robbin Blokpoel (DYN)

List of reviewers

Alejandro Correa (UMH)

Julian Schindler (DLR)

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

Document revision history	2
Table of contents	4
1 Introduction	5
1.1 About TransAID	5
1.2 Purpose of this document	6
1.3 Structure of this document	6
1.4 Glossary	7
2 Setting up traffic management implementations	8
2.1 Develop an application	8
2.1.1 Building the project	8
2.1.2 Setup/Preparation	9
2.1.3 Adapt applications to iTETRIS	9
2.1.4 The behaviour classes	11
2.1.5 Entry points for actions	12
2.1.6 Actions	13
2.2 Creating test suites	14
3 Conclusions	17

1 Introduction

In the following sections, we first give a concise overview of the TransAID project, then highlight the purpose of this document, and finally present its structure.

1.1 About TransAID

As the introduction of automated vehicles 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, where automated vehicles of all SAE (Society of Automotive Engineers) levels¹, connected vehicles (able to communicate via V2X, vehicle-to-everything) 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, highly complex situations, etc. Moving between those areas, there will be areas where 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 find optimal 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 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 transition areas (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 https://www.sae.org/misc/pdfs/automated_driving.pdf

1.2 Purpose of this document

This deliverable D4.3 relates to task T4.3 of WP4, and explains how simulations of both the baseline (WP3) and the traffic management schemes (WP4) are to be ported from the SUMO simulation environment (with the help of the TraCI interface and Python scripts) to the iCS environment (using the C++ language).

1.3 Structure of this document

The document starts with an explanation on how to set up the creation of a traffic management application in the context of the iCS. This entails the preparation to develop an application (which is what the traffic management schemes are called within the iCS), i.e. which files need to be created where within the software repository. Once this has been done, the document explains how testing schemes are to be defined for WP6. This is needed to check if the implemented traffic management logic complies with the previously obtained results from WP4. Note that the required communication protocols are, at this point, already elaborated in WP5.

1.4 Glossary

CAM	Cooperative awareness message
CPM	Cooperative perception message
iCS	iTETRIS control system
RSU	Road-side unit
SAE	Society of Automotive Engineers
SUMO	Simulation of Urban Mobility
TA	Transition area
TMC	Traffic management centre
TraCI	Traffic Control Interface
TransAID	Transition Areas for Infrastructure-Assisted Driving
V2X	Vehicle-to-everything
WP	Work package

2 Setting up traffic management implementations

The previous simulations of both the baseline (WP3) and the traffic management schemes (WP4) were directly implemented for the SUMO simulation environment with the help of the TraCI interface and Python scripts to set up multiple runs (to smooth out random variations and obtain stable results).

After these preliminary simulations and evaluations of traffic management strategies, the traffic management logic now has to be translated to the C++ language to be used in the iTETRIS platform. The work provides a direct input – and runs partly in parallel – to WP6, where the full integrated platform for the simulation and the assessment of traffic management procedures is used. In that sense, the work in Task 4.3 of WP4 is dedicated to generating code that can be used for simulations in WP6.

In the following sections, we first look at how to set up the creation of a traffic management application in the context of iTETRIS, and then describe how tests are to be defined for WP6 to check the performance of the implemented traffic management logic compared to previously obtained results from WP4.

Note that the structure of iTETRIS was already explained in Deliverable D6.1². Here, we provide extra information for porting the existing SUMO applications to the iCS that was not present in the aforementioned D6.1.

2.1 Develop an application

The TransAID version of the iTETRIS platform defined in WP6 includes a basic application known as baseApp that manages the exchange of information between the applications and the iCS modules. The application developed for the different services of the TransAID project will inherit from this baseApp and extend the functionality with the traffic management procedures defined in WP4. In order to develop these applications, a new branch (transaid-apps) is added to the git repository. Note that all TransAID applications developed share the same baseApp. Hence, commits to the baseApp should be strictly separated from the commits to the TransAID applications in development. Changes to the baseApp as well as other iTETRIS modules like iCS or ns-3 should be integrated into the transaid-dev branch.

2.1.1 Building the project

This goes similar to the instructions from iTETRIS_quick_installation_ubuntu.txt (branch transaid-dev), which have been explained in Deliverable D6.1.

Building with clang for extended error messages:

To configure an clang++ build for the iCS, which reports a lot of additional warnings, we can use the following command:

```
./configure CXX=clang++ CXXFLAGS="-stdlib=libstdc++ -fsanitize=undefined,address,integer,unsigned-integer-overflow -fno-omit-frame-pointer -fsanitize-blacklist=$SUMO_HOME/build/clang_sanitize_blacklist.txt" --prefix="$PWD/../../.."
```

² An integrated platform for the simulation and the assessment of traffic management procedures in Transition Areas (2018). TransAID Deliverable D6.1

To suppress the repeating warnings from the mersenne-twister, add `-Wno-deprecated-register`:

```
./configure CXX=clang++ CXXFLAGS="-stdlib=libstdc++ -  
fsanitize=undefined,address,integer,unsigned-integer-overflow -fno-  
omit-frame-pointer -fsanitize-  
blacklist=$SUMO_HOME/build/clang_sanitize_blacklist.txt -Wno-  
deprecated-register" --prefix="$PWD/../../.."
```

2.1.2 Setup/Preparation

To start with an application for a specific use case, we can use commits [b658a946³](https://gitlab.imet.gr/hit/transaid/commit/b658a946c40db28d960116afae4cdf1f49862a55) and [ba933001⁴](https://gitlab.imet.gr/hit/transaid/commit/ba933001f39e3275c89c267fe44a14762e015254) (and also [913d7b58⁵](https://gitlab.imet.gr/hit/transaid/commit/913d7b58ba05351693079d0e2ead196e48e83415)) as a template and rename the occurrences of `uc1` with the appropriate use case (e.g., `uc5`). A helpful snippet for this to be executed in the `transaidUCAApp` directory:

```
find . -type f -exec sed -i -- 's/UC1/UC5/g' {} +  
find . -type f -exec sed -i -- 's/uc1/uc5/g' {} +
```

(this replaces all occurrences of `uc1` in the sources with `uc5`.)

2.1.3 Adapt applications to iTETRIS

As previously mentioned, the earlier simulations of WP3 and WP4 were developed employing the SUMO simulation environment and Python scripts that interact with SUMO employing the TraCI interface. These scripts need to be modified in order to implement the traffic management measures of the different services within the iTETRIS platform. The latter coordinates the applications with the traffic simulator SUMO and the communications simulator ns-3 employing the iCS module (see Figure 1). To do so, the iCS module of the iTETRIS platform coordinates the execution of the different simulators and the applications. Following a cyclic sequence, the iCS commands to the connected modules (i.e. SUMO, ns-3, and the applications) to execute the simulation for a given time period known as a simulation step. Once each connected module is executed, the next iteration of the simulation is prepared in the iCS by updating the connected modules with the information of the other connected modules (i.e. if a new node is created in SUMO, this information is sent to the applications and ns-3 to create also the node in the other modules). This sequential execution implies that the information that an application may request to the iCS in one simulation step, will be sent to the application during one of the next time steps. Thus, the applications must know in advance the information needed at each simulation step in order to request this information from the iCS.

³ <https://gitlab.imet.gr/hit/transaid/commit/b658a946c40db28d960116afae4cdf1f49862a55>

⁴ <https://gitlab.imet.gr/hit/transaid/commit/ba933001f39e3275c89c267fe44a14762e015254>

⁵ <https://gitlab.imet.gr/hit/transaid/commit/913d7b58ba05351693079d0e2ead196e48e83415>

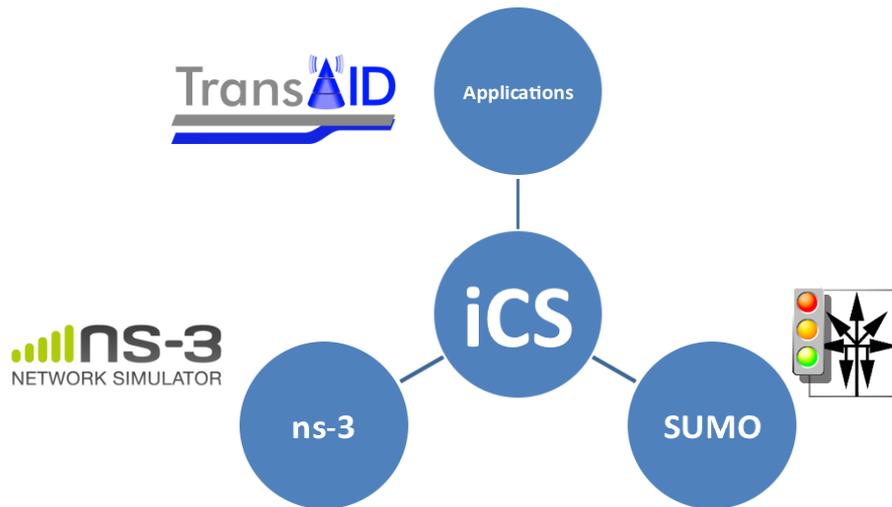


Figure 1: Schematic structure of the iTETRIS platform. The iCS manages the coupling of traffic simulation (SUMO), communications simulation (ns-3), and traffic management applications (diagram reproduced from Deliverable D6.1).

The iTETRIS platform individually simulates the different nodes involved in the simulations, thus instead of having a single application that runs with all the information of all the nodes of the simulation environment, we have different applications for vehicles and for RSUs running. If these applications want to share information, then a message must be sent between the nodes in which these applications are installed. Figure 2 shows the interactions between the TMC and vehicles simulated in the applications layer with the iCS module and the SUMO and ns-3 modules. If the TMC wants to send a message to a vehicle (i.e. a lane change advice) it needs to request from the iCS to schedule the transmission of a message in ns-3. Then, the message will be simulated in ns-3 and if the message is received by the vehicle, the iCS will pass the message to the application of the vehicle. Similarly, if the application of a vehicle wants to modify the behaviour of the vehicle in SUMO, it will need to request to the iCS that sends the command to SUMO. Afterwards, SUMO will modify the behaviour of the vehicle and then it will inform to the TMC application that the behaviour of the vehicle has been modified.

In order to adapt the scripts developed in WP3 and WP4 to the iTETRIS platform, the code will need to be modified to adapt to the event-driven interactions described in iTETRIS. The applications must employ two different types of functions: (i) functions that are executed at every simulation step, and (ii) functions that are executed when an event is triggered. The functions that are executed at each simulation step are the main core of the applications and that is where the traffic management measures of the services are implemented. On the other hand, the applications will also include functions that respond to the occurrence of events triggered by the iCS. These functions will complement the definition of the traffic management measures by for example providing information. Examples of triggering events can be the reception of a CAM message or the detection of a vehicle by a road sensor. In both cases, the information received will be processed and based on this information new action of the application may be triggered. The applications need to use a scheduler function that will schedule the execution of the different functionalities of the application depending on the current simulation step. For example, the application can schedule the periodic request of information to the iCS or schedule the transmission of messages such as the CAM or the CPM.

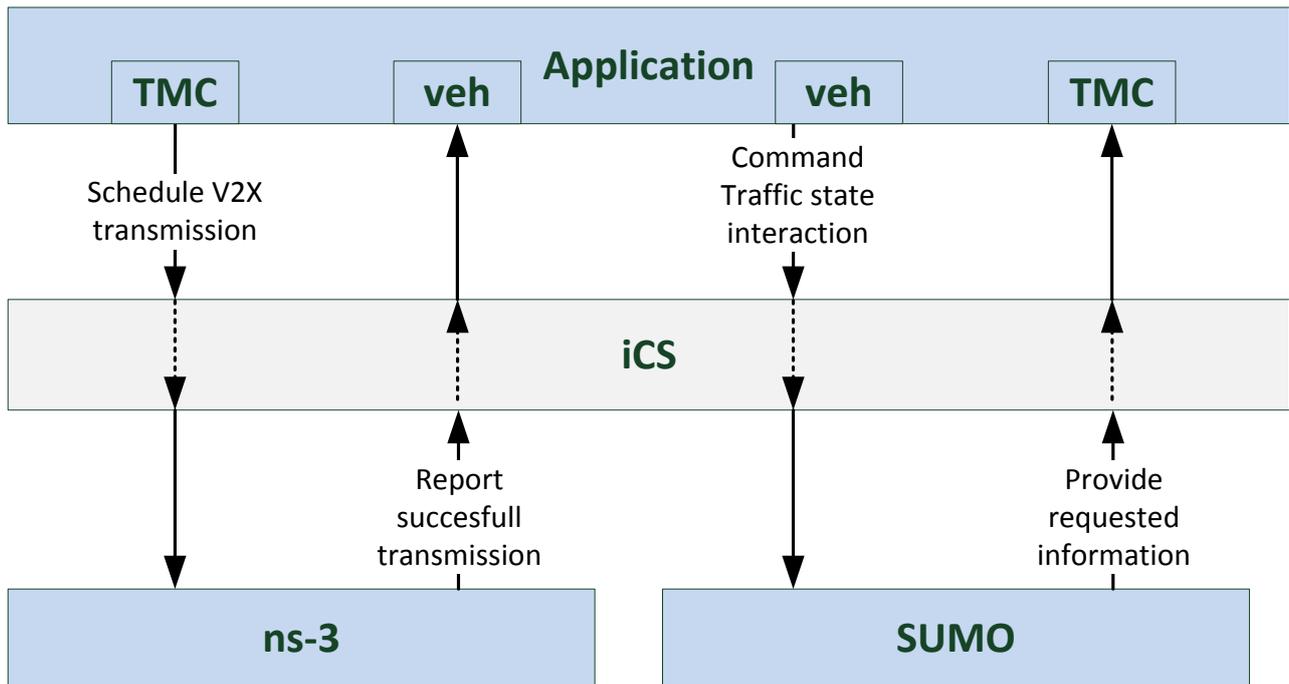


Figure 2: Interactions between the different iTETRIS modules.

In the following sections we describe the main classes and functions employed by the applications and the interfaces defined for the interactions between the applications and the iCS.

2.1.4 The behaviour classes

As of now, three behaviours are used in the applications:

- **RSU Behaviour:** this class is executed in the RSU nodes and includes the functions that model the behaviour of the RSU. This class will obtain information about the traffic stream, define the appropriate traffic management measures according to the TransAID service executed, and send the adequate advices to vehicles in order to execute the defined traffic management measures.
- **Vehicle Behaviour:** this class is executed in the vehicle nodes and includes the functionality of the applications that needs to be executed in the vehicles. This includes basic functionality as the transmission of CAMs and CPMs and also the response of the vehicles to the advices received by the infrastructure.

In the future we can also consider:

- **TMC Behaviour:** this class will manage multiple RSU nodes (as current we work with a single RSU Behaviour node that acts as the TMC and defines the traffic management procedures of the simulation).

The ‘behaviour’ classes inherit from their respective base classes BehaviourNode and BehaviourRsu (located in baseApp/application/model). Here we will put the logic for the application. The behaviours implement different event triggered methods, see next paragraph.

2.1.5 Entry points for actions

The behaviour class defined in the TransAID applications is called from the baseApp at different events. At each one of this events we need to program the response of our applications to the information received.

Initially, the interface calls

- **Start()** - executed at application initialisation. This method must initialise the application and schedule the first actions to be done. For example, at the start of the application the transmission of periodic CAM messages should be scheduled.

Regular events, which are fired on every simulation step are:

- **onAddSubscriptions()** - called at the beginning of each simulation step when the iCS asks the application whether it requires interaction. The interaction between the applications and the iCS are done using subscriptions. This subscriptions can be of different types, for example, triggering information retrieval, e.g. calling TraCI-getter methods or commanding the transmission of a messages.
- **Execute()** - called at the end of each simulation step. This is the main function of each application that should coordinate the traffic management measures defined by the application. In the case of an RSU this function must contain the basic functionality of the infrastructure in order to execute the service. Here the behaviour has the possibility to transmit result data back to the iCS, which may be used by different applications. Note that results of TraCI requests (direct information retrieval from the SUMO may be accessed via `GetLastTraCIResponse()` of the Behavior class).

Truly event-driven methods which are called in between **onAddSubscriptions()** and **Execute()** are:

- **Receive()** - called on reception of a unicast or broadcast message.
- **processCAMmessagesReceived()** - called on reception of CAM messages.
- **processTraCIResult()** - called on reception of a TraCI response requested from `onAddSubscriptions()` in the current simulation step or from another behaviour method in the previous simulation step.

2.1.6 Actions

The behaviour class can access the iCSInterface, which allows it to interact with the iCS to send messages, retrieve simulation state information or manipulate the simulation. It can be accessed via the method **GetController()**. In the following we list the most important available methods:

One-shot subscriptions:

- Send()
- SendTo()
- AddTraciSubscription()
- AddTraciStop()
- commandTraciOpenGap()
- setTraciParameter()
- requestToC()

Persistent subscriptions:

- StartSendingCAMs()
- receiveCAMs()
- receiveUnicast()
- receiveGeobroadcast()
- requestMobilityInfo()

Other:

- GetLastTraCIResponse() (in Behaviour)
- GetCurrentTimeStep()

2.2 Creating test suites

When porting the traffic management code from the WP4 to the WP6 environments, we need to make sure that the same logic is preserved. In order to guarantee this, alle applications implemented in the use cases should create test suites, similarly as described in Deliverable D6.1. We use the same testing framework, called *TextTest*⁶. An example of running a suit of such tests is shown in Figure 3.

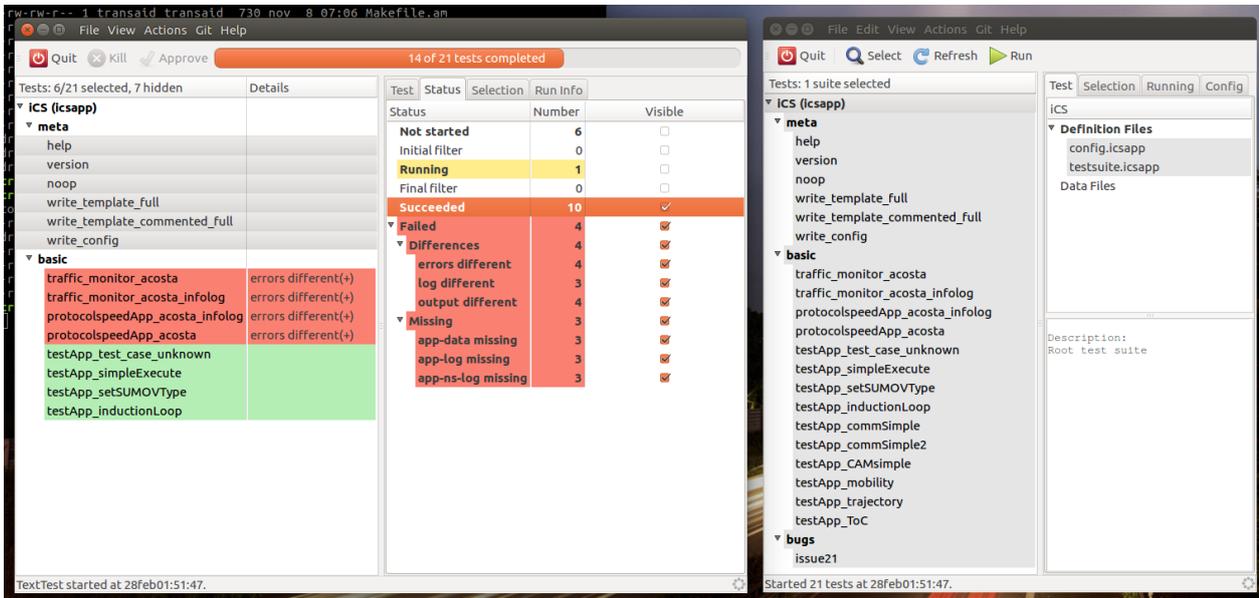


Figure 3: An example of a test suite running in TextTest.

Tests are created in the `transaid-apps` branch of the repository, separated for each use case individually. All tests are stored in the `transaid/TransAIDScenarios/tests/scenarios` folder. There, they are split between:

- **baseline**: the scenario without traffic management,
- **TM**: the scenario with traffic management, as implemented in SUMO, and
- **full**: the scenario with traffic management, as implemented in the iCS.

All relevant data pertaining to a specific use case (i.e. SUMO networks, configuration files, ...) are copied to the relevant scenario in the tests folder.

Just as before, 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`). However, here we need to be a bit more careful and considerate of the complexity involved with comparing the various iCS traffic management applications to their previously created SUMO counterparts.

⁶ <http://texttest.sourceforge.net/>

Answering the question “What to test?” yields a plethora of possibilities. The underlying idea however is that what is newly created for the iCS mimics the implementations already done for SUMO. This means that we need a way to compare entire simulation results. The most straightforward way for doing this, is by comparing results on an aggregated level. Each simulation run yields quantities like the average network speed, the total throughput, ... These can directly be compared with each other. In this spirit, we should allow some flexibility in interpreting the results, that is to say tests do not fail because an exact comparison yields a false result. Rather, we put a small margin on all obtained quantities, and check whether or not the tested ones fall within these margins (similar to how floating-point numbers are compared by using an epsilon, albeit a larger one in this case).

A first set of quantities that can be compared, are the ones listed by SUMO’s simulation output summary. This output contains the simulation-wide number of vehicles that are loaded, inserted, running, waiting to be inserted, have reached their destination and how long they needed to finish the route. The last value is normalised over all vehicles that have reached their destination so far. Table 1 presents an overview of these quantities.

Table 1: Overview of SUMO simulation output summary.

Quantity name	Type	Description
time	(simulation) seconds	The time step described by the entry.
loaded	#	Number of vehicles that were loaded into the simulation so far (including reported time step).
inserted	#	Number of vehicles inserted so far (including reported time step).
running	#	Number of vehicles that were running within the reported time step.
waiting	#	Number of vehicles which were waiting for insertion (could not be inserted) within the reported time step.
ended	#	Number of vehicles that have reached their destination so far (including reported time step).
meanWaitingTime	s	The mean time all vehicles up to now and within the reported time step had to wait for being inserted; -1 if no vehicle has been inserted, yet.
meanTravelTime	s	The mean travel time of all vehicles that have left the simulation within the previous and the reported time; -1 if no vehicle has been removed from the simulation, yet
halting	#	The number of vehicles in the network with speed below 0.1m/s (which are not waiting at a <stop>).

meanSpeed	m/s	The mean speed over all vehicles in the network (which are not waiting at a <stop>).
meanSpeedRelative	m/s	The mean speed over all vehicles in the network relative to the speed limit (which are not waiting at a <stop>).
duration	ms	The computation time for that simulation step in milliseconds).

A more detailed comparison of simulation outputs, would be to use detector measurements and/or explicit vehicle trajectories, create time-space diagrams from these (of average speeds or flows), and then compare these with each other and define whether or not the deviation is significant. An example of this is a study⁷ whereby a first-order macroscopic model was compared to a traffic cellular automaton, comparing the time-space diagrams, as shown in Figure 4.

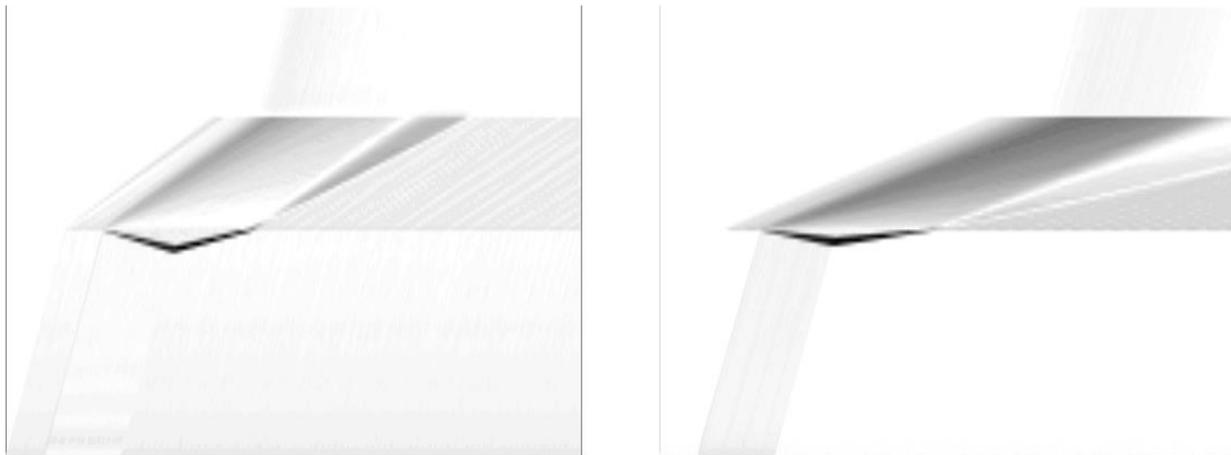


Figure 4: Time-space diagrams showing the differences in local traffic densities for a first-order macroscopic model and a traffic cellular automaton. The modelled road consists of three sections, with the middle one having a lower speed, hence leading to the formation of a congestion region. Darker regions indicate large differences between both modelling approaches (the left and right plots are to illustrate various parametrisations).

Even though this type of analysis would certainly allow us to detect deviations in the time-space plane (e.g. congested areas that may appear/disappear as artefacts, ...), it would be out of scope for now. In addition, such analyses have not been done widespread before, as they are also difficult to interpret, and still require some aggregation in order to test these ‘automatically’.

⁷ Maerivoet S., (2006) Relating the dynamics of the STCA to the LWR model, in Modelling Traffic on Motorways: State-of-the-Art, Numerical Data Analysis, and Dynamic Traffic Assignment, Catholic University of Leuven.

3 Conclusions

This deliverable D4.3 explained how simulations of both the baseline (WP3) and the traffic management schemes (WP4) can be ported from the SUMO simulation environment (with the help of the TraCI interface and Python scripts) to the iCS environment (using the C++ language).

We first gave an explanation on how to set up the creation of a traffic management application in the context of the iCS. Details were given on how to prepare the development of an application, based on the source code in the repository. We also explained the interactions between the iCS, SUMO, ns-3, and the various applications, using subscriptions and the exchange of messages.

We then provided information on how to test the implemented traffic management schemes, i.e. checking of the newly implemented logic would comply with the previously obtained results from WP4. We explained this via a method of aggregate quantities, rather than explicitly comparing time-space diagrams.