Specifying a middleware for distributed embedded vehicle control systems

Andreas Reschka[1], Marcus Nolte[1], Torben Stolte[1], Johannes Schlatow[2], Rolf Ernst[2] and Markus Maurer[1]

Abstract— **The software of electric / electronic vehicle control systems is static in current series vehicles. Most of the systems do not allow maintenance or functional updates, especially in the field of driver assistance systems. Main causes are the testing effort for a software release and the wide variety of different configurations in different vehicle models. In this paper we take a closer look at the requirements for a middleware which allows such updates, verifies new software versions, and adds reconfiguration mechanisms for singular control units and distributed sets of control units.**

To derive the requirements we consider the general vehicular context with limitations in space, electric power, processing power, and costs together with four exemplary road vehicle control applications (cruise control, automatic parking, stability control, force feedback), and a full x-by-wire target vehicle for implementing these applications. The analysis of these three different sources of requirements results in desired middleware functionalities and requirements, especially concerning runtime timings and update timings. The requirements cover an update functionality with integrated verification, the exchange of applications on singular control units, and the degradation of functionality by switching between control units.

I. Challenge

The complexity of vehicle control systems increases with each new generation. This includes an increasing degree of dependability between individual system components. In contrast, hardware resources are still limited in vehicles due to restrictions in terms of space, electric power, and costs. Advanced driver assistance systems with environment perception and Vehicle-to-X-communication in particular demand a lot of processing power and memory. In order to reduce energy, space, and cost demands, a singular or small number of electronic control units (ECU) could be used for multiple applications, if the applications currently relevant are alone stored and processed in the ECUs. (In this context the term *application* is used for the implementation of a specific vehicle control function). Thus, inactive software components could be stored in slower and cheaper memory and be loaded into the ECUs on demand. Additionally, different implementations of a single application depending on the equipment level of a vehicle or the current capabilities could be used and loaded into the main memory of a single ECU when needed.

[1]Andreas Reschka, Marcus Nolte, Torben Stolte, and Markus Maurer are with the Institute of Control Engineering, Technische Universität Braunschweig, 38106 Braunschweig, Germany reschka,nolte,stolte,maurer@ifr.ing.tu-bs.de

[2]Johannes Schlatow and Rolf Ernst are with the Institute of Computer and Network Engineering, Technische Universität Braunschweig, 38106 Braunschweig, Germany schlatow,ernst@ida.ing.tu-bs.de

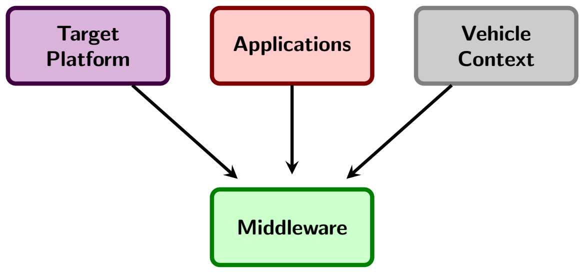

Fig. 1. Sources of constraints for the resulting requirements

In this paper we describe how we analyze requirements and constraints from the vehicle context in general, the desirable functionality of infrastructure software (operating system and middleware), functional requirements from exemplary safety-critical and non safety-critical applications, and the constraints we have in our target platform *MOBILE*, a fully x-by-wire controlled car (see Figure 1). The analysis results in a set of requirements for the infrastructure software of a single ECU or a set of ECUs. The work presented here is part of the *Controlling Concurrent Change* (CCC) project funded by the *Deutsche Forschungsgemeinschaft* (DFG).

A. Work Context

In the past, if applied at all, updates for automotive systems exclusively featured bug fixes for software errors induced during the design process. Currently the idea of automatic, incremental updates (concurrent change) is becoming more and more popular within the automotive community. Inspired by a growing app market in the smartphone sector, almost all automotive OEMs are currently starting to provide deep integration of tablet computers and smartphones into vehicle infotainment systems. Some OEMs also offer custom-tailored solutions and provide new features and bug fixes to their customers via over-the-air updates [1], [2]. This trend is also supported by suppliers like nVidia [3], who start offering (re)configurable hardware platforms for driver assistance and infotainment systems.

However, these updates do currently not consider the modification or extension of safety-critical systems which have direct influence on the driving-behavior of the vehicle. Due to potentially life-threatening failures, safety-critical systems still have to be thoroughly lab-tested and updates cannot be applied on the fly. Since the side effects of software changes are often diffuse, a reliable management-layer (middleware) is crucial when trying to apply concurrent change to complex embedded systems.

A component-based approach for dynamic updates of embedded automotive systems is described in [4]. The authors present a plugin-based approach for AUTOSAR-based ECUs. While they state that, theoretically, their approach also suits mixed-critically systems, they focus on non safety-critical system-components.

Martorell et al. [5] also present an AUTOSAR-based approach for enabling dynamic software updates for automotive ECUs during runtime, while still relying on offline lab-based tests. They demonstrate their solution with the help of non safety-critical components, while CCC aims at in-field verification and the exchange and / or update of safety-critical components.

In [6] Bosch and Eklund present a general architecture for the in-field deployment of new software components for automotive applications. However, they only provide infotainment-related use cases and explicitly exclude safety-critical systems from their approach.

Kim et al. [7] use a software redundancy system called SAFER for an autonomous vehicle. Their approach is focused on reliability and allows the exchange of software apps in real-time in order to switch from failing to functioning components. In contrast to the more general CCC approach, they only consider the exchange of software components with the same functionality in a single ECU rather than different applications. Limited computational resources or distributed applications are not considered. Last not least, SAFER does not consider protection mechanisms which are required according to automotive safety standards [8].

B. Controlling Concurrent Change

The CCC[2] project targets the management of changing applications in complex embedded system networks for automotive and aerospace applications. The special focus of this paper is on the automotive domain.

Today, typical development processes in the automotive industry can be divided into three stages at the most abstract level: specification, implementation, and verification (e.g. as described by the V-model, according to the ISO 26262 standard) (see also [9], [10]). Due to the complexity of modern automotive systems consisting of an increasing number of system components, the verification of a complete system is becoming more and more complex and time-consuming. Integration tests in particular often show hidden dependencies between the requirements for single system components, which makes the integration of updated components even more challenging. Additionally, updates of multiple ECUs complicate the update process because of dependencies between the updated software components.

One key aspect of the CCC project directly targets the reduction of offline testing effort in a development process. A model-based verification of the complete system is essentially based on the use of *contracts*. These are guarantees and assumptions which the components provide and / or make with respect to their interfaces to the environment. They are derived from detailed functional requirements for the system-components. In the CCC context contracts for different views of the system are considered. These views are *safety*, *availability*, *timing*, and *security*. The definition of contracts for each software component allows a formal analysis if a desired configuration of the system is feasible.

[2]Project homepage: `http://www.ccc-project.org`

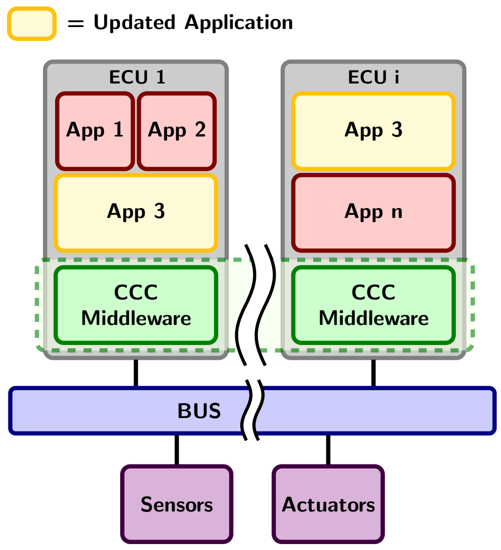

Fig. 2. Example of a distributed, CCC-enabled system; Applications can be mapped to multiple ECUs

If a component is rejected in the verification process, e.g. due to unsatisfiable safety constraints, a conflict-resolution strategy is applied which keeps the system in an operable state. With the help of the contracting approach, the aforementioned exhaustive testing process can be partly replaced with formal system-level analyzes which can be integrated into the (in-field) update process. Ideally it can be moved to the field, since the integrity of the whole system can be automatically verified whenever a new component is introduced.

As a part of the project, several driver assistance functions are implemented as applications which can be loaded and unloaded into a single ECU or a small set of ECUs.

II. Vehicle Development Constraints

Integrating ECUs and communication systems into a vehicle is subject to a variety of constraints, which have to be considered in the conceptual phase of an (advanced) driver assistance system. These constraints cover the available physical space in the vehicle for the integration of sensors, actuators, ECUs, and wiring. Less used space is desirable. A reduced number of ECUs reduces the required physical space for the ECUs themselves, the space for wiring, and the power consumption. These factors decrease the total hardware cost of the ECU network.

Additionally, the workload and consequently the efficiency of the ECU hardware increases. As a result the development and deployment costs of the ECUs can be reduced. Thus, a reduction or at least a constant number of ECUs has many obvious advantages in the overall vehicle context. Exemplary, this trend is currently visible in the development towards a central driver assistance ECU (German: zentrales Fahrerassistenzsteuergerät, zFAS). This ECU acquires data from multiple sensors and combines several driving functions [11]. A similar approach is used in the current S-Class from Mercedes-Benz, where several functions are integrated into the multipurpose camera system [12], [13].

Besides efficiency and resource issues, the potential of reducing development costs by reducing hardware development effort and especially offline lab testing effort, less ECUs seem to have more advantages.

III. Application-specific requirements

The aforementioned approach is applied to four exemplary use cases in different types, i.e. three driver assistance applications *Cruise Control* (CC), *Automated Parking* (AP), and a *Stability Control System* (SCS) for a passenger car. These are typical applications in current series cars, but they are still in the focus of research activities to improve their performance. Additionally, a *Force Feedback* (FF) application is covered, which is useful for a steer-by-wire system to provide feedback to a human driver from the torques and forces at the tires. These four applications are going to be implemented and tested in different updateable versions for our experimental vehicle MOBILE (s. section IV).

A. Cruise Control

In series cars, cruise control is realized either by a user input of a desired velocity or as Adaptive Cruise Control with an additional radar or lidar sensor and / or a camera to adapt the vehicle's velocity to other traffic participants in front of the equipped car. For our application we consider cruise control exclusively relying on user-input (Type CC.1) and a cruise control with a camera system, detecting speed limit signs (Type CC.2). The system controls the vehicle to the minimum of user input and speed limit. Of course, the system can be overruled by the driver. The type usable in a car depends on the availability of a camera system. If a camera is not connected, the cruise control only uses user inputs for its functionality. If a camera is available, the user input is used as a desired velocity and the detected velocity is an upper limit of the velocity. The ECU should detect the available type by itself. Both cruise control applications use the engine and brakes to control the velocity to the desired value. Lateral control is performed by a human driver.

In a first approach the exchange of the necessary software components is done offline, e.g. in a car workshop. Therefore, no real-time requirements exist and the software is exchanged in a controlled situation. An update functionality has to be integrated into the operating system / middleware to allow a safe exchange of software.

In a second approach, a failure of the camera system can be detected and the system degrades itself from CC.2 to CC.1 in a safe state, which maintains whenever the cruise control is deactivated.

B. Automatic Parking

The automatic parking application should park the car in a parking spot. As a cruise control functionality is not required while parking, the same ECU should be used for both applications. When activating the automatic parking application, the cruise control application is automatically unloaded and its resources can be used for parking. Again two different types of parking are planned. The first one uses on-board sensors to detect a parking space like in many series vehicles available (AP.1). The second type uses Vehicle-to-Infrastructure-communication to ask for a free parking space near the car and to receive the coordinates (AP.2). In both scenarios the car is parked automatically. Both types control the vehicle laterally and longitudinally to follow a calculated trajectory from a standstill to the final parking position.

The exchange from cruise control to automatic parking functionality is done online in a safe state, in this case a standstill of the car. If the Vehicle-to-Infrastructure-communication component is not available, AP.1 is automatically activated. Otherwise AP.2 is available as well and the driver has to decide, which type should be used. An automatic degradation from AP.2 to AP.1 is possible as well.

C. Stability Control System

This use case covers a stability control system, which stabilizes the car and additionally prevents the car from collisions. The stability control system has three different types of operation. Type SCS.1 is based on differential braking as used in series cars. It uses sensory feedback to keep the car controllable for the human driver. The SCS.2 type uses a side slip angle controller to allow an even better performance, especially in critical driving situations [14]. The side slip angle control is based on sensory feedback and actuation of the brakes, the engine(s), and the steering of a car. The third type (SCS.3) uses environment perception components. These are used to detect obstacles in the way of the car and thus allow the car to avoid collisions even in critical driving situations. Additionally they can be used to determine the road conditions.

Again as with cruise control and automatic parking the exchange of software components can be executed offline in a safe state, e.g. in a workshop, and online in a safe state, e.g. in a standstill or an uncritical driving situation. Additionally it is imaginable to exchange software components during operation to react to failures in the system and degrade the stability control functionality accordingly.

D. Force Feedback

To provide feedback to a driver from a steer-by-wire system, a force feedback system is useful. Depending on the cornering force at the tires, a torque is applied at the steering wheel against the steering torque of the human driver. This application will also be provided in two different variants: A standard mode which provides a predefined strength of the force feedback as well as a variant with user-definable strength.

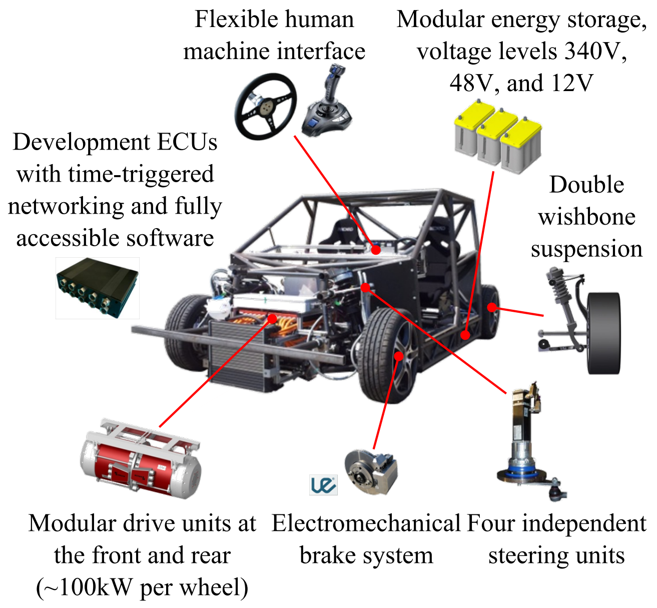
Flexible human machine interface
Modular energy storage, voltage levels 340V, 48V, and 12V
Development ECUs with time-triggered networking and fully accessible software
Double wishbone suspension
Modular drive units at the front and rear (~100kW per wheel)
Electromechanical brake system
Four independent steering units

Fig. 3. Set-up of the experimental vehicle MOBILE

IV. Target platform MOBILE

The experimental vehicle MOBILE[3] was custom built by the Institute of Control Engineering and the Institute of Engineering Design at the TU Braunschweig. The intended purpose of the vehicle is to serve as a tool for a variety of research projects on vehicle dynamics and mechanical or electric / electronic components. Still, the vehicle is also a demonstrator for use cases in the context of the CCC project.

The basic actuator set-up of MOBILE is given in Figure 3. The most important components are:

- the drive units at the front and rear axle, which allow to drive each wheel individually with a peak power of approximately 100 kW and a fixed gear ratio,
- the electric steering units at each wheel, which allow to control each wheel individually within a range of approximately +/- 43°,
- the electromechanical brakes,
- the flexible user interface to control the car, and
- the modular power supply based on two independent lead-acid battery packs.

The control of the actuators is implemented solely by wire. As a result, MOBILE offers the opportunity to evaluate control systems whose capabilities are beyond what a human driver can control utilizing inputs such as steering wheel, brake, and accelerator pedal.

Of course, pure by-Wire systems introduce additional functional risk into a system. Thus, the electronic system of MOBILE is designed redundantly for safety-critical tasks, i.e. each control unit and the data bus connections are available twice. To achieve an appropriate safety level based on this structure, an additional diagnostic and decision making system was developed to guarantee proper reconfiguration of the system in case of failures of individual control units [15], [16]. Also ongoing research deals with functional safety of basic functionality of MOBILE, e.g. the brake system [17].

[3]Project homepage: `http://www.ifr.ing.tu-bs.de/mobile`

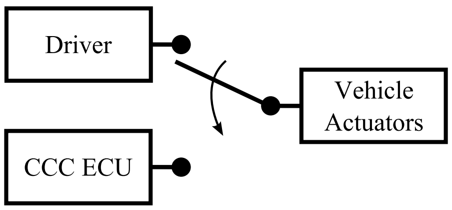

Fig. 4. Switching the control of MOBILE's actuators between driver and CCC application

Related to the CCC project, additional ECUs are added to the vehicle network of MOBILE. The ECUs implement all the functionality described in section III as well as the required mechanisms for the safe exchange of software. To ensure safe test operation, the interface between ECU and vehicle is designed as demonstrated in Figure 4. During manual operation the driver directly controls the actuators of MOBILE with steering wheel, brake, and accelerator pedal. Starting the currently active CCC application switches the control of the vehicle's actuators to the CCC ECU. Still, the switch is released only in standstill of the vehicle. In contrast, switching back to the driver is always possible and occurs as soon as the driver takes over control of one of the inputs. To realize the CCC showcases (section III) the CCC ECU has to run in parallel of the manual control, to allow an activation of the cruise control application and the parking application while driving manually. The stability control system application has to run in parallel to manual driving and all other applications.

The most important constraint in MOBILE is the cyclic rate of the FlexRay bus, which runs at a rate of 250 Hz. The control messages from an (advanced) driver assistance system ECU or a set of (advanced) driver assistance system ECUs have to meet this requirement. Additionally, the FlexRay configuration is static by design. Thus, it is not possible to connect new ECUs without reconfiguring all attached ECUs. A central FlexRay gateway which separates the FlexRay network from the ECUs running the middleware is necessary. In this case the additional ECUs can use the messages for vehicle control which are already provided in MOBILE's FlexRay cluster.

As the cruise control application is restricted to controlling the longitudinal dynamics of the vehicle, it is not highly critical to the vehicle's stability. Longitudinal vehicle dynamics possess typical bandwidths of about 1-2 Hz [14]. For the automatic parking application the speed is quite low and thus, instability of the vehicle is also not an issue. Therefore, for both aforementioned applications, a cyclic rate of 20 Hz should be sufficient for control messages from the (advanced) driver assistance system ECUs to the FlexRay gateway. The stability control system application requires faster execution with a rate of at least 100 Hz [14] [18].

The force feedback application is safety-critical, since a malfunction or a large lag can reduce the human driver's ability to control the vehicle. According to [19] and [20] typical bandwidths of force-feedback systems for steer-by-wire applications are 100 Hz.

V. Desired middleware functionality

As described above, a reduced number of ECUs leads to reduced costs for an embedded system network. However, with a reduced number of ECUs, the total computational power of the network decreases. In order to overcome this restriction, a suitable middleware has to provide means of *resource management and distribution*. This includes system-wide CPU time and memory assignment.

The applications using the middleware can be manifold. For an optimal usage of the computational power of the system, several applications have to share the same hardware. Thus, the middleware is responsible for running the necessary and desired applications and to distribute them on unused hardware. Besides resource distribution, *application distribution* is another desired functionality.

For safety reasons it is imaginable to move certain applications from one ECU to another. This improves the availability of certain applications, which may be safety-critical and provides a hardware redundancy mechanism. In order to provide a safe and efficient configuration of the system, the middleware has to apply suitable optimization algorithms.

Data exchange between ECUs is a *communication* task. Since the middleware already provides means of resource distribution, the developer cannot know which ECU is executing which tasks. Because of this, the middleware must also be able to provide transparent inter-ECU communication over the network. This also requires suitable interfaces to communication-hardware such as bus transceivers. Thinking of complex automated functions, typical examples of the need for transparent inter-ECU communication are shared components such as components for the calculation of a vehicle dynamics model or an environment model.

In order to enable online updates of applications in the sense of subsection I-A, the middleware must be able to verify pending updates. In order to maintain system integrity with respect to security and safety, malicious or faulty components have to be rejected. To integrate the update functionality into the software infrastructure to the largest possible extent, an *automatic application verification* is necessary which reduces offline lab testing effort for multiple different software versions in the same vehicle model and type.

While today, system function integration is thoroughly tested in lab environments and prototypes, this will no more be possible with automatic verification in the field. So, the middleware has to provide robustness against integration errors. The main instrument shall be monitoring which shall be used to supervise system timing and functionality according to the application contract specifications. Such monitors will be a main research target of CCC.

In future advanced driver assistance systems, which use environment perception and control the car laterally and longitudinally, a *degradation functionality* will be necessary to avoid complete outages or uncontrolled behavior of applications [21]. In case of malfunctions, other applications, which may work faultlessly, but reduce the overall functionality of the application's purpose (see [7]), should be automatically loaded.

If thinking about exchanging software components in an ECU and by that reconfiguring the system during runtime, fast exchange is necessary, particularly when considering applications which influence vehicle dynamics. The time needed for the exchange of components is therefore a metric for the performance of such a system. The *exchange time* metric is the time from unloading a piece of software to the start of execution of a newly loaded piece of software into the ECU.

Example: While updating at runtime or degrading the stability control system type because of technical failure, theoretically the cycle times from section IV apply to the speed of the update mechanism, too. As this seems to be quite difficult, it is more important that the degraded version of an application runs properly, instead of allowing a complete outage of the stability control system. A short lag of the change from the failing version to the degraded version is inevitable if the stability control system version is switched on the same ECU. A different solution could be to run two different versions of the stability control system in parallel on two different ECUs. In normal operation, the more sophisticated variant is used until it fails. In this case switching to the other version restores a basic stability control functionality. An approach for recognizing failing system components has been described in [15].

As applications like cruise control and automatic parking are not necessary for vehicle stability, an immediate degradation is not required for a safe operation of the vehicle. Due to this fact we consider an exchange time of one second to be appropriate for these use cases.

As a general requirement, the middleware must not interfere unintentionally with timings and availability within the network since these are important issues with regard to safe driving. These requirements depend on the respective application as described in section IV.

VI. Conclusions and Further Research

The requirements derived in this paper serve as an input for the other work groups in the CCC project in order to extend the respective middleware and adjust it to the needs of typical automotive applications. Additionally we will use the application-specific requirements as a basis for defining the contract interfaces for the respective software components. In summary, the resulting required features are:

- A guarantee as well as a monitoring of cyclic execution times at a rate of at least 20 Hz for automatic parking

and cruise control, and 100 Hz for stability control and force feedback,
- a reliable inter-ECU communication,
- an online application verification functionality,
- the guarantee of an appropriate online application exchange time (one second for non-critical applications and the cycle time for stability control and force feedback).

In our future research, applications under development will be integrated into MOBILE's onboard network on dedicated ECUs. With the help of MOBILE as an experimental platform we will verify the suitability of the CCC-approach for its application in automotive embedded networks and evaluate its performance with respect to the online exchange of applications.

ACKNOWLEDGMENTS

This work is part of the DFG Research Unit Controlling Concurrent Change, funding number FOR 1800.

We would like to thank the other work groups in the CCC project and the members of the MOBILE team for their support in our research.

REFERENCES

[1] W. Cunningham, "Tesla's path to the upgradable car," 2013. [Online]. Available: http://www.cnet.com/news/teslas-path-to-the-upgradable-car/

[2] K. Bullis, "Tesla motors over-the-air repairs are the way forward," 2014. [Online]. Available: http://www.technologyreview.com/view/523621/tesla-motors-over-the-air-repairs-are-the-way-forward/

[3] "Tegra k1 powers advanced driver assistance system for cars | NVIDIA." [Online]. Available: http://www.nvidia.in/object/tegra-k1-car-advanced-driver-assist-jan5-2014-in.html

[4] J. Axelsson and A. Kobetski, "On the conceptual design of a dynamic component model for reconfigurable AUTOSAR systems," *SIGBED Rev.*, vol. 10, no. 4, p. 4548, 2013. [Online]. Available: http://doi.acm.org/10.1145/2583687.2583698

[5] H. Martorell, J.-C. Fabre, M. Roy, and R. Valentin, "Dynamic software updates vs AUTOSAR," in *Embedded Real Time Software and Systems (ERTS 2014)*, Toulouse, France, 2014.

[6] J. Bosch and U. Eklund, "Eternal embedded software: Towards innovation experiment systems," in *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, ser. Lecture Notes in Computer Science, T. Margaria and B. Steffen, Eds. Springer Berlin Heidelberg, 2012, no. 7609, pp. 19–31. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-34026-0_3

[7] J. Kim, R. R. Rajkumar, and M. Jochim, "Towards dependable autonomous driving vehicles: A system-level approach," *SIGBED Rev.*, vol. 10, no. 1, pp. 29–32, 2013. [Online]. Available: http://doi.acm.org/10.1145/2492385.2492390

[8] "ISO 26262:2011 Road vehicles – Functional safety," International Organization for Standardization (ISO), Genf, Schweiz, 2011.

[9] M. Maurer, "Forward collision warning and avoidance," in *Handbook of Intelligent Vehicles*, A. Eskandarian, Ed. Springer London, 2012, pp. 657–687. [Online]. Available: http://dx.doi.org/10.1007/978-0-85729-085-4_25

[10] A. Reschka, J. R. Böhmer, J. Gačnik, F. Köster, J. M. Wille, and M. Maurer, "Development of software for open autonomous automotive systems in the Stadtpilot-project," in *8th International Workshop on Intelligent Transportation (WIT 2011)*, Hamburg, Germany, 2011, pp. 81–86.

[11] B. Kaiser, B. Augustin, and C. Baumann, "Von der Komponenten- zur Funktionsorientierten Entwicklung in der Funktionalen Sicherheit," *VDI-Berichte*, vol. 2188, 2013.

[12] M. Schopper, L. Henle, and T. Wohland, "DISTRONIC PLUS mit Lenk-Assistent und Stop&Go Pilot," *ATZextra*, vol. 18, no. 5, pp. 106–114, 2013.

[13] J. M. D. Mehren, M. Reichmann, M. Lallinger, W. Brenzen, and G. Weikert, "Intelligent Drive - vernetzte Intelligenz für mehr Sicherheit," *ATZextra*, vol. 18, no. 5, pp. 96–105, 2013.

[14] C. E. Beal, "Applications of model predictive control to vehicle dynamics for active safety and stability," Ph.D. dissertation, Stanford University, 2011.

[15] P. Bergmiller, M. Maurer, and B. Lichte, "Probabilistic fault detection and handling algorithm for testing stability control systems with a drive-by-wire vehicle," in *2011 IEEE International Symposium on Intelligent Control (ISIC)*, Denver, CO, USA, 2011, pp. 601–606.

[16] P. Bergmiller, "Design and safety analysis of a drive-by-wire vehicle," in *Automotive Systems Engineering*, M. Maurer and H. Winner, Eds. Springer Berlin Heidelberg, pp. 147–202. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-36455-6_8

[17] T. Stolte, P. Bergmiller, and M. Maurer, "Ensuring functional safety by networking systems from different domains," in *5th International Munich Chassis Symposium*, P. E. Pfeffer, Ed., vol. 2. Springer Vieweg, 2014, pp. 591–610.

[18] Continental, "Electronic stability control," 2014. [Online]. Available: http://www.conti-online.com/www/automotive_de_en/general/chassis/safety/safety_sicherheit_leben_en/hidden/elektr_stabilitaetskontrolle_en.html

[19] C. Wilwert, Y. Song, F. Simonot-Lion, Loria-Trio, and T. Clement, "Evaluating quality of service and behavioral reliability of steer-by-wire systems," in *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference*, vol. 1, 2003, pp. 193–200.

[20] R. E. Ellis, O. M. Ismaeil, and M. G. Lipsett, "Design and evaluation of a high-performance haptic interface," *Robotica*, vol. 14, no. 3, pp. 321–327, 1996.

[21] A. Reschka, J. R. Böhmer, T. Nothdurft, P. Hecker, B. Lichte, and M. Maurer, "A surveillance and safety system based on performance criteria and functional degradation for an autonomous vehicle," in *2012 IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Anchorage, AK, USA, 2012, pp. 237–242.