Proceedings of 7th Transport Research Arena TRA 2018, April 16-19, 2018, Vienna, Austria

A flexible automotive systems architecture for next generation ADAS

Johannes Hiltscher[a]*, S. V. N. Phanindra Akula[a], Robin Streiter[b], Gerd Wanielik[a]

[a]Chemnitz University of Technology, Reichenhainer Straße 70, 09126 Chemnitz, Germany
[b]Naventik GmbH, Reichenhainer Straße 70, 09126 Chemnitz, Germany

Abstract

The evolution of Advanced Driver Assistance Systems (ADASs) increasingly paves the way towards autonomous driving. The increase in sensor data to be processed as well as the complexity of processing algorithms demands more powerful processing platforms. Additionally, aspects like functional safety and platform security have to be taken into account. Algorithms should be re-usable to facilitate rapid development. Current solutions are very much focused on single sub-systems and incompatible with next generation processing platforms available for ADASs. In this paper we therefore propose a novel architecture for future ADAS development, taking into account the aforementioned aspects. We also outline tools to aid developers in dealing with the real-time and functional safety aspects of ADASs. The utilization of the presented architecture in the EU-funded project ADAS&ME is also described.

Keywords: ADAS, functional safety, ISO 26262, real-time

* Corresponding author. Tel.: +49-371-531-33353; fax: +49-371-531-8-33353
E-mail address: johannes.hiltscher@etit.tu-chemnitz.de

1. Introduction

Driver Assistance Systems (DASs) have been one of the core fields of innovation in the automotive industry in recent years. Starting with assistance systems to aid drivers in critical but infrequently appearing situations (like the anti-lock braking system, ABS), they evolved into very sophisticated ones like automatic lane keeping assistants. This evolution goes along with increasingly complex sensors, rising amounts of data to be processed and algorithm complexity. Current systems often employ cameras, which can generate several hundred Megabits of data per second. Looking towards autonomous and connected driving, the amount of sensor data will be in the range of Gigabits per second. The current approach for dealing with the resulting problems of data transmission and routing is fusing sensor and algorithm into one device. This relaxes the transmission problem as data processing algorithms typically reduce its size drastically. But this approach has some evident drawbacks:

- Possible **sensor duplication** due to inability to share between (A)DASs with different data processing;
- **Unpredictable and potentially unnecessary latencies** due to the inability of combining sensor data processing and ADAS algorithms into one processing system;
- **Increased overhead** for system integrity testing due to the distribution of system components;
- **Complex sensor systems**, e.g., LIDAR, are hard to replace due to proprietary Application Programming Interfaces (APIs) requiring interfacing code to be rewritten, even if their functionality is equivalent;
- **Inflexibility** combining sensor and data processing algorithm, potentially lowering performance.

At best, these drawbacks lead to increased costs. From a safety perspective, however, this is an alarming situation as an evaluation of the ADAS's correctness is complicated or could even become impossible. We address the identified problems with a novel systems architecture for complex ADASs. Our focus is on flexibility, extensibility and functional safety. We outline our motivation for designing it in Chapter 2. A set of design goals for a future ADAS middleware is derived from an evaluation of the developments in this domain. Afterwards, we evaluate related work and analyse the suitability of existing solutions regarding the design goals. Our approach is presented in Chapter 3. We describe how the design goals will be realized and present a holistic system architecture. A system implementing this architecture is going to be realized in the scope of the EU-funded project ADAS&ME; in Chapter 4, information regarding this implementation is presented. Chapter 5 gives a conclusion of the paper.

2. Analysis of next generation ADAS requirements and challenges

Although assistance functions like lane-keeping or emergency breaking assistants are available, most of today's vehicles provide drivers very limited assistance. Common systems like ABS or ESP only require low-bandwidth. Middleware is already employed in the development of today's systems, allowing the creation of reusable and hardware-independent code. The concepts employed are, however, not easily adaptable to the demands of future, highly automated vehicles, which we analyse in the following, focusing on technical and economic aspects.

2.1. Increasing ADAS complexity

The increasing autonomy of assistance functions increases the demand for sensor data. ADASs often process several sensors' data to increase robustness and avoid malfunctions in case of sensor failure. To reduce costs it is desirable to operate as many ADASs as possible from each sensor. However, different systems often require different data processing. Such different requirements and the availability of hardware platforms with ample processing power (refer to Section 2.3) favour strongly software-based solutions. Raw data is processed by dedicated software modules on the hardware platform rather than in the sensor device. Software-based solutions allow for generating a plethora of information from each sensor and allows ADAS developers to select the best available algorithm for each problem, enabling them to benefit from the active research on algorithms. Companies can reduce their in-house knowledge and rely on suppliers specialized on data processing. This can also be an incitement for innovators as they can concentrate on small domains rather than developing a whole ADAS stack.

Achieving these goals requires decomposing an ADAS into modules with limited functionality. This approach provides maximum flexibility and new ADASs can be added to an existing platform. Duplicate processing can be avoided by distributing the results of each module to all others. This approach is beneficial in further, probably less obvious, ways: it aids the testability of the overall system. Development paradigms like Model-Based Design (Mosterman, Schieferdecker, & Zander, 2012) commonly accepted as best practice when developing safety critical systems follow a modular approach. Furthermore, by relying on ready and tested blocks, both development and testing time of a new ADAS can be reduced (Reedy & Lunzmann, 2010). A similar technique, Behaviour-Driven Development (BDD, c.f. (Solis & Wang, 2011)), became famous in computer programming in recent years. Its focus is on providing an automatically assessable description of an algorithm's expected behaviour. This implies that designers put increased effort into the algorithm design phase and formulate unambiguous requirements.

2.2. Safety and security requirements, exponentially increasing testing time

State-of-the-art ADASs are already capable of taking over major driving tasks, paving the way to fully autonomous driving. They have to deal with complex environments and require extremely low failure rates as they have extensive control over the vehicle. Their high autonomy reduces drivers' involvement in the driving task so they will often be incapable of compensating failures. Testing of an ADAS encompasses the following aspects which we summarize under the term "safety": correctness of the application code, well-defined reactions to environmental situations, completeness of the environmental situations the system has to deal with and assurance that the application code handles all situations in the defined manner. In summary, safety means that a system and its behaviour are well-defined and it is assured that the system behaves accordingly. The first two points are addressed with coding guidelines like MISRA C (Motor Industry Reliability Association, 2013) and formal specifications like ISO 26262 (International Organization for Standardization (ISO), 2011) while the others demand extensive – and appropriate – testing. Functional safety, as specified in ISO 26262, is one of the core concepts to specify and assess the risks of electrical and electronic systems in vehicles.

Partitioning of an ADAS – as already mentioned in Section 2.1 – can be employed to minimize the development and testing time by avoid re-testing already established code. Furthermore, a sound formulation of functional safety items allows to semi-automatically assess new components. An architecture for future ADASs has to account for this by integrating these aspects. We consider BDD a promising approach as it encompasses formal languages for translating the natural language of specification documents into descriptions suitable for automatic testing.

Security – encompassing measures to avoid altering a system's intended function – is a further crucial aspect of ADAS platforms which have extensive control over a vehicle, including drivetrain and steering. Probably the best-known example of highly critical security flaws in an automobile is the remote hack of a Jeep Cherokee (Miller & Valasek, 2015). This example underlines the importance of considering safety at global system level during the whole development cycle. Currently a lot of work is spent on identifying illegitimate accesses on vehicles' control buses using network intrusion detection and virtual networks (Kleberger, Olovsson, & Jonsson, 2011). However, this approach is insufficient as exploitable issues in privileged software components still allow for accessing safety critical systems. Ideally, each system is designed to minimize the likelihood of being compromised.

2.3. Changing hardware platforms and hardware sharing

Current Electronic Control Units (ECUs), the vehicular computers, are classic embedded systems with relatively low processing power. Future ADASs require orders of magnitude greater processing power. New hardware vendors are hence pushing into the automotive market. However, though providing unprecedented processing power, their devices are problematic from a real-time perspective. While classic microcontrollers are true real-time devices, Systems-on-Chip (SoCs) like NVIDIA's Drive PX (NVIDIA Corporation, n.d.) do not exhibit tight predictability of their timing behaviour. Platform sharing of different ADASs and the advent of distributed systems in highly autonomous vehicles are developments likely to be observed. Given an adequate architecture, development environments can aid ADAS designers in dealing with the challenges of real-time requirements.

2.4. Related work

The automotive industry has developed solutions and frameworks for ADAS and ECU development. These are, as we will point out in the following, not suited for the demands of novel ADASs. Other, especially open source, solutions are hardly available. Probably best known is AUTOSAR, which, at a first glance, seems to satisfy all our stated requirements. However, for our work, we see several problems with AUTOSAR: its complexity (which also is a concern of developers, c.f. (Martínez-Fernández, Ayala, Franch, & Nakagawa, 2015)) and missing support for the hardware platforms required to implement complex ADASs. The latter point is the primary show stopper as vendors like NVIDIA do not supply device drivers for AUTOSAR platforms. Nevertheless, we consider AUTOSAR state-of-the-art regarding safety. Readers familiar with it will notice several analogies in our proposal.

TTTech offers a commercial middleware and integration framework with a nearly identical concept. This solution also focuses on functional safety and real-time capabilities (Lang, 2016). Further analysis is infeasible as the system is proprietary which also makes it inattractive for the academic research community. Irrespective of the topic's importance there has been few activity in this direction in research. The only publication with a similar focus was published by BMW (Bulwahn, Ochs, & Wagner, 2013). Several publications report about the integration of ADASs into experimental vehicles, e.g. (Lusserau, et al., 2015). This indicates that several groups use custom systems for running their ADASs, but obviously without intentions to fomalize them. While the BMW publication accounts for the real-time requirements of ADASs, this aspect lacks coverage in other publications. Taking a closer look at the architecture proposed by BMW considering the previously discussed requirements reveals that several of them are not met. The security aspect is not accounted for and the Robot Operating System (ROS) is neither real-time capable nor efficient. ROS has furthermore received much criticism regarding security (McClean, Stull, Farrar, & Mascarenas, 2013) which have been addressed in parts (c.f. (Breiling, Dieber, & Schartner, 2017)). The proposal also lacks support mechanisms for developers as it does not consider a development framework.

3. Design of an architecture for next-generation ADASs

Based on the analysis from the previous chapter we developed a novel systems architecture specifically for complex ADASs. Our approach differs from others in terms of design paradigms. By starting from scratch, we are able to avoid limitations of pre-existing solutions. We broadened the concept of a sensor, which we consider a data producer. We also advocate data format unification and a provision of machine-readable interface definitions to allow for automatic generation of interfacing code and assistance in safety assessment. Finally, we designed a concept for a middleware that enables strong vertical partitioning of ADASs and ensures the timing and ordering constraints of the different components are met. The different pillars of the architecture are presented in separate sections in the following after a short introduction of the generic concepts.

We propose a modular design approach which simplifies testing and cooperation, as will be elaborated upon later. The module concept is a producer-consumer model. Each module has an integer number N of inputs ($N \geq 0$) and M outputs ($M \geq 0$). Inputs are connected to other modules' outputs; one output may be connected to several inputs. In this view, sensors – or similar devices like Human-Machine-Interface (HMI) inputs – are modules with N = 0 inputs. They do not rely on other modules for generating data which is input from the environment. Likewise, actuators or HMI outputs are modules with M = 0 outputs, they do not generate any information. Each module consists of its respective code and a machine-readable description, preferably in XML format (Extensible Markup Language (XML) 1.0 (Fifth Edition), 2013). This description contains the module's crucial parameters (e.g., the required update rate and the data generation rate), generalized functional safety requirements and assurances as well as a description of its interfaces. Module parameters can also be inherited from a higher instance module.

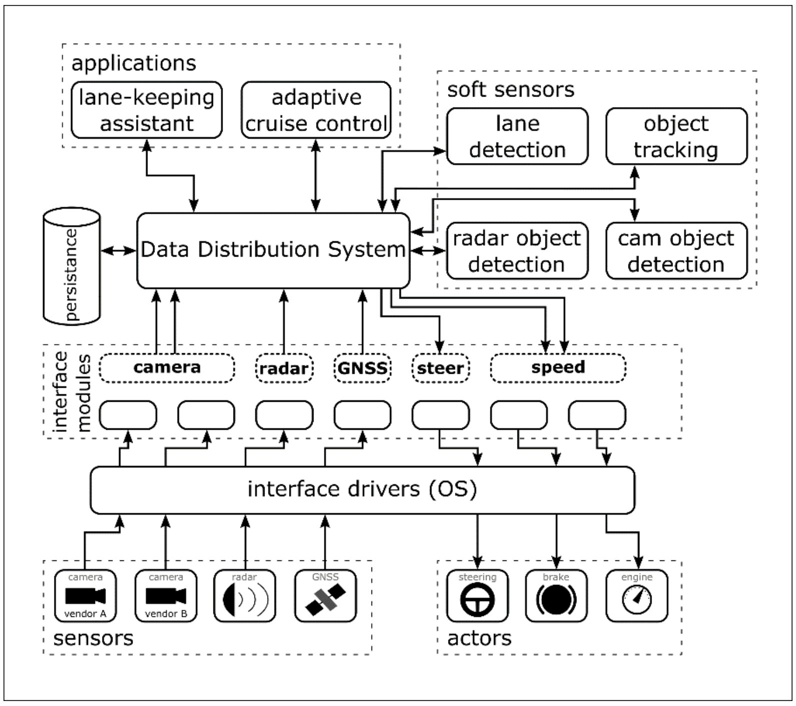

Fig. 1 example system diagram of an assistance system with two independent ADAS.

A system description (which may contain several ADASs) is generated as a directed graph G = (N, E) with N being the modules used in the system and E the directed connections between inputs and outputs. This allows for automatic system testing, partitioning and packaging. For the sake of better understanding we refer to an example system consisting of two ADASs, a lane-keeping assistant and an Adaptive Cruise Control (ACC). This system is meant for illustrative purposes only and should not be considered realistic. We use two different depictions of the system for better illustration. Fig. 1 shows a system diagram illustrating the setup of the operating system and middleware and the integration of the modules of the two ADASs. Fig. 2 depicts their data flow as well as their composition from modules. It is a graphical representation of the aforementioned graph description of the systems.

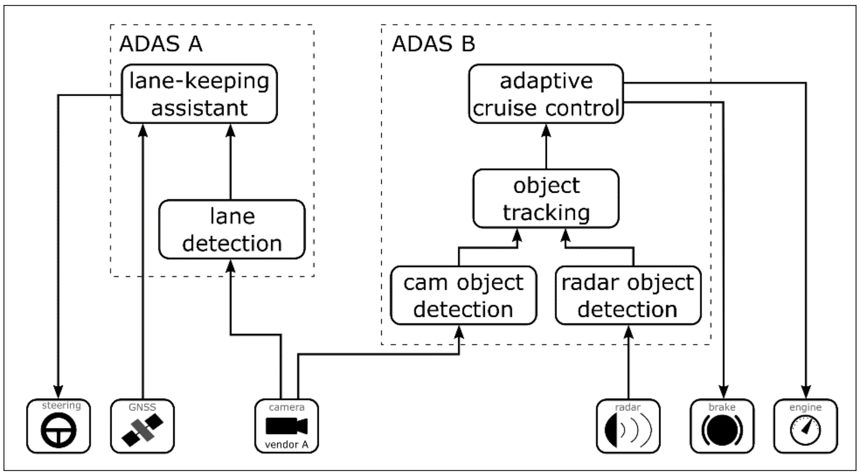

Fig. 2 data flow diagram of the example assistance system.

3.1. Sensor concept

Traditionally, a control system can be divided into sensor, processor (the actual control component implementing a function mapping the sensor readings to actuator settings) and actuator. Sensor and actuator are physical devices. In contrast, in our understanding, a sensor is a data producer. This may be a physical device, but it can be a data processing algorithm as well, which we refer to as a “soft sensor”. This view was developed from the insight that several ADASs may operate on the same basic data, e.g., camera images as shown in Fig. 2, and require a partially similar processing path. A processing path, in our understanding, is a sequence of processing steps, which is fed input data from a sensor and outputs data to control components. Duplication of processing paths should be avoided in order to minimize resource requirements. Additionally, this approach encourages developers to modularize their code, thereby improving testability as small and simple modules ease the definition of test-cases. A suitable set of test-cases either covers all possible input sets or a sub-set known to cover a sufficient proportion of the relevant situations the code may encounter. This is easier to assure if the unit under test is kept simple.

The same holds true for functional safety evaluations. The overall correctness and functional safety of a complex system composed from simpler ones can then be determined by inferencing over the sub-systems. On the downside a highly modular approach likely increases the number of events in the safety analysis. Using mechanisms to automatically check accordance to the safety standard and aid the preparation of the certification process (c.f. (Panesar-Walawege, Sabetzadeh, & Briand, 2013)), this increase can nevertheless be handled. Re-useable modules are supported by ISO 26262 and referred to as Safety Elements out of Context (SEooC) as defined in Part 10 of ISO 26262. Another benefit arises from the possibility of tapping the processing path at different stages to access data from within the processing chain. For run-time determination of data integrity, it can be crucial being able to access raw data (Streiter, Hiltscher, Bauer, & Jüttner, 2016). This is enabled by our proposed modular approach, while at the same time the amount of data passed is tailored to the specific demands of the respective consumer. Designers of intermediate processing algorithms need not anticipate the possible requirements of higher level components.

3.2. Interface unification and automatic generation

Sensors and actuators are connected using almost the whole range of computer interfaces. This situation is more pronounced in the sensor domain while actuators commonly have automotive interfaces like CAN or FlexRay. Besides different interfaces, developers are commonly forced to using a vendor-specific API for configuration of and data exchange with a device. This situation unnecessarily complicates the development of generic algorithms, which have to be independent of the specific sensor or actuator used. We therefore propose abstracting these into classes with common data formats, as depicted in Fig. 1 (interface modules block). Furthermore, the interface and data structure definition is part of the module description. This allows for automatic generation of conversion code which removes possible error sources. Data conversion can happen at any connection between two modules in Fig. 1. Sensors and actuators are attached to the system using so-called Interface Modules. An Interface Module encapsulates all the relevant code for retrieving data from respectively sending it to the hardware device and inserting it into a class specific data structure. It is fitted for the specific device and ideally provided by the vendor.

3.3. Data distribution system

Managing the data flow between the modules of a system is the third pillar of our proposed systems architecture. It is handled by a dedicated Data Distribution System (DDS, shown in the centre of Fig. 1) which implements a subscription mechanism. Modules advertise their available output data and can subscribe to the data produced by others. Subscriptions can be handled either as push or pull requests. A push request leads to the invocation of the module code by the DDS whenever new data becomes available. In contrast, if a module subscribes using pull requests, code invocation has to be performed otherwise, e.g., using a timer. Data is inserted into a queue from which the module consumes it. To simplify prototyping we seek accordance to the Open Management Group's Data Distribution Service (Open Management Group, 2015) to facilitate interoperability with other DDSs.

Taking into account the peculiarities of safety-critical systems, some further considerations are necessary. Due to performance reasons, data distribution should be as light-weight as possible. ROS, for example, employs the Internet Protocol (IP) for all inter-module communications. This is problematic both due to the overhead imposed as well as from real-time requirements. We therefore take a more requirements-oriented approach and select the most appropriate solution for each communication. As long as source and destination can access shared memory, this is our preferred communication mechanism. The DDS handles the management of the respective resources. Another notable difference to other solutions is that we are not aiming at providing a dynamic system, where modules could be added at run-time. Such an approach imposes an unnecessary burden for a safety-critical real-time system. As scheduling is an NP-complete problem, often only suboptimal solutions will be found at run-time, thereby restricting system efficiency. A static system can use resources efficiently as unlimited scheduling time is available (Liu, 2000). Modification of the system can be achieved by changing the system schedule.

At run-time the DDS performs two further tasks besides data distribution: system supervision and time keeping. Supervision monitors the outputs of all modules for rate deviations to detect malfunctions. As the time keeper, the DDS performs a consistent time-stamping of all data items passed between components. If the system is distributed to several platforms it also performs time synchronization between these.

3.4. Data persistence

Personalisation will be one important component of higher-level ADASs. Regulators are also likely to demand data logging for autonomous functions to enable crash analysis. This can be concluded from the recommendations of the National Transportation Safety Board's analysis of the fatal Tesla crash of 2016 (c.f. (Williston, 2016)). The ADAS&ME project also requires a storage mechanism for sensor data to implement a history function. These requirements are accounted for by introducing a dedicated persistence module, shown left in Fig. 1.

3.5. System description, integrity testing and commissioning

The previously described architecture is designed to allow for automatic integration testing and partitioning. This is performed by an ecosystem of software and a system description similar to the module descriptions. The system description includes all the modules required and their interdependencies, i.e., the input-output mapping. It also specifies some global parameters like required update rates and tolerable variations of the same. Together with a platform description the system description is used to commission the final system package consisting of application binaries and system schedule. This system description is used in the integration flow described in the following. Integration starts with profiling the different modules using the supplied test cases to assess whether the platform can meet the requested timings. This is shown left in Fig. 3 as *module benchmarking*. It is performed on the target platform, the results are employed to perform partitioning and scheduling afterwards. The partitioning and scheduling process, shown right in Fig. 3 as *system scheduling*, assigns modules to the platform's resources (processor cores, processors in distributed systems or software/hardware if applicable). Resource utilization as well as module interdependencies and timing requirements have to be taken into account to derive feasible schedules which are then compared. This process also defines the modules' order of execution. Interfacing code is also generated during this step, allowing to account for communication costs.

Each schedule is again tested on the target system to verify the overall system meets the specified requirements. This is a mandatory step as unexpected interdependencies can occur due to modern processor systems' complexity, resulting, e.g., from cache conflicts. Research concerning the real-time properties of SoCs employed in complex ADASs underpins the necessity of analysing the resource demands of the different components and evaluating different schedules. An analysis of the NVIDIA Jetson TX 1 found an increased variance of the execution time of computer vision algorithms when competing for usage of the device's Graphics Processing Unit (GPU). Depending on the resource requirements, however, hardware utilization can be increased (Otterness, et al., 2017).

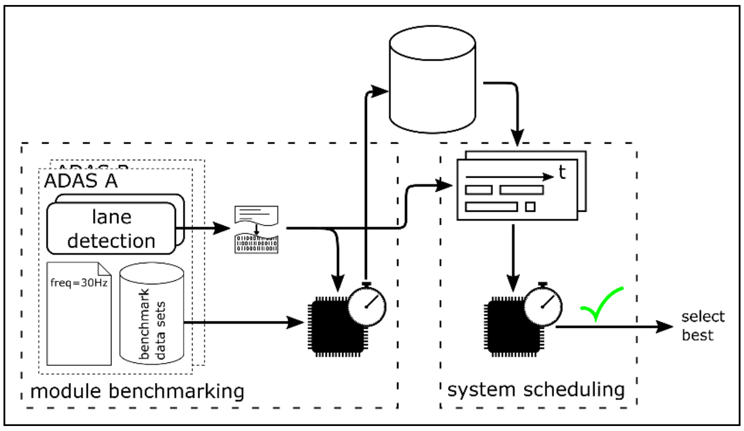

Fig. 3 simplified view of the system integration flow for a specific target hardware.

3.6. Operating System

Due to the complexity of the targeted hardware platforms employing an elaborate Operating System (OS) is inevitable. Considering real-time, safety and security aspects, the preferred solution would be a real-time micro kernel with a custom middleware. This combination provides the most reliable real-time behaviour, is easy to verify due to the commonly small code base and has a small attack surface due to limited functionality. However, hardware manufacturers typically support only more “common” OSs like Linux. For example, to use the graphics hardware in NVIDIA devices, the CUDA library is required which is not available for real-time OSs.

The Linux kernel has integrated support for real-time tasks via the PREEMPT-RT patch. Data available from the Open Source Automation Development Lab (OSADL) shows quite good performance regarding critical real-time parameters. Xenomai provides a dedicated real-time kernel running besides the Linux kernel (Xenomai Project, 2017), promising even better real-time performance. To meet the safety requirements, using a “hardened” kernel with all unnecessary components removed is advisable. Using the available mechanisms for process isolation, like cgroups, and rigorous control of communication with other systems aid in further reducing the attack surface.

4. Implementation at ADAS&ME

The architecture described in the previous chapters is being realised within the ADAS&ME (“Adaptive ADAS to support incapacitated drivers & Mitigate Effectively risks through tailor made HMI under automation”) Project funded by the European Union under the Horizon 2020 framework. This project aims to develop adapted ADASs incorporating driver/rider state, situational/environmental context and adaptive interaction for automatic transfer of control between vehicle and driver/rider to ensure safer and more efficient road usage. The work in this project spans across seven provisionally identified Use Cases for cars, trucks, buses, and motorcycles, aiming to cover a large proportion of European road traffic. System requirements for realising these Use Cases lay the foundation for designing the overall system architecture. System requirements within the ADAS&ME project are specified in terms of Tasks, Communication, Timing, and Priorities (c.f. (Ganssle, 2008)). Based on this information the overall ADAS&ME system is divided into several subsystems. These are further subdivided into modules which consist of hardware or software components. Let us now look at realization of the proposed architecture and its flexibility aspects at different instants. Use Case C: Driver state-based smooth and safe automation transitions, Use Case D: Non-Reacting Driver Manoeuvre are two of the seven use cases that are realised in the scope of the ADAS&ME Project on the DLR Car Demonstrator. In this chapter, first we briefly describe Use Cases C and D of the ADAS&ME Project, second we look into the details of the architecture configuration of these Use Cases.

Use Case C aims at designing smooth and safe transitions between different levels of automation. The focus is on handovers (higher to lower level of automation) as well as temporary takeovers (transition from a lower to higher level of automation). The Use Case will concentrate on automatically initiated transitions from automated to manual driving mode. During this period of automated driving the driver might be busy doing some other tasks without knowing the current vehicle situation. Hence the transition of control back to the driver is supervised by a constant assessment of her state (stress/workload, focus of attention, emotions such as frustration and anger) and the situation around the vehicle. Similarly, Use Case D addresses situations in which the driver does not react to any warnings/requests coming from a car driving in automated mode and consequently the situation becomes safety critical. Hence Use Case D concentrates on automatically initiated takeovers. More specifically, Use Case D deals with minimal risk manoeuvres and emergency manoeuvres. It can be seen as an escalation of Use Case C.

The general ADAS&ME system architecture consists of the following five subsystems, covered in detail below:

- Sensor Subsystem (SS)
- Driver State Monitoring Subsystem (DSMS)
- Environmental Situation Awareness Subsystem (ESAS)
- ADAS&ME Core (ADAS&ME C)
- Vehicle Automation Subsystem (VAS)

Fig. 4 shows the ADAS&ME general system architecture. The Sensor Subsystem assists all the subsequent ones by providing the necessary data from the different classes of sensors as input. At the input side, SS consists of a Smart Eye pro camera, Electrocardiography (ECG), microphone array and ultra-wide band sensors interfaced to the Interface Module (IM) to constantly measure the drivers' physiological parameters like facial expression, heart rate, voice in the car and respiration rate. Similarly, Denso Wireless Safety Unit (WSU) and Cellular connectivity unit, LIDAR, RADAR, Mobileye camera, NOVATEL SPAN-CPT, NAVENTIK Endurance, TomTom high definition (HD) Maps are sensors connected to the IM to observe the environmental situation surrounding the car. HMI elements are also coupled to this Module. The IM at this subsystem also provides a mechanism for subsequent modules to subscribe for publishing new sensor data. Software algorithms present at DSMS and ESAS process the input sensor data and come up with an estimation of the current driver state and environmental situation.

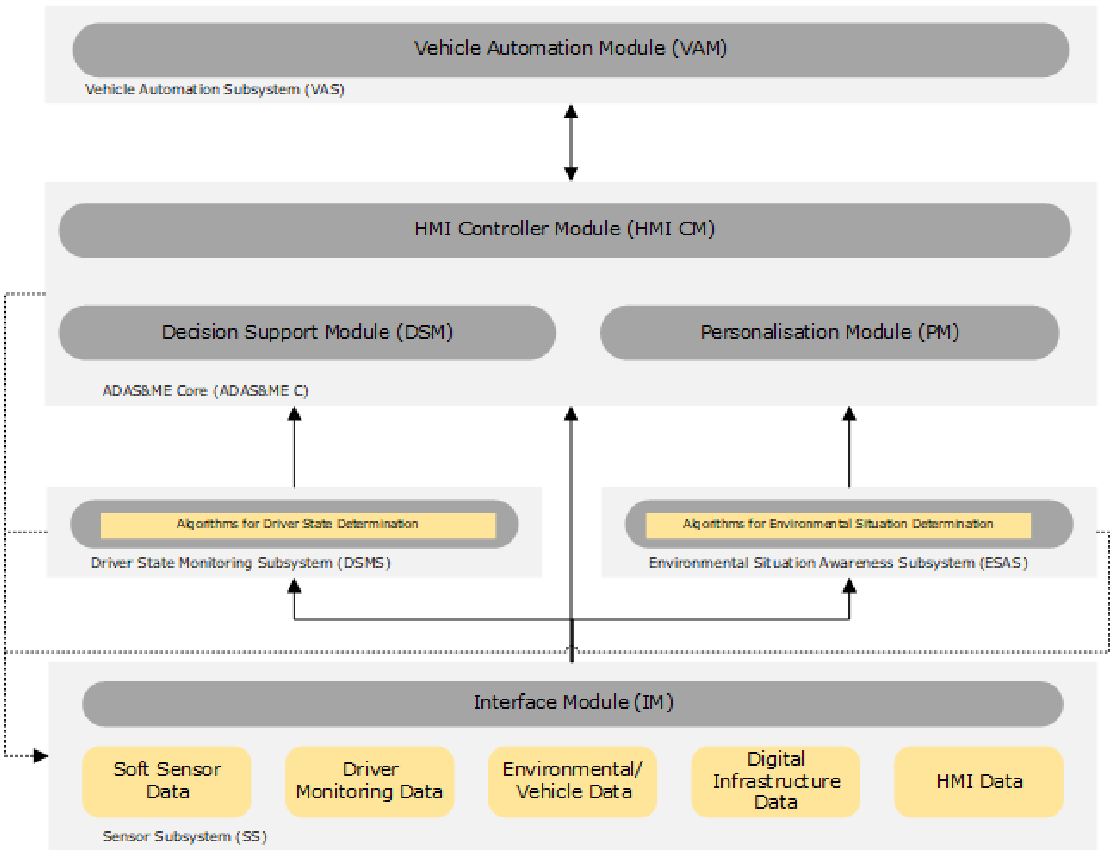
Vehicle Automation Module (VAM)
Vehicle Automation Subsystem (VAS)
HMI Controller Module (HMI CM)
Decision Support Module (DSM)
Personalisation Module (PM)
ADAS&ME Core (ADAS&ME C)
Algorithms for Driver State Determination
Driver State Monitoring Subsystem (DSMS)
Algorithms for Environmental Situation Determination
Environmental Situation Awareness Subsystem (ESAS)
Interface Module (IM)
Soft Sensor Data
Driver Monitoring Data
Environmental/ Vehicle Data
Digital Infrastructure Data
HMI Data
Sensor Subsystem (SS)

Fig. 4 ADAS&ME General System Architecture.

The ADAS&ME Core acts as the central subsystem by: receiving inputs from other subsystems, processing the received sensor data to compute a decision strategy, actuating the HMI or Automation functions. It encapsulates Personalisation Module (PM), Decision Support Module (DSM), and HMI Controller Module (HMI C). PM checks if the current driver of the car has already driven the same car or not. If so, this module will supply the driving profile calibrated while the driver has driven the vehicle for the first time. Otherwise, a new driving profile is created for future use. The DSM works on a decision strategy based on the output of other algorithms received from DSMS, ESAS and the real-time sensor data from the SS. This decision strategy will serve as an input to the HMI controller module in order to notify the user about the system's strategy and get the drivers feedback on whether to follow as per the computed strategy or not. Based on the user feedback, the ADAS&ME C instructs the VAS either to continue in the current driving mode (manual mode) or switch to autonomous driving mode.

In real time all the subsystems work in parallel and the flow of data happens in a sequential manner between subsystems. In order to facilitate intercommunication between modules of different subsystems, the current specified architecture has a special feedback channel wherein all modules can send the output of their respective software algorithms to the SS. The software algorithms thereby act like Sensors. This feedback channel is specified by a dotted line in Fig. 4. At the SS, all data received via this channel is referred to as soft sensor data. Apart from personalising the driver's driving profile, PM at ADAS&ME C also facilities input data history storage. Similar to the other subsystems this one also shares the output of its contained software algorithms to subsequent layers and implements a feedback to the SS. The feedback channel at this subsystem cannot only share the algorithms' output at the current time instant but also from the previous instants by using a database. Whenever any subsystem is in need of some data this is not available at the current time instant it can be queried from that database using this feedback channel. In the following we will describe how Use Cases C (driver-state based automation transitions) and D (non-reacting driver) will be realized based on this system architecture.

The system will be realized as a push system (as described in Section 3.3); therefore, all subsystems subscribed to a sensor get a notification from the SS each time new data is available. Converting the sensor data into a common format (ADAS&ME data format) at the IM enables all subsystems to function independent of the sensor hardware interface and data layout. During the start of communication between subsystems or modules, each of them exchanges the communication data layout as part of a handshake procedure. At the DSMS software algorithms assess the driver state for sleepiness, inattention, emotion or stress on a discrete scale. Correspondingly software algorithms at the ESAS first compute the vehicle state and vehicle context which are then combined to arrive at an Environmental State based on the sensor data. These algorithm outputs are shared to the ADAS&ME C and also as a feedback to SS. Sharing the output of algorithms back to the SS enables inter-process communication at subsystem level. As estimation of driver state and situational awareness are complex tasks and sensor data cannot be guaranteed to be error-free, each algorithm also computes the current state estimation's confidence level based on input data. This estimation value computed at any subsystem helps the subsystem to decide whether to process this data or opt for the reliable approach specified as per the safety management.

Having discussed the building blocks of ADAS&ME Use Cases C and D architecture, we now shift our focus on realizing their high-level functionality on top of it. Let us assume you are sitting in your car equipped with the proposed ADAS&ME system, all the sensors are set on and fully functional. You start driving by keying your desired destination, then after travelling for some time you hit the highway. The car finds out that you are a bit stressed and an HMI element requests to change the driving mode from manual to automatic. On accepting the request you are informed that the vehicle will be "Changing the Driving Mode to Automatic Mode in 10 seconds". On giving the control to the car you feel very relaxed and would like to have a short nap. Sensors interfaced to ADAS&ME System continue monitoring your behaviour. Your current behaviour will be estimated as sleepy by the DSDM. After travelling some distance on the highway the car realizes from the available HD digital map that there happens to be a road work ahead. It can no longer continue driving autonomously and wishes to transition to manual driving. As you are sleepy and not aware of the current situation inside and outside the car, it tries to bring you into an active state by using available HMI elements. At a suitable instant where the environmental situation allows for handover, you get the control back. This example illustrates one solution to achieve a smooth transition from a higher level of automation to a lower one, as planned to be realized in the scope of the ADAS&ME project.

Use Case D can also use the same background discussed above until the point where the car requests a change to manual control when approaching a road work ahead. If, at this point, you are unaware of the situation, Use Case D mitigates the consequences arising when you are not able to accept a requested handover. A solution to this can be achieved with the same hardware configuration that was used in Use Case C, but with a minimal modification of the software algorithms. One possible mitigation strategy option planned as part of the project is to enable a V2X communication with a car in the front and follow it. If no car is available in the front then the car will perform an emergency manoeuvre and come to a halt at a nearby safe parking area as displayed in the HD map.

5. Conclusion and future work

We have presented a novel and open design of a systems architecture for future ADASs along with tools for assisting designers. It is derived from an analysis of the complex systems' requirements providing extensive driver support. We received positive feedback from both academia as well as the industry in the ADAS&ME project, and look forward to seeing the first system implementing our approach. During the development phase we concentrate on supporting developers with novel tools for integrating real-time systems on SoCs and handling functional safety. We will further report on these tools and experiences from the implementation phase.

Acknowledgements

The authors would like to thank the ADAS&ME Project consortium and the European Union for their support. Johannes Hiltscher is supported by Deutsche Forschungsgemeinschaft (DFG) within SPP1835.

6. References

Breiling, B., Dieber, B., & Schartner, P. (2017). Secure Communication for the Robot Operating System. *2017 Annual IEEE International Systems Conference (SysCon)* (pp. 1-6). IEEE.

Bulwahn, L., Ochs, T., & Wagner, D. (2013). *Research on an Open-Source Software Platform for Autonomous Driving Systems.* Munich: BMW Car IT GmbH.

Extensible Markup Language (XML) 1.0 (Fifth Edition). (2013, February 7). Retrieved from W3C: https://www.w3.org/TR/REC-xml/

Ganssle, J. (2008). *The Art of Designing Embedded Systems, Second Edition.* Elsevier Inc.

International Organization for Standardization (ISO). (2011). *ISO 26262 Road vehicles - Functional safety.*

Kleberger, P., Olovsson, T., & Jonsson, E. (2011). Security aspects of the in-vehicle network in the connected car. *2011 IEEE Intelligent Vehicles Symposium (IV)* (pp. 528-533). Baden-Baden: IEEE.

Lang, M. (2016, July). Integration of ADAS Into One Central Platform Controller. *Auto Tech Review, Volume 5, Issue 7*, pp. 32-35.

Liu, J. W. (2000). *Real-time systems.* Dehli: Pearson Education.

Lusserau, J., Stein, P., David, J.-A., Rummelhard, L., Nègre, A., Laugier, C., . . . Othmezouri, G. (2015). Integration of ADAS algorithm into an Experimental Vehicle. *2015 IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO)* (pp. 1-6). IEEE.

Martínez-Fernández, S., Ayala, C. P., Franch, X., & Nakagawa, E. Y. (2015). A Survey on the Benefits and Drawbacks of AUTOSAR. *Proceedings of the First International Workshop on Automotive Software Architecture* (pp. 19-26). ACM.

McClean, J., Stull, C., Farrar, C., & Mascarenas, D. (2013). A Preliminary Cyber-Physical Security Assessment of the Robot Operating System (ROS). *Proceedings SPIE 8741, Unmanned Systems Technology XV, 874110.* Maryland.

Miller, C., & Valasek, C. (2015, August 10). *Remote Exploitation of an Unaltered Passenger Vehicle.* Retrieved from Security Zap website: https://securityzap.com/files/Remote%20Car%20Hacking.pdf

Mosterman, P. J., Schieferdecker, I., & Zander, J. (2012). *Model-based testing for embedded systems.* Boca Raton, Fl.: CRC Press.

Motor Industry Reliability Association. (2013). *MISRA C 2012: Guidelines for the Use of the C Language in Critical Systems.*

NVIDIA Corporation. (n.d.). Retrieved from www.nvidia.com/object/drive-px.html

Open Management Group. (2015). *Data Distribution Service Version 1.4.* Retrieved from http://www.omg.org/spec/DDS/1.4/PDF/

Otterness, N., Yang, M., Rust, S., Park, E., Anderson, J. H., Smith, F. D., . . . Wang, S. (2017). An Evaluation of the NVIDIA TX1 for Supporting Real-time Computer-Vision Workloads. *Real-Time and Embedded Technology and Applications Symposium (RTAS)* (pp. 353-363). IEEE.

Panesar-Walawege, R. K., Sabetzadeh, M., & Briand, L. (2013). Supporting the Verification of Compliance to Safety Standards via Model-Driven Engineering: Approach, Tool-Support and Empirical Validation. *Information and Software Technology 55.5*, pp. 836-864.

Reedy, J., & Lunzmann, S. (2010). Model Based Design Accelerates the Development of Mechanical Locomotive Controls. *SAE Technical Paper 2010-01-1999*.

Solis, C., & Wang, X. (2011). A Study of the Characteristics of Behaviour Driven Development. *37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, (pp. 383-387). Oulu, Finland.

Streiter, R., Hiltscher, J., Bauer, S., & Jüttner, M. (2016). Probabilistic Integration of GNSS for Safety-Critical Driving Functions and Automated Driving—the NAVENTIK Project. In T. Schulze, B. Müller, & G. Meyer, *Advanced Microsystems for Automotive Applications 2016* (pp. 19-29). Springer International Publishing.

Williston, F. (2016, May 7). *Collision between a Car Operating with Automated Vehicle Control Systems and a Tractor-Semitrailer Truck.* Retrieved from National Transport Saftey Board: https://www.ntsb.gov/news/events/Documents/2017-HWY16FH018-BMG-abstract.pdf

Xenomai Project. (2017, September 28). *How does Xenomai deliver real-time?* Retrieved from Xenomai Project Website: https://xenomai.org/start-here/