D5.4

TeamMate System Architecture incl. open API for 3rd Cycle

Project Number:	690705
Classification	Public
Deliverable No.:	D5.4
Work Package(s):	WP5
Document Version:	Vs. 0.4
Issue Date:	31.12.2018
Document Timescale:	Project Start Date: September 1, 2016
Start of the Document:	Month 27
Final version due:	Month 28
Deliverable Overview:	**Main document:** D5.4 Appendix 1: D5.4-appendix_1_confidential.docx
Compiled by:	Stefan Suck, OFF
Authors:	Adam Knapp, BIT Stefan Suck, OFF Andrea Castellano, REL Elisa Landini, REL
Technical Approval:	Fabio Tango, CRF
Issue Authorisation:	Andreas Lüdtke, OFF

DISTRIBUTION LIST		
Copy type[1]	Company and Location	Recipient
T	AutoMate Consortium	all AutoMate Partners

[1] Copy types: E=Email, C=Controlled copy (paper), D=electronic copy on Disk or other medium, T=Team site (Sharepoint)

RECORD OF REVISION		
Date	Status Description	Author
28.11.2018	Deliverable structure	Stefan Suck
04.12.2018	V2X messages definition	Adam Knapp
11.12.2018	Multimodal HMI message and recapitulation of Teammate overview	Andrea Castellano, Elisa Landini
19.12.2018	Include further messages, figures, and description of date flow and inter action	Stefan Suck
20.12.2018	Include feedback from BIT	Adam Knapp

Table of Contents

List of Abbreviations

A2H ..... *Automation to Human Cooperation*
DENM ..... *Decentralized Environmental Notification Message*
DIR ..... *Driver Intention Recognition*
H2A ..... *Human to Automation Cooperation*
protobuf ..... *Google Protocol Buffers*
V2I ..... *Vehicle to Infrastructure*

List of Figures

List of Tables

Executive Summary

This deliverable D5.4 presents the updates to the global AutoMate System architecture and the TeamMate Application Programming Interface (API) for the third cycle since D5.1. Moreover, a short recapitulation of the cooperation modes is given, including the concept of Automation to Human (A2H), as well as Human to Automation (H2A) communication.

An explanation of the overall architecture and the dataflow within the SW/HW construct is provided since there were some minor changes for cycle three, while data structures and protocols for communication stayed unchanged. Additionally sequence charts are provided to give a more detailed view on the inter actions between enablers inside the architecture.

Concerning the TeamMate API, updates of messages for the communication between enablers are provided.

1 Introduction

The purpose of the Automate project is to build a complete software/hardware concept for the teammate car. Therefore, each module has to be developed and programmed and finally all modules have to be put together. Due to the number and complexity of these modules, the composition can be become a complex task and needs to be well coordinated. Also one needs to make sure, that the architecture works on all demonstrators of the Automate project considering their different initial architectures. For this reason WP5 deals only with the issues mentioned above and this document presents the updates since the previous deliverable for the global TeamMate architecture, the interfaces between the modules, as well as the common data formats standards and communication protocols.

This document presents the global AutoMate System architecture focussing on changes and updates for the 3rd cycle in comparison to the status reported in D5.1 "TeamMate System Architecture incl. open API for 3rd Cycle". The data structures reported in D5.1 stayed unchanged. Thus, there is no section dedicated to those.

This document is structured as follows. In Section 2 a brief recapitulation of the enablers and the TeamMate cooperation modes including the concepts of automation to human (A2H) and human to automation (H2A) communication is given. Section 3 describes the global TeamMate architecture concept, concerning updates of dataflow within the software construct and a clarification of the interaction between the enablers. Also updates of

protocols for communication. Subsequently in Section 5 describes updates of the TeamMate Application Programming Interface (API) concerning data formats and messages for interfacing the modules and the communication of information between components in the TeamMate ecosystem.

2 Global overview of the TeamMate system

AutoMate is based on the concept of mutual complementarity between the driver and the automation: the support between the human and the technological agents is given through the cooperation of the agents considered as team members.

The cooperation is bidirectional: while the Automation to Human Cooperation (A2H) is used to complement the human limits, the Human to Automation Cooperation (H2A) is implemented to allow the driver to support the automation to overcome its limits.

According to AutoMate concept, the cooperation is made of two types of support: in perception and in action. The complementarity between the driver and the automation is the conceptual solution to compensate the reciprocal limitations, while the cooperation is how the complementarity is implemented. A more detailed description of the project's concept and an overview of the cooperation are given in D5.1.

In order to concretely implement the cooperation, technical and conceptual enablers are needed: the enablers too are listed and described in D5.1. In this document any possible changes to the role of the enabler is described: moreover, the concrete implementation after the 3rd development and integration cycle, including communication and data-structure, are reported.

3 TeamMate functional Architecture

As described in the previous deliverable D5.1 "TeamMate System Architecture incl. open API for 2nd Cycle" the overall TeamMate functional Architecture is based on several components called enablers.

Main requirements for the architecture are:

- flexibility in terms of configuration
- portability in terms of integration into existing platforms.

The configuration flexibility is reach via the component based approach where the components are separated dependent on their concerns, allowing different setups to implement different interactions. It is supported by the publisher-subscriber messaging pattern for the communication between the components. More precise each component may act as a server that provides services to other components. Simultaneously, each component may act as a client by requesting services from other components.

The portability is supported by fact that there are only few interfaces to surrounding systems which have to be provided by the vehicle or the simulator and the usage of the communication protocols.

For the sake of clarity in the overall architecture shown in Figure 1 the message exchange is illustrated as communication via an implied message bus, more details a provided later in separate pictures. The figure shows the relations of the automate enablers among each other and together with a given platform (vehicle or simulator) during the third cycle. The depicted

layout is kept close to the current implementation while staying general enough to be applicable for all demonstrators.

The AutoMate enablers support different functional steps: "data processing & fusion", "interpretation", and "planning & actions" which are represented by the corresponding sections of Figure 1 ordered from left to right. The existing vehicle or simulator may already have modules which also perform one or more of the aforementioned functional steps, e.g., a platform often includes the sensors themselves and the corresponding data fusion components. However, the TeamMate system does not need to know how these internal modules of the platform work or interact. Thus, wide parts of the existing platform are considered as a black box.

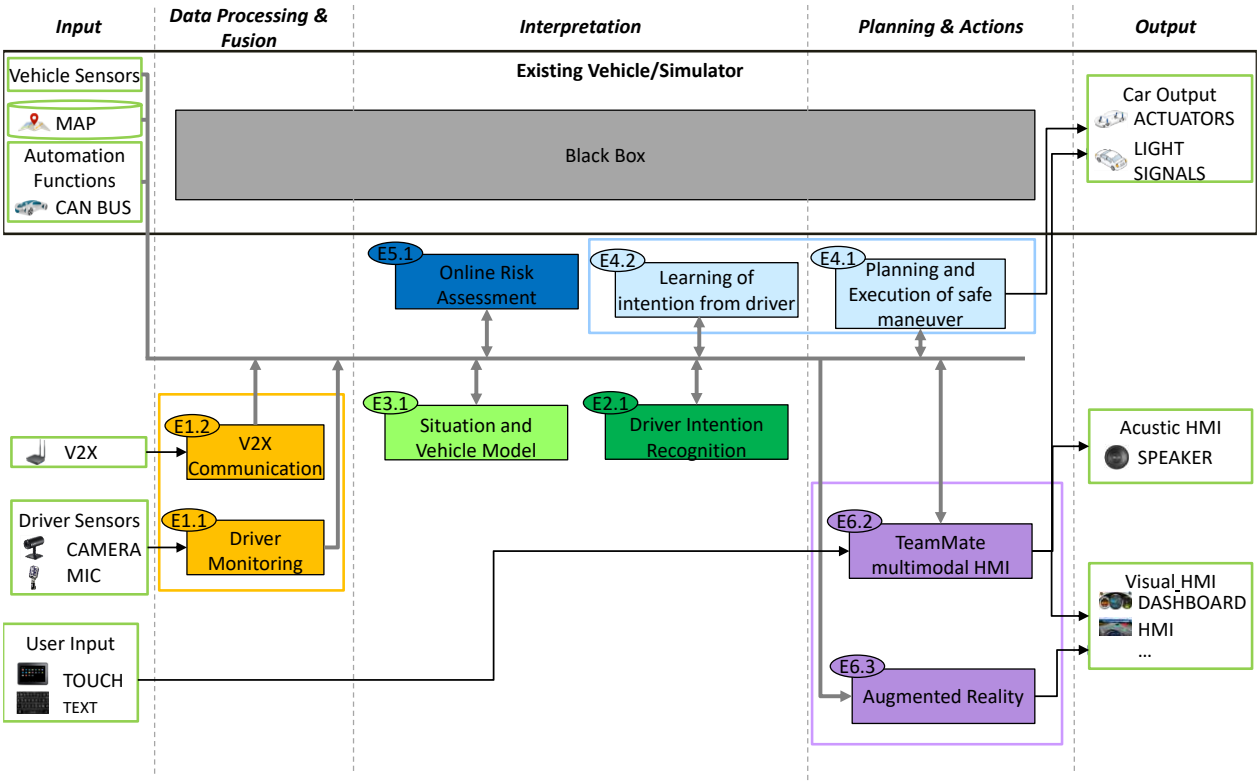

Figure 1: TeamMate overall functional architecture for cycle three

The most prominent changes in comparison to D5.1 and concerning the depictured overall TeamMate Architecture are first of all, that now the whole communication between enabler components is carried out via the message bus. Communication outside of the message bus happens only for direct communication between enablers and certain input or output interfaces, like the driver sensors or the visual HMI etc.

Further, the enablers "Instruments Cluster", "Central Display", "Ambient Lights", and "HUD" were merged into the single enabler "TeamMate multimodal HMI". This was decided in WP4 since these enablers are designed

in a harmonic and interrelated way, where each of the original enabler interacts with the others according to the overall HMI strategy.

Additionally, the enabler "Interaction Modality" was removed since it is no longer considered as a stand-alone software module, instead it provides guidelines which should be implemented by the corresponding HMI software enablers. Therefore, the enabler "TeamMate multimodal HMI" is now directly connected to the user input, the natural interaction input like pedal positions and steering wheel angle are received via the TeamMate bus from the CAN of the demonstrator platform.

Inside the architecture there are several data flows. There were only slight changes compared to the description from D5.1. For the sake of the document the flows shall now be described from the perspective of the enablers.

The **Driver State Monitoring (E1.1)** receives its data directly from sensors related to the driver like a camera. The module infers the driver state in terms of drowsiness, distraction, attention to predefined Areas of Interest. The driver state is provided as output data and can be consumed by other enablers. Currently the *TeamMate multimodal HMI (E6.2)* trigger messages for the driver, e.g., to suggest a transition to automatic mode, based on the diver state received from E1.1.

The **V2X Communication (E1.2)** is directly connected to a V2X data receiver. The received data is interpreted and as shown in Figure 1 the information is communicated to other enablers. Currently the *TeamMate*

multimodal HMI (E6.2) uses V2X data about certain conditions on the route to inform the driver and trigger suggestions for mode transitions to manual or shared control.

The **Driver Intention Recognition (E2.1)** consumes map data, ego vehicle data, and object data from the car or simulator via the message bus. The recognized intention probabilities, for example for a lane change, are then sent to the bus. As shown in Figure 2 this information is consumed by the *Augmented Reality (E6.3)* and the *TeamMate multimodal HMI (E6.2)* where the intention of the driver can be visualized in order to inform the driver that the TeamMate car is aware of his intention. Additionally the *TeamMate multimodal HMI (6.2) can* give the driver an opportunity to confirm or decline an overtaking manoeuvre if the corresponding intention was detected.

The **Situation and Vehicle Model (E3.1)** receives its input data, i.e., map data, ego vehicle data, and object data via the message bus from the demonstrator platform. The output data, an interpretation of the traffic situation and a spatial and temporal prediction of traffic participants within sensor range can then be received by further enablers. Currently this data is consumed by the *Online Risk Assessment (E5.1)* which is also illustrated in Figure 2.

The **Planning and Execution of safe maneuver (E4.1)** receives map data, ego vehicle data, and object data from the car or simulator via the

message bus. It attempts to plan a trajectory which is then evaluated by the *Online Risk Assessment (E5.1)*. After the evaluation result is received the trajectory is directly provided to the output interface of the simulator or car so that the vehicle can follow the planned path. Additionally, the trajectory data is also consumed by the *Augmented Reality (E6.3)*. This interaction is also illustrated in Figure 2.

The **Learning of intention from driver (E4.2)** is closely related to the *Driver Intention Recognition (E2.1)*. It receives map data, ego vehicle data, and object data from the vehicle or simulator. Additionally it has also access to the driver model storage of the DIR. The enabler updates the parameters of the DIR model based on observed evidence. After an update the new model parameters are stored to the DIR model and the DIR is informed about the update to load the new model in order to operate with the new parameters. This interaction is also illustrated in Figure 3.

The **Online Risk Assessment (E5.1)** consumes traffic prediction data from the *Situation and Vehicle Model (E3.1)*, as well as map data, object data, and ego vehicle data from the car or simulator. All data is again received via the message bus. Additionally, trajectories from the *Planning and Execution of safe maneuver (E4.1)* are received in order to assess the safety of these trajectories. The assessment result of the trajectories is then made available for other enablers via the message bus. As shown in Figure 2 currently the

Planning and Execution of safe maneuver and the *Augmented Reality* receive the risk assessment.

The **TeamMate multimodal HMI (6.2)** receives map data, ego vehicle data and objects data from the vehicle or the demonstrator. As illustrated in Figure 4 the HMI also consumes data from the *VX2 Communication* and the *Driver State Monitoring*. As mentioned before, the data from both enablers might be used to trigger suggestions for the driver to change the current automation mode. As shown in Figure 2 HMI also receives the output of the *Driver Intention Recognition (E2.1)*, which may trigger a dialog with the driver to confirm or decline a manoeuvre. The decision of the driver is then provided to other and enablers and currently consumed by the *Planning and Execution of safe maneuver (E4.1)*. Additionally the HMI may process data from the user input interfaces, e.g., text or touch. It will also send the natural input, like steering wheel or pedal interaction directly to the vehicle. The output of the HMI is directly provided to the driver via the corresponding visual or acoustic interface.

The **Augmented Reality HMI (E6.3)** receives, as shown in Figure 2, the inferred intention from the DIR, the trajectory from the *Execution of safe maneuver (E4.1)*, and the risk assessment result for the trajectory from the *Online Risk Assessment (E5.1)*. Besides the visualization of the intention of the driver, or the chosen trajectory the Augmented Reality HMI can combine

this information to warn the driver if an intention or manoeuvre would be currently unsafe.

3.1 Enabler Interaction Sequences

The following sequence charts which have already been referenced before shall illustrate main interactions between the enablers inside the TeamMate architecture and give a more detailed view on the underlying data flows than the high level picture given in Figure 1. The charts are focussed on the current implementation. The *Initialization* part refers to data which is usually received just once, in contrast to messages of the *Processing* part which might occur multiple times or in loops. While it can be argued that the map message is probably received more than once it is currently implemented in way that the map of the whole track is contained in a single message. The messages that are shown in the charts are not necessarily on all demonstrators implemented as messages as those. For example, the *trafficPrediction* message from the *Situation and Vehicle Model* to the *Online Risk Assessment* (Figure 2) is sometimes carried out via a shared object, since it is more feasible is some cases and the message is not consumed by other enablers.

In Figure 2 the interaction sequence for the creation and confirmation of an overtaking is illustrated. This for example will take place during the Peter scenario while the vehicle is in automated mode. All shown enablers continuously receive their input messages. To implement the H2A cooperation the *multimodal HMI* asks the driver to confirm or decline an overtaking manoeuvre, if the DIR detects a lane change intention. If the

driver confirms, the HMI sends *releaseOvertaking* which is received by the *Planning and Execution of safe manoeuvre* enabler. An overtaking trajectory is planned and made available to the enablers in the TeamMate system. The trajectory is received by the *Online Risk Assessment* and evaluated. The assessment result is sent to the *Planning and Execution of safe manoeuvre.* If the trajectory is considered as safe, it is sent directly to the vehicle.

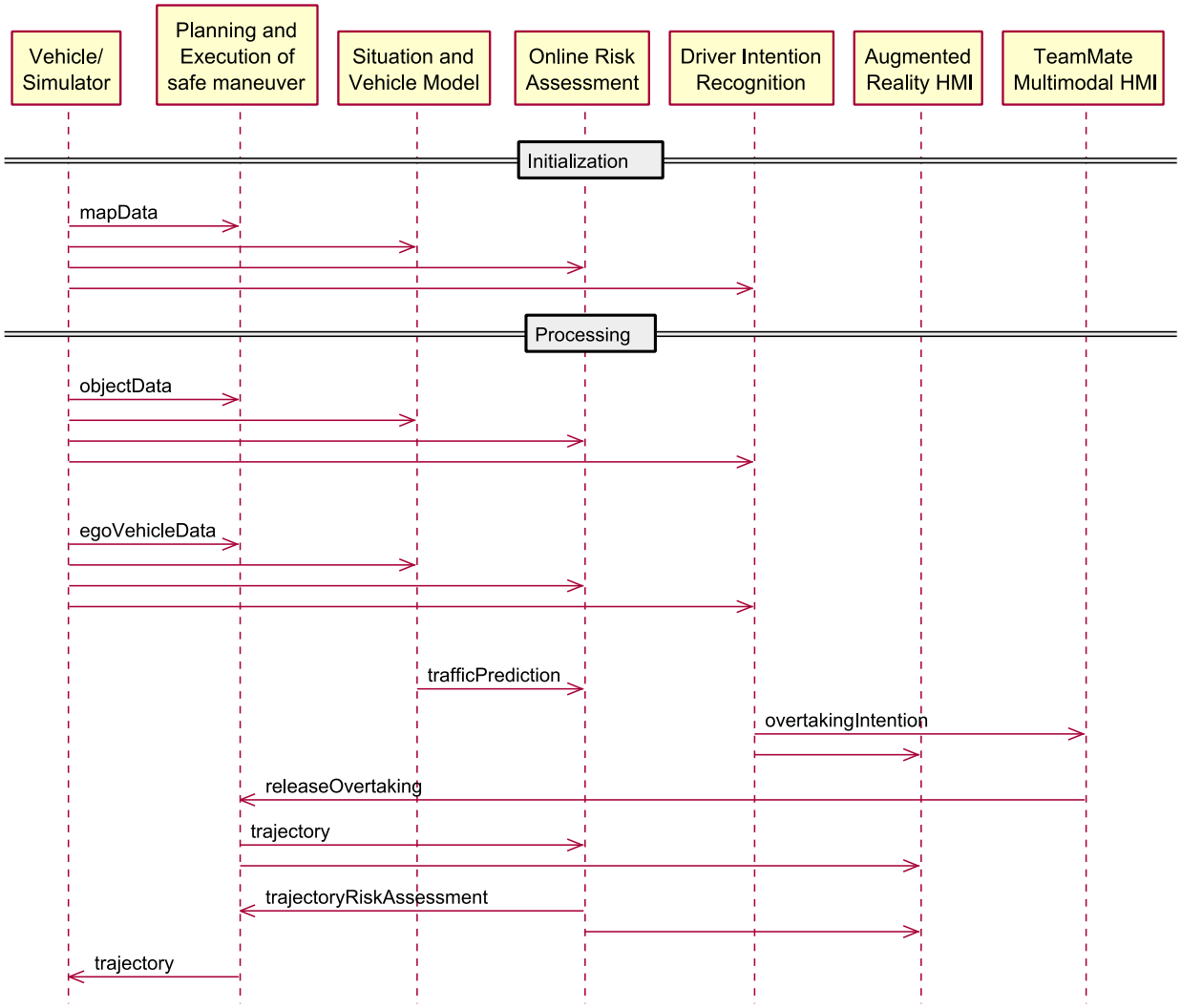

Figure 2: Interaction of enablers and components which are involved in the assistance for an overtaking manoeuvre

Figure 3 focuses on the interaction of *Driver Intention Recognition* and *Learning of intention from driver*. Both enablers initially load the same driver model and map data, and then continuously receive their input data. While the DIR infers the current intention of the driver, the *Learning of intention from driver* gathers data which is used to update its instance of the driver

model. After this the stored driver model is updated and the DIR is informed via *updatedModelReady* about this update. Then the DIR will reload the driver model in order to work with the new model parameters.

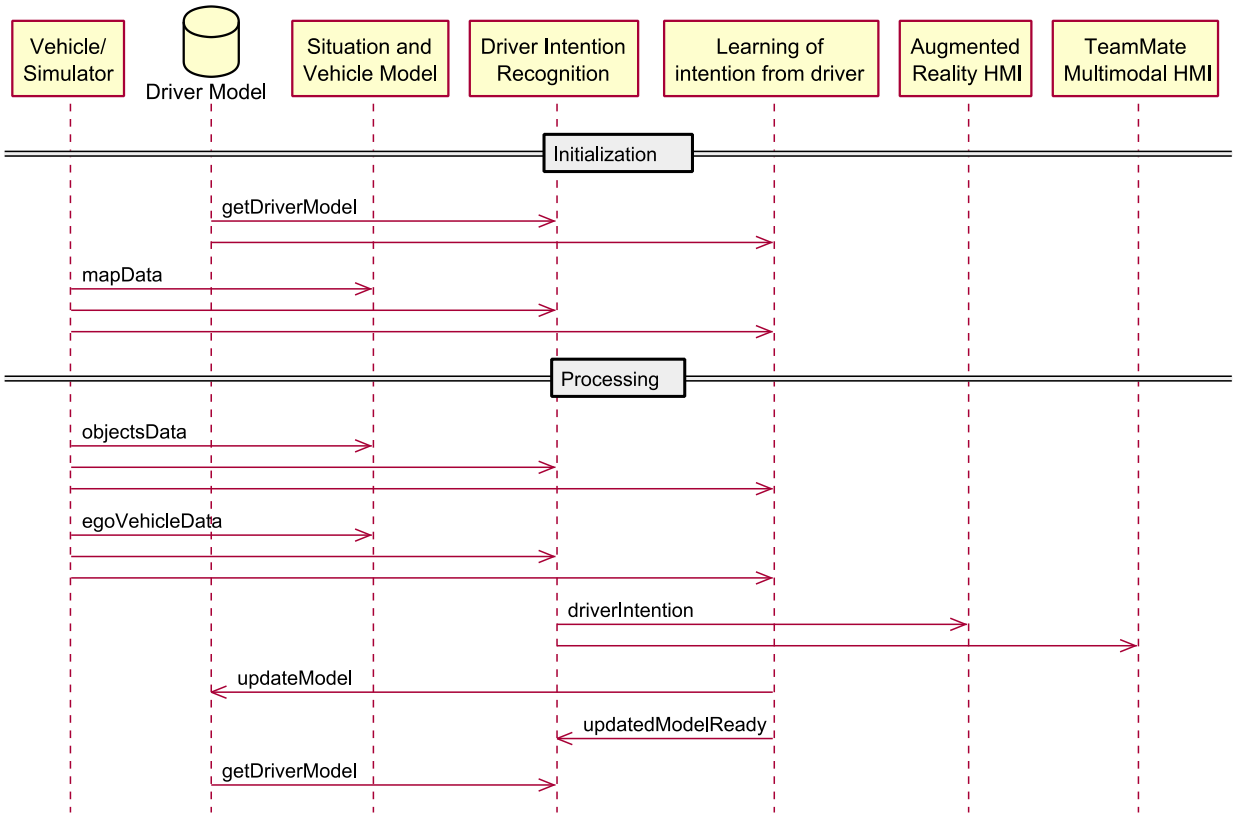

Figure 3: Interaction of Driver Intention Recognition and Learning of intention from driver.

In Figure 4 the interaction of the *Driver State Monitoring* and the *V2X Communication* with the *TeamMate Multimodal HMI* is shown. The HMI continuously receives its input data. If a distraction of the driver is detected the HMI will inform the driver, e.g., to switch from manual to automated control in order to provide A2H support. If certain Road Work Warnings are

received which would bring the automation to its limit the HMI also informs the driver to request H2A support.

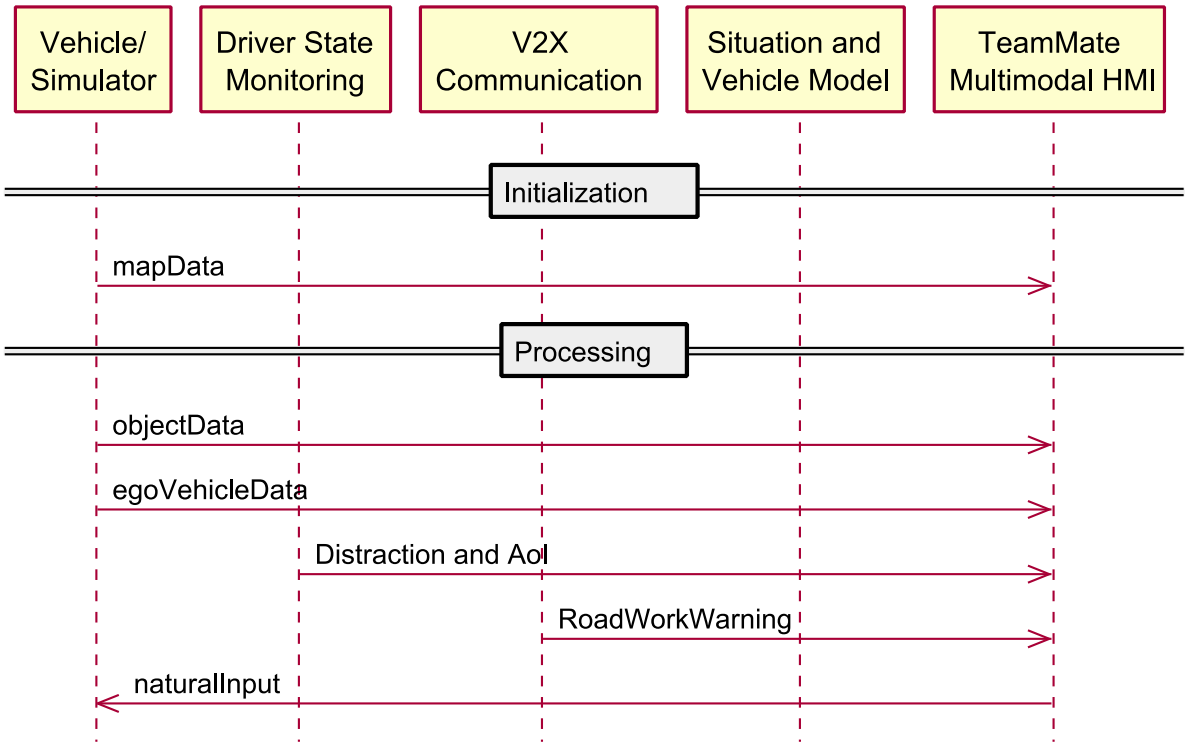

Figure 4: Interaction of Driver State Monitoring and V2X Communication with the TeamMate multimodal HMI.

Concerning the deployment the overall TeamMate architecture is in general very flexible. Due to the fact every enabler a separate component, which are dependent on their concerns it should be possible to compile each of them into a stand-alone module. The publisher-subscriber messaging pattern, which was already introduced in the previous deliverable D5.1 in theory even, allows running each enabler on separate hardware.

Thus, the actual deployment is more dependent on the corresponding demonstrator platform. For the second cycle the actual implementation and deployment of the TeamMate architecture was reported in "D5.3 – TeamMate Car Demonstrators after the 2nd cycle", for the third cycle this shall be described in D5.6 which is dedicated to the actual implementation of the TeamMate architecture in the TeamMate car demonstrators after the third cycle.

4 TeamMate API and V2X communication

In in D5.1 an initial definition of the general API of the TeamMate system was proposed. To be able to provide to third party a way to interoperate with other enablers or sensors messages were defined using Google Protocol Buffers[2] (protobuf). Thus, the enablers could be defined in any language (C/C++, JAVA etc.), in that way we have a convenient way to integrate TeamMate system messages in their implementations.

The API was extracted from the main inputs of the TeamMate architecture and described the exchanged information between the system and the sensors in order to define the TeamMate system functional and ready to be integrated in a specific car. The first messages defined in D5.1 were about the information requested by the enablers to operate computations, the main topics were the environment definition in terms of static (`MapMessage`) and dynamic (`ObjectMessage`) information and the ego-vehicle state (`EgoVehicleMessage, TrajectoryMessage`).

The updated for the third cycle provide message that are sent by the TeamMate components to provide their results to other enablers inside then TeamMate architecture.

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

The Driver Monitoring Message provides the detected distraction state and attended Areas of Interest of the driver to other enablers.

Driver Monitoring Message

Name	Type	Units	Comments
timestamp	Long (64 bits)	Ms	Expressed since 01/01/1970-00:00:00
drowsinessState	Float (32 bits)	-	drowsiness state
drowsinessLevel	Float (32 bits)	-	drowsiness level
drowsinessConfidence	Float (32 bits	-	quality of drowsiness
microsleep	Integer (32 bits)	-	microsleep event (1 when it takes place, else 0)
VTS	Float (32 bits	-	Visual Time Sharing
VTSConfidence	Float (32 bits)	-	quality of the Visual Time Sharing
attentionState	Integer (32 bits)	-	attention state
attentionStateLevel	Float (32 bits)	-	attention state level
attentionStateConfidence	Float (32 bits)	-	attention state quality
instrumentID	Integer (32 bits)	-	id of the visualized instrument
instrumentIDConfidence	Float (32 bits)	-	quality of the eye gaze diag
instrumentIDLookTime	Integer (32 bits)	-	Time the driver has been looking the instrument
attentionToTheRoad	Integer (32 bits)	Ms	Attention to road ahead
attentionToTheRoadConfidence	Float (32 bits)	-	Attention to the road quality
attentionToTheRoadLookTime	Integer (32 bits)	Ms	Time the driver has been looking the instrument

Table 1: Driver Monitoring System message content

The protobuf message translation is the following:

```
package eu.automate.openapi.messages;

message DriverMonitoringMessage {
       required uint64 timestamp                                    =    1;    /*<!
```

```
timestamp (ms)*/

       message DrowsinessMessage{
              required int32 drowsinessState                                =1;       /*<!
drowsiness state */
              required float drowsinessLevel                                =2;       /*<!
drowsiness level */
              required float drowsinessConfidence                           =3;       /*<!
quality of drowsiness */
              required int32 microsleep                                     =4;       /*<!
microsleep event (1 when it takes place, else 0) */
       }

       message VisualAttentionMessage{
              required float      VTS                                                =1;
/*!< Visual Time Sharing */
              required float      VTSConfidence                             =2;       /*!<
quality of the Visual Time Sharing */
              required int32  attentionState                                =3;       /*!<
attention state */
              required float      attentionStateLevel                       =4;       /*!<
attention state level */
              required float      attentionStateConfidence          =5;   /*!< attention
state quality */
       }

       message InstrumentIdMessage{
              required int32  instrumentID                                  =1;   /*!<  id
of the visualized instrument*/
              required float  instrumentIDConfidence                =2; /*!< quality of
the eye gaze diag */
              required uint32 instrumentIDLookTime                  =3;   /*!<  Time  the
driver has been looking the instrument (ms)*/
              required int32  attentionToTheRoad                            =4;       /*!<
Attention to road ahead */
              required float      attentionToTheRoadConfidence      =5;   /*!<  Attention
to the road quality */
              required uint32 attentionToTheRoadLookTime            =6;   /*!<  Time  the
driver has been looking the instrument (ms)*/
       }

}
```

The Intention Recognition messages makes the inference result of the Diver Intention Recognition model and therefore, e.g., lane change intentions available for other enablers.

Intention Recognition Message

Name	Type	Units	Comments
timestamp	Long (64 bits)	Ms	Expressed since 01/01/1970-00:00:00
continuous_belief_states	ContinuousMarginalizedBeliefState	-	Vector of marginalized belief states over all continuous query variables
discrete_belief_states	DiscreteMarginalizedBeliefState	-	Vector of marginalized belief states over all discrete query variables

Table 2: Driver Intention Recognition message content

The Driver Intention Message consists of vectors for continuous and discrete marginalized belief states:

Name	Type	Units	Comments
variable_name	String	-	Name identifier of a continuous variable
variable_mean	Float (32 bits)	-	Mean of the marginalized belief state
variable_variance	Float (32 bits)	-	Variance of the marginalized belief state

Table 3: ContinuousMarginalizedBeliefState message content

Name	Type	Units	Comments
variable_name	String	-	Name identifier of a continuous variable
variable_cardinality	Integer (32 bits)	-	Cardinality of the marginalized belief state
probabilities	Vector of Float (32 bits)	-	Vector of probabilities, with one probability of each possible assignment of the discrete variable

Table 4: DiscreteMarginalizedBeliefState message content

The protobuf message translation merging the tables is the following:

```
package eu.automate.openapi.messages;

message IntentionRecognitionOutputMessage {

	required int64 timestamp 		= 1;			// Timestamp,
expressed in milliseconds since 01/01/1970-00:00:00

	message ContinuousMarginalizedBeliefState {
		required string variable_name 		= 1;		// Name
identifier of a continuous variable
		required float variable_mean 		= 2;		// Mean of
the marginalized belief state
		required float variable_variance = 3;		// Variance of the
marginalized belief state
	}

	message DiscreteMarginalizedBeliefState {
		required string variable_name 		= 1;		// Name
identifier of a discrete variable
		required int32 variable_cardinality 	= 2;		//
Cardinality of the marginalized belief state
		repeated float probabilities 		= 3;		// Vector of
probabilities, with one probability of each possible assignment of the discrete
variable
	}

	repeated ContinuousMarginalizedBeliefState continuous_belief_states 	= 2;
	// Vector of marginalized belief states over all continuous query variables

	repeated DiscreteMarginalizedBeliefState discrete_belief_states 	= 3;	//
Vector of marginalized belief states over all discrete query variables

}
```

The Online Learning message can be used by enabler 4.2 to inform the Driver Intention Recognition (E2.1) about the availability of a new model update. Currently both enablers are compiled into one component, thus an explicit message is not necessary.

Online Learning Message

Name	Type	Units	Comments
timestamp	Long (64 bits)	Ms	Expressed since 01/01/1970-00:00:00
updated	Boolean	-	signalize if driver model was updated
model_path	String	-	path were the new model is stored

Table 5: Learning of intention from driver (Online Learning) message content

The protobuf message translation is the following:

```
package eu.automate.openapi.messages;

message OnlineLearningOutputMessage {

       required int64 timestamp         = 1;               // Timestamp, expressed in
milliseconds since 01/01/1970-00:00:00
       bool  updated                    = 2;            // signalize if driver model was
updated
    string model_path              = 3;            // path were the new model is stored
}
```

The Situation and Vehicle Model Message provides the traffic prediction for other enablers. Since, this data is currently only used by the Online Risk Assessment and both enablers are compiled into one component there is currently no explicit message sent.

Situation and Vehicle Model Message

Name	Type	Units	Comments
timestamp	Long (64 bits)	Ms	Expressed since 01/01/1970-00:00:00
size	Integer (32 bits)	-	Number of objects in list
object_list	Object[]	-	A vector of Objects, specifying the evolution of the object's state at specific points in time.

Table 6: Vehicle and Situation model message content containing the Traffic prediction

Name	Type	Units	Comments
id	Long (64 bits)	int	The object id
timestamp	Long (64 bits)	Ms	Time to which the object is predicted Expressed since 01/01/1970-00:00:00
absolute_position_x	Float (32 bits)	m	The x-coordinate center of bounding box Expressed in the UTM referential
absolute_position_y	Float (32 bits)	m	The y-coordinate center of bounding box Expressed in the UTM referential
yaw	Float (32 bits)	degree	The orientation of the bounding box
yaw_rate	Float (32 bits)	m.s-1	The rate of change of the heading angle of the bounding box in degree per second
longitudinal_velocity	Float (32 bits)	m.s-1	The velocity of the object
longitudinal_acceleration	Float (32 bits)	m.s-2	The acceleration of the object
cov_matrix	Float * 36 (32 bits)	-	vectorization of the covariance matrix of absolute_position_x, absolute_position_y, yaw, yaw_rate, longitudinal_velocity, and required float longitidinal_acceleration

Table 7: predicted position and orientation of an object at certain time

The protobuf message translation is the following:

```
package eu.automate.openapi.messages;

message TrafficPredictionOutputMessage {

       required int64 timestamp                =   1;   //   Timestamp   expressed   in
milliseconds since 01/01/1970-00:00:00
    size int32                                = 2; // Number of objects in list

    message Object {
              required int64 id                          = 1;  // An object ID
         required int64 timestamp                  = 2;   // Time to which the object is
predicted expressed since 01/01/1970-00:00:00

              required float absolute_position_x       = 10; // The x-coordinate center
of bounding box, expressed in meters (UTM referential)
              required float absolute_position_y       = 11; // The y-coordinate center
of bounding box, expressed in meters (UTM referential)

         required float yaw                          = 12;      // The orientation of the
bounding box expressed in degree
              required float yaw_rate                  = 13; // The rate of change of
the heading angle of the bounding box in degree per second

              required float longitudinal_velocity     = 14; //  The  velocity  of  the
object
              required float longitidinal_acceleration = 15; // The acceleration of the
```

```
object

        repeated float cov_matrix                       = 16;   // vectorization of the
covariance   matrix   of   absolute_position_x,   absolute_position_y,   yaw,   yaw_rate,
longitudinal_velocity, and required float longitidinal_acceleration
       }

       repeated Object  object_list             = 100;

}
```

The Risk Assessment Message provides the safety assessment of a provided trajectory to other enablers.

Risk Assessment Message

Name	Type	Units	Comments
timestamp	Long (64 bits)	Ms	Expressed since 01/01/1970-00:00:00
output	OutputInterpretation	-	Output of the online risk assessment, indicating whether a provided trajectory is regarded as save, unsafe, or not assessed.

Table 8: Online Risk Assessment message content

The protobuf message translation is the following:

```
package eu.automate.openapi.messages;

message RiskAssessmentOutputMessage {

       required int64 timestamp                = 1;   //    Timestamp,    expressed    in
milliseconds since 01/01/1970-00:00:00

       enum OutputInterpretation {
              NOT_ASSESSED = 0;
              TRAJECTORY_SAFE = 1;
              TRAJECTORY_UNSAFE = 2;
       }

       required OutputInterpretation output    = 2;   //  Output   of   the   online   risk
assessment, indicating whether a provided trajectory is regarded as save, unsafe, or
not assessed.

}
```

4.1 Communication of the TeamMate car with its environment

This subsection describes the API extension with Vehicle-to-Infrastructure (V2I) message. In D5.1 the specific Decentralized Environmental Notification Message (DENM) was presented that is used for the MARTHA scenario. In this cycle the API definition is extended with the relevant parts of the DENM Road Works Warning message to efficiently share this information with other components inside the TeamMate system.

V2X DEN Message

Name	Type	Units	Comments
timestamp	Long (64 bits)	Ms	Expressed since 01/01/1970-00:00:00
duration	Long (64 bits)	Ms	Validity period
event_latitude	Long (64 bits)	Tenths of microdegrees	Latitude in tenths of microdegrees
event_longitude	Long (64 bits)	Tenths of microdegrees	Longitude in tenths of microdegrees
event_altitude	Long (64 bits)	Centimetres	Expressed in centimetres
relevance_distance	Integer (32 bits)	-	1=less than 100m, 2=less than 200m, 3=less than 500m,4=less than 1000m, 5=less than 5km, 6=less than 10km, 7=over10km
closed_lanes	Integer (32 bits)	-	List of up to 13 lanes specifying; 1=closed, 0=open. [hardShoulder, outerMost, ..., innerMost]
speed_limit	Integer (32 bits)	km/h	
speed_limit_latitude	Long (64 bits)	Tenths of microdegrees	Indicating where the speed limit starts

speed_limit_longitude	Long (64 bits)	Tenths of microdegrees	Indicating where the speed limit starts
speed_limit_altitude	Long (64 bits)	Centimetres	Indicating where the speed limit starts

Table 9: V2X DEN message content

The protobuf message translation is the following:

```
package eu.automate.openapi.messages;

message V2XDENMessage {

    required int64 timestamp               = 10; // expressed in milliseconds since
01/01/1970-00:00:00
    required int64 duration               = 11; // expressed in milliseconds

    required int64 event_latitude         = 20; // latitude in tenths of microdegrees
    required int64 event_longitude                = 21; // longitude in tenths of
microdegrees
    required int64 event_altitude         = 22; // expressed in centimetres

    required int32 relevance_distance     = 23; // 1=less than 100m, 2=less than 200m,
3=less than 500m, 4=less than 1000m, 5=less than 5km, 6=less than 10km, 7=over10km

    optional int32 closed_lanes            = 24; // List of up to 13 lanes specifying
1=closed, 0=open. [ hardShoulder, outerMost, ..., innerMost ]

    optional int32 speed_limit            = 30; // expressed in km/h

// indicating where the speed limit starts
    optional int64 speed_limit_latitude   = 31; // latitude in tenths of microdegrees
    optional int64 speed_limit_longitude  = 32; // longitude in tenths of microdegrees
    optional int64 speed_limit_altitude   = 33; // expressed in centimetres

}
```

4.2 TeamMate Multimodal HMI

This paragraph describes the API extension for E6.2 TeamMate Multimodal HMI. Since the enabler has been developed to be integrated in different demonstrators with different requirements, a flexible data-structure has been built. Some of the data are common for all the scenarios (e.g. the timestamp, vehicle speed, scenarioID, mode of automation). Slots for other

data have been created in order to allow the customization of the information to be communicated.

TeamMate Multimodal HMI Message

Name	Type	Units	Comments
timestamp	Double	Ms	Expressed in 01/01/1970-00:00:00
val_d1	Double		these values are all arbitrary fields which can be adapted according to the different information provided by the different demonstrators
val_d2	Double		
val_d3	Double		
val_l1	Long (32 bits)		
val_l2	Long (32 bits)		
val_l3	Long (32 bits)		
val_l4	Long (32 bits)		
mode	Char (int at 8 bits)	n	vehicle mode (manual, automate, safe mode and the different support type)
speed	Char (int at 8 bits)	Km/h	Vehicle speed from CANbus
scenarioID	Char (int at 8 bits)	n	Indicates, according to the scenario (i.e. Peter, Martha and Eva, the information – such as the video- that should be displayed). It is a mutually exclusive variable of the demonstrators
Val_c3	Char (int at 8 bits)		these values are all arbitrary fields which can be adapted according to the different information provided by the different demonstrators
Val_c4	Char (int at 8 bits)		
Val_c5	Char (int at 8 bits)		
Val_c6	Char (int at 8 bits)		
Val_c7	Char (int at 8 bits)		

The protobuf message translation is the following:

```
package eu.automate.openapi.messages;

message HMIMessage {

    double timestamp             = 1; // expressed in milliseconds since 01/01/1970-
00:00:00
    double val_d1               = 2; // this value and the following are all arbitrary
fields which can be useful in future
    double val_d2              = 3;
    double val_d3              = 4;
    double val_d4              = 5;

    uint32 val_l1              = 6;
    uint32 val_l2              = 7;
    uint32 val_l3              = 8;
    uint32 val_l4              = 9;

    uint32 mode                = 10; // vehicle mode (manual, automated, etc)
    uint32 speed               = 11; // in Km/H
    uint32 scenarioID          = 12; // indicates different scenarios in the same Mode
    uint32 val_c1              = 13;
    uint32 val_c2              = 14;
    uint32 val_c3              = 15;
    uint32 val_c4              = 16;
    uint32 val_c5              = 17;
    uint32 val_c6              = 18;
    uint32 val_c7              = 19;
}
```

The protobuf output message:

```
package eu.automate.openapi.messages;

message HMIOutputMessage {

    double timestamp             = 1; // expressed in milliseconds since 01/01/1970-
00:00:00
       enum OvertakingRelease{
              NO_RESPONSE = 0;
              ACCEPTED = 1;
              REJECTED = 2;
       }

       required OvertakingRelease output= 2;   //   Output   of   the   HMI   Overtaking
Manoeuvre Selection dialog indicating if the driver acceptes, rejects the overtaking or
did not react.
}
```

5 Conclusions

In this deliverable D5.4, we presented the global AutoMate System architecture, showing that it works on all demonstrators of the AutoMate project considering their different initial architectures. Based on D5.1, we refined the functional blocks of the TeamMate Car, including their interconnections and the data flow between them. In this way, we have also formally specified the system architecture. The explanation of dataflow within the software construct was updated and a clarification of interaction sequences between the enablers is given as well.

Another important part is the definition of interfaces between the modules, as well as common data formats standards and communication protocols. Therefore, the TeamMate Application Programming Interface (API) has been defined in terms of principles, standards, interfaces and data structure that enable the communication of information between components in the TeamMate ecosystem. Thus, we presented a common design principle for the communication between components in the TeamMate ecosystem, based on exchanging messages in a publisher-subscriber messaging patterns. Messages are defined in terms of data structures with fixed semantics.

This document constitutes the basis of the 3rd cycle for the description of the baseline cars (D5.5) and of the implementation of the TeamMate cars in the three demonstrators with a specific focus on the deployment (D5.6).