This project has received funding from the ECSEL Joint Undertaking under grant agreement No 101007326.
This Joint Undertaking receives support from the European Union's
Horizon 2020 research and innovation programme and the ECSEL member states.

Automotive Intelligence for/at Connected Shared Mobility

Deliverable	**Report on simulation architectures for testing, verification, and validation of SC1 building bricks.**		
Involved WPs	WP2	**Deliverable type**	Public
Project	***AI4CSM***	**Grant Agreement Number**	101007326
Deliverable File	D2.1	**Last Modified**	02.05.2023
Due Date	30.04.2023	**Actual Submission Date**	02.05.2023
Status	Draft	**Version**	1.0
Contact Person	Gerhard Dorn	**Organisation**	Virtual Vehicle (VIF)
Phone	+43 316 873 9761	**E-Mail**	gerhard.dorn@v2c2.at

Document history			
V	**Date**	**Author**	**Description**
0.01	11.10.2022	Jorge Perez (ZF)	Initial Version
0.02	19.10.2022	Gerhard Dorn (VIF)	First adaption for SC1
0.03	15.02.2023	Gerhard Dorn (VIF)	First input for SCD1.2
0.04	23.02.2023	Karin Festl (VIF)	Adding module
0.05	24.02.2023	Gerhard Dorn (VIF)	Adding modules
0.06	28.02.2023	Moritz Schaffenroth (OTH)	Input for SCD1.3
0.07	03.03.2023	Lorenz Klampfl (AVL)	First Description of SCD1.1 and AVL contributions
0.08	09.03.2023	Stefan Jaksic (AIT)	AIT contributions to SCD1.1 and partner level contributions
0.09	24.03.2023	Franz Wotawa, Iulia Nica, Ledio Jahaj, Liliana Marie Prikler (TU GRAZ)	TUG contributions to SCD1.1 and partner level contributions
0.10	27.03.2023	Gerhard Dorn (VIF)	Update for SCD1.2
0.11	28.03.2023	Lorenz Klampfl (AVL)	Merging contributions of SCD1.1 partners and final adaptions in chapter 4.
0.12	29.03.2023	Mantas Makulavicius (VGTU)	Update module
0.13	31.03.2023	Gerhard Dorn (VIF)	Including SCD1.1
0.14	03.04.2023	Gerhard Dorn (VIF)	Including SCD1.3
0.15	18.04.2023	Gerhard Dorn (VIF)	Finalizing
0.16	20.04.2023	Gerald Fritz-Mayer (TTTAUTO)	Adding partner contributions
0.20	26.04.2023	Ovidiu Vermesan (SINTEF)	Review
1.00	02.05.2023	Gerhard Dorn (VIF)	Final and reviewed Version

Disclaimer

The opinion stated in this document reflects the opinion of the authors and not the opinion of the European Commission. The Agency is not responsible for any use that may be made of the information contained (Art. 29.5 of the GA).

Table of contents

1 Executive/ Publishable summary

This document is intended to give an overview of smart connected shared mobility for urban area, collected from all partners participating in task T2.1 of AI4CSM WP2.

Task T2.1 results are fundamental to Supply Chain 1 (SC1), which targets to provide three demonstrators: **SCD 1.1: Lessons-learned based (critical scenario) update of ADAS/AD Controller**, **SCD 1.2: Robo taxi automated operation in challenging urban use cases** and **SCD 1.3: Virtual city routing**.

The system level design defined in the task will outline the framework to develop these demonstrators. AI4CSM's technology enabler supply chains SC4 to SC7 and their related demonstrators target to integrate their results into the output enabler supply chains SC1 to SC3.

We use the following scheme to collect and compile T2.1 work products of the task members:

1. Partners concentrate on their individual work package tasks and contribute a chapter to D2.1 describing the work executed for T2.1.
2. Each demonstrator lead compiles the results for each demonstrator.
3. Inputs are reworked and consolidated. Further, partners discuss cooperation aspects of system level design and align in regular meetings.
4. The task leader compiles an aggregation of the system level design into this document and documents the approach and achievements.

Chapter 3 of this deliverable D2.1 describes the scope of the document and gives an introduction and overview. The following three chapters contain the system level design of all demonstrators. Finally, a conclusions chapter sets the work in context to related AI4CSM tasks and summarizes the impact and contributions to the work packages and supply chains.

2 Non publishable information

All the information below is publishable.

3 Introduction & Scope

3.1 Purpose and target group

The AI4CSM project will develop advanced electronic components and systems (ECS) and architectures for future mass-market ECAS (electric, connected, autonomous, shared) vehicles. This fuels the digital transformation in the automotive sector to support the mobility trends and accelerate the transition towards a sustainable ecosystem.

SC1 will demonstrate ECAS vehicles in future shared mobility situations. To enable this, SC1 will develop smart edge- and cloud-based building bricks for autonomous mobility interconnected with secure communication architectures and systems. SC1s **vision** to enable a safe, efficient, and green autonomous mobility in urban areas through connected mobility represents the central element of the entire SC. In that sense,

- **safe** stands for safety enhancement via cooperative integration of cloud knowledge into edge perception and vehicle intelligence solutions.
- **efficient** stands for maximize traffic throughput with minimum latency time together with minimizing active cars in urban environments.
- and **green** stands for consideration of shared resources for minimizing energy consumption.

Based on this vision the **key challenge** represents the development of smart edge- and cloud-based building bricks for autonomous mobility interconnected with secure communication architectures and systems addressing the overall project objectives 1, 2 and 6. To realize the specified vision and tackle the key challenge SC1 focuses on the following objectives depicted in Figure 1Figure 1.

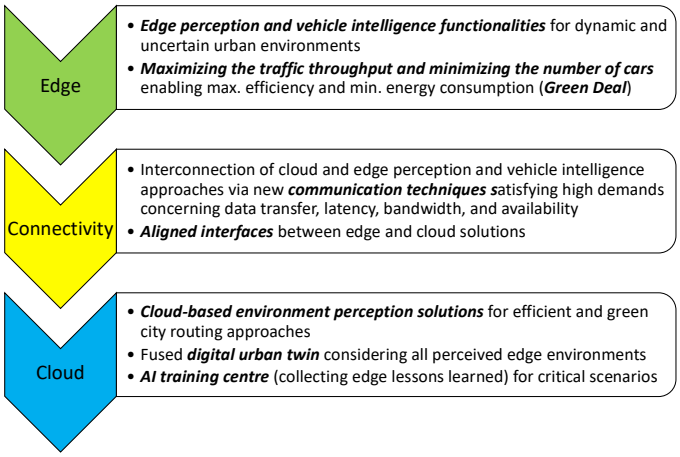

FIGURE 1: SC1 OBJECTIVES

All SC1 objectives can be categorized into edge-, connectivity- and cloud specific objectives. Results of task T2.1 activities form the framework towards fulfilment of these objectives, finally leading to three demonstrators **SCD 1.1: Lessons-learned based (critical scenario) update of ADAS/AD Controller, SCD 1.2: Robo-taxi** and **SCD 1.3: Virtual city routing**. The system level designs collected in the task will be the blueprint to develop these demonstrators.

3.2 Contributions of partners

Individual partners summarized their contributions in the following table and provided detailed contributions in the referenced chapters.

TABLE 1: OVERVIEW PARTNER CONTRIBUTIONS

Chapter	Partner	Contribution
1, 2, 3, 5, 7	VIF	Deliverable structure and draft outline, demonstrator SCD 1.2 description, system level design for demonstrator SCD 1.2, conclusion section
3.2, 4.8, 7.2, 7.3, 7.4, 7.5	TUGRAZ	Demonstrator SCD1.1 description, partner level contributions, conclusion section
3, 4, 7	AIT	Demonstrator SCD1.1 description, technical contribution description, conclusion section
3, 4, 7	AVL	Demonstrator SCD1.1 description, contributions on partner level, conclusion section
3, 6, 7	OTH	System level design for demonstrator SCD 1.3
3, 6, 7	VGTU	RSU Simulation module

3, 5, 7	TTTAUTO	Description of communication platform design and interfaces to SCD1.2 modules, HW platform description, requirements mapping.

3.3 Relation to other activities in the project

SC1 represents one of the core output enabler supply chains and therefore acts as a first integration platform for the component demonstrators developed in the technology enabler SCs (SC4 to SC7) to be tested and validated in urban areas, see Figure 2.

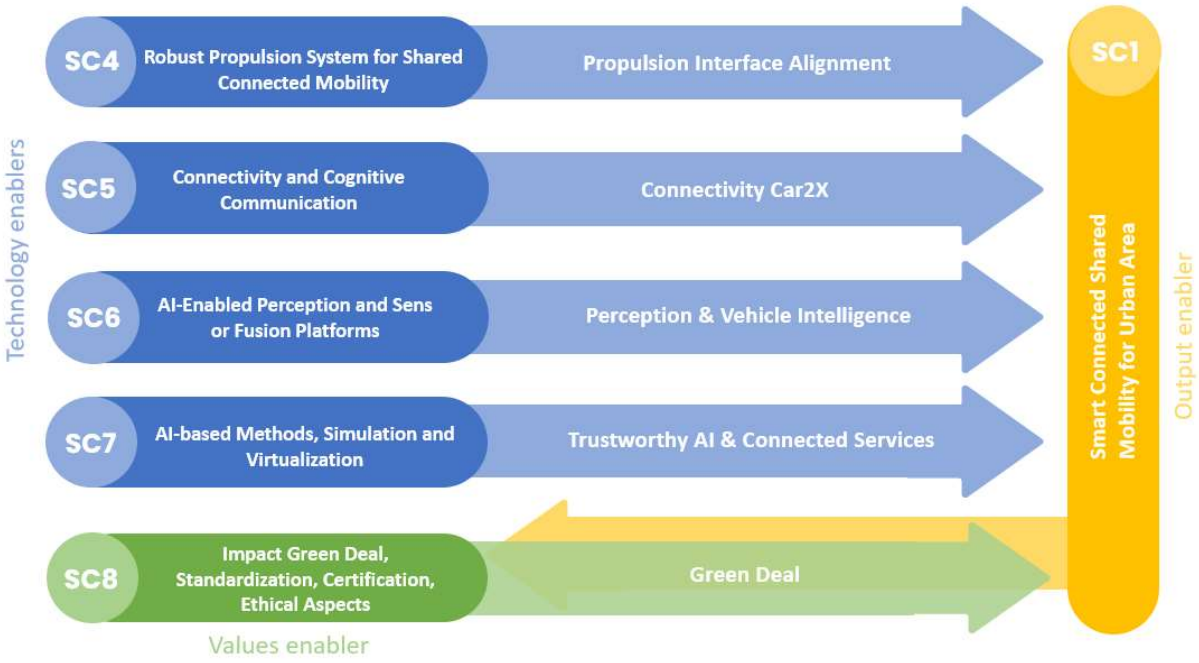

FIGURE 2: OVERVIEW SC1 INTERCONNECTIONS

Therefore, this supply chain will very closely collaborate with the following supply chains:

- Propulsion interface alignment with **SC4**
- Integration of connectivity solutions of **SC5** into the demonstrator SCD 1.2 for real-world approval
- Application of **SC6** related perception and vehicle intelligence approaches for urban use cases
- Synchronization of **SC7** cloud based digital twins for urban areas enabling trustworthy AI (Artificial Intelligence) and efficient interconnection between the edge and the cloud.
- Bidirectional exchange and import of **SC8** related Green-Deal principles and potentials to maximize the impact of SC1 demonstrators.

3.3.1 Input from WPs, SCs and tasks

T2.1 is dedicated to defining the system level design for the development of the **smart edge- and cloud-based building bricks** for autonomous mobility interconnected with **secure communication** architectures and systems which encompasses three (3) demonstrators within SC1:

- **<u>Demonstrator SCD 1.1:</u> Lessons-learned based (critical scenario) update of ADAS/AD Controller (lead: AVL, partner: TUG, AIT)**

- **<u>Demonstrator SCD 1.2:</u> Robo taxi automated operation in challenging urban use cases (lead: VIF, partner: TTTA, IFAG)**
- **<u>Demonstrator SCD 1.3:</u> Virtual city routing (lead: OTH-AW, partner: TUG, VIF, AVL)**

To collect the system level design for the demonstrators, a **collaborative** approach was followed, building on the demonstrator descriptions and specified requirements (D1.1).

Dedicated demonstrator workshops with all contributing SC1 partners in the middle of the task period helped to co-define necessary interfaces and the baselines for the system level design.

Special attention was put on the specific links to the technology enabler supply chains SC4 to SC7 as their developed systems and results will be integrated later into SC1 real-world demonstrators. In that manner, the targeted SC1 system level design will be shared and synchronized with the technology enabler supply chain requirements collection tasks T1.4 to T1.7.

3.3.2 Output from these results

The presented results comprise the output of task T2.1 (System level design for smart connected mobility) and will feed other WPs within AI4CSM that entail the implementation of the smart edge- and cloud-based building bricks together with their communication approaches in between. This deliverable will be particularly useful to exchange experience from a planning perspective of how common standards can be implemented. As such, these results will be linked to five (5) SCs and in three (3) WPs:

<u>Supply chain links:</u>

- Interface alignment with technology enabler supply chains:
  - SC4: Robust Propulsion System for Shared Connected Mobility
  - SC5: Connectivity and Cognitive Communication
  - SC6: AI-Enabled Perception and Sensor Fusion Platforms
  - SC7: AI-Based Methods, Simulation and Virtualization
- Integration of technology enabler (SC4 to SC7) supply chain results into SC1 demonstrators
- Synchronization of SC1 activities with Green Deal principles developed and evaluated in SC8

<u>Work package links within SC1:</u>

- **WP4** (T4.1 - Embedded HW/SW for Smart Connected Mobility) will focus on the development and implementation of the software building bricks according to the now presented system level design.
- The developed software solution of WP4 will be integrated into the specified demonstrators within SC1 in **WP5** (T5.1 - Integration of systems for Smart Connected Mobility).
- Finally, deliverable D2.1 will provide the blueprint for **WP6** (T6.1 - Validation and tests of Systems for Smart Connected Mobility) for testing and verification purpose to meet all stated requirements defined in D1.1 in the SC1 demonstrators. This deliverable D2.1 contains the strategy of how the developed technologies, modules and systems will be validated against the requirements, specifications and KPIs defined D1.1.

Figure 3 depicts the relation of this deliverable with other WPs, SCs and tasks regarding inputs and outputs.

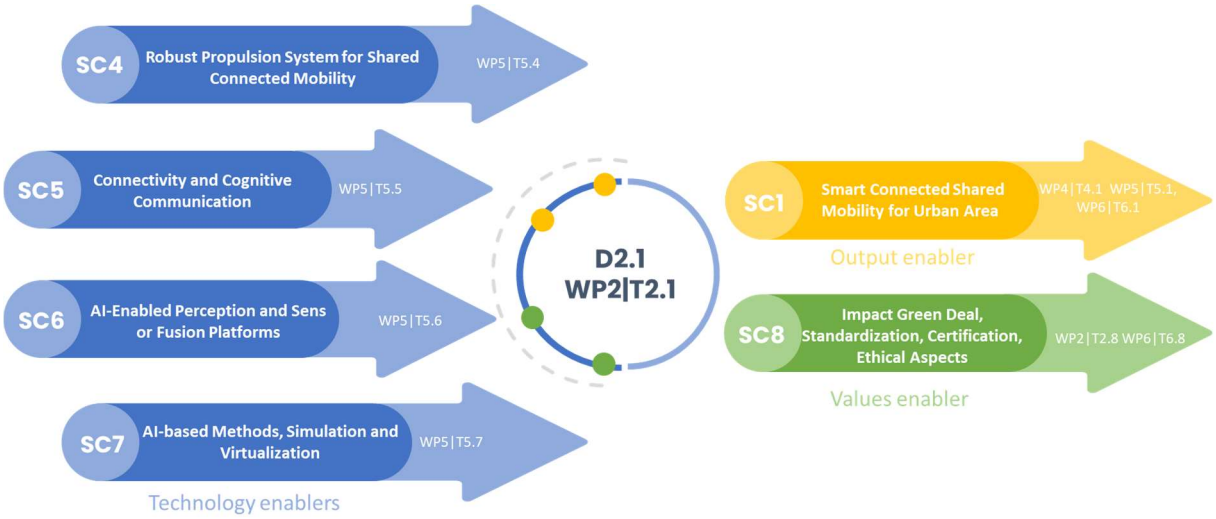

FIGURE 3: RELATION OF DELIVERABLE D2.1 WITH OTHER ACTIVITIES WITHIN AI4CSM.

4 Demonstrator 1 (SCD1.1 Lessons-learned based (critical scenario) update of ADAS/AD controller)

In recent years, the field of ADAS/AD has seen impressive technical advancements which has paved the way for their increasing integration within modern vehicles. Although these developed and integrated functions are already versed at addressing various traffic scenarios, there may still arise situations where the controller is not able to handle a situation correctly and thereby necessitating the intervention of a human driver. As a result, there is still a need to validate the functionality of ADAS/AD and explore untested scenarios in the enormous search space to ensure reliable and trustworthy systems.

With demonstrator SCD1.1, the partners AVL, TUG, and AIT contribute to this problem and will demonstrate a lessons-learned based approach for updating and validating ADAS/AD by monitoring deviations between and ADAS/AD function and the behaviour of an expert driver during operation. With this approach, valuable situations for the verification and validation of ADAS/AD functions are detected, recorded, and post-processed for transferring the scenario into a simulation environment. Within an AI training center, the converted scenarios are further enhanced by generating concrete test cases for evaluating the performance of the ADAS/AD under test.

In the following sections an overview of the system level design of the key building blocks of the demonstrator is given as well as their interfaces and connections to other supply chains.

4.1 High-level architecture

Figure 4 depicts the high-level architecture of our demonstrator. As it can be seen, demonstrator SCD1.1 can be divided into three main parts, i.e., test scenario definition, test case generation, safety evaluation. Figure 4 also shows the parts of the demonstrator where each partner's focus lies as well as connections to SC7. The first part of the demonstrator is dealing with the definition of a test scenario by detecting situations of an ADAS/AD where the system has not performed well and its followed virtualization of the scenario. The next step is the generation of concrete test cases in the scenario search space by applying different algorithms and strategies. The output is in a last step to the safety evaluation part where the generated concrete test cases are executed and with the help of safety monitors diagnoses & verdicts are made. A more detailed explanation of each part is included in the following subsections.

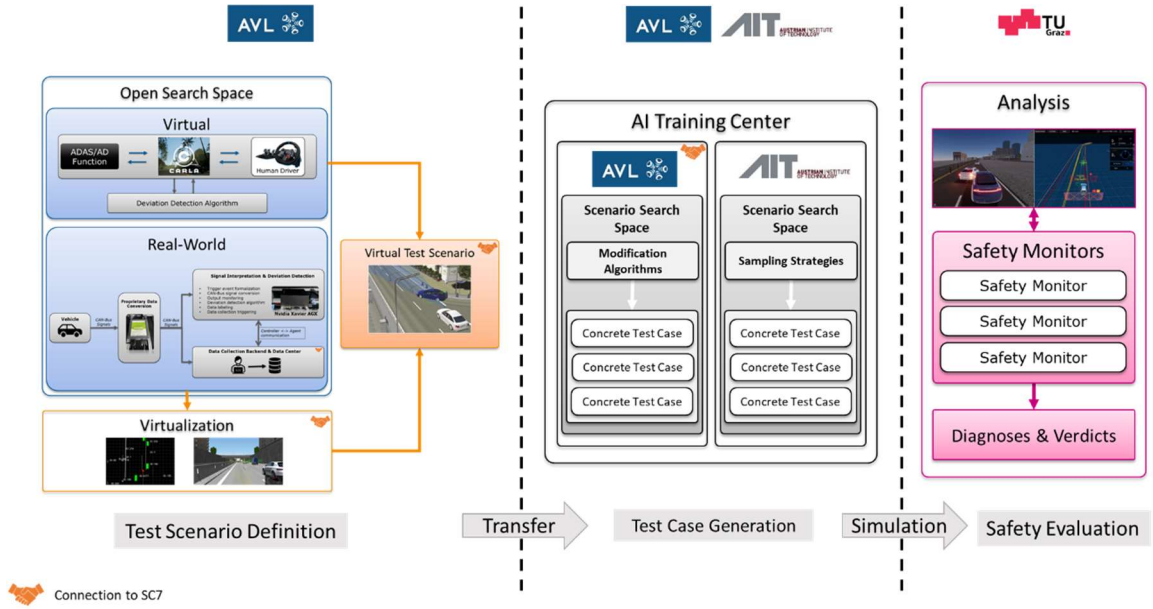

FIGURE 4: HIGH-LEVEL ARCHITECTURE OF DEMONSTRATOR SCD1.1

4.2 Relevant requirements for system, subsystems, and components

Several requirements which were defined in WP1, T1.1 are considered within WP2 and the system level design process of demonstrator SCD1.1 to ensure the collective validation within WP5 and WP6. Table 2 gives a summary of the relevant requirements. For a detailed list of all requirements related to demonstrator SCD1.1 we refer to deliverable D1.1 [1].

TABLE 2: SUMMARY OF RELEVANT REQUIREMENTS

	WP2	WP4
ADAS/AD function availability (simulation & in-vehicle)	●	
ADAS/AD function integrateability (simulation & in-vehicle)	●	
Sensor signal availability (in-vehicle)	●	
Deviation detection (simulation & in-vehicle)		●
Data collection (simulation & in-vehicle)		●
Standardized virtual scenario formats	●	
Identification of critical test case parameters		●
Sub-block interoperability	●	
Availability of expected sensor signals	●	
Model-based diagnosis completeness		●
Model-based diagnosis correctness		●
Model-based diagnosis performance		●

There are no requirements addressed in WP3 since SC1 is not involved there.

4.3 Design flow/methodology

As seen in Figure 4, the demonstrator is split into three main parts and the output of each part is passed to the consecutive demonstrator part. Within each part, the different building blocks of the demonstrator are developed at a partner level. However, to achieve the overall goal of demonstrator SCD1.1 it is necessary that a close collaboration between the partners is established. Within this collaborative work, special focus is set on the decision of correct interfaces and standardized formats so that the output of each part can be further processed without any additional work of the partners working on the following part of the demonstrator.

Collaborative system level design: To align on interfaces and standardized formats, bi-weekly meetings are organized by the SCD1.1 lead (AVL). Within these meetings, the overall system level design was established in an iterative process by elaborating what each partner needs to fulfil their respective tasks and by finding acceptable solutions for all partners.

4.4 Relevant standards, norms, and ethical aspects

Standards considered within the development of SCD1.1 are mostly related to the first part of the demonstrator, i.e., test scenario definition. Here relevant standards are **ASAM MDF** [2] and **ASAM OSI** [3] which are related to the data collection process. **ASAM MDF** is a compact binary file format to store recorded data for post-processing activities and offers efficient and high performance of huge amounts of measurement data. **ASAM OSI** is an object-based environment description utilizing the protocol buffer library developed by Google and provides standardized interfaces to driving simulation frameworks.

In addition, **ASAM openSCENARIO** [4] and **ASAM openDRIVE** [5] are used when dealing with the virtualization of the collected data. The **ASAM openSCENARIO** standard is used to describe the dynamic contents of a scenario whereas the **ASAM openDRIVE** format is necessary to describe the road network of the scenario.

4.5 Overview of components/subsystems/systems in demonstrator

As already stated in the previous sections, demonstrator SCD1.1 can be separated into three main parts, i.e., test scenario definition, test case generation, and safety evaluation. Each of these parts consists itself of different building blocks. Within the next sections a more detailed description of the system level design of each part and building blocks is given.

4.6 Demonstrator part 1: Test Scenario Definition

4.6.1 Concept description

The starting point for our demonstrator dealing with lessons-learned based (critical scenario) update of ADAS/AD controller builds the definition of a test scenario which should be evaluated in more detail to reveal possible issues in the controller and take appropriate counter measures. The test scenario definition is done in two different setups, first virtually within a simulation environment and second in an in-vehicle setup.

4.6.1.1 Virtual

Figure 5 displays the methodology for the intelligent extraction of interesting test scenarios within a simulation environment. The system level design for scenario extraction can be separated into three main sub-blocks, i.e., the simulation environment, the deviation detection, and the virtualization. Tasks within SC1 mostly are related to the deviation detection and its interconnection with the simulation sub-block, therefor the focus within this deliverable lies on those. For more detailed information about the virtualization of collected data we refer to D2.8 (Report on architecture with AI-based methods for simulation and virtualization) which is part of SC7.

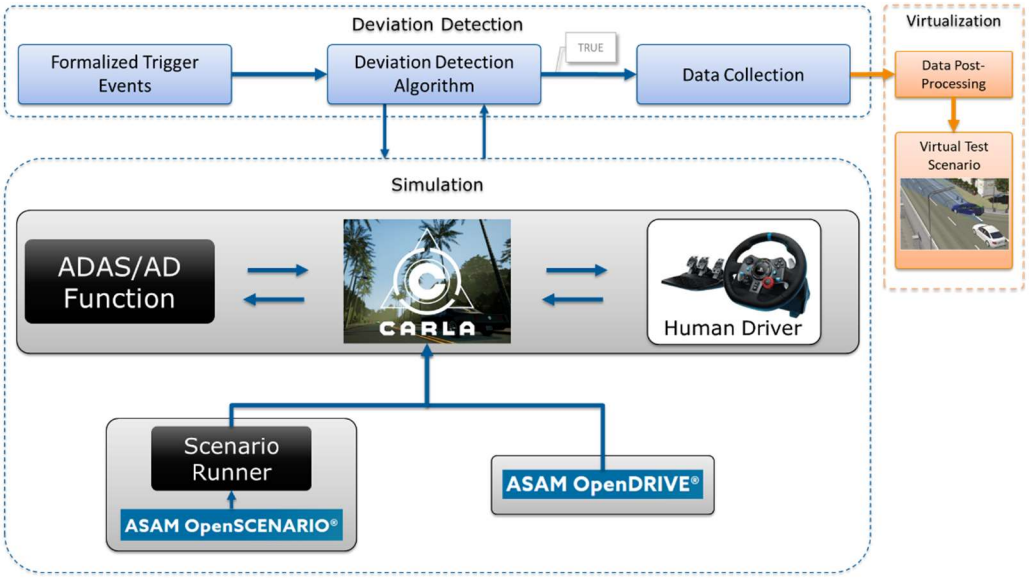

FIGURE 5: METHODOLOGY FOR TEST SCENARIO DEFINITION WITHIN SIMULATION

Simulation: To identify deviations between an ADAS/AD controller and a human driver within simulation a suitable simulation environment needs to be arranged that is capable of providing interfaces for the integration of an ADAS function as well as interfaces for receiving input from a human driver. In addition to that it has to be ensured that it is able to use the defined standard formats. Furthermore, interfaces are necessary to include submodules, i.e., the deviation detection. As seen in Figure 5, it was decided to use the CARLA simulator for extracting test scenarios. Within the simulation framework, CARLA is receiving as input an ASAM openSCENARIO and ASAM openDRIVE file which are used to initialize CARLA with the defined environment as well as a base scenario, e.g., driving on a highway, number of included traffic participants. Within CARLA the EGO vehicle is driven by a human driver with a physical steering wheel, acceleration pedal, and brake pedal. The ADAS/AD function under test is integrated in passive mode, which means that it receives simulation data, i.e., sensor data, from CARLA and outputs the respective action the system under test would perform in a certain situation without being connected to the actual acceleration and braking signals of the EGO vehicle within simulation.

Deviation detection: The deviation detection block is used to extract only interesting scenarios where an actual discrepancy between the human driver and an ADAS/AD system was observed. It communicates with the simulation framework and feeds back its output to the simulator to indicate if a misbehaviour of the system under test was identified. In addition to the simulation data itself, the developed algorithm takes formalized trigger events as input. Those are for instance used to reduce the detection of false positives. If we take as an example an autonomous emergency breaking (AEB)

function which should be tested, a formalized trigger could be that the braking pedal of the human driver needs to be pressed for a certain amount of time to ensure that the driver is really breaking and not by mistake. Otherwise without such formalizations we would identify a misbehaviour although the software subsystem (SUT) might has performed a correct action.

As soon as a deviation is detected, the developed algorithm reports to the data collection submodule and the data recording of the specific scenario where the problem occurred is initiated to extract the scenario for further in detailed investigations.

Details about the actual development and implementation of the simulation framework and deviation detection algorithm will be part of the upcoming deliverables in WP4.

4.6.1.2 Real-World

The next logical step builds the transfer of the previous described method into an in-vehicle setup. Figure 6 displays the system design to perform deviation detection within a vehicle during driving on the road and extract real-world scenarios. Similar as before, the tasks concerned within SC1 mostly deal with the in-vehicle signal interpretation and deviation detection. Virtualization topics are mostly dealt with within SC7 and we refer here to the respective deliverables.

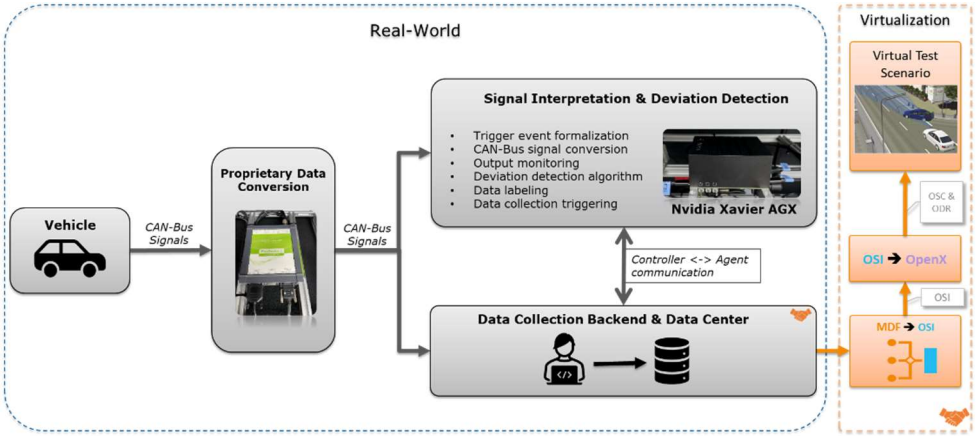

FIGURE 6: METHODOLOGY FOR TEST SCENARIO DEFINITION FROM REAL-WORLD DRIVING DATA

As outlined in Figure 6, the starting point for extracting scenarios from real-world are the CAN-bus signals delivered by the vehicle. Those signals are in a next step converted from a proprietary data format to readable CAN signals. The pre-processed signals are then forwarded to a computing platform, i.e., Nvidia Xavier AGX, as well to a data collection backend. On the computing platform signal interpretation and deviation detection is performed. This includes the conversion of the raw CAN-signals into an interpretable format, the formalization of the trigger events, similar than in the virtual use case, output monitoring, and data collection triggering. Based on the output of the deviation detection algorithm, the computing platform initializes the data recording process to extract the scenario where a discrepancy between the human driver and the SUT was identified. With the recorded data, the scenario is in a next step virtualized to perform in depth analysis in part two and three of the overall demonstrator.

4.6.2 Computation platform and toolchain

CARLA: to run Carla a conventional workstation with build in GPU was used.

Algorithms developed: the algorithms developed in part one of demonstrator SCD1.1 are implemented in Python. However, this is open to changes of the programming language for minor submodules of the algorithms if we encounter during evaluations that a different programming language would suit better for a specific task.

FlexDevice-S: used to convert the proprietary CAN signals to a readable format.

Nvidia Xavier AGX: this energy efficient and high-performance computing platform is used in the in-vehicle setup to execute the developed and implemented algorithms during real-world driving. This includes the raw CAN signal interpretations as well as deviation detection algorithms and communication between the platform and data collection backend.

4.7 Demonstrator part 2: Test Case Generation

Test case generation phase is an intermediate step in our methodology defined for SCD1.1. In prior phase, we have gathered real-world data of a car driving in a city, an urban area, or highway, and where we observed the discrepancy between ADAS decision and an expert driver decision. We record this situation and convert it to a virtual test scenario, which is represented by a set of waypoints on a realistic map, for further examination in a simulator. The following concern follows immediately: Is the observed discrepancy a real problem in the system? Are similar test cases also resulting in an unsafe behaviour?

We can formalize our concerns into two major requirements for our testing process: (1) does the observed discrepancy represent a real issue with the ADAS, or is this problem induced by the environment conditions and specifics of a concrete case and (2) in case we have a real bug: what are the critical parameters of the ADAS that represent the root cause of the issue and how to find these parameter values efficiently.

These two questions are non-trivial to answer because they require extensive testing over a large multi-dimensional parameter space. Moreover, collecting large amounts of diverse and representative data to produce test cases is cumbersome. The test cases need to generalize well w.r.t. different driving conditions. For these reasons a direct application of model-based test case generation would remain unfeasible.

4.7.1 Systematic testing for critical parameters

To answer these challenges, we perform *systematic testing for critical parameters* and respective values using *scenario-based test case generation*. We start with a recorded ADAS behaviour and convert it to an abstract scenario.

A scenario is a specification of a vehicles and their properties and behaviours which allows systematic scene generation. For example, we can have a scenario with two vehicles, a lead and an ego, where a lead vehicle is misbehaving and the ego, who is following the lead, needs to perform an emergency break to avoid direct collision. An abstract scenario is a scenario with parameters that are subject to randomizations. Using such an abstract scenario we can generate many scene instantiations, given

different weather conditions, time of the day, vehicle physics, which means we can perform testing in a very general way. This way, we can take the failing controller and test it in related, similar scenarios, to understand the criticality and the reasons for failure. Therefore, such a methodology is a solution to challenge (1).

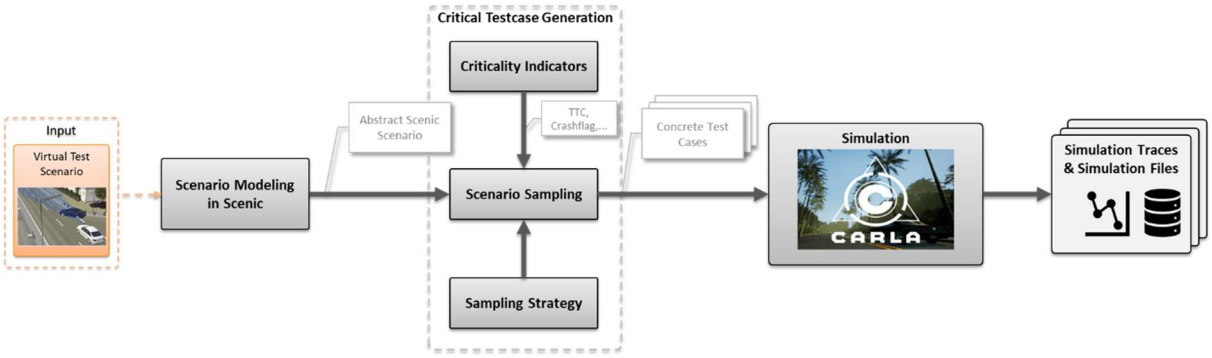

FIGURE 7: METHODOLOGY FOR CRITICALITY-BASED TEST CASE GENERATION

4.7.2 Scenic abstract scenario modelling

Scenarios involve controllers which depend on a set of parameters, and also the scenarios themselves can introduce parameters. For example, we can have emergency braking scenario with different values for the safety distance or time-to-collision, which we then model as a scenario parameter. Testing the entire space of such parameters is resource-intensive and does not scale. From testing point of view, we want to understand which of these parameters are critical for the deviation we have witnessed in real-world data, to identify a critical part of parameter search space.

Abstract scenarios allow different exploration strategies, we can ensure that we have sufficiently explored parameter space and that we can obtain relevant and minimal set of test cases needed to test for a deviation we have observed. We distinguish two classes of sampling strategies: passive, which systematically explore the parameter space, and active which are guided by the feedback from the simulated scene and achieve targeted exploration.

4.7.3 Concrete scenario based on criticality

In order to instantiate scenarios, we need to allow our tools, Scenic and VerifAI to sample concrete values of parameters in an abstract scenario. This involves a sampler which is trying to sample new values according to a sampling strategy. In our case we base our strategy on a metric for criticality. This metric is defined as a robustness of satisfaction of safety requirements. In other words, the safer the autonomous driving, the better the robustness, and vice versa. As our first measure of safety, we will adopt time-to-collision (TTC).

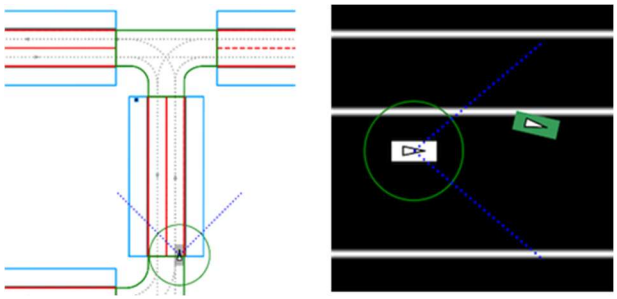

FIGURE 8: RESULTS OF SAMPLING AN ABSTRACT SCENARIO (SOURCE: SCENIC WEBSITE [6])

4.7.4 Sampling strategies

The modelled scenario parameters are subject to sampling during the instantiation of a concrete scenario. Sampling is conducted by selecting specific parameter values from the set of values defined in an abstract scenario, according to a sampling strategy. There are two major classes of sampling algorithms: active sampling, where the next sample depends on the feedback from a previous sample; and passive, where each sample is sampled independently.

Depending on the chosen kind of sampling, we can search the parameter space in breadth or in depth. Combining both techniques gives the most confidence in testing results.

In order to explore parameter search space in a systematic fashion we need to choose a uniform sampling which does not favour a specific part of the search space. This way, we can perform an initial sensitivity analysis of the search space to look for candidate critical regions. Depending on system dynamics, we can also apply this sampling to achieve certain guarantees in terms of system correctness. One kind of this sampling is the so-called Halton sampling.

In other cases, we want to aggressively search around a specific failed test case to explore its surroundings and estimate the size of the parameter search space which contains test cases that fail. This way we can estimate the impact of certain parameter values on the system correctness. With this kind of analysis, we can also determine if a failed test case was a simple glitch in the system which can be handled separately or this failing test case is not an exception, but rather a part of a group of a similar test cases which fail. In that case, this is a cause for major investigation and system debug. One strategy is to calculate the *robustness* of satisfaction of certain requirement and to optimize the search of the parameter space to minimize the *robustness*. This way we are guiding the parameter search towards failing test cases and we are searching around the failing cases. To perform this kind of search we would use *active* sampling strategies, i.e., cross-entropy sampling.

4.7.5 Tools used

As our tool of choice, we use Scenic, a probabilistic language for modelling the environments of autonomous vehicles as well as a paired tool for scenario-based testing. For sampling and parameter space exploration we use VerifAI, a falsification-based testing tool built on top of Scenic. For simulating scenarios and evaluating criticality we run our scenarios in CARLA simulator, de facto standard tool for testing of Autonomous Driving.

4.7.6 The advantages of our approach

The benefits of our approach: enabling virtual testing of scenarios based on real-world data, and identifying the critical parameters which lead to a discrepancy and generating a set of relevant test cases to test for such a critical scenario.

4.8 Demonstrator part 3: Safety Evaluation

To check the test cases for being passing or failing, we rely on properties. These properties capture expected behaviour, e.g., like the conformance of objects detected by two sensors, or a time-to-collision value that is always larger than a predefined boundary value. The underlying idea is to formalize the properties and to use simulation results to check their fulfilment. We perform this check using a theorem prover and in particular the answer set programming prover Clingo. In this subsection, we outline the overall approach, its structure, and the current status of the implementation. A high-level overview of our safety evaluation methodology is presented in Figure 9.

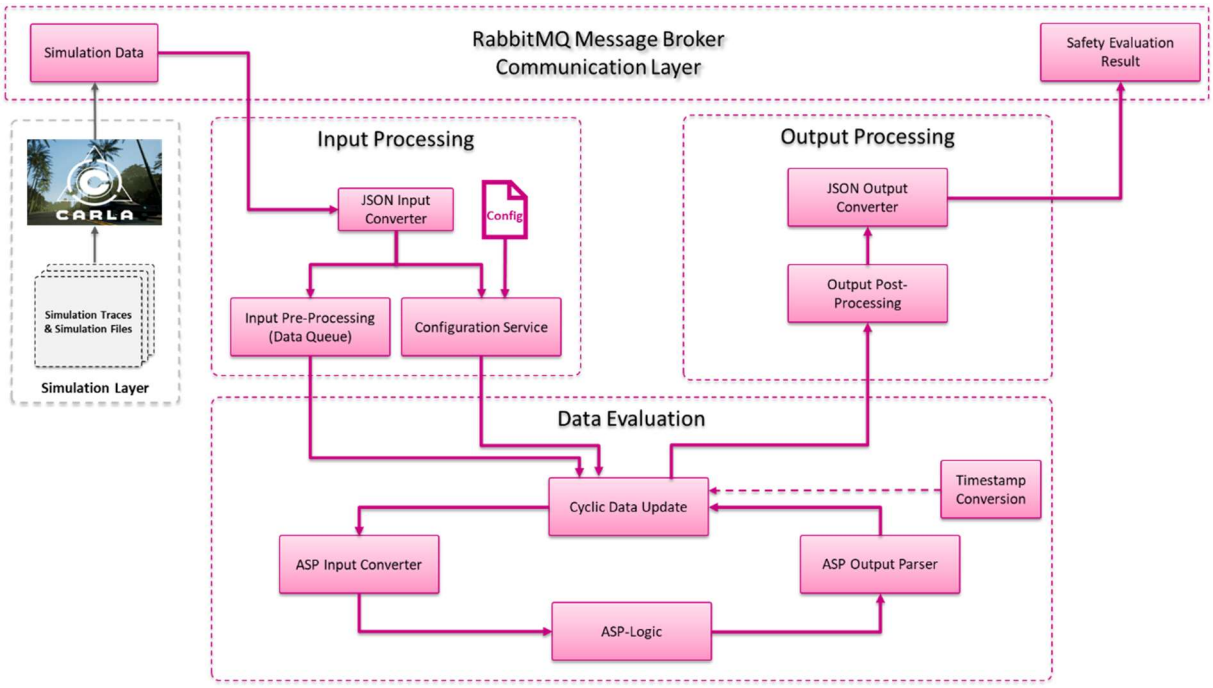

FIGURE 9: GENERAL OVERVIEW OF THE SAFETY EVALUATION METHODOLOGY

4.8.1 Software concept description

Safety evaluation is achieved through sensor data validation. We introduce a qualitative justification of data received by the sensors, meaning that an autonomous system receives more information about the quality of data and can set different actions. Our approach to create this behaviour is the introduction of logic constraints. Every data sample is processed through a set of logic rules. These rules define the correct behaviour of a data source. Constraints may be defined over time, in relation to other data sources, or both. On constraint violation, model-based diagnosis (MBD) is used to identify sources with unintended behaviour.

To close the gap between continuous sensor data and discrete logic statements, we have developed a monitoring system. We tackle the acquisition of input data from CARLA simulator and the

transformation to a common representation. To do so, we apply means of data normalization to create consistent logic predicates. These predicates are evaluated by an external solver in combination with defined rules, stating the correct behaviour of the data sources. We then publish the received result, to be processed by other interested parties.

We present a modular approach to integrate the monitoring system into existing environments. Loose connections within the used components guarantee extensibility and interchangeability of the utilized units. Furthermore, we present a communication scheme, which allows us to build hierarchies with the monitoring system. This enables data validation on different levels.

Having this modular-built tool enables us to connect it to the CARLA simulator. We developed a user interface in Python that gets the output sensor data from a CARLA simulation and uses this data as input for the monitoring system. The interface enables us to evaluate the behaviour of any simulation from Carla.

4.8.2 Computation platform and toolchain

We use CARLA version 0.9.11. to run simulations. A parser written in Python extracts data from CARLA's logs and publishes it to our monitoring system implemented in Java 11, using a REST API to register sensors and RabbitMQ 3.11.2. for transmitting their data. Internally, this monitoring system invokes Clingo 3 to perform reasoning tasks.

4.8.3 Classical (non-AI) software functions

Figure 10 depicts the architecture of the monitoring system connected to the Carla simulator.

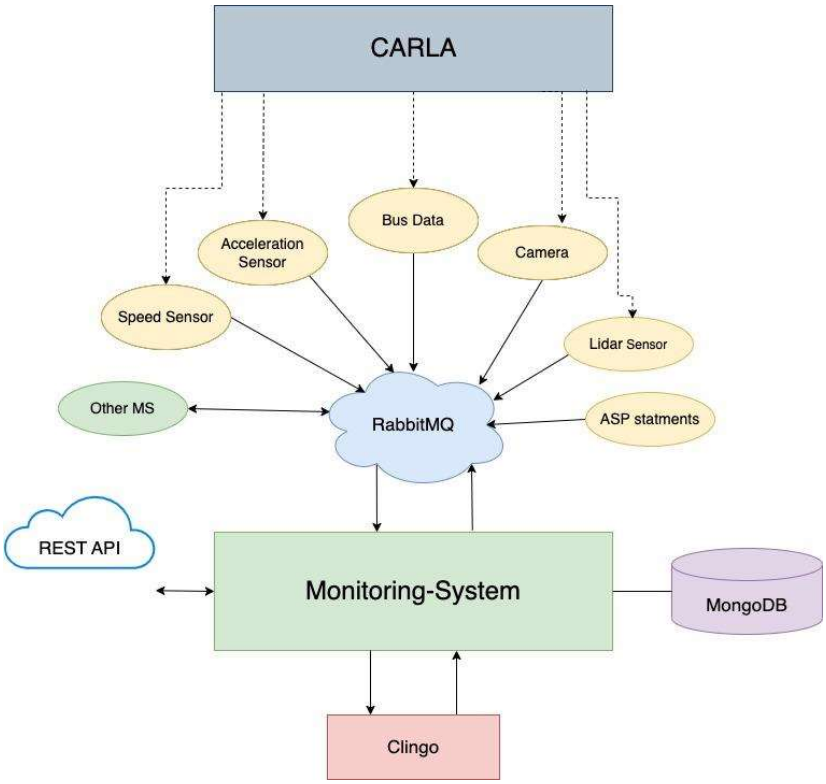

FIGURE 10: MONITORING SYSTEM CONNECTED WITH CARLA SIMULATOR

In this section, we will further describe the functionality of each component. Our first step in the evaluation framework workflow is to run a simulation in CARLA and receive the output. After receiving

the output from CARLA in text format, we developed a framework in Python, which parses the data, converts it into JSON format, and inserts it as input to the monitoring system. Our interface connects with REST API to get, create, update, and delete originators and also connects with RabbitMQ to publish the data from CARLA.

Input data into the monitoring system: A multiplicity of data is produced during automated driving simulation. To address all the data producing components, the term *originator* is introduced. An originator defines a data source in an abstract manner. This enables a flexible way for components to interact with the system. The following originators are considered representational for different kinds of data sources. Simple sensors, which measure speed or acceleration data, form one end of the range of sensors. Intelligent LiDAR sensors or cameras conclude the other end. Besides data from sensors, further sources report to the system. The internal bus system within the car, reporting general information about the current state of the vehicle; logic constraints, to adapt the behaviour of the monitoring system during operations; and the result of other monitoring systems operating in the environment.

Some constraints restrict the data produced by the originators. All these originators have to report their data in a high-level format, as data decoding is not part of the project and therefore referred to as given. Every component must report its data with a unique identification and the timestamp of the data sample as shown in Figure 11. We introduce the format of Originators, OriginatorGroups and DataPoints to represent various kinds of information about the originators while keeping a general and extendable format. The database schema that maps these three components is shown in Figure 12. The value scheme defines the specific formats of how originators publish data. Figure 13 presents these available formats. We introduce objects to store basic data types, like integers, decimal numbers (with double precision), and booleans. More specialized objects store common data produced in the automated driving domain. We provide the possibility to directly inject answer set programming statements through the AspValue-object. Furthermore, we assure input-output conformity with the AspResult-object, which defines the output format of the monitoring-system.

```
DataDto {
  "originatorId":String,
  "timestamp":Timestamp,
  "value":JSON-object
}
```

FIGURE 11: INPUT JSON-FORMAT OF ORIGINATOR DATA

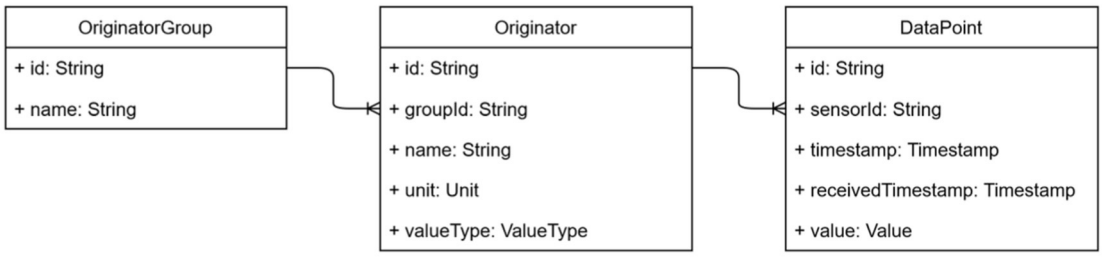

FIGURE 12: DATABASE SCHEMA STATING THE RELATION BETWEEN ORIGINATORS, ORIGINATOR GROUPS AND DATAPOINTS

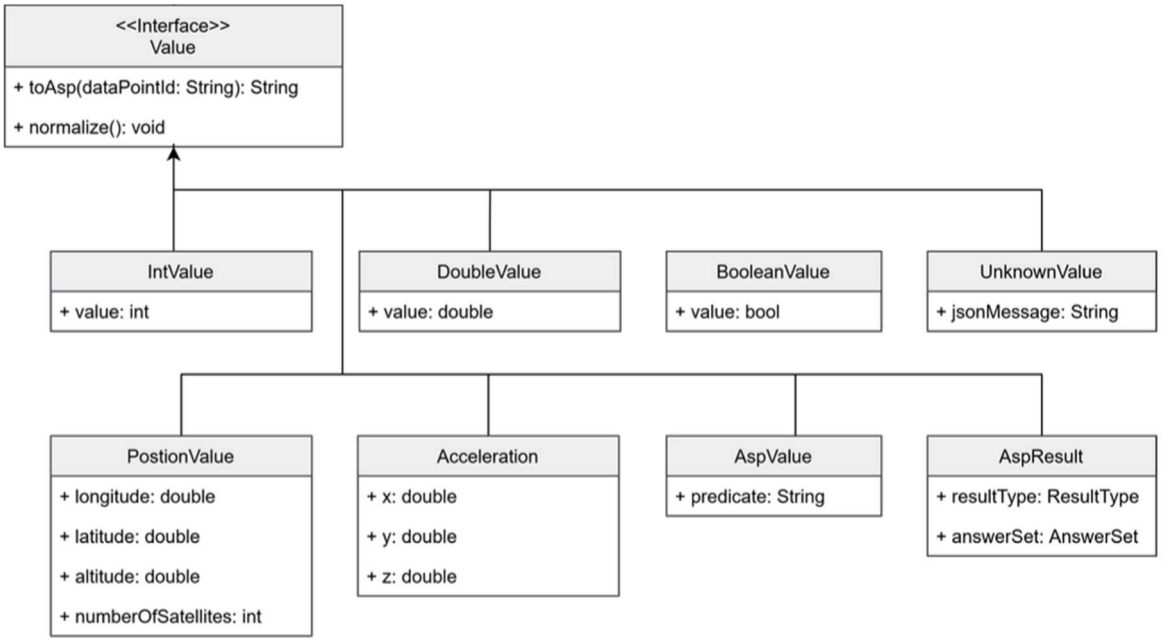

FIGURE 13: VALUE FORMAT STATING THE DISTINCT VALUE OBJECTS

The Value-interface assures that all specific Value-types are representable in answer set programming (ASP). We provide an AspStatementBuilder-class to ensure a uniform and syntactically correct format throughout all conversions. This builder creates statements in the format: < predicateName > ([< variable >,] ∗ [< variable >]). by taking all data types used in the specific value implementations as input. As numbers are limited to integers in Clingo, we provide a dedicated conversion of numbers. We scale all numbers to the same precision and represent them as a single integer. We relied on a precision of 3 digits (milli-fraction) of all numbers for our experiments (see Figure 14).

```
decimal value | scaled number
      1.3     |  1300
     -1.003   | -1003
```

FIGURE 14: CLINGO NUMBER SCALED TO FIXED PRECISION

We use the established REST standard to provide endpoints to configure the monitoring system. Many tools and libraries exist to conveniently interact with such API's. Machine-to-machine (m2m) communication is guaranteed as well, due to well defined JSON message bodies and unique resource identifiers (URI's). We provide create, read, update and delete (CRUD) functionality for Originators and OriginatorGroups. The springfox swagger2 library is used to create a Swagger2 configuration and thereby a web-frontend to interact with the endpoints. The appropriate functionality (see Figure 15) is automatically generated through the combination of the springfox library and Spring Boot on application startup.

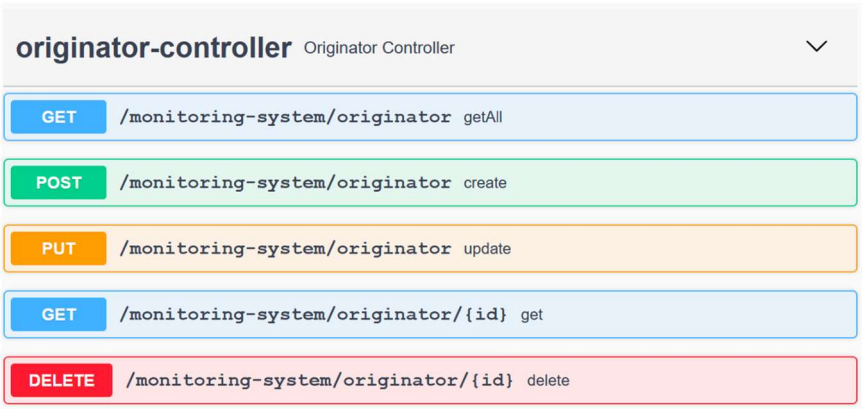

FIGURE 15: SWAGGER UI TO MANIPULATE ORIGINATORS

The input processing workflow handles data retrieval. It is triggered on receiving a data sample from the RabbitMQ and adds the received sample to a temporal storage, the data queue. All pre-processing steps, which are possible within the scope of a single sample, are executed throughout this workflow (see Figure 16).

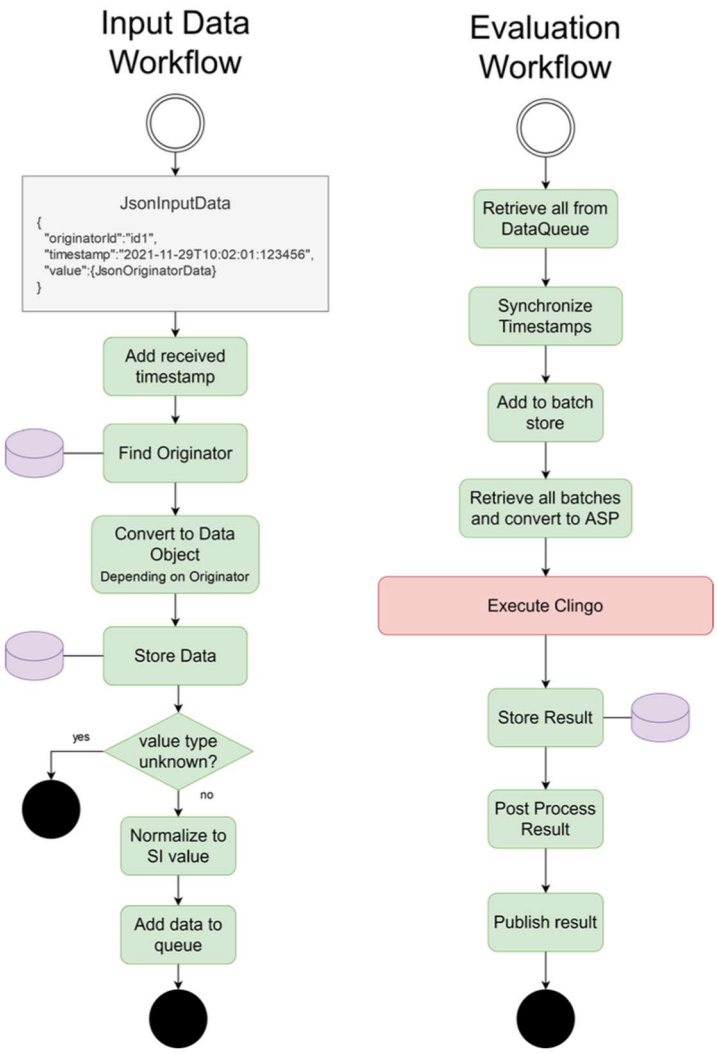

FIGURE 16: WORKFLOWS TO PROCESS INPUT DATA AND DATA EVALUATION

Data is received through the subscription to certain RabbitMQ topics. The utilized RabbitMQ-exchange is defined in the configuration file of the monitoring system and must exist in the RabbitMQ environment. We introduce the scheme [originator.<groupId>.<originatorId>.data] to identify and subscribe to originators.

On receiving data, we first set the received timestamp of a new DataPoint-object to the current one. This enables statistics and decisions on the timing behaviour of the measurements. We search the database for the specified originator and try to construct a valid DataPoint. This results in the following scenarios:

- If the originator does not exist, we create a new one with the received id, valueType set to UNKNOWN and groupId set to UNKNOWN as well. The provided value is converted to an UnknownValue, which holds the JSON-object as string representation. Therefore, we can handle unknown data sources as well and can correct them afterwards.
- On successfully retrieving the originator, we try to transform the provided value-object to a specialized value-format, which is defined by the originator.
  - If this fails (due to input data corruption) we again create an Unknown Value-object with the plain JSON data and assign the provided value to it.
  - Otherwise, we successfully transformed the received value.

We store the constructed DataPoint in all cases in the database. The workflow terminates at this point if the constructed value-object is of type unknown because we are not able to further operate on this unknown data sample.

We perform data normalization by converting data points within the same value-type to a common unit. In this process we eliminate differing units within data points of a common group. For instance, speed data received in mph and km/h is transformed to the common unit m/s. Different value types allow for distinct normalization techniques. To achieve this, we define the method normalize(unit) within the Value-interface. The parameter unit states the original unit of the data sample. The different normalization techniques are then implemented in the Value-sub-classes. This specifies the value-format and its possible normalization at a single point. Further examples are found in position data, which can differ in the points of origin, or boolean values, which need no normalization at all. The monitoring system transforms speed and length units to SI scales in the current implementation.

The workflow concludes with adding the sample to a data queue. The data queue contains all data points which are ready to be processed by the evaluation workflow. For data integrity purposes, we process only one sample per originator in each execution of the evaluation workflow. Internally, we use a map-like structure of originators and data samples to implement this behaviour. On data insertion, the sample of the corresponding originator is updated. We choose the sample with the latest timestamp to be stored in the data queue and remove the other.

Evaluation of the sensor data is done in two dimensions. First, we compare data points of different originators within the same time frame to validate their consistency. Second, we compare samples over time to be able to reason in the timing dimension as well. In doing so, we tackle the discrete nature of answer set programming. Data points with continuous timestamps are grouped to corresponding discrete time frames. This transformed data is then passed to a running Clingo instance, to evaluate the data. The evaluation result is then post-processed and published to RabbitMQ.

We split the integration of Clingo into two parts: One for preparing the environment and parsing the result of Clingo, and a second one for handling the external process itself.

Environment preparation: To conduct data evaluation with Clingo, we need additional input specifying constraints to which the data samples must conform. This input is retrieved in a file based manner. We call these files containing the logic constraints written in ASP, rule files. We provide an additional filter predicate, which filters the files within the directory. The filter is specified in the application properties and supports the glob-pattern. This pattern supports simplified regex expressions. The default value is set to ∗.lp to only include files containing ASP-code. The file ending .lp is suggested by the developers of Clingo and therefore inherited [7]. The files are then provided to Clingo via their file name. The previously created ASP-string representation of all da points is stored in a separate file named data_predicates.lp.internal in the same folder and provided to Clingo in the same manner. Clingo also supports input via the standard input of the console. We experimented with this feature to provide the data predicates to Clingo (and thereby omitting the storage procedure). But it turned out to be more consistent to provide all input via dedicated files. The file based approach is especially beneficial during the development of the ASP constraints. It supports isolated testing of the logic constraints on real input data without adding handcrafted predicates.

Clingo process handling: As Clingo is executed in an external process, a new instance of such a process must be started. Creating a new process for every execution cycle is not feasible as process creation is time-consuming. We therefore create a single process with a running shell-environment, which we keep executing until application shutdown. The Clingo command is executed within this shell. As the process never terminates, we rely on the input, output, and error stream for process-interaction, and therefore introduce the following schema.

We write the command to execute Clingo to the output stream of the process-object. Then, we read the input stream until we receive the last line of the expected output of the evaluation. This concludes a successful Clingo execution. The last line is specified by Clingo and states the "CPU Time" used for the execution. If the last line is received, we parse the received output. At the same time we evaluate a timeout. If the last line is not received within the specified timeout, the execution gets terminated. We log the error stream of the process and return a timeout error to the result parser.

Clingo result parsing: We retrieve the output lines of Clingo or an error and convert it into a ClingoResult-object. This object consists of the evaluated result type and a list of answer sets which contain the different solutions. The result type refers to one of the following, stating general information about the result:

- SATISFIABLE: The evaluation of the ASP statements leads to one or more answer sets which satisfy all constraints without any errors. The corresponding answer sets are stored in the result object.
- UNSATISFIABLE: The evaluation of the ASP statements was possible without error, but no answer set exists to satisfy all constraints.
- UNKNOWN: The input passed to Clingo contains an error, most likely syntax errors. No evaluation is possible. The log output contains information about the error.
- TIMEOUT: The execution did not terminate within the specified timeout. This is either due to too complex statements or an error within the Clingo command, indicating that some configuration parameters of the application are wrong. The log output states details in the second case.

- ERROR: A general, unexpected error occurred within the monitoring system on handling the execution.

Post processing: We transform the fully parsed result to be published to RabbitMQ. To do this, every instance of a monitoring system is an originator itself. Its id and groupId are defined within the configuration file. The specialised AspResultValue (which implements the Value-interface, see Figure 13) is the output format of the monitoring system. It contains the result type and one answer set of the ClingoResult-object. We include one answer set per result object to comply with our input format, which expects single valued data. We publish and store all retrieved answer sets. This leads to an output of at least one result and up to the defined maximum number of them. The RabbitMQ-exchange, to which the results are published, is defined in the configuration file and can differ from the input one. Furthermore, the self produced results are filtered out during input processing, as this would lead to an endless loop of self evaluations otherwise. This filtering is applied on the topic-level of the received data.

4.8.4 AI methods and techniques; training, verification, and validation plan; Explainability for AI components

We use logic programming, in particular answer set programming via Clingo, to establish a rule-based system which checks whether given properties are satisfied or validated. This constitutes artificial intelligence in the traditional meaning of the term, which is not based on machine learning. As such, concerns regarding the training of such methods do not apply. It further allows us to use traditional software testing approaches such as property-based testing or mutation testing to assess the quality of our software.

As prior art already established rules for visual sensemaking over time (Suchan, Bhatt and Varadarajan, “Out of Sight But Not Out of Mind”, Wotawa and Klampfl “Explaining Object Motion Using Answer Set Programming”), we focus on defining notions of *agreement* and *consensus* between multiple data sources within a single time frame. We plan to merge these approaches later.

Intuitively, a data source *agrees* with another with respect to given objects, if the object perceived by the latter is also perceived by the former modulo some transformations defined through a noise model. *Consensus* exists between the sensors when this agreement is mutual.

Under “normal” operating conditions, we would expect multiple sensors to reach consensus on the world they perceive under the above definition of the term. Thus, when they fail to do so, we can raise a warning and try to figure out why this is the case (e.g., using model-based diagnosis to distinguish between known failure modes). We further derive desirable properties for sensor fusion based on these notions of agreement and consensus. In particular, we constrain the sensor fusion to report objects that agree with the sensor readings, while also stating that readings that all sensors agree on should occur in the fusion.

4.8.5 Training, verification, and validation plan

As has been stated in the previous section, there is no training and we plan to apply traditional software testing techniques.

4.8.6 Explainability for AI components

Solvers for answer set programs produce-like solvers for other declarative programming problems, such as boolean satisfiability-models, that can themselves be checked. The explainability of an answer set program relies on the explainability of the models it creates. In any case, all the facts that are assumed to be true can be output by the solver.

An important benefit of answer set programming over other declarative problem solving approaches such as SAT, is that an answer set solver does not “hallucinate” facts unless it is explicitly allowed (by the programmer) to do so. As a consequence, the models it produces follow more or less directly from the observations made (in this case the values obtained from sensors).

The explainability of answer set programs can be hurt in two major ways. First, the models themselves can become big enough to require help for interpretation. Here, both divide-and-conquer patterns and search-based techniques may help ease the cognitive load. Second, rules may become arbitrarily complicated, in particular including arithmetic expressions. Such rules are not easy to debug and could well be improved by making facts about those computations explicit. However, establishing such facts usually incurs a performance penalty, and thus a delicate balance must be struck.

4.9 Milestones

Internal milestones include:

- First execution of the complete toolchain based on a scenario extracted within simulation. (est. PM 26/27)
- Successful transfer of developed algorithms to the vehicle setup. (est. PM 30)
- First execution of the complete toolchain based on a scenario extracted during real-world driving. (est. PM 33)
- Validation of the complete toolchain. (est. PM 36)

5 Demonstrator 2 (SCD1.2 Robo Taxi)

The system level design for demonstrator SCD1.2 focusses on **demonstrator platform B** (DP B) which is a closed loop simulation platform realized within CARLA and on **demonstration platform C** (DP C) which is the implementation of a full AD function in our demonstrator vehicle Ford Mondeo.

What are these two DPs?

DP B is a purely virtual test environment – a fully closed loop 3D simulation that allows to test different sensor models, verify perception and intelligence modules.

DP C represents the final demonstrator – a real robo-taxi (Ford Mondeo) that is capable to handle the three defined use cases (see **Fehler! Verweisquelle konnte nicht gefunden werden.**):

- **A:** Urban overtaking
- **B:** Crosswalk handling
- **C:** Customer Fetch-Up

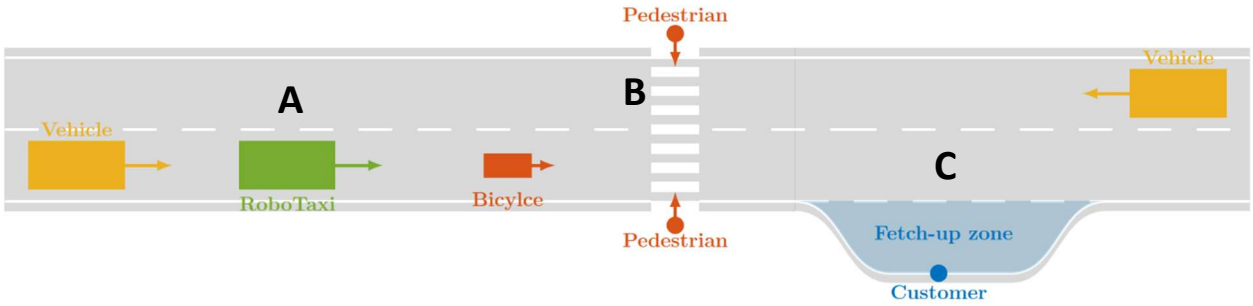

FIGURE 17: SCENARIOS FOR THE URBAN ROBO-TAXI USE CASE

5.1 High-level architecture

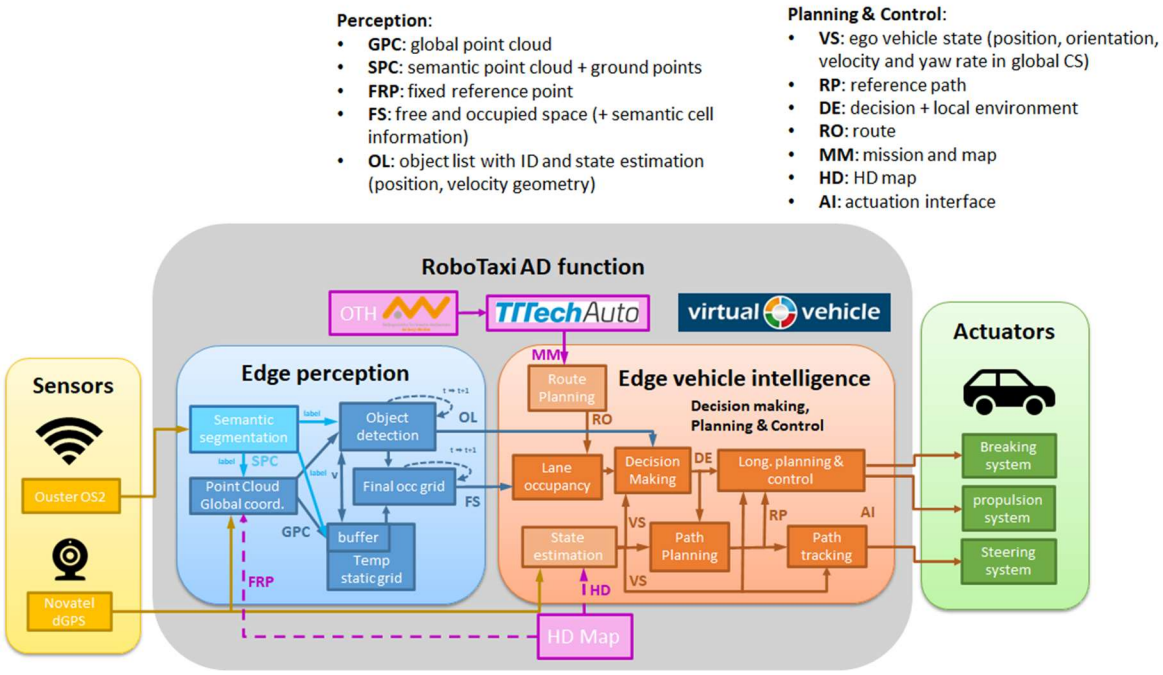

FIGURE 18: FUNCTIONAL OVERVIEW OF THE DEMONSTRATOR SCD1.2

Figure 18 shows the full architecture of Demonstrator SCD1.2 consisting of the main four edge modules:

- Sensor module (simulated in DP B)
- Perception module (Software)
- Intelligence module (Software)
- Actuation module (simulated in DP B)

5.2 Relevant requirements for system, subsystems, and components

The system level design of demonstrator platforms B and C explicitly considers qualitative and feature requirements defined in D1.1 directly in such a way, that features necessary to fulfil these requirements are integrated in the design (such as the existence of a module for semantic segmentation).

The following table points out which system level design components (such as occupancy grid, sensor data, etc.) are fundamental for assessing the requirements allowing a later validation.

7 requirements for edge perception (3 qualitative, 4 quantitative)	DP B has a **Carla semantic lidar** sensor type implemented that provides ground truth data in order to verify semantic and ground segmentation of the sensor data. The static occupancy grid of a test drive will be checked against cumulated lidar data (**HD map**) in DP C Object detection of dynamic objects will be verified using the available **Carla object list** (DP B) and **recordings** (DP C)
16 requirements for edge vehicle intelligence	The quantitative requirements of the intelligence module such as minimal distance to other objects will be verified

(7 qualitative, 9 quantitative)	using the **object lists** including bounding boxes and positions (refer to object list format) and the **lidar data** available as ground truth in DP B. In DP C **real lidar data** and the output of the perception module in combination with visual assessment (**recordings**) will be used to validate the intelligence requirements. Qualitative requirements such as decision-making features will be considered in the design and later validated such as not violating traffic rules will be validated via **recordings**.
3 requirements concerning communication and connectivity	**Architectural support for Cloud to Edge communication:** The design considerations reflect the requirements emerged from the data flow and amount of the robo-taxi use-case. The system design reflects the required services and interconnections northbound (cloud) as well as the southbound (controller) connections, their timing and execution constrains and on the HW side the interfaces useable for integration in WP5. **Mixed Criticality Functionality:** To ensure mandatory system requirements like freedom of interference between safety critical and non-critical performance applications the ECU design with its SW stack is setup to support mixed criticality per design. This is achieved by hardware as well as software measures (e.g., in case of communication channels via TSN, separated media, task execution planning and global scheduling). The remaining requirement is mainly handled in SC5, WP4 activities.

There are no WP3 tasks in SC1.

5.3 Design flow/methodology

For demonstrator SCD1.2 some building blocks are developed at the partner level internally whereas for collaboration exchange formats are organized.

System level design between partners:

To define the interfaces of the demonstrator components a two days' workshop was organized by the SCD1.2 lead (VIF) in Linz in September 2022 where all partners of SCD1.2 and SCD1.3 participated. This workshop helped a lot in the definition of communication protocols also between different demonstrators.

In addition, regular meetups with partners of SC1 are organized by the SC1 lead (VIF) to exchange best practice in the use of the open-source Carla simulation engine which is relevant in all three demonstrators.

System design on partner level:

VIF: the edge components of SCD1.2 (the car, the software and the simulation environment) are developed within VIF. On demand system design meetups take place at least once a week using flow chart diagram tools such as Miro boards, software revision systems such as git and digital communication tools such as Teams. The system design and development process follows a very agile style working in a core team of three people in a very flexible manner on both platforms DP B and DP C in parallel. This allows to consider hardware specifications of demo vehicle DP C (such as available Lidar sensor types or computing power) in the simulation development of DP B and to align interfaces of different formats (e.g., sensor raw data, …)

5.4 Relevant standards, norms, and ethical aspects

The most relevant standards in the development of SCD1.2 for DP B are the **ASAM openDRIVE** [5] and **ASAM openSCENARIO** [4] standards which are file formats defining a road map and a road scenario that are the baseline to generate a digital twin of real-life driving scenario.

The recent **EU regulation 2022/1426** [8] regarding "uniform procedures and technical specifications for the type-approval of the automated driving system (ADS) of fully automated vehicles" acts as guideline to consider which data will be used in future and will be collected and undermines the importance of the virtualization of real driving scenarios as they will be key for admission and registration of AD functions in the next years.

Problems with used standards:

Severe problems were encountered when realizing that the new ASAM OpenSCENARIO 2.x standard is not implemented in the used open-source 3D simulation engine CARLA.

The older ASAM openSCENARIO 1.x standard is only partly implemented in CARLA hindering description and creation of scenarios with the full functionality foreseen in the standard. This lack of proper implementation of open standards in open-source simulation engines slows down the development with driving scenarios.

To be mentioned is ESMINI – a basic openSCENARIO player with ASAM openSCENARIO 2.x support.

Benefits of used standards:

Using the same road map standards enables demonstrators SCD1.2 and SCD1.3 to collaborate in such a way that the same part of the city Amberg in Bavaria is simulated.

In addition, SC8 provided useful information regarding standards about processes how to develop trustworthy AI which are considered and partly implemented in the development of AI algorithms within VIF.

5.5 Overview of components/subsystems/systems in demonstrator

Demonstrator SCD1.2 consists in total of three demonstrator platforms, DP A, DP B and DP C (see also D1.1) where DP A corresponds with the software modules PERCY (perception) and ROMPAC (intelligence). In addition, there is a communication module that allows to exchange data with the cloud or with other Robo Taxis. The following list contains all relevant demonstrator platforms and modules:

- Demonstrator platform B: Carla simulation framework
- Demonstrator platform C: Ford Mondeo with RTMaps framework

- Perception module: PERCY handling semantic segmentation, occupancy grid and object detection
- Intelligence module: ROMPAC handling state estimation decision making, planning and control
- Communication module: Hardware box, 5G modem, communication protocol for cloud edge communication

5.6 Demonstration platform B: Carla simulation framework

The demonstration platform B is a virtual closed loop simulation that allows to test AD functionality and in particular perception and intelligence modules in a virtual environment. It is particularly useful for verification of AD functionalities as it provides ground truth data in the fields of object detection, semantic segmentation and decision making.

The used framework is based on the CARLA simulator – an open-source simulator for autonomous driving (AD) research.

Figure 19 gives an overview of the whole demonstration platform B, depicting the data flow and in particular the output files which are relevant for the evaluation.

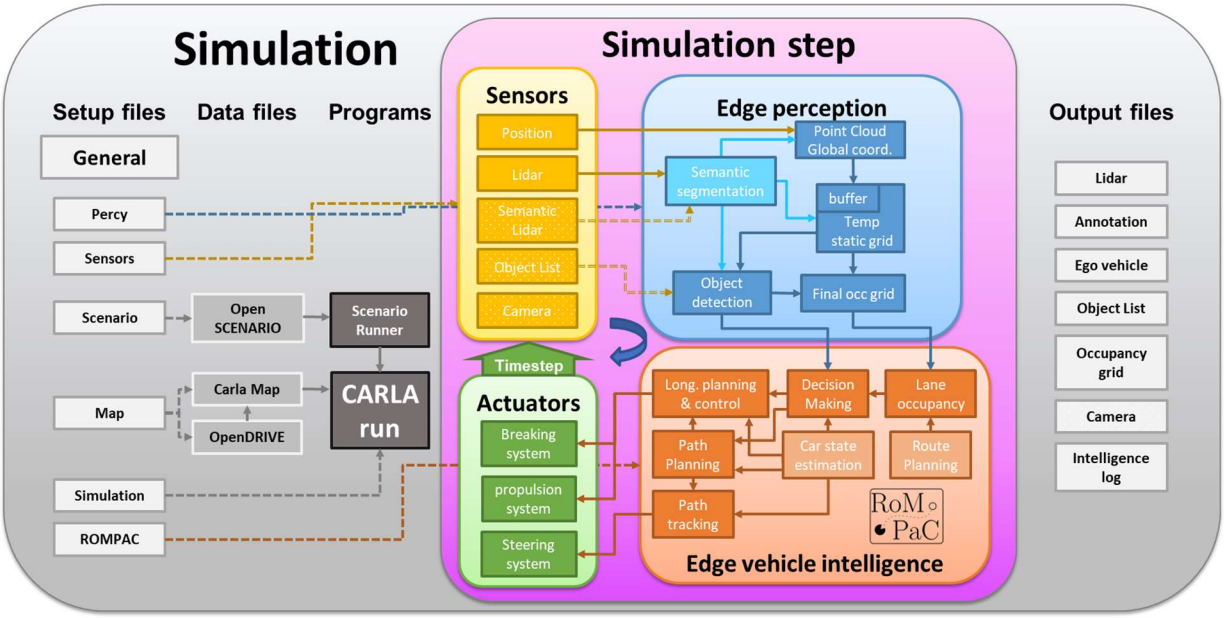

FIGURE 19: SCHEMATICS OF DEMONSTRATION PLATFORM B

DP B consists of the following parts:

- **Configuration files** (json file format) for setup of different program parts and modules
- **Data files** corresponding to the openDRIVE and openSCENARIO standards and the 3D map
- the **ScenarioRunner** and **Carla** interpreting the data files and running the scenario in the defined map
- The **simulation step** comprising the four main modules (sensor, perception, intelligence and actuation) of the demonstrator
- **Output files** of raw sensor data and perception and intelligence output for validation

5.6.1 Software concept description

The main challenging part in this demonstrator platform is to define scenarios and make them tuneable with an easy interface.

To do so, a pipeline using the python library *scenariogenerator.py* was established that creates partly parametrizable openSCENARIO files, which can be tuned via the configuration file "*Scenario*".

5.6.2 Computation platform and toolchain

To run the Carla simulation a conventional desktop computer with build in GPU and sufficient RAM was used.

5.7 Demonstration platform C: Ford Mondeo and RTMaps framework

The demonstration platform C is a real-world demonstrator that allows to apply the AD functionality via drive-by-wire in a real urban environment.

The demonstrator vehicle is a representative hybrid-electric passenger car (Ford Mondeo), equipped with steer/break-by-wire systems, an environmental perception sensor setup and dedicated computational platforms to perform highly automated driving, owned by VIF. The sensor setup consists of Lidar sensors, cameras as well as a dGPS and IMUs. It is intended to demonstrate at least one scenario of the defined scenario setup on a test track or at the TU Graz-Campus in mixed-reality.

5.7.1 Hardware concept description

The hardware concept builds on an existing demonstrator owned by VIF which will be adopted with an appropriate LIDAR sensor (presumably Ouster OS2) to match the desired needs for the defined scenarios.

In addition a communication demonstrator platform will be provided by TTTAuto. The HW platform provides a safety host based on a Renesas RH850P/1H-C (ASIL D MCU with lockstep cores) and two performance host based on Renesas R-Car H3 (ASIL B SoC with 4x Cortex A57, 4x Cortex A53, 1x Cortex R7, 1x IMP-X5, 1x IMG GX6650 GPU). In addition for the SC demonstrator purposes the platform provides extensive connectivity interfaces (e.g. CAN-FD, Ethernet, digital & analog IOs, etc.) to integrate it in modern cars. For cloud connectivity the platform is extended with an external peripheral 4G/5G modem sufficient for providing sufficient bandwidth for this use-case.

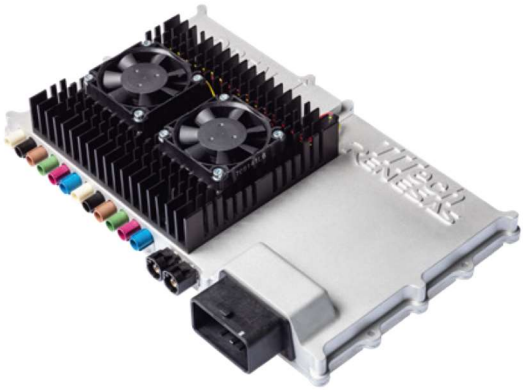

FIGURE 20: COMMUNICATION & COMPUTING PLATFORM BY TTTECH AUTO AG

In addition, a communication interface will be provided by TTTAuto.

The car uses RTMaps as an internal communication platform between sensors, the different software parts, the drive-by-wire interface and the communication module.

5.8 Perception module: Percy

The challenge within the perception module is to process approximately a million LIDAR points within a second and to robustly describe the car environment with respect to objects and obstacles.

To achieve this task a probabilistic 2D occupancy grid is used that ignores ground (task of ground segmentation) and distinguishes between static and dynamic environmental objects.

Furthermore, an object list containing 3D bounding boxes and inferred velocities is provided to help the intelligence to make decisions taking other traffic participants into account.

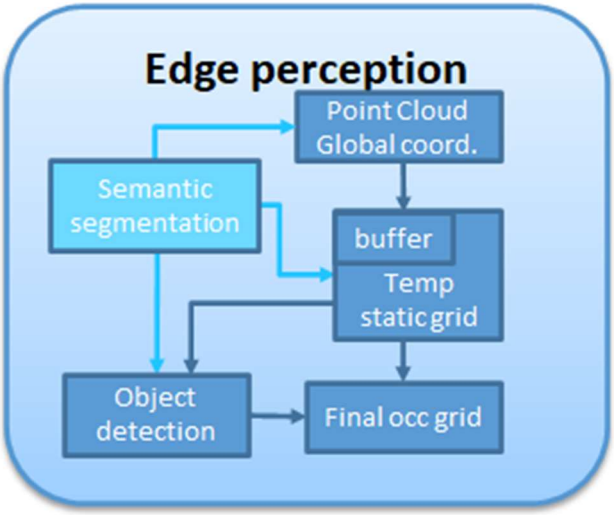

FIGURE 21: PERCEPTION MODULE ILLUSTRATING MAIN BUILDING BLOCKS AND DATA FLOW

Figure 21 depicts the main components of the perception module. The input - the raw lidar data (x, y, z, reflectivity) and the position of the car are processed to create the output: a probabilistic occupancy grid and an object list.

The submodules are:

- *Point Cloud Global coord.*: transforms the relative point cloud data into global UTM coordinates using the cars position, which might have deviations to bad GPS signals
- *Buffer, Temp static grid*: stores the data of previous time steps to make velocity estimation, object detection and semantic segmentation more robust
- *Semantic segmentation*: labels each lidar point using a trained deep neural network model
- *Object detection*: estimates size, translation, rotation and velocity of detected objects
- *Final occ grid*: Merges static grid and objects (including prediction) with data from the communication module (perception data from the cloud or other cars)

5.8.1 Software concept description

The *semantic segmentation* submodule uses a convolutional neural network architecture to label each lidar point. This is done by switching from Euclidean coordinates to a 2D representation using a

spherical projection. Several input layers like depth, reflection, and other information extracted from the raw data are used to label each point. Labels include ground, static, objects etc.

The *Point Cloud Global coord.* submodule transforms the relevant relative lidar points to a global static coordinate system.

The *Buffer, Temp static grid* submodule uses static points to build up a 2D static occupancy grid using uncertainty quantification.

The *object detection* submodule uses object points and also previous information to infer on type, size, velocity, translation and rotation of recognized objects from a BEV perspective.

Finally, the *Final occ grid* submodule fuses object list, static occupancy grid and also information from the communication module to provide a real-time occupancy grid for the intelligence module.

5.8.2 Computation platform and toolchain

The perception module is implemented 100% in Julia allowing to work directly on the GPU and to create convolutional neural networks from scratch which enables fast prototyping and high-performance in a combined way.

Interfaces to the demonstrator platforms (Carla and RTMaps) and to the intelligence module are created using Python bridges which were already tested and which work straight forward.

5.8.3 Classical (non-AI) software functions

The coordinate transformation and filling of occupancy grid is done with classical algorithms.

5.8.4 AI methods and techniques

The core of the semantic segmentation builds a UNET that is well suited to semantic segmentation tasks.

5.8.5 Training, verification, and validation plan

The UNET is trained with data from the NuScene database. In addition, virtual scenarios realized with DP B will be used to train the network with the appropriate sensor configuration that matches the final configuration of DP C. Carla scenarios can also be used for verification and validation of the semantic segmentation.

Validation of the UNET in DP C will work with a previously recorded point cloud HD map and manually by relabelling the classified lidar images.

5.8.6 Explainability for AI components

In order to characterize explainability of the used AI algorithms saliency maps will be investigated.

5.9 Intelligence module: RoMPaC

The task of the intelligence module is to process the perceived environment information to decide, plan and execute the behaviour of the Robo Taxi. To successfully accomplish the use case scenarios, the intelligence module classifies the scene based on road data and an object list. With this, it is able to decide whether it is appropriate to overtake, necessary to stop or time to fetch a customer. Based

on this decision, a safe and comfortable trajectory is planned, that defines the desired path of the Robo Taxi and corresponding velocities at each point in time. Using state-of-the-art control algorithms, the actuation signals (steering, propulsion and braking) are set such that the Robo Taxi follows the planned trajectory with sufficient accuracy. The control feedback is provided by the car state estimation which estimates the vehicle position, velocity and orientation based on several sensors.

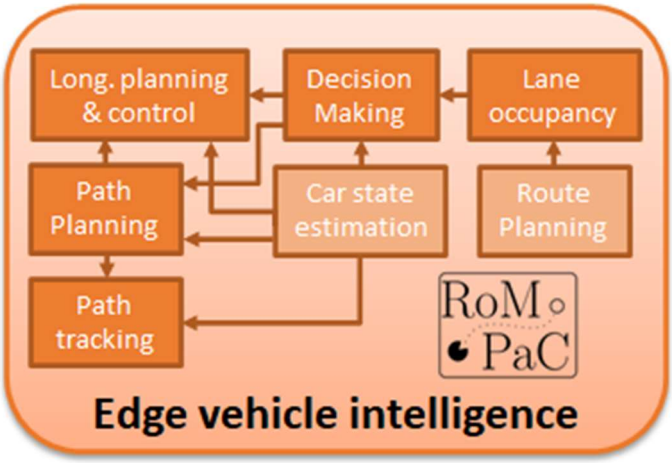

FIGURE 22: INTELLIGENCE MODULE ILLUSTRATING MAIN BUILDING BLOCKS AND DATA FLOW

Figure 22 depicts the structure of the intelligence module. It can be summarized as a three steps procedure: the *decision making*, *trajectory planning* (Longitudinal and path planning) and the *control* part.

5.9.1 Software concept description

Decision making and Lane occupancy

The *decision making* submodule processes the observed environment information to decide the upcoming maneuver. The whole scene understanding procedure is based on the road data and object lists. For the object list, the objects perceived by Percy are supplemented by fictive objects that are created to incorporate additional information into the decision making process. For example to consider blind spots in a passive driving manner, ghost objects are placed at these spots – the Robo Taxi is thus primed to avoid collisions with any obstacle coming from this direction. The placement of these additional objects is done in the Lane occupancy submodule.

The decision making is implemented as a modular state machine. Each maneuver (e.g. follow the lane, overtake, stop, …) is implemented as a state that can be add or removed to the state machine. The output of the decision submodule is a maneuver containing the following information:

- type of the maneuver
- start and length of the maneuver (in m path length)
- target velocity
- priority (comfort, emergency, …)

Trajectory planning

According to the upcoming maneuver that has been set by the decision making submodule, a trajectory is planned to execute it. The trajectory consists of a path that is generated by the lateral planning submodule, and a velocity profile that is generated by the longitudinal planning submodule.

The path is represented by Bézier curves and can be parameterized regarding the order of continuity and magnitude of curvature. This way, the path can be adapted to fit best to the chosen controller and to modify the comfort and safety properties of the resulting behaviour of the Robo Taxi.

The velocity profile is defined in form of polynomials starting at the current velocity of the Robo Taxi and ending at the target velocity. They can be parameterized to satisfy constraints regarding the maximum acceleration, deceleration and jerk.

Longitudinal and lateral control

To steer the vehicle along the planned trajectory, two separate controllers are used: a controller for the lateral movement (the Robo Taxi following the planned path) and a controller for the longitudinal movement (the Robo Taxi following the planned velocity profile).

For the lateral controller, multiple state-of-the art path tracking controllers are implemented that can be exchanged easily. These controllers show different advantages and disadvantages especially in presence of disturbances. While some controllers are more sensitive to actuation delays, others are more keen to persistent control errors. Therefore, it is left to decide for the final controller during the evaluation.

The longitudinal controller is implemented as a PID control. An important component here is the pedal-to-torque map that defines the relation between positions of the gas and brake pedal and the resulting propulsion or braking torque. This relation is stored as dedicated static maps for each demonstrator.

5.9.2 Computation platform and toolchain

The vehicles edge intelligence is implemented in Python. It can run on any Linux or Windows computer and does not require specific hardware or a GPU. The three submodules (decision making, planning and control) can be either executed sequentially in a single Python process or each module in its own Python process. For the latter, RTMaps is used to start the Python processes, synchronize the submodules and transfer the data in-between them. The advantage of each submodule running in its own process is that they can be executed parallel. This way, for example, the control submodule can be executed at a high frequency and is not blocked by computations of the planning submodule.

5.9.3 Classical (non-AI) software functions

The whole edge intelligence is implemented with classical (non-AI) functions. With these non-data-driven approaches, it is easy to adapt the functions to the dedicated use case and design domain. The decision making is implemented with a classical state of the art machine. The path planning and velocity planning submodules compute the trajectory analytically based on heuristics for the geometric conditions and system dynamics. The trajectory tracking controllers are state of the art algorithms with a feedback law in closed form.

5.10 Communication module

The communication module will allow to exchange of perception and intelligence data with other cars and customer requests and routing data with the cloud represented by demonstrator SCD1.3. It will

especially be used to connect demonstrators SCD1.2 and SCD1.3 exchanging customer status (edge) and optimized routes to the next customers (cloud).

The communication module consists of a software submodule that uses aligned interfaces for route and customer request exchanges (see Figure 23) and a hardware submodule that allows fast communication satisfying high demands concerning data transfer, latency, bandwidth, and availability.

5.10.1 Software concept description

The cloud intelligence provides routes to customers and to the customers destinations while the edge intelligence steers the Robo Taxi to follow these routes. An exemplary scenario for executing a customer request (picking up the customer and bringing him to his destination) is shown in Figure 23.

The procedure in this scenario is as follows:

- the passenger requests a Taxi from his/her location to a goal location
- the cloud intelligence computes the corresponding routes and sends them to the edge intelligence
- the Robo Taxi drives to the customer and picks him/her up
- the cloud intelligence again computes the corresponding route, the edge intelligence executes it and the passenger can leave the Taxi at the goal location

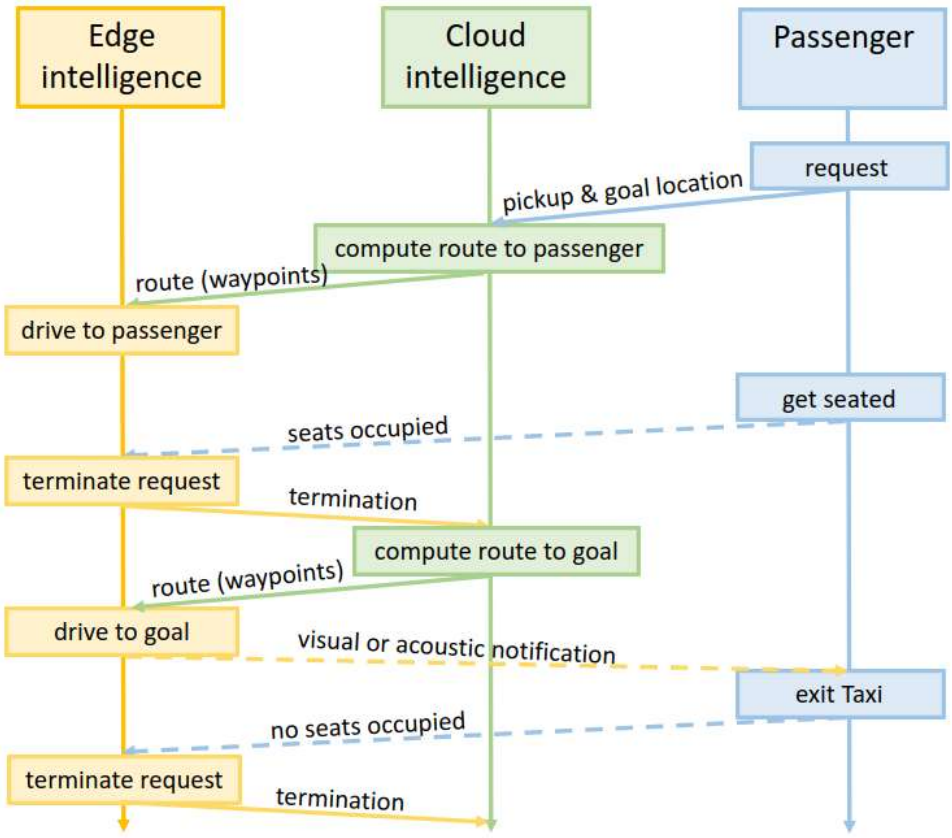

FIGURE 23: SKETCH OF THE COMMUNICATION PROTOCOL FOR A STANDARD CUSTOMER TRANSPORTATION

To implement the interaction between edge- and cloud intelligence, these modules send messages to each other at a constant frequency. An example message from edge intelligence to cloud intelligence is given in Table 3.

TABLE 3: MESSAGE STRUCTURE EDGE TO CLOUD

Message part	Explanation
`"id": 66,`	unique number to identify the request
`"taxiID": 0,`	unique number to identify the Robo Taxi

`"pose": [`
`  39.128,`
`  1.749`
`],`
	the position of the Robo Taxi in UTM coordinates

`"passengers": 0,`	number of seats occupied
`"problem": false`	has a problem occurred? e.g. if the route is blocked

An example message from cloud intelligence to edge intelligence is given in Table 4.

TABLE 4: MESSAGE STRUCTURE CLOUD TO EDGE

Message part	Explanation
`"id": 66,`	unique number to identify the request
`"taxiID": 0,`	unique number to identify the Robo Taxi

`"route": [`
`  [40.21, 2.05],`
`  [44.53, 2.04],`
`  …`
`]`
	list of waypoints (UTM coordinates) describing the route from taxi to a target location

`"cancel": false`	discard the request (e.g. plans have changed)

5.10.2 Computation platform and toolchain

In the in-vehicle edge intelligence module, the messages are sent and received with a dedicated communication platform provided by TTTAuto. The cloud intelligence provides via a message-based protocol the necessary requests and routes from customer requests as depicted in Figure 23. The communication module handles the bi-directional data flow which is then further processed in the edge vehicle intelligence. The connectivity between the two in-car modules in Figure 24 is provided by an ethernet backbone. The bi-directional cloud-edge communication is designed utilizing a service-oriented architecture principle that manages the following to services:

- Data exchange (routing information / robo-taxi related data exchange

- Transparent data handling (comm. abstraction) through the middleware stack
- Communication management northbound (cloud) and southbound (vehicle controller)
- Execution and process health monitoring to guarantee deterministic system behaviour

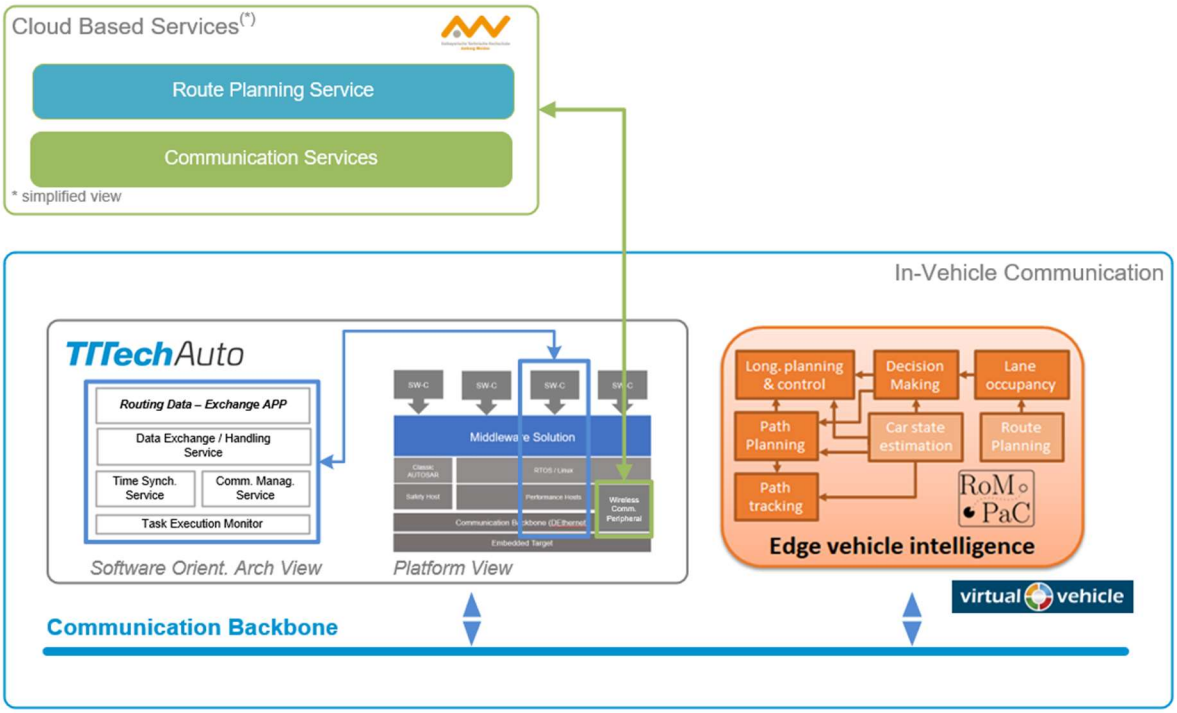

FIGURE 24: SYSTEM DESIGN W.R.T. COMMUNICATION BETWEEN VEHICLE INTELLIGENCE AND CLOUD INTELLIGENT SERVICES

The communication module provides the following features and automotive specific domain standards for the development in the project. The framework is able to handle communication between applications (platform view in Figure 24), between standard automotive ECUs and vehicle components such as sensors or actuators. The framework is compatible with AUTOSAR Classic RTE, AUTOSAR Adaptive on POSIX-based hosts and supports the OMG DDS standard. High performance IPC is realized via shared memory access on the service-oriented architecture. The SW stack provides zero-copy implementation, data transparency, data allocation, and a complete memory abstraction for the application development across all software abstraction layers.

5.11 Milestones

Milestones will be:

- Realization of 3 urban environment scenarios on a street in Amberg in Carla (PM 25)
- First closed loop simulation with these three scenarios (PM 26)
- First benchmarks of semantic segmentation component (PM 27)
- First benchmarks of object detection component (PM 30)
- Validation of DP B (PM 28)
- Run time test of perception module in DP C (PM 26)
- State machine and decision test of intelligence module in DP C based on perception (PM 28)
- First closed loop run in DP C (PM 31)
- Validation of DP C (PM 36)

6 Demonstrator 3 (SCD1.3 Virtual city routing)

As described in deliverable D1.1 [1] the Virtual City Routing demonstrator shows that efficient routes in urban areas for electric vehicle fleets can be determined while considering dynamic changes in the environment.

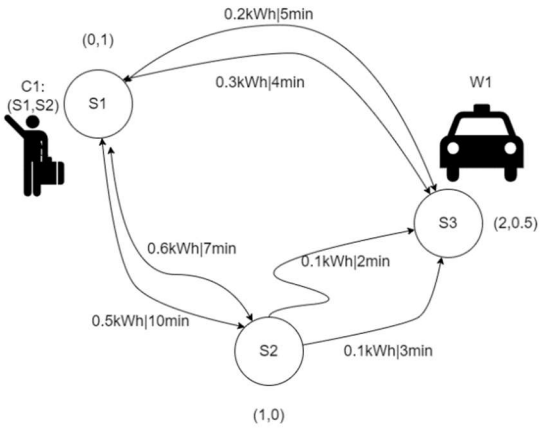

FIGURE 25: OVERVIEW ROUTING

Figure 25 shows a scenario where a customer wants to be transported in a network of stations. A vehicle is able to transport him via one of the possible routes. The goal of this demonstrator is to optimize the routes of vehicle fleets with pooling.

In addition, traffic simulations of roadside unit (RSU) vehicle-related data gathering are implemented together with TraCI.

In the following sections the system level design of the demonstrator is described in detail. This includes a high-level architecture 6.1, the relevant requirements for this deliverable 6.2, and the building blocks.

6.1 High-level architecture

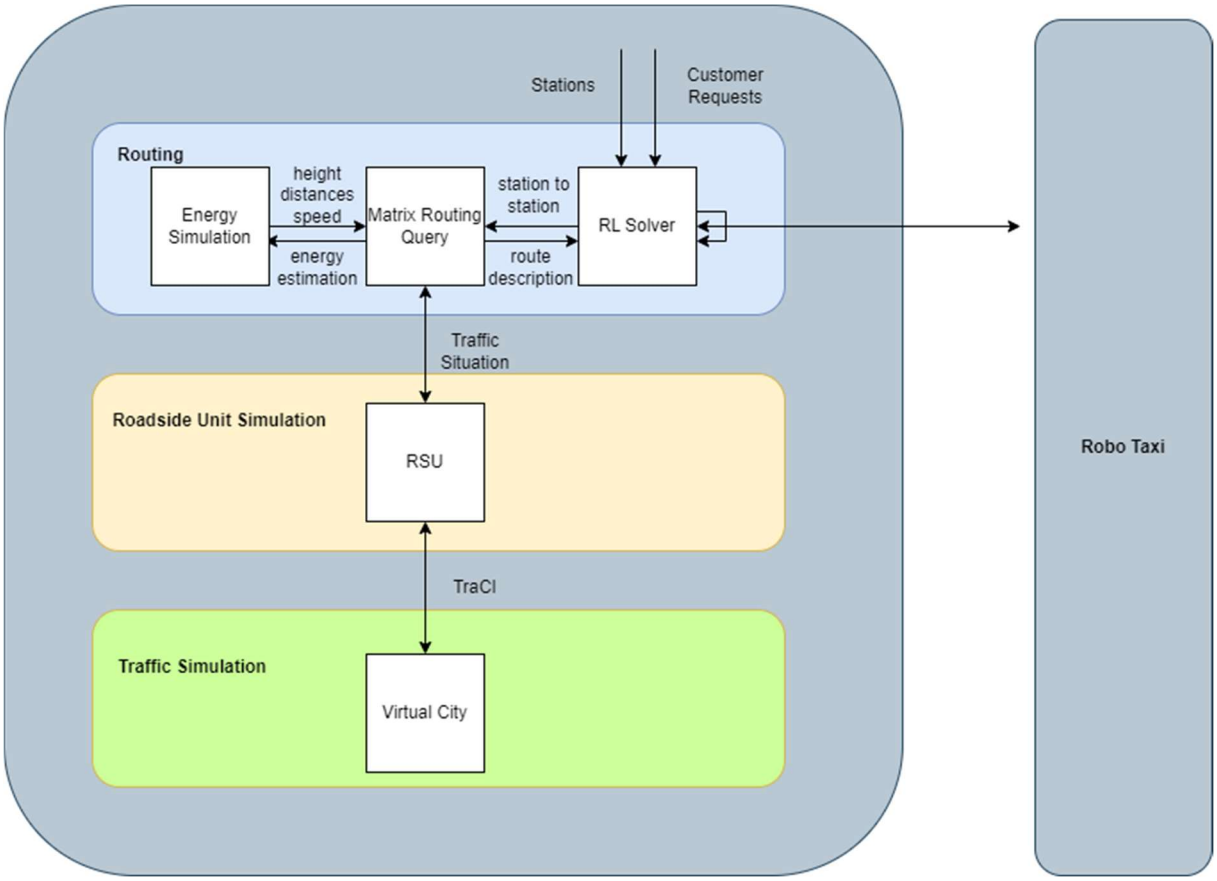

FIGURE 26: HIGH-LEVEL ARCHITECTURE VIRTUAL CITY ROUTING

The full architecture of SCD1.3 can be viewed in Figure 26. It contains:

- A Routing module
- A roadside unit simulation module
- Traffic simulation module

6.2 Relevant requirements for system, subsystems, and components

The following mentioned requirements are from deliverable D1.1 and the section for SCD 1.3. They have the prefix AI4CSM_WP1_SCD1.3.

The requirements relevant for this deliverable are those handling the data sources which is number 14, validation environment (SUMO), which are number 15, 16, and 17.

Requirement number 13, which deals with the interface to the Virtual Vehicle Test Car will be part of this deliverable.

The requirement number 12, which is the interface to the EQC test vehicle from SC2 will not be considered any more, since we want to focus on the interface to the Virtual Vehicle Test vehicle.

The requirements 1-11 will be dealt with in WP4.

The implementation of the requirements will be shown via recordings.

From a derivable, some data transfer requirements of key performance indicators (KPIs) for three different traffic communication scenarios: normal, faulty information, and overloaded information are defined:

For normal scenario:

- Throughput (the amount of data that can be transferred per unit time): Minimum throughput of 100 Mbps (megabits per second)
- Latency (the time it takes for data to travel from one point to another): Maximum latency of 50 ms (milliseconds)
- Packet loss (the percentage of data packets that are lost during transmission): Maximum packet loss rate of 1%

For faulty Information scenario:

- Error rate (the percentage of data packets that contain errors): Maximum error rate of 5%.
- Recovery time (the time it takes to recover from a faulty transmission): Maximum recovery time of 1 second.
- Redundancy (the amount of backup data available in case of faulty transmission): Minimum redundancy of 2x (twice the amount of data being transmitted).

For overloaded Information scenario:

- Congestion (the percentage of data packets that are delayed due to network congestion): Requirement: Maximum congestion rate of 10%.
- Fairness (the amount of data transferred to each user in a fair manner): Minimum fairness of 80%.
- Bandwidth (the amount of data that can be transferred simultaneously): Minimum bandwidth of 1 Gbps (gigabits per second).

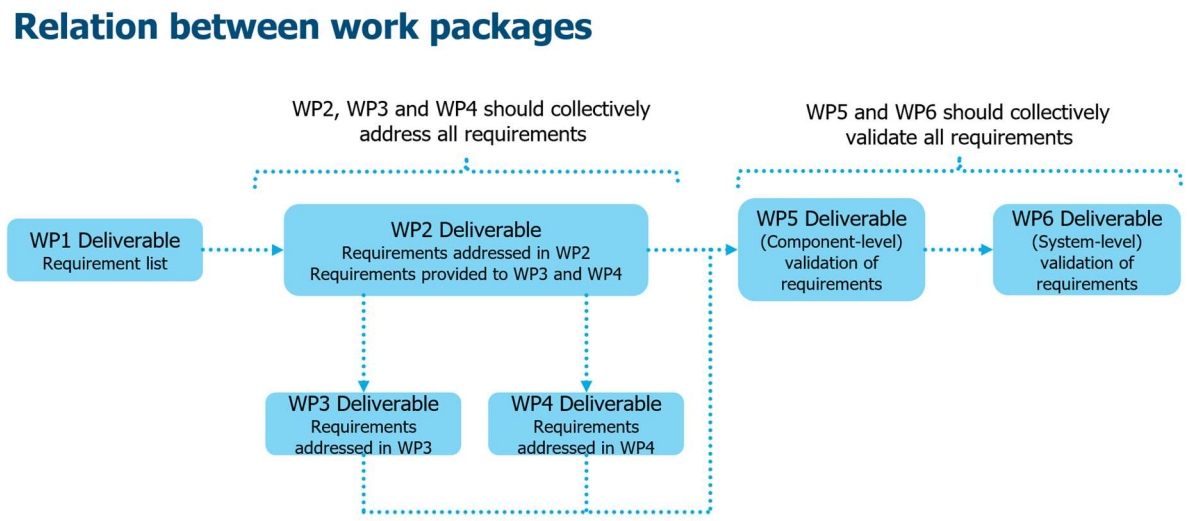

FIGURE 27: MAP IN WHICH WORKPACKAGES REQUIREMENTS ARE ADDRESSED AND VALIDATED

6.3 Design flow/methodology

For demonstrator SCD1.3 most of the components were developed on partner level or reused from predecessor projects

Exchange between partners:

- To define interfaces between SCD1.2 and SCD1.3 OTH and VGTU took part in a two days' workshop organized by VIF.
- The project partners participated regularly in the SC1 Meetings.
- For synchronization between VGTU and OTH a call took place to figure out the collaboration on the demonstrator.
- Exchange with AVL took place to clarify the usage of the energy estimation module.

System Design on partner level:

- **OTH:** The simulation the city of Amberg and the routing service are developed within OTH. Weekly meeting was carried out together with the SC2 portion of the OTH mainly offline with the help of whiteboard and sometimes online - via Big Blue Button - with help of software revision tools such as git and diagrams via draw io.
- **VGTU:** The simulation of the cities of Vilnius and Amberg and the routing using SUMO are developed within VGTU. One time a week meeting in person was carried out with the VGTU team with the help of whiteboard and smart board.

6.4 Relevant standards, norms, and ethical aspects

To exchange the SUMO street network the ASAM OpenDrive standard was utilized.

6.5 Overview of components/subsystems/systems in demonstrator

The virtual city routing demonstrator overview can be seen in the Figure 26 and contains three main building blocks – a routing module, a roadside unit simulation module and a traffic simulation module. The *routing module* has connections to the robo-taxi and the roadside unit module has links to the routing module. The connection to the robo-taxi is described in 5.10.

The routing module is built up from the building blocks depicted in Figure 28. The simulation module is a SUMO Simulation described in Section 6.6.1.

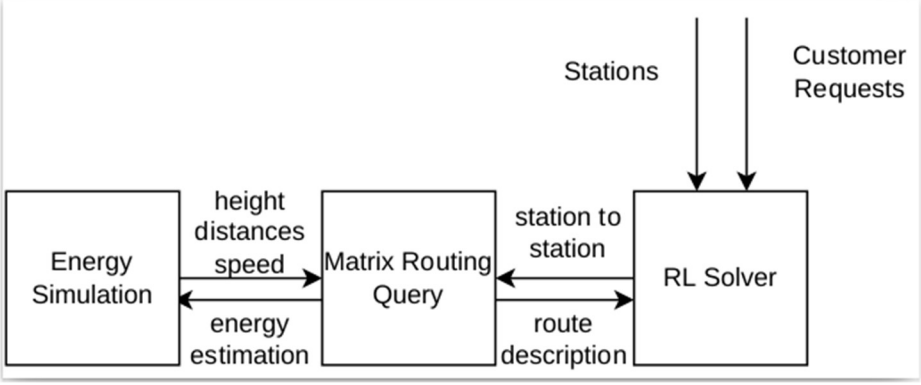

FIGURE 28: OVERVIEW ROUTING SYSTEM

The overall flow is that from outside stations and customer requests are given, the RL solver utilizes the matrix routing query to figure out the best combinations of routes and the energy simulation evaluates the energy consumption to be able to optimize for energy consumption.

The overall traffic at specific regions is evaluated by obtaining relevant information from the vehicles by the roadside units (RSUs). This information can be transmitted to the processing unit to make Machine learning calculations and decisions.

6.6 Traffic simulation

6.6.1 Software concept description

To build a digital twin of the city of Amberg the simulation environment SUMO [9] (Simulation of Urban MObility) was used.

The authors describe it as an open source, highly portable, microscopic and continuous multi-modal traffic simulation package designed to handle large networks.

To build a simulation of the city of Amberg the following steps were performed:

1. Read in Map from OpenStreetMap

   To get a basic map the 3-click scenario generation tool [10] of SUMO was used to generate a street network out of an OpenStreetMap file of the city of Amberg.

2. Post processing of map data

   After the initial map was generated it was fine-tuned with the following steps

   - Removal of isolated edges
   - Check of turning lanes
   - Removal of illegal u-turns

   The result of the imported map can be seen in Figure 32.

3. Generation of traffic load from traffic count over 24 hours
4. To generate a realistic scenario, it was enriched with traffic participants. Vehicles were generated according to traffic data from the “Stabsstelle für Mobilität und Verkehr der Stadt Amberg” [11] from 1997 since no more recent data was available, but have received conformation to get up to date data in future. Open data availability in the sector of traffic movements turned out to be a problem for the project, since the available real-time sources are very costly.
5. Placement of Pickup-Stations throughout the city. See figure Figure 29.
6. The stations that the customers can go to be picked up were evaluated by considering the walking distance of 500 m which takes about 10 minutes and the number of houses in the surrounding area. Areas without houses didn’t get a station assigned. See for the process of finding stations and Figure 30 for the result of the applied k-means clustering algorithm to assign the stations. The result of 210 assigned stations in the city area of Amberg can be viewed in Figure 31.

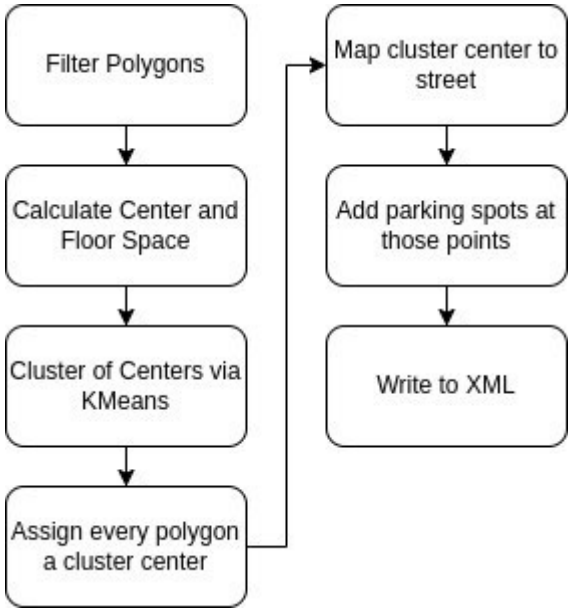

FIGURE 29: PROCESS OF FINDING STATION LOCATIONS

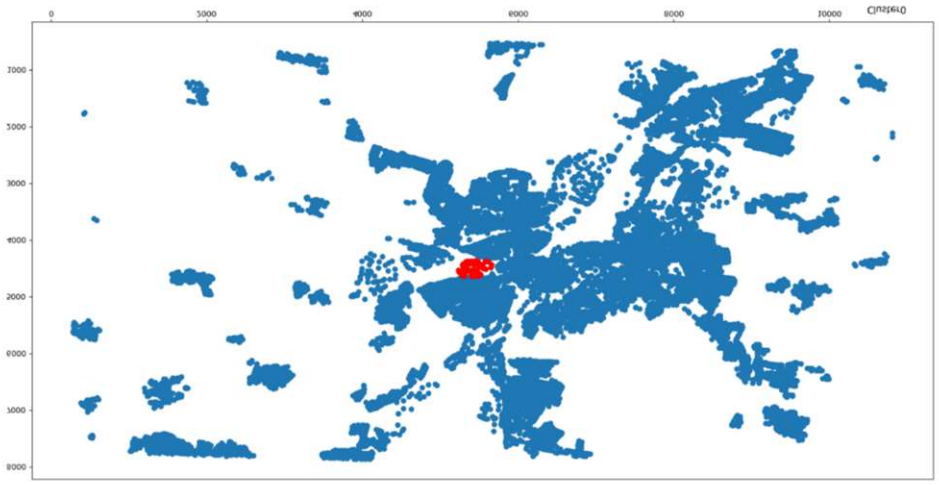

FIGURE 30: K-MEANS ALGORITHM TO FIND STATION PLACEMENT

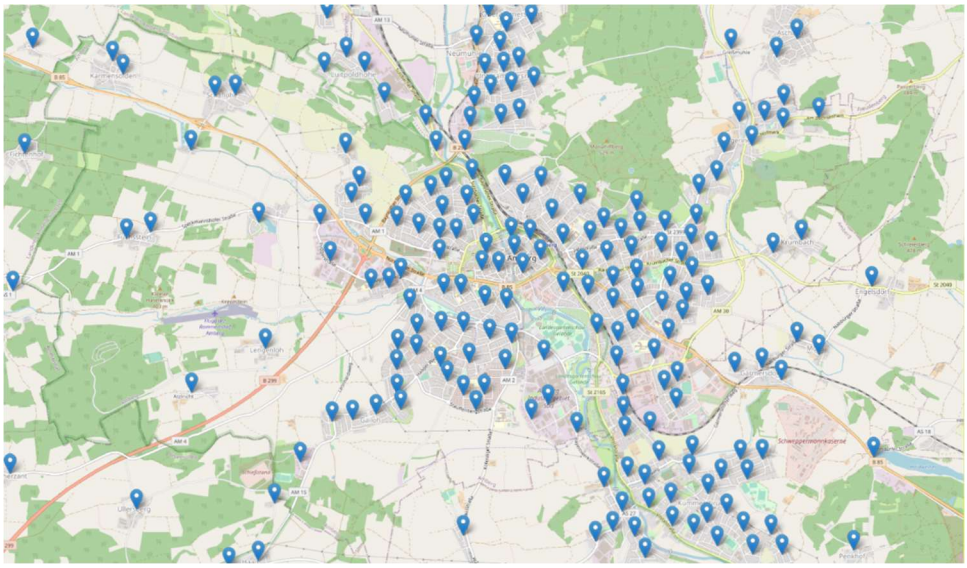

FIGURE 31: PICKUP STATIONS PLACED IN AMBERG

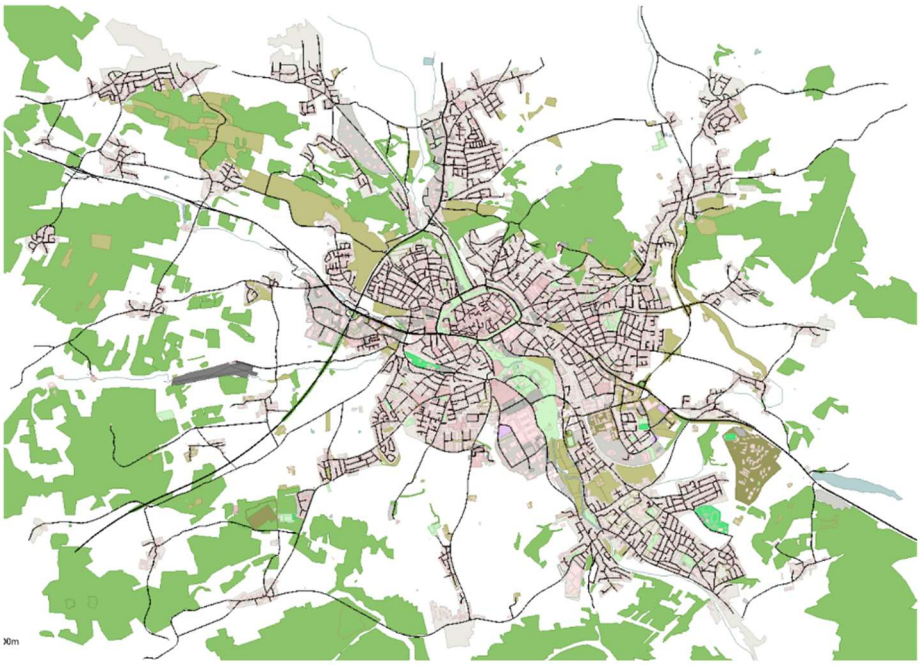

FIGURE 32: SUMO SIMULATION OF AMBERG

6.6.2 Computation platform and toolchain

To create the simulation no special hardware, but current consumer hardware was used. The toolchain consists of SUMO itself and python scripts to build the scenario.

6.6.3 Classical (non-AI) software functions

The whole simulation contains no AI components.

6.7 Routing Module

The routing module consists of three components as can be seen in Figure 28:

The RL solver is a reinforcement learning bases solver that figures out the best combination of routes by trying out different possible actions and learn from a given reward and a given set of stations and customer requests.

The matrix routing query is a component that queries the shortest routes from station to station. It is called matrix routing, since the result are multiple matrices that contain information about the cost minimized paths.

The energy simulation component evaluates the energy consumption for a given track section.

6.7.1 Software concept description

The routing system takes care of routing autonomous cabs and their passengers to their destinations. An example of a routing setting can be seen in Figure 34. The example contains five passengers with current and intended position. A vehicle has a set of orders and drives from station to station, to deliver the customers to their intended position. The route the vehicle takes is optimized. During the ride, multiple passengers can be transported in the same vehicle (pooling). The factors to be considered during optimization are energy consumption as well as trip duration or waiting time of the passengers.

The solution is calculated via a machine learning algorithm based on reinforcement learning, which is a self-learning approach, where an agent has possible actions that allow him to explore an environment and make observations and gain a reward for the taken actions.

The decided environment and action-space can be viewed at Figure 33.

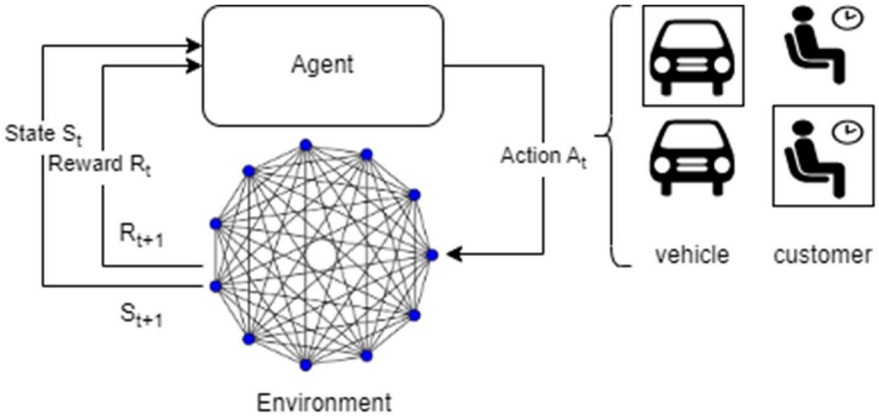

FIGURE 33: REINFORCEMENT LEARNING SETUP

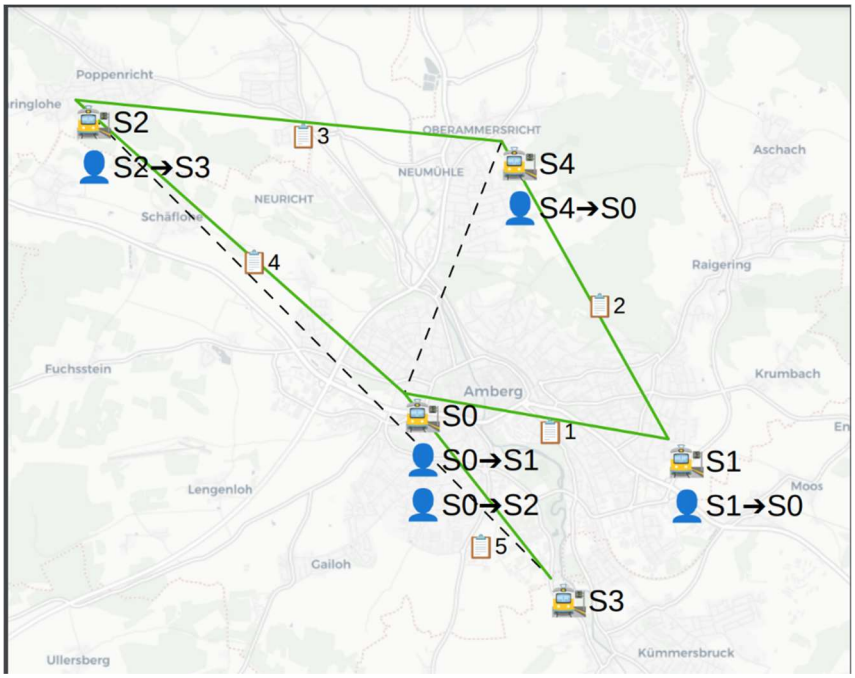

FIGURE 34: ROUTING EXAMPLE WITH 5 CUSTOMERS

6.7.2 Computation platform and toolchain

The energy consumption module is built with MATLAB Code wrapped with a Python Webserver build on FastAPI [12]. The matrix routing query is built in python using the open source routing software Valhalla. The Reinforcement solver is built with the reinforcement framework Stable Baselines [13] and the environment library, Gym from OpenAI [14]. The computation platform is server with an AMD EPYC 7452 32-Core Processor, 256GB RAM and 2xQuadro RTX 6000 graphics units.

6.7.3 Classical (non-AI) software functions

6.7.3.1 Energy Consumption Calculation

To optimize the energy consumption, an exact estimation of the energy consumption for a given route is required. Therefore, the building block of the energy estimation was established. As shown in Figure 35 it takes as input a speed profile and slopes, filters those, simulates the energy consumption with the input of weather conditions and outputs the energy consumption, which will be used for the routing. This building block was developed as part of the project 1000kmPLUS in cooperation with AVL and is extended to support the city use case requirements. This means adjustments of the filtering, since in the city area only short routes are used whereas 1000kmPLUS focused on longer routes. Apart from that, weather conditions like wind speed is hard to measure in the city area, due to swirls at the house fronts. Therefore, the influence of wind was neglected.

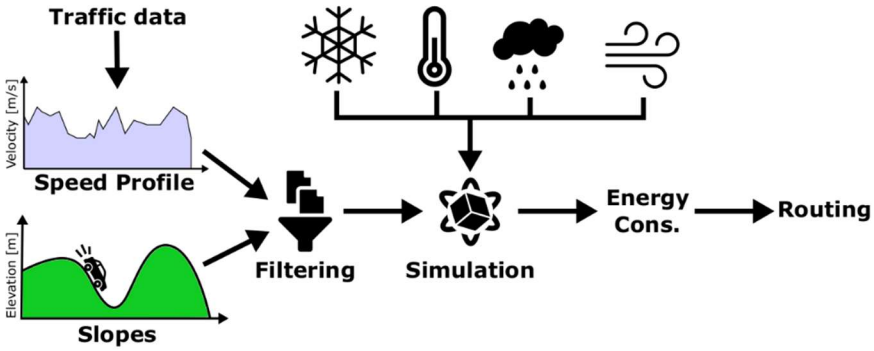

FIGURE 35: ENERGY SIMULATION PROCESS

6.7.3.2 Matrix Routing Query

In the demonstrator the routing between stations is optimized. To achieve that, costs between the stations have to be evaluated. Costs means distance, duration or energy consumption between the stations. This can be evaluated every time the reinforcement solver uses one connection of the routing graph. To speed up the process, the costs are pre-computed. This is done in the block matrix routing query. It calculates different cost types from each station to each station. To achieve this, the Matrix Query API from the routing engine Valhalla will be used[4].

6.7.4 AI methods and techniques

The Reinforcement Solver component is based on the control method of reinforcement learning and utilizes machine learning to estimate good policies to produce high quality solution to the fleet routing problem. Different reinforcement learning algorithms like PPO and A2C are investigated.

6.7.5 Training, verification, and validation plan

The AI is supposed to be trained via hyperparameter optimization where the hyperparameters of the model as well as additional hyperparameters like used features and reward parameter values are optimized. The verification is done by comparison to heuristics or optimal results, if possible to calculate.

6.7.6 Explainability for AI components

Explainability for the ML module will be delivered via so called SHAP values which explain the influence of input features for the action. The break down of the influences will help to optimize the feature selection and also help to make the agents decisions more transparent for humans.

6.8 RSU simulation module

RSU simulation module for vehicles-related data gathering in SUMO traffic was utilized.

6.8.1 Software concept description

According to command lines, a Python code is written to randomly generate vehicles with different autonomous level and routes for N number of vehicles and for specific simulation time. Roadside units (RSU) can be defined at the specific place of the map as representations of data-gathering infrastructure units from their surrounding vehicles within a specific range and over a specific period of time using TraCI toolkit in Python. In simulations different automation vehicle levels can be defined with Krauss Model in SUMO. Collected data correspond to a number of vehicles and their average speed over a specific period of time. As well, other data can be collected according to documentation [9].

6.8.2 Computation platform and toolchain

To create the simulation no special hardware, but current consumer hardware was used. The toolchain consists of SUMO itself and TraCI toolkit in Python scripts to run the scenario and obtain the data.

6.8.3 Classical (non-AI) software functions

The whole simulation contains no AI components.

7 Conclusion

7.1 Contribution to overall picture

All three demonstrators contribute fundamentally to support upcoming mobility trends and to the transition towards a sustainable mobility ecosystem by setting the stage for realizing autonomous vehicles and providing the possibility to share and optimize transport.

The hereby delivered system level design of the partners will be the blueprint for developing and implementation in the last period of the project.

Already realized parts of the development within the demonstrators ranging from first evaluation results, first simulations and benchmarks show the advanced progress within the project.

7.2 Relation to the state-of-the-art and progress beyond it

All three demonstrator realize exciting state-of-the-art technologies ranging from digital twin creation, complex route optimization towards in-house development of AI based perception and AD function development. Approaches ranging from advanced probabilistic fault-tolerant perception algorithms

via safe detection of human-AD function deviations including its analysis towards new reinforcement algorithms in fleet route optimization represent beyond state-of-the-art developments.

Partner/Topic	Description
VIF	The prototype and integration workflow of machine learning algorithms from virtual environments into real demonstrator vehicle is beyond state of the art as a relative new programming language and cutting edge HPC infrastructure (Vienna Scientific Cluster) were used.
VIF	The developed algorithms for real time semantic segmentation using ML techniques combined with a probabilistic semantic occupancy grid are beyond state of the art.
OTH	The state of the art in routing of passenger transportation fleets is methods based on linear programming. Current solution techniques include branch-and-cut, column generation and dynamic programming for exact solving and classic heuristics like insertion, meta heuristics like tabu-search, variable neighbourhood search or genetic algorithms for approximation. No techniques based on reinforcement learning are known yet in the literature. Only in the related field of the VRP and EVRP have those methods been used
VGTU	The developed algorithm for real time data simulation from roadside units is integrated in SUMO maps to record the status of vehicles, which can be saved or transmitted for further computations or ML-based decision making.
AVL	Testing ADAS/AD is a complex and challenging process that most of the time includes simulation-based testing and real-world testing. While each of both is an important part of testing ADAS/AD and is currently seen as state-of-the-art, the combination of detecting deviations of an ADAS/AD and a human driver in a safe manner (either in simulation solely or in the vehicle) as well as recreating those scenarios in simulation to further investigate possible additional problems in an expanded search space is beyond state-of-the-art.
AIT	The integration of criticality-based testing using probabilistic scenarios with an industrial platform for virtual testing of Autonomous Driving.
TUGRAZ	State of the art defines logic programs for visual sensemaking over time, but no qualitative approach for cross-checking multiple data sources against each other. (There exist quantitative approaches, like the well-known Kalman filter and its variants.)
TTTAUTO	Based on the communication platform design the service-oriented architecture utilizing exchangeable and thus configurable zero-copy implementation on the middleware abstraction is beyond state of the art. The major improvements however are dedicated in supply-chain 5.

7.3 Impacts to other WPs, Tasks and SCs

As output enabler SC1 partners are in close contact to technology enablers within SC4 – SC7. Standards and ethical issues rising up in the development autonomous functions are of great relevance for SC8.

Partner/Topic	Description
VIF	Output of SCD1.2 will be very relevant for SC8 (standards and ethical issues)
Carla integration	Relevant for many project partners using the same simulation engine
VGTU	Relevant for many project partners using simulation data about vehicles status on road.
AVL	The outcome of T2.1 is used for the upcoming tasks in WP4, the actual implementation of the developed algorithms and frameworks and its later integration and verification in WP5 and WP6. In addition, the outcome will be also used in the tasks within SC7 which is strongly connected to SC1.
AIT	Impacts on SC8 (standardization), putting scenario-based testing standards such as ASAM open scenario standard in focus.
TUGRAZ	The outcome of T2.1 builds the basis for the upcoming tasks within WP4 responsible for the implementation of the proposed framework. The results are later integrated and verified within WP5 Task 5.1 and are verified within WP6 Task 6.1.
TTTAuto	The communication and connectivity platform (originated in SC5) is utilized in SC1 and SC2 for general edge cloud communication abilities. Most of the design and implementation effort is dedicated to the SC5 related tasks. However, output enabler specific functionalities are implemented in SC1 WP4 and WP5 activities.

7.4 Contribution to demonstration

The following table summarizes the partner contributions to the three demonstrators.

Partner/Topic	Description
VIF	Lead of development of demonstrator SCD1.2
OTH	Lead of development of demonstrator SCD1.3
AVL	Lead of demonstrator SCD1.1. Responsible especially for the first part of the demonstrator dealing with test scenario definition in simulation and real-world.
AIT	Implementation of scenario-based testing loop, integration in the virtual test centre, selection and implementation of different sampling strategies, interfacing with other building blocks.
TUGRAZ	TUGRAZ provided a monitoring system that allows checking test scenarios to be passing or failing with respect to given formal properties.
TTTAuto	Provision of the communication platform and services to develop/test and integrate in later project activities.

7.5 Other conclusions and lessons learned

The following table summarizes some of the lessons learned by the partners ranging from new approaches to speed up innovation in software development of real-time applications to the current status of the implementation of common standards in popular software applications.

Partner/Topic	Description
VIF	Integration of Python with Julia works very well
VIF	OpenSCENARIO implementation within CARLA is not fully supported and lacks behind the newest standards
OTH	It is hard to get public available real-time traffic data. To enable further research, data sources need to be publicly available or available for research with affordable costs.
VGTU	Integration of SUMO with TraCI in Python works very well, although more available data sources is needed for better evaluations.
AVL	Although there exist common standards like ASAM openSCENARIO, simulator frameworks often are behind with their development and therefor with the adaption to the newest standard versions. Dealing with CAN signals can be quite difficult in case of occurring errors since the reported error messages are not always expressive and small errors in the configuration can lead to significant different outputs.
AIT	Scenic, VerifAI and CARLA loop runs smooth, and the integration takes moderate effort. Writing customized behaviours in Scenic can yield unpredictable results if not done as specified according to the guidelines.
TUGRAZ	Explicitly establishing all facts in an answer set program can quickly become counterproductive as it tremendously increases the time spent grounding and the size of the grounded program.
TTTAuto	The application of automotive domain specific design principles in R&D projects is challenging due to the lack of a common integration framework.

8 References

[1] "public Deliverable 1.1 (AI4CSM)," [Online]. Available: https://ai4csm.automotive.oth-aw.de/index.php/project/dissemination/public-deliverables/109-d1-1-report-on-sc1-use-cases-scenarios-kpis-and-requirements.

[2] "ASAM Measurement Data Format standard," [Online]. Available: https://www.asam.net/standards/detail/mdf/.

[3] "ASAM Open Simulation Interface standard," [Online]. Available: https://www.asam.net/standards/detail/osi/.

[4] "ASAM OpenSCENARIO Standard," [Online]. Available: https://www.asam.net/standards/detail/openscenario/.

[5] "ASAM openDRIVE standard," [Online]. Available: https://www.asam.net/standards/detail/opendrive/.

[6] "Scenic - a domain-specific probabilistic programming language for modeling the environments of cyber-physical systems," [Online]. Available: https://scenic-lang.readthedocs.io/en/latest/.

[7] M. G. e. al., "Answer Set Solving in Practice," in *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2012.

[8] "Commission Implementing Regulation 2022/1426 on automated driving systems (ADS)," [Online]. Available: https://op.europa.eu/en/publication-detail/-/publication/94bfefa8-24e9-11ed-8fa0-01aa75ed71a1.

[9] "SUMO User Documentation," [Online]. Available: https://sumo.dlr.de/docs/index.html.

[10] “3-click scenario generation,” [Online]. Available: https://sumo.dlr.de/docs/Networks/Import/OpenStreetMap.html#3-click_scenario_generation.

[11] "Official webpage of town Amberg relating to mobility," [Online]. Available: https://www.amberg.de/mobil .

[12] "FastAPI - a modern web framework for building APIs with Python," [Online]. Available: https://fastapi.tiangolo.com/.

[13] "Stable Baselines docs - a reinforement learning library," [Online]. Available: https://stable-baselines.readthedocs.io/en/master/.

[14] "Github repository on OpenAI training gym for reinforcement learning," [Online]. Available: https://github.com/openai/gym.

List of figures

List of tables

- Last page of the document is intended to be blank! -