kantSaar
Kooperatives, automatisiertes Fahren
im neurokognitiven Testfeld Saarland

kantSaar

Kooperatives, automatisiertes Fahren im neurokognitiven Testfeld Saarland

Deliverable D1

Architecture, Data Structure and Data Preparation

Version	1.0
Dissemination level	public
Project Coordination	htw saar

Funded by

Federal Ministry
of Transport and
Digital Infrastructure

Project Coordination

Prof. Dr. Horst Wieker
Head of ITS Research Group (FGVT) at the
htw saar – Hochschule für Technik und Wirtschaft des Saarlandes,
University of Applied Sciences
Department of Telecommunications
Campus Alt-Saarbrücken
Goebenstr. 40
D-66117 Saarbrücken
Germany

Phone +49 681 5867 195
Fax +49 681 5867 122
E-mail wieker@htwsaar.de

Authors:

J. Golanov – htw saar

C. Metzner – htw saar

A. Otte – htw saar

J. Staub – htw saar

A. Revision and History chart

Version	Date	Description
0.0	2019-01-30	**Created Document Root**
1.0	2020-02-29	Document Release

B. Table of Content

Figures

Tables

Executive Summary

This document describes the architecture and data processing in the kantSaar project. The kantSaar project aims to evaluate traffic situations, if they are suitable for a handover from automated to manual driving function. Architecture-related requirements have been defined based on visionary scenarios. Based on the requirements, the architecture has been defined. It describes where and which data needs to be aggregated.

Four data classes have been defined: *Environmental data* such as weather information, *traffic data* in the form of V2X messages, *vehicle data* in the form of sensor values and *driver-related data* such as the current stress level of the driver. All gathered data is sent to the *data fusion*, which combines them to an *overall traffic situation description*. Since a Big Data approach has been chosen and therefore many data records are generated, it is important that the data is transferred as sparingly as possible. Therefore, all data is compressed by Google Protobuf [1] before it is sent. HTTP with REST [2] is used to transmit the data to the data fusion in the backend.

1 Introduction

The level of automation in vehicles will significantly increase over the next decade. As automation will become more and more common, vehicles will not be able to master all traffic related situations for a long time by themselves. In such situations, the driver must take over and steer the vehicle through the situation. One of the important questions is when the takeover should be performed. Many decisive factors must be considered. On the one hand, the current traffic situation including roads, traffic lights and other road users, especially vulnerable road users, and on the other hand, the state of the driver must be considered as well. The goal is to combine neurocognitive measurements of the drivers' state and the static and dynamic traffic related data to develop an interpretation of the current situation. This situation analysis should be the basis for the determination of the most suitable point for a takeover maneuver.

One goal of the research project kantSaar is to record traffic situations holistically. Therefore, it is necessary to identify which data needs to be gathered. This goal has been defined, using a requirement analysis based on visionary scenarios, described in Chapter 2. An architecture is needed to describe how the data gathering is performed. It defines data sources (aggregators), where data is combined and evaluated. The architecture and its components are described in Chapter 3.

Traffic data, vehicle data, environmental data and driver-related data: Each data class has different characteristics. A challenge of the project is to combine those data using a unified data model. The data model is described in Chapter 4. Due to the different characteristics, the data set is not complete in every traffic situation. Therefore, the data needs to be supplemented with interpolation and extrapolation processes, described in Chapter 5.

1.1 Project goals

The goal is to build up a sensor infrastructure and to combine neurocognitive data with environmental and traffic-related data to evaluate the overall traffic situation. By using camera systems, which can track micro expressions in the driver's face, a detailed driver state can be detected, including emotions and vital signs. With this information, it is possible to assess traffic situations and adapt the handover strategy according to the drivers' condition. In addition, a "stress map" can be created, that indicates which areas in urban and non-urban environments are challenging for a driver of semi-automated vehicles. This information can later be provided to the vehicles and the infrastructure to make automated driving safer and more reliable.

1.2 Approach

To accomplish the previously described project goals, a five-step approach, containing *requirement analysis*, *data collection*, *data preparation*, *data fusion* and *evaluation* was developed. This document describes the steps requirement analysis, data collection and data preparation in detail.

2 Requirement analysis

2.1 Scenarios

The base of the requirement analysis are screnario descriptions. They describe five situations, that users of automated vehicles can experience in the near future.

2.1.1 SCN-00: Visionary Scenario

IT manager Emilia E. wants to drive with her car from her city apartment to an IT company located at the edge of the conurbation. Her navigation system suggests the route with the highest automation mode. Additionally it shows her on which sections of the route she has to be extra attentive and on which sections she needs less attention and can keep attention to other tasks. These sections are important for Emilia E. as she can prepare her upcoming meeting. During the ride, the vehicle supports her on certain sections and takes full control of speed and longitudinal and transverse steering. The latter is for example, on the nearby main road. On the way, she approaches a complex subway construction site that the vehicle cannot manage independently due to constantly changing traffic flows. The control system of Emilia E.'s vehicle is informed about the situation through the exchange of information with the traffic infrastructure and traffic information services and can hand over the driving task to her at an appropriate point at an early stage. On the basis of neuro-cybernetic expected values of the degree of attention as well as information about the current traffic situation, the system decides to prepare and carry out the handover task 300m before the construction site. Emilia E. receives up-to-date information on the situation and the entertainment system of her vehicle will be suppressed. After the construction site, the vehicle system takes over again. In the further ride, she reaches a section of road which shows a large number of road damages leading to a very uncomfortable driving. She prefers to take control of the vehicle by herself and informs the vehicle about her decision. On the basis of the measured values from the learning machine's archive, the system prepares her for the handover at an early stage/time (200 m). The rest of the ride the vehicle can drive again fully automated, because her destination is located in a well developed industrial area. She reaches it safely and well prepared.

2.1.2 SCN-01: Stationary vehicle blocks lane

A vehicle drives automatically at 30 km/h on the left lane of a two-lane one-way street in dense traffic in the evening. The driver is tired and therefore his attention is affected. The left lane is blocked by a broken down, cooperative vehicle, which alerts the driver to the dangerous situation by transmitting DENM. The following traffic is forced to avoid the vehicle by using the right lane. The automated vehicle receives the DENM. Due to the complexity of the situation (dense traffic with high percentage of legacy vehicles), the vehicle initiates the handover to the driver.

2.1.3 SCN-02: Pedestrians at the intersection

A vehicle drives automatically at 45 km/h towards an intersection with traffic lights. It is sunny and warm. Because of the late morning, there is sparse traffic. Due to the focused work on his tablet, the driver's attention is constrained. The vehicle is going to turn right at the intersection. There are two lanes: one turning lane to the right and one turning lane to the left. The same traffic light controls both directions. The traffic light switches to green while the vehicle is driving towards the intersection. There are pedestrians who are going to cross the road the vehicle will turn in. The intersection is equipped with a camera system that detects the pedestrians. Their positions are sent

to the road users using CPM messages. Due to the dynamic behavior of pedestrians, the handover to the driver is made before the turning.

2.1.4 SCN-03: Drive towards fog

A vehicle drives automatically at 70 km/h on an empty rural road with a wet road surface. The driver observes the driving situation. The vehicle moves towards a dense wall of fog. It receives a corresponding warning via DAB. Bad weather conditions limit the function of the vehicle sensors. Therefore, the handover to the driver is performed before the vehicle arrives at the fog.

2.1.5 SCN-04: Bad road conditions

A vehicle drives automatically at 50 km/h in the city. Due to road damage, the ride is rough. The vehicle recognizes that the driver feels uncomfortable and slows down automatically. The driver disagrees with the vehicle's decision because he wants to arrive in time. Therefore, the driver would like to take control of the vehicle. Using the HMI the driver informs the vehicle, whereupon the vehicle initiates a handover.

2.2 Gathered Requirements

Based on the scenarios, requirements are defined by the project team. Each scenario is stripped down to single statements that are analyzed, if they contain any important information that can be used for handover decision algorithm.

Each requirement is mapped to a scope-category and a class. The scope describes whether the requirement addresses the architecture or the implementation of the prototype. The class provides information whether the requirement is technical or organizational. Requirements that are architectural-scoped and classified as technical are used both for the creation of the architecture and for its later evaluation. This approach was used in the research project iKoPA [3][4].

The following table contains all defined requirements.

Req.-No.	Subject	Source	Scope	Class	Description
REQ-SYS-001	Vehicle recognizes the level of traffic density	SCN-01	Implementation	Technical	The vehicle system shall be able to recognize the level of traffic density.
REQ-SYS-002	Vehicle recognizes the level of light	SCN-01	Implementation	Technical	The vehicle system shall be able to recognize the level of light.
REQ-SYS-003	Vehicle receives DENM	SCN-01	Architecture	Technical	The vehicle shall be able to receive DENM via G5.

REQ-SYS-004	Vehicle sends DENM	basic	Architecture	Technical	The vehicle shall be able to send DENM via G5.
REQ-SYS-005	Vehicle-aggregated data send to backend	SCN-01, SCN-02, SCN-03, SCN-04	Architecture	Technical	The data aggregated by the vehicle shall be send to the backend.
REQ-SYS-006	Vehicle recognized its position vector	basic	Implementation	Technical	The vehicle shall be able to recognize its position vector. The position vector consists of the position, the course, the heading, speed, position accuracy and time.
REQ-SYS-007	Lane-precise position	SCN-01, SCN-02	Implementation	Technical	The vehicle shall be able to match its position to the current lane.
REQ-SYS-008	HMI is available	all SCN	Architecture	Technical	The vehicle shall provide a human-machine-interface.
REQ-SYS-009	vehicle requests manual driving function	SCN-01, SCN-02, SCN-03	Architecture	Technical	The vehicle shall be able to request manual driving function.
REQ-SYS-010	vehicle receives MAP	SCN-02	Architecture	Technical	The vehicle shall be able to receive MAP messages via G5.
REQ-SYS-011	vehicle knows its route	SCN-02	Architecture	Technical	The vehicle shall know its route.
REQ-SYS-012	intersection determines pedestrian position	SCN-02	Implementation	Technical	The intersection shall be able to determine the existence and position (-vector) of pedestrians.

REQ-SYS-013	intersection sends CPM	SCN-02	Architecture	Technical	The intersection shall be able to send CPM's via G5, containing the position of obstacles.
REQ-SYS-014	intersection send MAP	SCN-02	Architecture	Technical	the intersection shall send MAP messages via G5.
REQ-SYS-015	intersection send SPAT	SCN-02	Architecture	Technical	The intersection shall send SPAT messages via G5.
REQ-SYS-016	Vehicle receives SPAT	SCN-02	Architecture	Technical	The vehicle shall be able to receive SPAT messages via G5.
REQ-SYS-017	Vehicle receives CPM	SCN-02	Architecture	Technical	The vehicle shall be able to receive CPM via G5.
REQ-SYS-018	Intersection-aggregated data send to backend	SCN-02	Architecture	Technical	The data aggregated by the intersection shall be send to the backend.
REQ-SYS-019	Pseudonymous vehicle data	all SCN	Architecture	Technical	The data send by the vehicle to the backend shall be pseudonymised.
REQ-SYS-020	Encrypted vehicle data	all SCN	Architecture	Technical	The data send by the vehicle to the backend shall be encrypted.
REQ-SYS-021	Discard of untrustworthy messages	SCN-01, SCN-02	Architecture	Technical	The vehicle shall discard untrustworthy messages (not signed) received by G5.
REQ-SYS-022	Vehicle receives weather information	SCN-03	Architecture	Technical	The vehicle shall be able to receive weather information.

REQ-SYS-023	Driver requests handover	SCN-04	Architecture	Technical	The HMI provides a mechanism to allow the driver to take control.
REQ-SYS-024	Route displayed on HMI	SCN-00	Implementation	Technical	The HMI shall provide the possibility to display a route.
REQ-SYS-025	Distinguish between automatic levels	SCN-VS	Implementation	Technical	The route shall be distinguished by their ability to drive automatically.
REQ-SYS-026	Stress map available	SCN-00	Architecture	Technical	The backend shall provide a map indicating stressful road sections on a planned route.
REQ-SYS-027	Handover start time	SCN-00	Architecture	Technical	The backend shall provide the start time of a handover.
REQ-SYS-028	Transition time	SCN-00	Architecture	Technical	The backend shall provide the transition time of a handover.
REQ-SYS-029	Minimal Risk Maneuver	basic	Architecture	Organizational	The vehicle shall perform a minimal risk maneuver if the driver does not take control when requested.
REQ-SYS-030	Suppress entertainment systems	SCN-00	Implementation	Technical	The vehicle shall suppress the entertainment systems if the driver is requested to take control.
REQ-SYS-031	Vehicle requests automatic driving function	SCN-00	Architecture	Technical	The vehicle shall be able to request the automatic driving function.

REQ-SYS-032	Vehicle recognizes level of attention	SCN-01, SCN-02, SCN-03, SCN-04	Implementation	Technical	The vehicle shall be able to recognize the level of attention of the driver.
REQ-SYS-033	Vehicle recognizes action	all SCN	Implementation	Technical	The vehicle shall be able to recognize what the driver is doing.
REQ-SYS-034	Driver's level of comfort	all SCN	Implementation	Technical	The vehicle shall be able to recognize how the driver is feeling.

Table 1 Requirement list

3 Architecture

The project kantSaar aims to design a system architecture, allowing the fusion of various data to an overall traffic situation description. The challenge is to merge data with different characteristics. The result of the requirement analysis described in chapter 2 requests to unify discrete data with continuous data. Most data is originated in the remote station plane at a *Roadside ITS Station (R-ITS-S)* or the *Vehicle Under Test (VUT)* and other *Vehicle ITS Station (V-ITS-S)* systems. Other data is collected from third party services or own cloud-based service providers preprocessing raw data.

The complete set of required data cannot be directly accessed at a local node in the remote station plane, because the information is distributed on various systems. Therefore, the data is transmitted to a central component in which the data fusion will be performed.

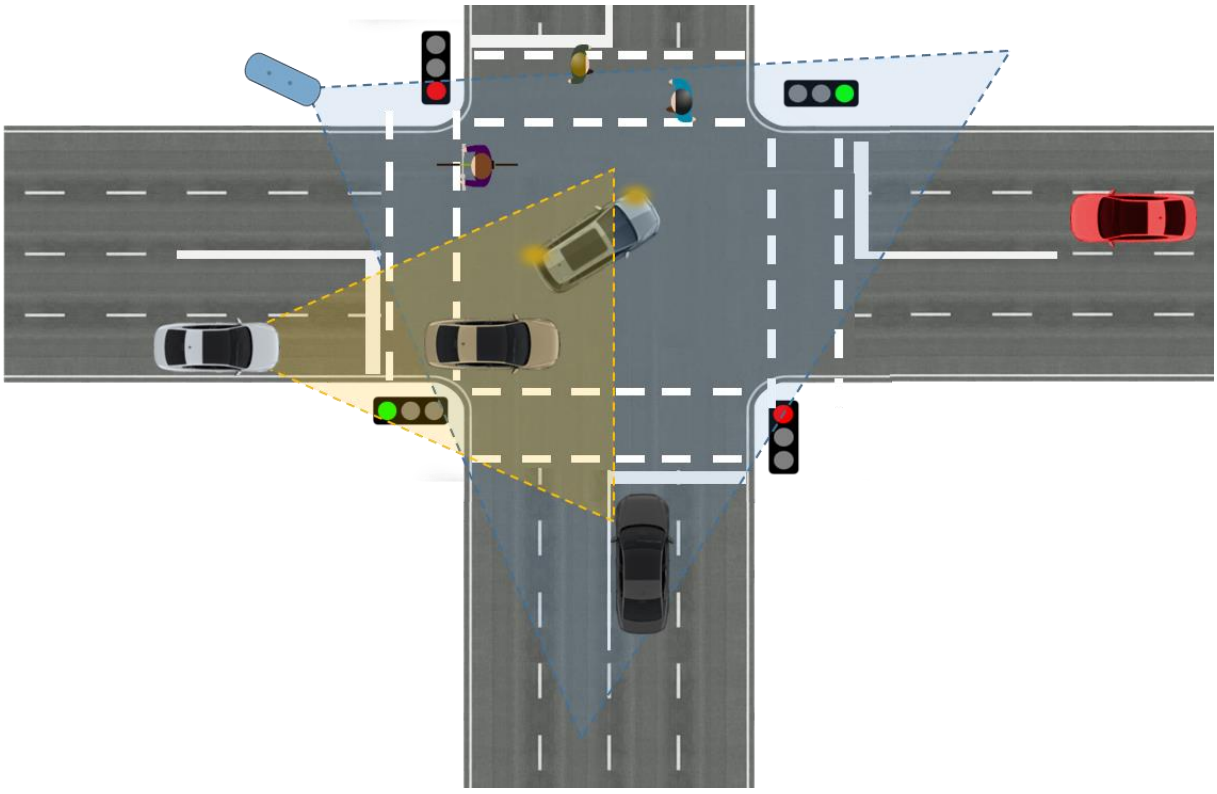

Figure 1 Data sources in the remote station plane

Figure 1 shows elements of the remote station plane. The RSU accesses the traffic light system (TLS), the direct short radio communications (DSRC) system, indication loops and optical sensors like traffic cameras. The TLS provides information about current signal phases of the lane topology. The sensors are used to detect common road users, DSRC like ETSI ITS-G5 and C-V2X receives information about Cooperative Vehicles (CV). Additionally, the VUT and CV also provide information about other road users using their in-vehicle sensors in combination with their Collective Perception Service (CPS). The simultaneous detection of the same road user by vehicles and the infrastructure causes the problem of a multiplication of detected objects, induced by the principle of distributed sensors [5].

The information of the remote station plane is supplemented with information from a traffic center and third party services. If the used TLS does not provide information about the signal phases or sensors locally, a corresponding traffic center may have that information.

3.1 Overview

Figure 2 shows the detailed architecture of kantSaar. It is split in two parts: The Remote Station Plane and the Backend Plane. The Remote Station Plane contains infrastructure systems called roadside ITS stations (R-ITS-S) and Vehicle ITS Stations (V-ITS-S). The VUT is a specialized V-ITS-S with access to a driver monitoring system. The backend plane consists of services for the pre-processing of locally gathered data and additional third party data suppliers as well as the Data Fusion and the Situation Evaluation. The raw, fused and evaluated data is stored in the Situation Storage.

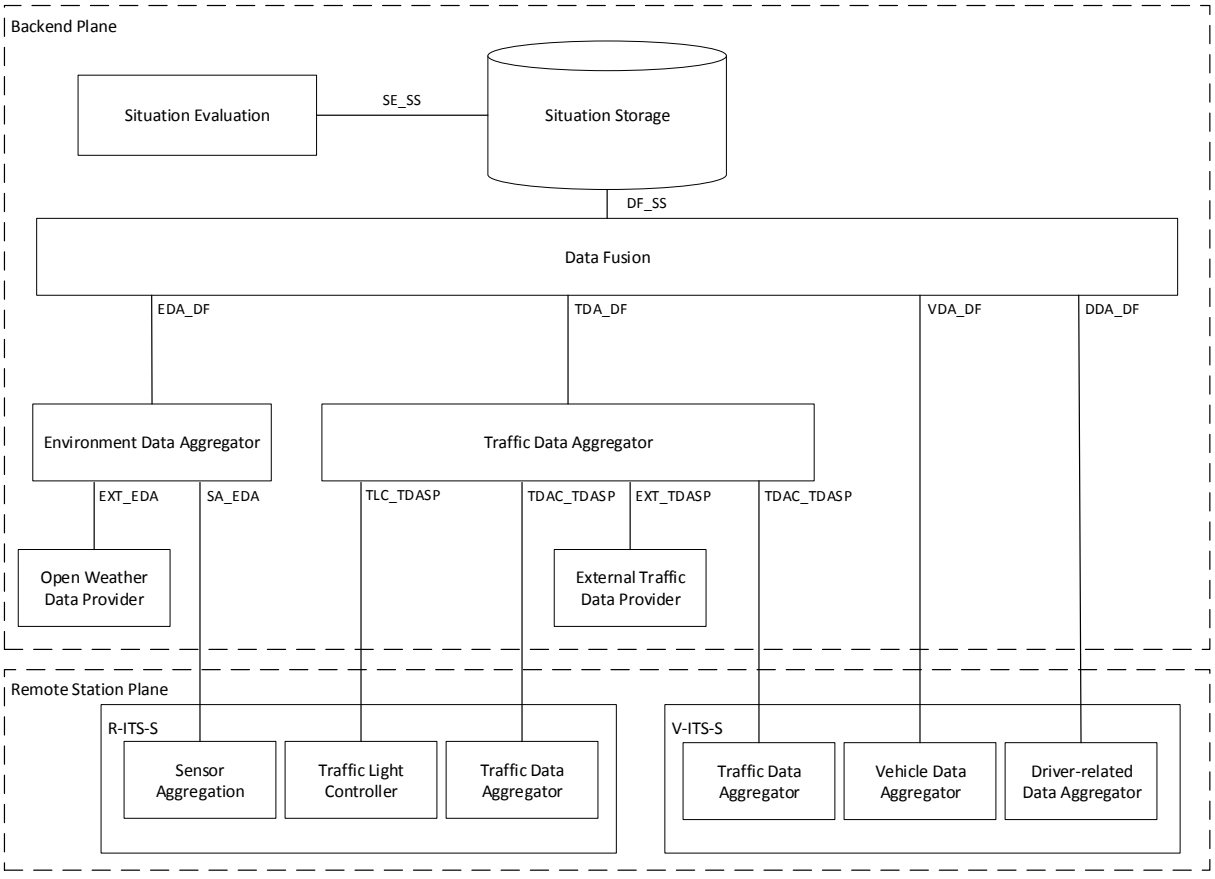

Figure 2 Detailed Architecture

3.2 Components

3.2.1 Driver-related Data Aggregator

The driver-related data aggregator is described in a separate document. When it is finalized, it will be linked here.

3.2.2 Environment Data Aggregator

The function of the Environment Data Aggregator is to collect environment data and provide it to the Data Fusion. This data is primarily weather data. Weather information is an important part of road traffic and can significantly affect a traffic situation. The weather data, which is collected in the Environment Data Aggregator is bound to a timestamp with a validity and a location area. It describes the current weather conditions using information about temperature, precipitation, wind, light and visibility conditions (and more, such as pressure, humidity, cloudiness, …).

The weather data, which represents the current weather conditions in the immediate vicinity, is provided by the abstract component Open Weather Data Provider in the backend plane. It serves as a service provider and supplies its data from the OpenWeatherMap.org website using a free weather API.

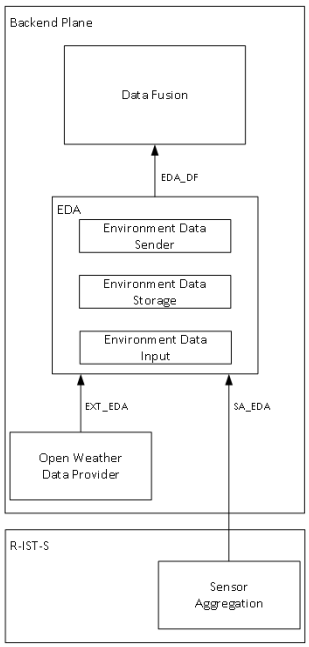

Figure 3 EDA inner structure and environment

The sensor aggregation component, which is settled in an ITS roadside station is another information provider for the Environment Data Aggregator. For this a weather station with sensors could be mounted to the ITS Roadside station and provide real time data about the weather condition in the correspondent intersection area.

3.2.3 Traffic Data Aggregator

The Traffic Data Aggregator is used to collect and aggregate traffic data in a cooperative V2X environment. This data is primarily the V2X messages sent and received by the remote stations via ITS G5 and the information contained therein about the current immediate traffic situation.

3.2.3.1 Traffic Data Aggregator Client

The Traffic Data Aggregator Client (*TDAC*) is a component that is present both in the vehicle and in infrastructure components, like ITS Roadside stations. Its purpose is to collect traffic-related data contained in V2X-messages (CAM, DENM, SPaT, MAP, CPM) sent or received by this station.

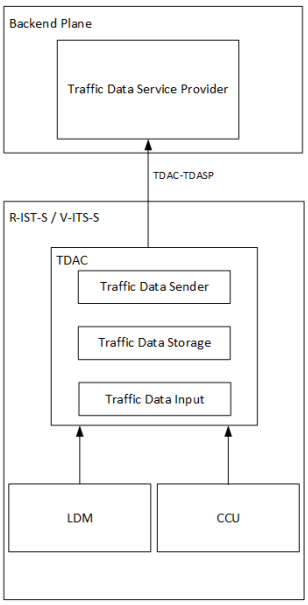

Figure 4 TDA inner structure and environment

To gather this information, the TDAC obtains all messages sent from the Local Dynamic Map, processes them and extracts the relevant information. The collected data is then compressed and sent (via interface TDAC_TDASP) to the Traffic Data Aggregator Service Provider located in the backend.

V2X-Message	Description
CAM	Cooperative Awareness Messages (CAMs) are messages exchanged in the ITS network between ITS-Ss to create and maintain awareness of each other and to support cooperative performance of vehicles using the road network. A CAM contains status and attribute information of the originating ITS-S. The content varies depending on the type of the ITS-S. For vehicle ITS-Ss the status information includes time, position, motion state, activated systems, etc. and the attribute information includes data about the dimensions, vehicle type and role in the road traffic, etc. On reception of a CAM the receiving ITS-S becomes aware of the presence, type, and status of the originating ITS-S. [7]
DENM	Decentralized Environmental Notification Message (DENM) is a facilities layer message that is mainly used by ITS applications in order to alert road users of a detected event using ITS communication technologies. A DENM is used to describe a variety of events that can be detected by ITS-Ss. This message contains information related to an event that has potential impact on road safety or traffic condition. An event is characterized by an event type, an

	event position, a detection time and a time duration. These attributes may change over space and over time. [8]
SPaT	Signal Phase And Timing (SPAT), is a message type which describes the current state of a signal system and its phases and relates this to the specific lanes (and therefore to movements and approaches) in the intersection. It is used along with the MAP message to allow describing an intersection and its current control state. [9]
MAP	The MAP message is used to convey many types of geographic road information. At the current time its primary use is to convey one or more intersection lane geometry maps within a single message. The map message content includes such items as complex intersection descriptions, road segment descriptions, high speed curve outlines, and segments of roadway. [9]
CPM	The Collective Perception Message (CPM) contains information about detected objects by the disseminating ITS-S. The message consists of information about the disseminating ITS-S, its sensory capabilities and its detected objects. These objects are classified in vehicles or pedestrians for example including information about their position, heading and movement speed. This message focuses on reporting changes in the dynamic road environment. [10]

3.2.3.2 Traffic Data Aggregator Service Provider

The Traffic Data Aggregator Service Provider (*TDASP*) is the backend component of the Traffic Data Aggregator. On the one hand it provides an interface (TDAC_TDASP) to receive traffic data collected by the TDAC of remote stations. On the other hand the TDASP is also able to receive and process data via the interface EXT_TDASP from other sources like an external supplier of traffic data. The collected traffic information is processed, checked for duplicates and subsequently merged. Eventually this data is forwarded to the Data Fusion via the specified interface TDA_DF.

3.2.3.3 External Traffic Data Provider

A solution for an external traffic data provider could be a federal or state transport service. In this case an own implementation called TravelTimeRecorder is used as a solution for external traffic data.

The implementation of the TravelTimeRecorder enables data sets to be completed on routes between the individual traffic nodes. The concept of the TravelTimeRecorder is to record each individual traffic node as a measure point at which it is possible to leave the section of the route. The road site units distribute these measure points to each individual vehicle using ITS-G5.

If a vehicle receives these measure points and passes one of these, a measurement is started automatically. This measurement runs until the next measure point is passed or the measurement is discarded due to a parking process or something else.

The measurements will be transmitted to the road side units using ITS-G5. A road site unit forwards these measurements to the backend for further evaluation.The measurements can be used to determine average travel times on a large number of sections. Analysis and evaluation of the traffic density on a given (partial) route in areas without road site units and their detection will be possible.

3.2.4 Vehicle Data Aggregator

The Vehicle Data Aggregator (VDA) collects sensor data of the VUT. The vehicle sensor data is originated of the vehicles CAN bus systems. CAN data is transmitted dynamically in the matter of time, e.g. the current lateral acceleration is transmitted more often as the state of the wiper system. Using a sensor data abstraction layer, the VDA registers for a subset of the available vehicle data listed in Table 2.

Value	Description
Brake System	Status of the brake system including brake actuation, status of Antilock Braking System and if a panic braking is performed
Clutch and Gear status	Status of the clutch pedal (if available) and the current gear
Door position	Position of each door (closed, ajar or open)
Exterior Lights	Low and High beam status, fog lights, emergency lights, hazard warning system, turning signals
Speed	Vehicle speed and longitudinal and latitudinal acceleration
Rain Sensor	Rain intensity sensor and wiper system
Yaw	Yaw rate, yaw velocity, steering wheel angle and steering wheel velocity

Table 2 VDA sensor values

Figure 5 shows the inner structure of the VDA. As the submission involves overhead, the sensor data is not directly forwarded on reception but stored in a local vehicle data storage. Therefore, the collected data is transmitted periodically via the interface VDA_DF. The period can be adapted on changing factuality e.g. mobile reception quality. If the transmission was successful, the storage is cleared.

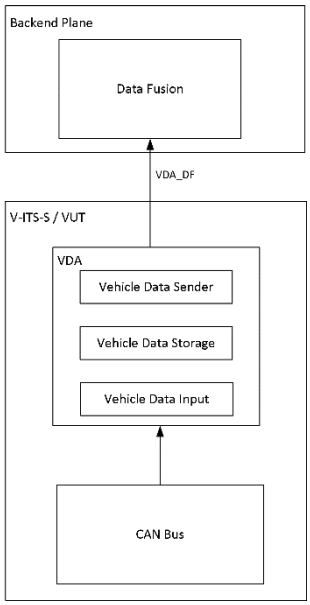

Figure 5 VDA inner structure and environment

Unlike the traffic data aggregator, VDA only consists of a V-ITS-S part executed on the VUT. There is no need to pre-merge sensor values of different VUT in the backend plane as they operate independently and no duplicates need to be filtered.

3.2.5 Data Fusion

After the collection of all traffic and neurocognitive data, a fusion and preparation of this data must take place. The merging of all recorded data takes place in the data fusion, which represents the central component of the project kantSaar. First, the data is persistently stored as raw data.

Figure 6 table structure of raw data

Subsequently, the raw data is evaluated and written to the so-called situation database. During the evaluation, classifications and groupings of the raw data are performed in order to transfer it into a simplified data pattern. As a result, traffic situations can be meaningfully evaluated and reproduced later on.

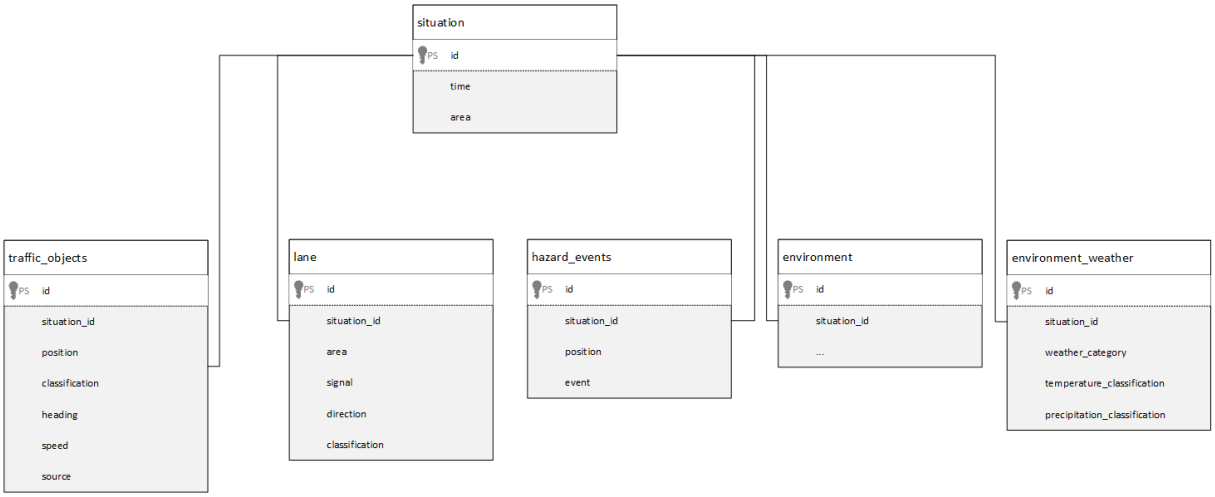

Figure 7 table structure of situation data

Finally, artificial intelligence algorithms are used to evaluate and analyze the data in order to make a decision about the likelihood of transferring driving control to the driver.

3.2.6 Situation Storage

The situation database serves to simplify the overall evaluation and enables easier decision-making when evaluating and analyzing the data. The raw data is evaluated in a simplified manner so that a decision can later be determined more efficiently.

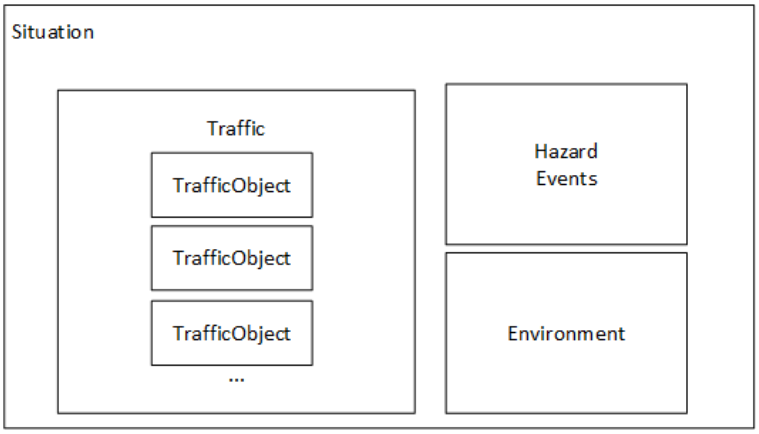

Figure 8 schematic draft of situation data

The diagram shows that objects from different sources (CAM or CPM) can be the same object or an object of the same type. This means that it has basically the same information, but in different formats and data structures. Therefore, this data must be standardized in order to enable evaluation, which is independent of the source and structure.

Every situation is determined and triggered by a geographical and temporal parameter. Therefore, a certain geographical area is observed at a given point in time.

3.2.7 Situation Evaluation

The purpose of the situation evaluation is to determine the degree to which a traffic situation at a certain time and place is suitable for a handover from automated to manual driving function. In order for the situation evaluation to make such a decision, it is necessary to have suitable data sets as a basis for the input data. The collected raw data of the aggregators and their unification and processing in the data fusion forms the resulting situation database. The information contained within the situation database serves as the basis for the evaluation of the individual traffic situations. The different traffic situations are the data sets that the situation evaluation needs as input.

An individual traffic situation is characterized by traffic objects, lanes, hazard events, the environment and weather as well as raw vehicledata. Each data sample has a situation ID, which is a reference to the metadata (time, geo-position and validity) of the traffic situation.

The time, the geo-position and its validity are bound as meta-data to each traffic situation. This metadata is not taken into account in the calculation (of the decision). This information remains unchanged in the output of the evaluation and thus determines at which time and at which place a transfer is suitable or not. The output data, i.e. the result of the situation evaluation will be a degree of suitability for a transfer from automated to manual driving function. The suitability degree will be a scale value that defines to which extent a traffic situation is suitable for a handover (for example, on a scale range from one to ten (1...10) for not suitable to highly suitable).

3.3 Interfaces

The previously described architecture (**Fehler! Verweisquelle konnte nicht gefunden werden.**) defines numerous interfaces to transport data between the different components.

3.3.1 SE_SS

Provider	Situation Storage
Consumer	Situation Evaluation
Type	MySQL Database query
Description	The Situation Storage requests situation data sets with MySQL queries, which serve as the basis for the evaluation of an individual traffic situation. The calculated result of the Situation Evaluation is a scale range number indicating the suitability for a handover. For each situation, the evaluated result is written back in the Situation Storage with another MySQL query.
Data format	SQL Statement (MySQL)

3.3.2 DF_SS

Provider	Situation Storage
Consumer	Data Fusion
Type	MySQL Database query
Description	The Data Fusion filters the collected data and writes grouped data sets into the Situation Storage. The filtering and grouping is based on parameters given by requests of users.
Data format	SQL Statement (MySQL)

3.3.3 EDA_DF

Provider	Data Fusion
Consumer	Environmental Data Aggregator (Service Provider)
Type	HTTP Rest
Description	The Environmental Data Aggregator sends HTTP Post requests to the Data Fusion. These requests contain the corresponding environmental data in their HTTP body. This data mostly contains information about the current weather. The Data Fusion confirms the reception using the status codes in the HTTP response.
Data format	Protobuf message “EnvironmentData”.

3.3.4 TDA_DF

Provider	Data Fusion
Consumer	Traffic Data Aggregator (Service Provider)
Type	HTTP Rest
Description	The Traffic Data Aggregator sends HTTP Post requests to the Data Fusion. These requests contain the corresponding traffic data in their HTTP body. This traffic data contains all the data received from the multiple Traffic Data Aggregator Clients.

	The Data Fusion confirms the reception using the status codes in the HTTP response.
Data format	Protobuf message “TrafficData”.

3.3.5 VDA_DF

Provider	Data Fusion
Consumer	Vehicle Data Aggregator
Type	HTTP Rest
Description	The Vehicle Data Aggregator sends HTTP Post requests to the Data Fusion. These requests contain the corresponding vehicle data in their HTTP body. This vehicle data consists of all the data, like CAN and positioning data, collected on the vehicle. The Data Fusion confirms the reception using the status codes in the HTTP response.
Data format	Protobuf message “VehicleData”.

3.3.6 DDA_DF

Provider	Data Fusion
Consumer	Driver-related Data Aggregator
Type	HTTP Rest
Description	The Driver-related Data Aggregator sends HTTP Post requests to the Data Fusion. These requests contain the corresponding driver-related data in their HTTP body. This set contains data on the current status of the driver. The Data Fusion confirms the reception using the status codes in the HTTP response.
Data format	Protobuf message “NeurocognitiveData”.

3.3.7 EXT_EDA

Provider	Environmental Data Aggregator
Consumer	Open Weather Data Provider
Type	HTTP Rest
Description	The Open Weather Data Provider sends HTTP Post requests to the Environmental Data Aggregator. These requests contain the corresponding environmental data in their HTTP body. This data contains weather information from a publicly available weather service. The recipient confirms the reception using the status codes in the HTTP response.
Data format	Protobuf message “EnvironmentData”.

3.3.8 SA_EDA

Provider	Environmental Data Aggregator
Consumer	Sensor Aggregation
Type	HTTP Rest

Description	The Sensor Aggregation sends HTTP Post requests to the Environmental Data Aggregator. These requests contain the corresponding environmental data in their HTTP body. This data contains weather and climate information, measured by the sensors of the sender. The recipient confirms the reception using the status codes in the HTTP response.
Data format	Protobuf message “EnvironmentData”.

3.3.9 TLC_TDASP

Provider	Traffic Data Aggregator (Service Provider)
Consumer	Traffic Light Controller
Type	HTTP Rest
Description	The Traffic Light Controller sends HTTP Post requests to the provider. These requests contain the corresponding traffic data in their HTTP body. This traffic data contains the V2X messages SPaT and MAP generated by the controller. The service provider confirms the reception using the status codes in the HTTP response.
Data format	Protobuf message “TrafficData”.

3.3.10 TDAC_TDASP

Provider	Traffic Data Aggregator (Service Provider)
Consumer	Traffic Data Aggregator Client
Type	HTTP Rest
Description	The Traffic Data Aggregator Client sends HTTP Post requests to the provider. These requests contain the corresponding traffic data in their HTTP body. This traffic data contains all the v2x messages received and generated by the vehicle. The service provider confirms the reception using the status codes in the HTTP response.
Data format	Protobuf message “TrafficData”.

3.3.11 EXT_TDASP

Provider	Traffic Data Aggregator (Service Provider)
Consumer	External Traffic Data Provider
Type	HTTP Rest
Description	The External Traffic Data Aggregator Provider sends HTTP Post requests to the provider. These requests contain the corresponding traffic data in their HTTP body. The service provider confirms the reception using the status codes in the HTTP response.
Data format	Protobuf message “TrafficData”.

4 Data model

4.1 Data definitions

Multiple different data types are getting used within the described system. These contain the relevant information for the particular system components and can be exchanged between them. A distributed system like this relies heavily on the performance of its components and the communication channels between them. Therefore the data needs to be defined in a way that allows fast serialization, processing and distribution.

4.1.1 Protocol Buffers

Protocol buffers or short “Protobuf” is a method of serializing structured data, developed by Google[1] and published under the BSD license. Its main design goal has been the creation of simple, high-performance data structures, both for the exchange and the storage of data. These structures are described in human-readable proto definition files (.proto) and are called messages. Protobuf delivers a compiler (protoc) to create the corresponding class files for common programming languages like Python, C++ or Java, which can be integrated in libraries and other software projects.

4.2 Data structures

The following messages have been defined for the storage and distribution of data, using the proto3-format.

4.2.1 Package: dataelements

The package “dataelements” contains the “Data” message, which defines a wrapper object, to allow the storage/distribution of one of the given data types as a unified object. It always contains only one of the listed objects.

[1] https://developers.google.com/protocol-buffers

```
syntax = "proto3";

package dataelements;

import "environment.proto";
import "neurocognitive.proto";
import "traffic.proto";
import "vehicle.proto";

option java_package = "de.htwsaar.fgvt.dataaggregator.library.dataelements";
option java_multiple_files = true;

message Data {
	oneof message {
		environmentaldata.EnvironmentData environmentData = 1;
		vehicledata.VehicleData vehicleData = 2;
		trafficdata.TrafficData trafficData = 3;
		neurocognitivedata.NeurocognitiveData neuroData = 4;
	}
}
```

4.2.2 Package: metadata

The package “metadata” contains the message “MetaData”. This message defines a set of information to put the attached data in context.

```
syntax = "proto3";

package metadata;

import "general.proto";

option java_package = "de.htwsaar.fgvt.dataaggregator.library.dataelements";
option java_multiple_files = true;

message MetaData {
	int32 sourceId = 1;
	int64 tsBegin = 2;
	int32 validity = 3;
	general.GnssPosition gnssPosition = 4;
}
```

4.2.3 Package: general

The package “general” contains multiple message definitions, which are used in messages of other packages, which is mostly the definition of messages to store geographic information.

```
syntax = "proto3";

package general;

option java_package = "de.htwsaar.fgvt.dataaggregator.library.general";
option java_multiple_files = true;

message GnssData {
	int32 ts_delta = 1;
	PositionVector gnssPositionVector = 2;
	GnssMetaData gnssMetaData = 3;
}

message Wgs84Position {
	int64 latitude = 1;
	int64 longitude = 2;
}

message GnssPosition {
	Wgs84Position position = 1;
	int32 altitude = 2;
}

message PositionDelta {
	sint32 deltaLatitude = 1;
	sint32 deltaLongitude = 2;
	sint32 deltaAltitude = 3;
}

message PositionVector {
	PositionDelta gnssPositionDelta = 1;
	int32 heading = 2;
	int32 speed = 3;
}

message GnssMetaData {
	int32 numberOfSatellites = 1;
}
```

```
message GeoArea {
	oneof message {
		CircularGeoArea circularGeoArea = 1;
		RectangularGeoArea rectangularGeoArea = 2;
	}
}

message CircularGeoArea {
	Wgs84Position centerPoint = 1;
	int32 radius = 2;
}

message RectangularGeoArea {
	Wgs84Position centerPoint = 1;
	double metersToShortSide = 2;
	double metersToLongSide = 3;
	double azimuthAngle = 4;
}


// General
message Tuple {
	string key = 1;
	oneof value {
		double doubleValue = 2;
		int32 intValue = 3;
		int64 longValue = 4;
		string stringValue = 5;
		bool booleanValue = 6;
		bytes bytesValue = 7;
	}
}

message Value {
	oneof message {
		double doubleValue = 1;
		int32 intValue = 2;
		int64 longValue = 3;
		string stringValue = 4;
		bool booleanValue = 5;
		bytes bytesValue = 6;
	}
}
```

4.2.4 Package: environmentaldata

The package “environmentaldata” defines one of the main datatypes, used in the system. The “EnvironmentData” message contains multiple entries of “WeatherData” describing the current weather, in addition it has a MetaData object with necessary supportive information.

```
syntax = "proto3";

package environmentaldata;

import "metadata.proto";

option java_package = "de.htwsaar.fgvt.dataaggregator.library.environmentaldata";
option java_multiple_files = true;

message EnvironmentData {
	metadata.MetaData metaData = 1;
	repeated WeatherData weatherDataSets = 2;
}

// Weather Data
message WeatherData {
	int32 ts_delta = 1;
	sint32 temperature = 2; // in 1/10 degree Celsius
	int32 humidity = 3; // in percent
	int32 airPressure = 4; // in 1/10 hPa
	int32 windSpeed = 5; // in km/h
	int32 windDirection = 6; // in degree
	int32 precipitation = 7; // in 1/10 l/m^2
}
```

4.2.5 Package: vehicledata

The package “vehicledata” defines among others the main datatype “VehicleData”, which is mostly used by the VehicleDataAggregator. It defines message types containing data collected from the vehicle.

```
syntax = "proto3";

package vehicledata;

import "general.proto";
import "metadata.proto";

option java_package = "de.htwsaar.fgvt.dataaggregator.library.vehicledata";
option java_multiple_files = true;

message VehicleData {
	metadata.MetaData metaData = 1;
	repeated CanData canDataSets = 2;
	repeated general.GnssData gnssDataSets = 3;
}

// CAN message types
message CanData {
	int32 ts_delta = 1;
	// different CAN data elements follow here
	repeated CanTuple canKeyValue = 2;
}

// General
message CanTuple {
	SensorKey key = 1;
	oneof value {
		double doubleValue = 2;
		int32 intValue = 3;
		int64 longValue = 4;
		string stringValue = 5;
		bool booleanValue = 6;
		bytes bytesValue = 7;
	}
}
```

```
enum SensorKey {
	VD_CAN_BRAKE_ACTUATION = 0; // Brake pedal applied signal. Type Boolean (false = OFF, true = On)
	VD_CAN_CLUTCH_SWITCH_ACTUATION = 1; // Signals the state of the clutch (switch). Type: Boolean (false = Off, true = On)
	VD_CAN_CURRENT_GEAR = 2; // Byte. Indicates the current gear of the vehicle.0 = Idle, 1 = One, ... 254 Reverse
	VD_CAN_DIRECTION_OF_DRIVING = 3; // Describes the direction of driving (0 = unknown, 1 = standstill, 2=forward, 3=reverse).
	VD_CAN_DOOR_POS_FRONT_RIGHT = 4; // Byte: 0 = Closed, 1 = Ajar, 2 = Open
	VD_CAN_DOOR_POS_FRONT_LEFT = 5; // Byte: 0 = Closed, 1 = Ajar, 2 = Open
	VD_CAN_DOOR_POS_REAR_RIGHT = 6; // Byte: 0 = Closed, 1 = Ajar, 2 = Open
	VD_CAN_DOOR_POS_REAR_LEFT = 7; // Byte: 0 = Closed, 1 = Ajar, 2 = Open
	VD_CAN_DOOR_POS_BOOT = 8; // Byte: 0 = Closed, 1 = Ajar, 2 = Open
	VD_CAN_EMERGENCY_LIGHTNING = 9; // The activity of emergency vehicle lighting (AKA light bar, beacon). Additional to standard vehicle lighting e.g. hazard warning lights for use on emergency, maintenance or transportation response vehicles. Type: Boolean (false = Off, true = On)
	VD_CAN_FOG_LIGHT_FRONT = 10; // Boolean
	VD_CAN_FOG_LIGHT_REAR = 11; // Boolean
	VD_CAN_HAZARD_WARNING = 12; // Boolean
	VD_CAN_HIGH_BEAM = 15; // Boolean
	VD_CAN_HORN = 16; // Boolean
	VD_CAN_LATERAL_ACCELERATION = 17; // Double, meter per seconds squared
	VD_CAN_LONGITUDINAL_ACCELERATION = 18; // Double meter per seconds squared
	VD_CAN_MODEL_TYPE = 19; // Byte, 0 = Notch, 1 = Hatch, 2 = Short, 3 = Station Wagon, 4 = Cabrio, 5 = Coupe, 6 = Offroad, 7 = Pickup, 8 = MPV, 9 = SUV
	VD_CAN_PANIC_BRAKING = 20; // Boolean
	VD_CAN_PEDAL_FORCE = 21; // Byte, percent
	VD_CAN_RAIN_INTENSITY = 22; // Short, intensity in percent, 0 = no rain, 250 max rain, 251 = Invalidity Border
	VD_CAN_SPECIAL_VEHICLE_TYPE = 23;// Byte, 0 = No special vehicle, 1 = Police, 2 = Fire brigade, 3 = Ambulance or emergency, 4 = Heavy transport or oversize load, 5 = Public transport, 6 = Taxicab, 7 = Slow vehicle, 8 = Vehicle for handicapped persons, 9 = Building site vehicle, 10 = Agricultural vehicle or machines, 11 = Accompanying vehicle, 12 = Unknown
	VD_CAN_SPEED = 24; // Double, km/h
	VD_CAN_STEERINGWHEEL_ANGLE = 25; // Double, degree
	VD_CAN_STEERINGWHEEL_ANGULAR_VELOCITY = 26; // Double, degree per seconds
	VD_CAN_TURN_SIGNAL_LEVER = 27; // Byte, 0 = idle, 1 = left, 2 = right
	VD_CAN_VEHICLE_HEIGHT = 28; // Integer. In millimeter
	VD_CAN_VEHICLE_LENGTH = 29; // Integer. In millimeter
	VD_CAN_VEHICLE_WIDTH = 30; // Integer. In millimeter
	VD_CAN_WIPER_FRONT = 31; //Byte, 0 = idle, 1 = normal, 2 = fast, 3 = intermittent
	VD_CAN_WIPER_REAR = 32; //Byte, 0 = idle, 1 = normal, 2 = fast, 3 = intermittent
	VD_CAN_YAW_RATE = 33; // Double, degrees per second
	VD_CAN_GNSS_ACCELERATION = 34; // Double, meter per seconds squared
	VD_CAN_GNSS_ALTITUDE = 35; // Double, meter
	VD_CAN_GNSS_HEADING = 36; // Double, degree
	VD_CAN_GNSS_LATITUDE = 37; // Double, degree -90 ... 90
	VD_CAN_GNSS_LONGITUDE = 38; // Double, degree -180 ... 180
	VD_CAN_GNSS_TIMESTAMP = 39; // Long, ms since epoch
	VD_CAN_GNSS_SPEED = 40; // Double, km/h
}
```

4.2.6 Package: traffichelper

The package “traffichelper” contains supportive message definitions for other packages, like topology information for crossroads. These data types are mostly used by the TrafficData Aggregators.

```
syntax = "proto3";

package traffichelper;

import "general.proto";
option java_package = "de.htwsaar.fgvt.dataaggregator.library.trafficdata.helper";
option java_multiple_files = true;

enum StationType {
	UNKNOWN = 0;
	PEDESTRIAN = 1;
	CYCLIST = 2;
	MOPED = 3;
	MOTORCYCLE = 4;
	PASSENGER_CAR = 5;
	BUS = 6;
	LIGHT_TRUCK = 7;
	HEAVY_TRUCK = 8;
	TRAILER = 9;
	SPECIAL_VEHICLES = 10;
	TRAM = 11;
	ROAD_SIDE_UNIT = 15;
}

message DetectedObject {
	int64 objectId = 1;
	general.Wgs84Position position = 2;
	ObjectClassification classification = 3;
	int32 speed = 4;
	int32 heading = 5;
}

enum ObjectClassification {
	VEHICLE_OBJECT = 0;
	PEDESTRIAN_OBJECT = 1;
	BUS_OBJECT = 2;
	TRAM_OBJECT = 3;
	TRAIN_OBJECT = 4;
}

message Lane {
	string title = 1;
	int32 laneId = 2;
	int32 laneWidth = 3;
	int32 speedLimit = 4;
	repeated int32 connectedLanes = 5;
	LaneType laneType = 6;
	AllowedManeuvers allowedManeuvers = 7;
	LaneDirection laneDirection = 8;
}
```

```
enum LaneType {
	OVERLAPPING_LANE_DESCRIPTION_PROVIDED = 0;
	MULTIPLE_LANES_TREATED_AS_ONE_LANE = 1;
	OTHER_NON_MOTORIZED_TRAFFIC_TYPES = 2;
	INDIVIDUAL_MOTORIZED_VEHICLE_TRAFFIC = 3;
	BUS_VEHICLE_TRAFFIC = 4;
	TAXI_VEHICLE_TRAFFIC = 5;
	PEDESTRIAN_TRAFFIC = 6;
	CYCLIST_VEHICLE_TRAFFIC = 7;
	TRACKED_VEHICLE_TRAFFIC = 8;
}

enum LaneDirection {
	INGRESS = 0;
	EGRESS = 1;
}

message AllowedManeuvers {
	bool maneuverStraightAllowed = 1;
	bool maneuverLeftAllowed = 2;
	bool maneuverRightAllowed = 3;
	bool maneuverUTurnAllowed = 4;
	bool maneuverLeftTurnOnRedAllowed = 5;
	bool maneuverRightTurnOnRedAllowed = 6;
	bool maneuverLaneChangeAllowed = 7;
	bool maneuverNoStoppingAllowed = 8;
	bool yieldAllwaysRequired = 9;
	bool goWithHalt = 10;
	bool caution = 11;
	bool reserved1 = 12;
}

enum SignalPhase {
	OUT = 0;
	GREEN = 1;
	ORANGE = 2;
	RED = 3;
	RED_ORANGE = 4;
	RED_ORANGE_GREEN = 5;
}

message StateTimeSpeed {
	SignalPhase phase = 1;
	int32 startTime = 2;
	int32 endTime = 3;
}

message MovementState {
	int32 signalGroupId = 1;
	repeated StateTimeSpeed stateTimeSpeeds = 2;
}
```

4.2.7 Package: trafficdata

The package “trafficdata” contains one of the main data types: “TrafficData”. It defines an object containing multiple data sets of V2xMessages, which have been collected by the TrafficDataAggregator clients.

```
syntax = "proto3";

package trafficdata;

import "general.proto";
import "metadata.proto";
import "traffichelper.proto";

option java_package = "de.htwsaar.fgvt.dataaggregator.library.trafficdata";
option java_multiple_files = true;


message TrafficData {
	metadata.MetaData metaData = 1;
	repeated V2xCam cams = 2;
	repeated V2xCpm cpms = 3;
	repeated V2xDenm denms = 4;
	repeated V2xMap maps = 5;
	repeated V2xSpat spats = 6;
}

message V2xCpm {
	int32 ts_delta = 1;
	int64 stationId = 2;
	traffichelper.StationType stationType = 3;
	repeated traffichelper.DetectedObject detectedObjects = 4;
}

message V2xDenm {
	int32 ts_delta = 1;
	int64 stationId = 2;
	int32 causeCode = 3;
	int32 subCauseCode = 4;
	general.GeoArea relevanceArea = 5;
	/* Removed: int 32 validity = 6. */
	int64 detectionTime = 7;
	int64 expiry_time = 8;
}

message V2xMap {
	int32 ts_delta = 1;
	int64 intersectionId = 2;
	string title = 3;
	repeated traffichelper.Lane lanes = 4;
}

message V2xSpat {
	int32 ts_delta = 1;
	int32 intersectionId = 2;
	map < int32, traffichelper.SignalPhase > laneSignalPhases = 3;
	int32 regionId = 4;
	repeated traffichelper.MovementState movementStates = 5;
}
```

```
message V2xCam {
	int32 ts_delta = 1;

	int64 stationId = 2;
	int32 cam_ts_delta = 3;
	traffichelper.StationType stationType = 4;
	general.PositionDelta referencePositionDelta = 5;
	int32 speed = 6;
	int32 heading = 7;
	enum SpecialVehicle {
		NO_SPECIAL_VEHICLE = 0;
		PUBLIC_TRANSPORT = 1;
		SPECIAL_TRANSPORT = 2;
		DANGEROUS_GOODS = 3;
		ROAD_WORKS = 4;
		RESCUE = 5;
		EMERGENCY = 6;
		SAFETY_CAR = 7;
	}

	SpecialVehicle specialVehicle = 8;
	enum DriveDirection {
		FORWARD = 0;
		BACKWARD = 1;
		UNAVAILABLE = 2;
	}

	DriveDirection driveDirection = 9;
	int32 vehicleLength = 10;
	int32 vehicleWidth = 11;
	enum VehicleLengthConfidenceIndication {
		NO_TRAILER_PRESENT = 0;
		TRAILER_PRESENT_WITH_KNWON_LENGTH = 1;
		TRAILER_PRESENT_WITH_UNKNOWN_LENGTH = 2;
		TRAILER_PRESENCE_IS_UNKNOWN = 3;
		UNAVAILABLE_VLCI = 4;
	}

	VehicleLengthConfidenceIndication vehicleLengthConfidenceIndication = 12;
	bool sirenInUse = 13;
	bool lightBarInUse = 14;
}
```

4.2.8 Package: neurocognivitedata

The package “neurocognitivedata” contains the main data type “NeurocognitiveData”, which holds information about the current state of the driver.

```
syntax = "proto3";

package neurocognitivedata;

import "metadata.proto";

option java_package = "de.htwsaar.fgvt.dataaggregator.library.neurocognitivedata";
option java_multiple_files = true;

message NeurocognitiveData {
        metadata.MetaData metaData = 1;
}
```

5 Data supplementation

5.1 Problem

Not all data sources provide data at any time. This can be caused by different things e.g. a missing sensor, which is available in another VUT, a defect sensor that sends wrong data, or missing data input from an external information provider such as a weather provider. The challenge for the data evaluation is to work with different partly filled data sets combined by the data fusion. An indication is needed, that is able to distinguish whether there is no data due to a malfunction or whether there is nothing to detect. In case of using different sensor sets in the vehicle, that indication can be implemented by a configuration. Supposing of missing cooperative awareness or collective perception messages, there is no way to detect a misbehavior of the communication system.

There are essentially two different groups of aggregators. One group can collect data autonomously without the help of other networked vehicles and infrastructure components; the other one depends in most cases on these networked components and vehicles. The infrastructure units with cameras for the detection of vehicles and pedestrians belong to the self-sufficient group, which can work in their field of vision without additional components or assistance. The cooperative vehicles need other vehicles with V2X communication to collect traffic data. Otherwise, they will only receive limited data from local sensors or their own vehicle data without knowing their exact environment. This creates data-gaps or data-shadows in the multitude of data that is collected. This means that incomplete data is available for certain measurements in subsections.

5.2 Approach

5.2.1 Data validity and expiry date

For every information provided by sensors, the communication system or external service providers, a validity can be defined. Some data records have a very short validity time e.g. an acceleration sensor in the vehicle, because the sensor value typically changes within milliseconds. Other information can be valid for several minutes e.g. weather conditions. That means every collected data record needs to be tagged with an expiration date. This expiration date should primarily ensure the time of how long a data record is meaningful. For example, the position of a traffic object, a warning or the weather condition are not kept in the evaluation for longer than this data is valid. The expiry date allows reducing the amount of records but also indicates if there is a lack of data. It also defines the amount of how often a data record needs to be updated.

Figure 9 shows an example of the usage of validity and update rate. *Data A* represents a misconfigured update rate for its validity, *Data B* represents a well configured update rate for the given validity. As the case of Data A demonstrates, if the time to the next update is not at least twice as long as the corresponding validity, every missing data record leads to a data gap in the data fusion. Data B is well configured as a missing data record can be compensated by the validity of the previous data record.

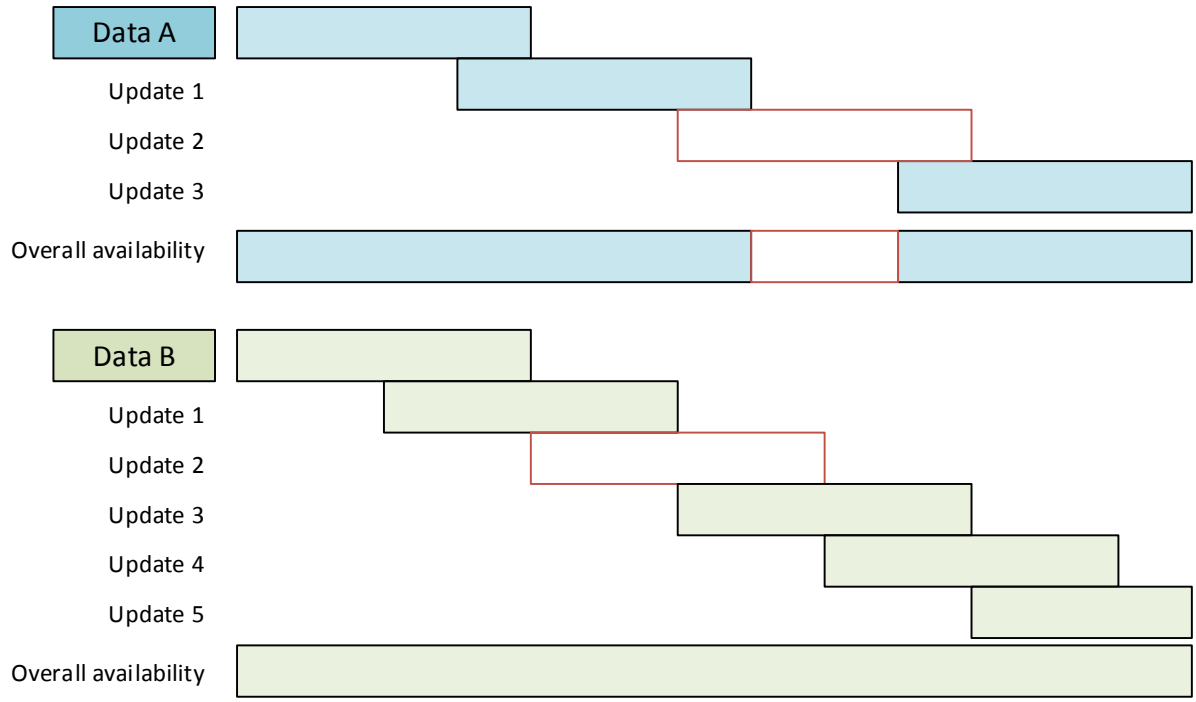

Figure 9 Validity example

5.2.2 Interpolation

Using Fast Fourier Transformation (FFT) missing data can be interpolated [6].

5.2.3 Artificial Intelligence

To supplement data sets with data gaps, artificial intelligence (AI) can be used. Based on a trained machine-learning model for example, the AI can use complete data sets as training data. The input of the supplementation AI is the data set with the missing gap and again after filling it. This is for improving the results of the AI by extending the trained data set. For every data type, a different set of data is used to reconstruct the gap. For example, for a missing speed value from a vehicle sensor, the position of the vehicle, its acceleration values as well as brake and clutch switch actuation are used to calculate the gap in the speed records and reproduce it.

C. Literature

[1] Google Developers - Protocol Buffers, https://developers.google.com/protocol-buffers

[2] REST

[3] iKoPA Deliverable

[4] iKoPA Veröffentlichung

[5] Ramanarayanan Viswanathan, Pramod K. Varshney, Distributed Detection With MultipleSensors: Part I—Fundamentals, PROCEEDINGS OF THE IEEE, VOL. 85, NO. 1, JANUARY 1997, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=554208

[6] SINGHAL, VLACH; Interpolation Using the Fast Fourier Transform; PROCEEDINGS OF THE IEEE, DECEMBER 1972; 1972; https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1450888

[7] ETSI EN 302 637-2 V1.4.1 (2019-04), Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service

[8] ETSI EN 302 637-3 V1.3.1 (2019-04), Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service

[9] SAE J2735™ (SEP2015), Dedicated Short Range Communications (DSRC) Message Set Dictionary

[10] ETSI TR 103 562 V2.1.1 (2019-12), Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Analysis of the Collective Perception Service (CPS); Release 2