The RACE Project: An Informatics-Driven Greenfield Approach to Future E/E Architectures for Cars

Alois Knoll, Christian Buckl, Karl-Josef Kuhn, and Gernot Spiegelberg

Abstract As cars are turning more and more into "computers on wheels," the development foci for future generations of cars are shifting away from improved driving characteristics toward features and functions that are implemented in software. Classical decentralized electrical and electronic (E/E) architectures based on a large number of electronic control units (ECUs) are becoming more and more difficult to adapt to the extreme complexity that results from this trend. Moreover, the innovation speed, which will be dictated by the computer industry's dramatically short product lifecycles, requires new architectural and software engineering approaches if the car industry wants to rise to the resulting multidimensional challenges. While classical evolutionary architectures mapped the set of functions that constitute the driving behavior into a coherent set of communicating control units, RACE (Reliable Control and Automation Environment) is an attempt to redefine the architecture of future cars from an information processing point of view. It implements a straightforward perception-control/cognition-action paradigm; it is data centric, striking a balance between central and decentralized control. It implements mechanisms for fault tolerance and features plug-and-play techniques for smooth retrofitting of functions at any point in a car's lifetime.

A. Knoll (✉)
Technische Universität München (TUM) and fortiss GmbH, München, Germany
e-mail: knoll@in.tum.de

C. Buckl · K.-J. Kuhn · G. Spiegelberg
Siemens AG, CT RTC RACE, München, Germany

© Springer Nature Switzerland AG 2019
Y. Dajsuren, M. van den Brand (eds.), *Automotive Systems and Software Engineering*, https://doi.org/10.1007/978-3-030-12157-0_8

1 Introduction[1]

Over the last decades of "vehicle electronification and digitalization," it has become clear that information and communication technology (ICT) will determine the vehicle of the future. ICT is becoming the dominant factor and will drive vehicle developments by itself—people will want their cars to be equipped with ICT as powerful as it is in their offices and homes. For this reason, architectures and technologies for vehicle ICT cannot be viewed merely as a framework for gradual evolutionary innovations as they once were—they will determine the future development of cars in terms of functionality, innovation speed, and value creation. Architectures designed with these insights in mind will make new approaches and functions possible—from greater autonomy in driving to a more complete integration of the vehicle into the ICT infrastructure—and thus help significantly to achieve socio-political goals like energy efficiency or lower accident rates.

Today's automotive electronic/electric (E/E) architectures are a result of a long evolutionary process. The number of electronic control units (ECUs) has risen dramatically since their first large-series introduction into car technology in the 1970s. At present, the value added by ICT to the car is between 30% and 40%, but 80% of the innovations[2]—from entertainment systems by way of driver assistance systems to advanced engine and chassis control—are due to ICT. While the first antilock braking system (ABS) in 1978 had a small processor with a few hundred lines of code, today's luxury cars have millions of lines of code running on high- performance processors and dedicated chips. Still, however, the potential of modern ICT is far from being fully exploited by the car industry. For example, the move to full "drive-by-wire" or even "drive-by-wireless," although offering numerous advantages, has not been made because the necessary fundamental software structures are not deemed to be sufficient for this industry's standards. The development in areas like infotainment or telematics is also much faster, which has now become a real threat to traditional car manufacturers. One of the reasons is that customers want their car to keep pace with their rapidly changing ICT environment over the lifetime of the car—which will increase rather than decrease as we move to maintenance-free electric powertrains—and that static architectures will not be able to meet the dynamic, and as of now unforeseeable needs that will arise in the future.

What can be foreseen, however, is that there will be an ever-increasing speed of development in customer electronics, innovative applications (purely defined in software), new business models based on "platforms," and much more emphasis on environmental protection and resource economy. These aspects can only be taken care of if automotive original equipment manufacturers (OEMs) can keep up with the pace of the development of processing power, storage capacity, and

[1]Note that parts of this section, including Figs. 1 and 2, are an updated extract from [1].

[2]http://aesin.org.uk/about/about-automotive-electronics/

communication bandwidth—in other words, if they can rely on an architecture base that can make practical use of these developments.

In summary, ICT architecture is increasingly becoming a barrier—or an enabler!—to innovation.

2 A Brief History of ICT E/E Architectures for Cars

A look at the history of car electronics can be helpful to understand why the time is now ripe to develop radically new architectures (see Fig. 1). In the evolution of vehicle architectures, there is a trend for the architecture actually to accidentally become much more complex than it is essential for the achieved increase in overall functionality. Hence, new functions to be added become more and more cumbersome and hard to integrate, and the innovation activities therefore tend to lag behind. Only a substantial revision of the architecture enforced by a disruptive technology leap can bring the accidental complexity back down to its essential level.

The only way of achieving this is to raise the level of functional abstraction at which new functions can be integrated. This has already been observed in the automotive industry. To reduce emissions and improve comfort, in the 1980s it was necessary to expand the use of microcontrollers. Complexity relatively quickly

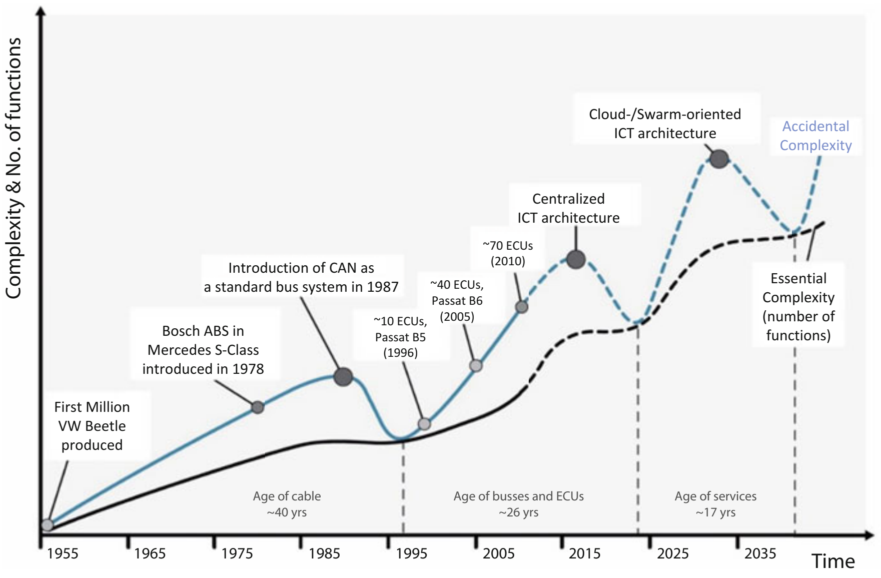


Fig. 1 Qualitative past and expected future growth of essential complexity due to increasing functionality to be supplied by electronic and software infrastructure, plotted against accidental complexity introduced by suboptimal mapping of ICT technology to vehicle technology

became a big problem because it was almost impossible to connect all these electronics modules together in a big cable harness. The solution was an abstraction of communication through shared media—buses like the CAN bus virtualized the physical connection. It thus became significantly easier to introduce new functions because integration no longer had to take place at the cable level, but at the information level.

While CAN is a networking standard originating from the automotive industry, the other important standard when it comes to architectures is Automotive Open System Architecture (AUTOSAR).[3] We are currently observing huge efforts in the industry to completely revise this standard in view of the requirements emerging from autonomous driving and an ever-increasing number of advanced driver assistance systems. This will be called the Adaptive AUTOSAR Platform, but when it will be published, it will be far from complete. Like AUTOSAR Classic, it will be subject to many revisions, and its initial version will be limited in scope.

The differences between AUTOSAR classic and RACE are manifold, and will become clear in the rest of this paper. Some differences are that the RACE RTE is dynamic to allow for plug and play. RACE provides support for fault tolerance and recovery, as well as a complete up-to-date software development environment, including separation kernels, as an underlying operating system layer. Moreover, the hardware structure based on central computers is a radical departure from the current architectures.

Notwithstanding, today's ICT architectures face problems because of the ever-increasing number of control devices. A new, centralized electric/electronics architecture, with a base middleware, might drastically reduce accidental complexity. New functions may then be integrated, not as physical electronic control devices but as software. The third step, finally, would be a further virtualization of the necessary total system of hardware and software (the hardware/software stack) into a service-oriented architecture. The underlying execution platform, composed of control devices and buses, would be entirely virtualized by middleware. The middleware would also implement non-/extrafunctional features, such as fault tolerance or communication delays. Then it would be possible to distribute functions as desired, even outside the vehicle; the car would thus become quite naturally part of a larger system.

At this point, a closer look at other sectors is in order. In the early 1980s, solutions in industrial controls and PCs proved that modular hardware and standard operating systems like MS-DOS and Unix can completely change entire industries. Open standards have resulted in increased innovations in hardware and software ever since. Economies of scale in production, with the associated cost reductions in modular hardware, made PCs attractive to end users. In the 1990s, a new architecture was introduced in aviation because of problems very similar to those in the automotive sector today. The "Integrated Modular Avionics IMA" [2] showed

[3]See, for example: O. Scheid, AUTOSAR Compendium, CreateSpace Independent Publishing Platform, August 2015.

that a new architecture can help reduce complexity and create a viable basis for future developments. Important concepts like centralizing and virtualizing computer architecture, local data concentrators, and "X by wire" can be adopted and adapted to the needs of automotive design—today more than ever. Robotics may also be of interest for a reorientation of ICT architecture in the automotive industry.

The logical architecture for controlling service robots, with its division into environmental perception, planning, and action, can particularly serve as a model for a logical architecture in the automotive industry. Important concepts from middleware architectures in robotics may also be of interest for the automotive sector, such as is being currently observed with the success of the Robot Operating System (ROS) [3], which in turn can only be a first step on the way to a quality-controlled industry-grade operating system for robots.

Let us now speculate briefly on how the future of ICT architectures would look if we learn from what we have observed in the past. As Fig. 2 shows, ICT architecture could thus develop in three steps. In an initial step, which is already going on today, ICT modules are integrated and encapsulated at a high level. In the second step, the ICT architecture could be reorganized with reference to all functions relevant to the vehicle. And finally, a middleware that integrates both the functions relevant to driving and the nonsafety-critical functions for comfort and entertainment would make it possible to customize vehicles for their drivers by integrating third-party software. Consequently, the automotive industry could manage the upcoming changes in two phases:

- **First Phase: Low Function/Low Cost**. This scenario is the most suitable for new market actors focusing on low-cost vehicles. The vehicle functionality and customer expectations for comfort and reliability are relatively low. The resulting requirement set is well suited for introducing a revised, simplified ICT architecture that is based on a drive-by-wire approach; actuator components are connected directly to the power electronics and the ICT. Actuators have a local energy supply and can be controlled via software protocols, reducing the number of cables and control devices.
- **Second Phase: High Function/Low Cost**. This assumes that ICT introduced in phase 1 has been optimized over the years and is now very reliable so that even customers with high expectations buy vehicles based on this architecture. This trend is reinforced by the ability to integrate new functions easily into vehicles and to customize them.

In summary, it is obvious that the necessary complexity of functions implemented by electronics and software will rise significantly:

1. Automated and autonomous driving will become state of the art very shortly. These functions are very complex and have high-performance requirements on the ECU. But even more exacting is a new requirement on E/E architectures: the system must become fail operational. While today it is state of the art to simply shut down a function as soon as an error is detected, in the future, fall-back functions must be provided to ensure that the vehicle can still be driven in a safe state even if there is a partial or even complete breakdown.

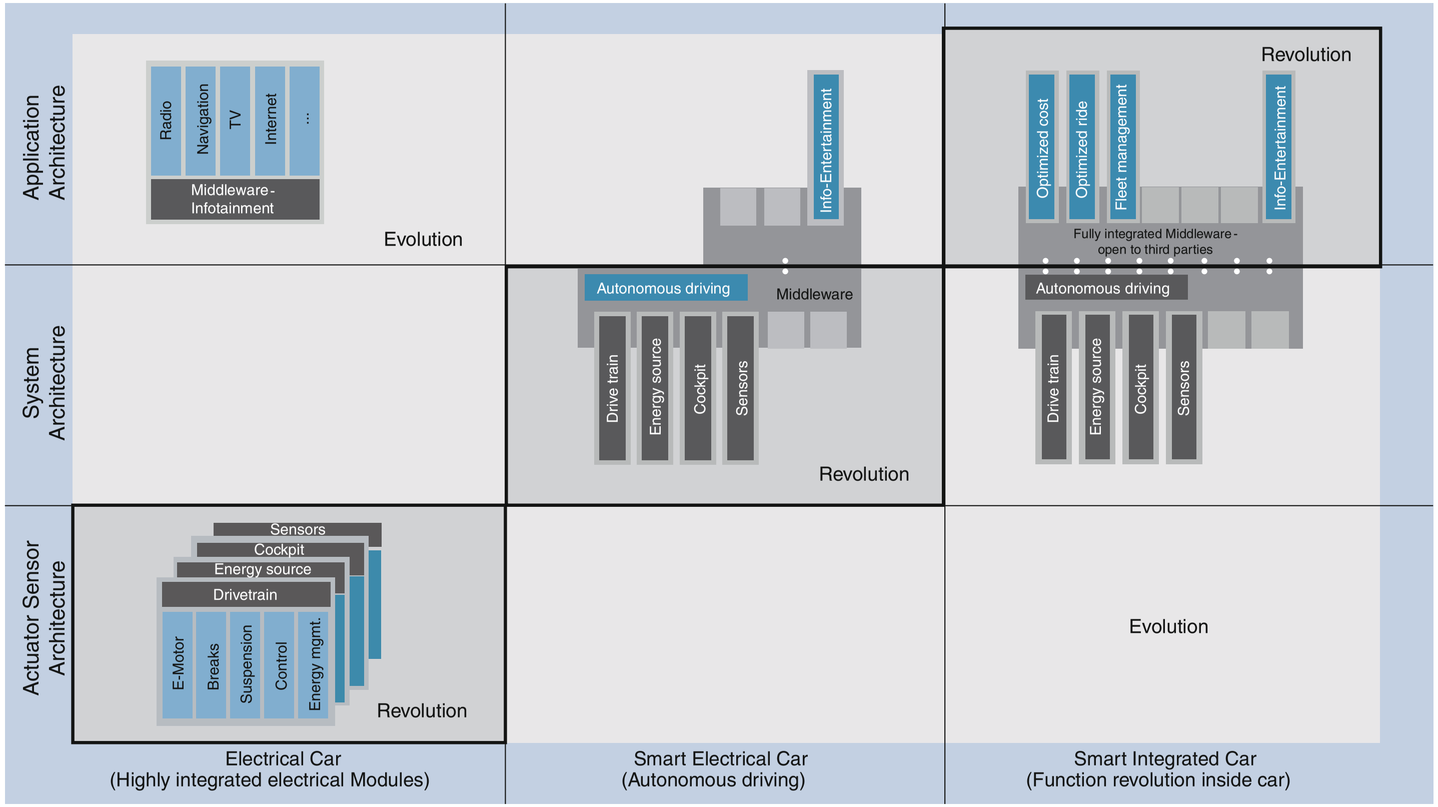


Fig. 2 Layers of ICT architectures and a possible development in evolutionary steps, as well as revolutionary “disruptive” steps. We expect the initial revolution at the level of the actuator-sensor architecture (and then evolutionary steps after that). Likewise, a disruptive development may take place at the system level and then finally at the application level

2. Connected mobility is another massive trend that will require an increasing interaction of the car with its infrastructure and with other IT-driven domains. This will not result in an increase in system complexity but generates another big challenge: security. While in the past cars were isolated systems, the growing openness of the E/E platform requires treating security as a key design criterion.

In summary, besides meeting additional functional requirements, future E/E architectures will have to provide additional extrafunctional properties, like a fail-operational system and security. Moreover, highly integrated mechatronic components are reinforcing the trend toward X-by-wire control or even X-by-wireless one. Through the introduction of middleware architectures and by encapsulating and abstracting from those mechatronic modules, integration can take place at a much higher logical level. Components to merge sensor data will become important elements of middleware architectures, along with mechanisms that ensure that safety-critical functions are separated from noncritical ones. They can be performed on a single computer without interfering with one another. This in turn will result in the centralization of all the computers in the car, similarly to server technology.

We will now look at the requirements in more detail. For an overview of the initial concepts at the time of the project's start, see [4].

3 A Set of Requirements for a New Architecture

In recognition of these trends, RACE[4] was started to establish a technology addressing the upcoming challenges. The main intention of RACE is to provide OEMs with an example of a platform technology that enables them to redesign their E/E architectures. By providing generic hardware components, a related runtime environment (RTE), and system engineering tools, OEMs can introduce new functions in most cases as software rather than the way it is today—as an ECU. This way, OEMs can benefit from faster innovation cycles of software. In the following subsections, the project goals will be listed in more detail.

3.1 Integration of New Functions in Software to Achieve Faster Development Times

The core platform will speed up the development of new automotive functions particularly related to:

- **Integration of new functions as software**: the core platform should be able to execute hardware-independent applications from all automotive domains; this includes but is not limited to body/comfort, driver assistance systems,

[4]For a short introduction, see http://w3.siemens.com/topics/global/en/electromobility/pages/race.aspx

power train, chassis control, and occupant and pedestrian safety. Hardware-independent applications are all those applications that do not require specific hardware components (e.g., actuators with power electronics). By integrating these functions on one single platform, RACE also addresses the trend that the functions of different domains are increasingly interacting with each other and the domain barriers vanish.

- **Software updates in the field**: users require continual updates to increase the functionality of their cars. Therefore, frequent software updates in the field of both of the platform and functions are an essential requirement. Agile development methods are to be supported.
- **Scalability and reuse**: the RACE core platform shall support the scalability to different platform variants. Reuse of applications must be supported.

This is accomplished by number of different concepts, most importantly a rigidly implemented publish/subscribe mechanism, along with a data-centric structure that not only ensures the necessary decoupling but also guarantees data consistency and sparsity—data are produced and stored at only one place across the whole system. Wherever possible, these concepts are supported by formal checks. For example, checks are made to determine whether a complete and unambiguous data flow graph can be obtained from the collection of application software components and modules. Moreover, if there is a subscriber with several potential (publishers in the computed data flow graph), then the RTE is checked to see if it has the appropriate data fusion methods available (used, for example, for redundant sensors). If this is not the case, the configuration will be aborted, and a corresponding message of the reason is generated.

However, it was beyond the scope of the RACE project to produce a complete guarantee of the overall behavior and timing of a RACE application architecture. To check whether each component is supplied with data is certainly not sufficient to ultimately ensure the functional integrity of the application architecture. This, however, is an interesting subject for follow-up research.

3.2 *Enabling New Business Models by Software Updates and Opening Function Development to Third Parties*

In order to meet the requirements arising from the OEMs' desire to permanently develop new business models around their cars in order to be able to react to changing market needs, the core platform shall support the integration of functions even in after-sales market. There should be no need to integrate these functions during the original design phase of the car. Moreover, the architecture should be open to the integration of third-party applications. This integration should not be restricted only to applications from today's Tier-1 suppliers but support the creation of new ecosystems (see Sect. 7).

3.3 Built-In Safety and Security

The core should be designed such that it provides the necessary safety and security for accommodating functions of the highest criticality level. It should support the development of dependable systems covering:

- Availability: readiness for correct service up to fail-operational quality
- Reliability: guaranteeing correct service
- Safety: absence of unreasonable risk up to Automotive Safety Integrity Level D (ASIL-D)
- Integrity: mechanisms to inhibit improper system alteration
- Maintainability: ability to undergo modifications and repairs
- Testability: simplifying verification strategies and analyzing misbehavior
- Security: protecting automotive software systems from unauthorized access, use, disclosure, disruption, modification, perusal, inspection, recording, or destruction

Furthermore, the core should provide the basis for functions with fail-operational requirements. The RACE core platform should support the execution of applications in fail-operational mode. Even in case of a subsystem failure, the core platform would guarantee the correct execution of the application.

The current software version of RACE runs under the PikeOS[5] hypervisor real-time operating system. This allows for third-party software to work "in isolation" and to prevent system crashes due to programming errors. While according to the above list of requirements there was some work undertaken to design a hardware security module for authentication, the implementation of security measures resulting from the integration of potentially dangerous and malignant software was not in the focus of the development work. However, it was made sure that no design decisions were taken that would prevent security measures to be implemented. By the same token, we expect this spatial and temporal separation to be a key component in meeting safety requirements: a hypervisor like PikeOS is used in many mixed-criticality environments and has stood the test of large-scale deployment and can, therefore, be expected to also meet the safety and software integrity needs in future cars.

3.4 Simplifying Migration from Other Platforms

Clearly, the core platform should support all applicable international automotive standards and state-of-the-art technologies. Furthermore, the core platform will support the collaboration between various partners by standardized data exchange

[5] https://www.sysgo.com/products/pikeos-hypervisor/

formats and support the integration of application software from various partners on a single ECU via a run-time environment and across the entire vehicle network.

Despite its superior functionality, production costs for a customer system platform based on the RACE core should not exceed the costs of a traditional E/E architecture. Moreover, the nonrecurring costs should be driven down to a minimum. Since these costs are hard to measure, it is important that production costs by themselves be equivalent or even lower.

4 RACE Architecture Concepts

To meet the requirements listed above, a completely new architecture has been developed, implemented as an advanced prototype, and integrated into a number of demonstrator cars (Fig. 3). The leitmotiv has been to design an architecture that fits the needs of information processing ("future cars will be computers on wheels"), capitalizes on the rich experience in Computer Science and Informatics in the design of mission-critical distributed systems, and makes it possible to keep pace with the rapid progress in methodology and tools for software design.

The central concept is that of a platform, centered about functions that are integrated very easily. The set of functions can hence change very easily and—taken as a whole—constitutes the complete functionality of the car controller across all domains. But not only that: since our focus is on communications and

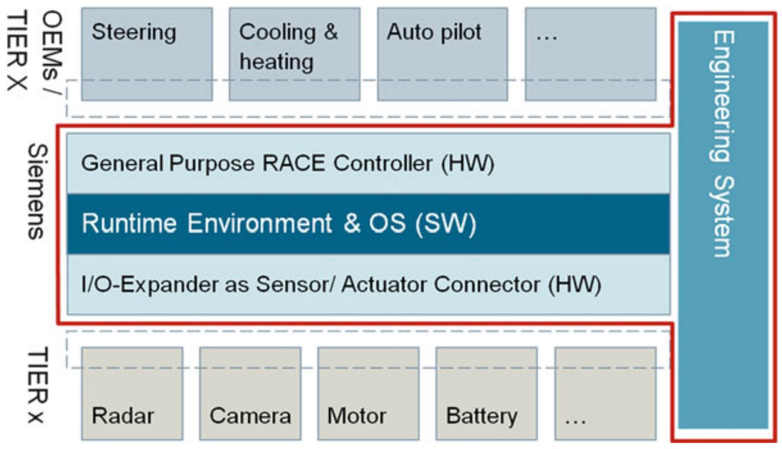

OEMs / TIER X
Steering
Cooling & heating
Auto pilot
...
Siemens
General Purpose RACE Controller (HW)
Runtime Environment & OS (SW)
I/O-Expander as Sensor/ Actuator Connector (HW)
TIER x
Radar
Camera
Motor
Battery
...
Engineering System

Fig. 3 Overview of the RACE architecture. The complete system consists of the controllers (GPRCs), the run-time environment (RTE) running on centralized hardware, and the IO expanders and interfaces. An integral part of the system is the engineering system, which provides all the software tools for programming the components in a certifiable way. The goal here is to abstract as much as possible from the hardware and to provide a "single system illusion" to the programmer

the communication protocols used within the car and those used outside the car are mostly identical, there is no longer a fixed barrier between the car and the infrastructure—this border becomes irrelevant.

4.1 General Structure and Communications

RACE is based on an execution platform consisting of General Purpose RACE Controllers (GPRCs), implemented as so-called Duplex Control Computers (DCCs) (see the following section), and a real-time run-time environment (RTE); see [5]. On top of this platform, OEMs can integrate their functions purely as software components. The architecture is designed to be scalable; if further processing power or resources, such as more memory, are required, additional RACE controllers can easily be added. The GPRCs communicate via a high-bandwidth, real-time Ethernet using the upcoming IEEE 802.1-TSN [6] standard.

Sensors and actuators can be integrated via common interfaces (CAN, local interconnect network (LIN), Ethernet, digital/analog input/output (IO)). If more IOs are required than what is offered by the GPRCs, IO expanders (IOXPs) can be attached to the GPRCs. An IOXP provides a number of IO interfaces, but in contrast to a GPRC it does not offer enough processing power to execute complex software functions. Its main purpose is to interface with sensors and actuators, which can provide any degree of local intelligence and/or data compression and/or preprocessing capabilities. In any case, the main benefit of using such a hierarchical system of data sources and data concentrators—as supported by our architecture—will substantially cut back on the cable harness, which may significantly reduce costs and failure probabilities.

Another concept that leads to significant cost and development time reduction is reusability. Software functions communicate via OEM-defined interfaces with each other and with the sensors and actuators. As a result, these functions can be reused across different vehicles. If an OEM so chooses, these interfaces can be shared throughout the industry (or collaboration partners), and reuse may even take place across the industry.

The system engineering tool chain includes a test system and a continuous integration solution. The tool chain is optimized for agile development. New software components can be tested seamlessly at all levels, from software in the loop, by way of hardware in the loop up to vehicle in the loop. The test system enables fault injection to test different scenarios, such as rare and typically irreproducible component errors. A configuration tool simplifies the integration and building of new vehicles, automating many steps that in the past had to be done by system integrators and were very cumbersome.

Altogether, the scalable platform, reusability of functions, and system engineering environment and tool chain offer a fast and flexible way to bring new functions or changes to future cars. We will now look at specific safety and security aspects in a little more detail.

4.2 Built-In Safety and Security

RACE implements a "Safety-Element-out-of-Context" approach as defined in ISO 26262. The run-time environment (RTE) offers a separation of all software functions to eliminate any unintended interactions between software components. As a consequence, the RACE platform can execute functions with mixed criticality on the same controller (see Fig. 4).

4.2.1 Separation Concept

To further simplify the development of safety-critical systems, there are several built-in safety mechanism patterns for different safety levels. GPRCs are currently designed with two-channel controllers, where each channel or "lane" has its own processing unit. For functions with no safety requirements (quality management), the function can be deployed on any channel. If the function has safety requirements but can be shut down in case of a failure, the function can be deployed to both channels of a GPRC. The replication can be realized as homogeneous replication (duplication the function) or diverse replication (two diverse implementations or a "productive function" and a "plausibility function/watchdog"); see Fig. 5.

The run-time environment monitors the consistence of the results on both channels and can shut down the function in case of an unwanted deviation. Functions with fail-operational requirements (meaning a function must still be operational even in case of component failures) are also available. These are implemented as a master-slave mechanism provided by the run-time environment. This mechanism allows the deployment of a function on two GPRCs to ensure the correct execution even if one GPRC fails. Depending on the required fail-over times, the system can be configured for cold or hot-standby mode.

4.2.2 Scalable Safety

All these safety patterns are based on an indication-based health-monitoring system built into the RTE. The RTE permanently monitors the data flow and execution of components. In case of a deviation from normal behavior, the RTE raises an error indication. A health monitoring component collects all these indications and determines the health status of the different fault-containment regions on application component, hardware, and network level. Several mechanisms are available to react to the failures of a component, ranging from fault masking if redundant results are available to the actual shut down and separation of the faulty component.

Besides these safety mechanisms, RACE also incorporates a configurable security mechanism, such as secure boot, authentication, authorization, and encryption. Several mechanisms of the RTE contribute to safety and security at the same time.

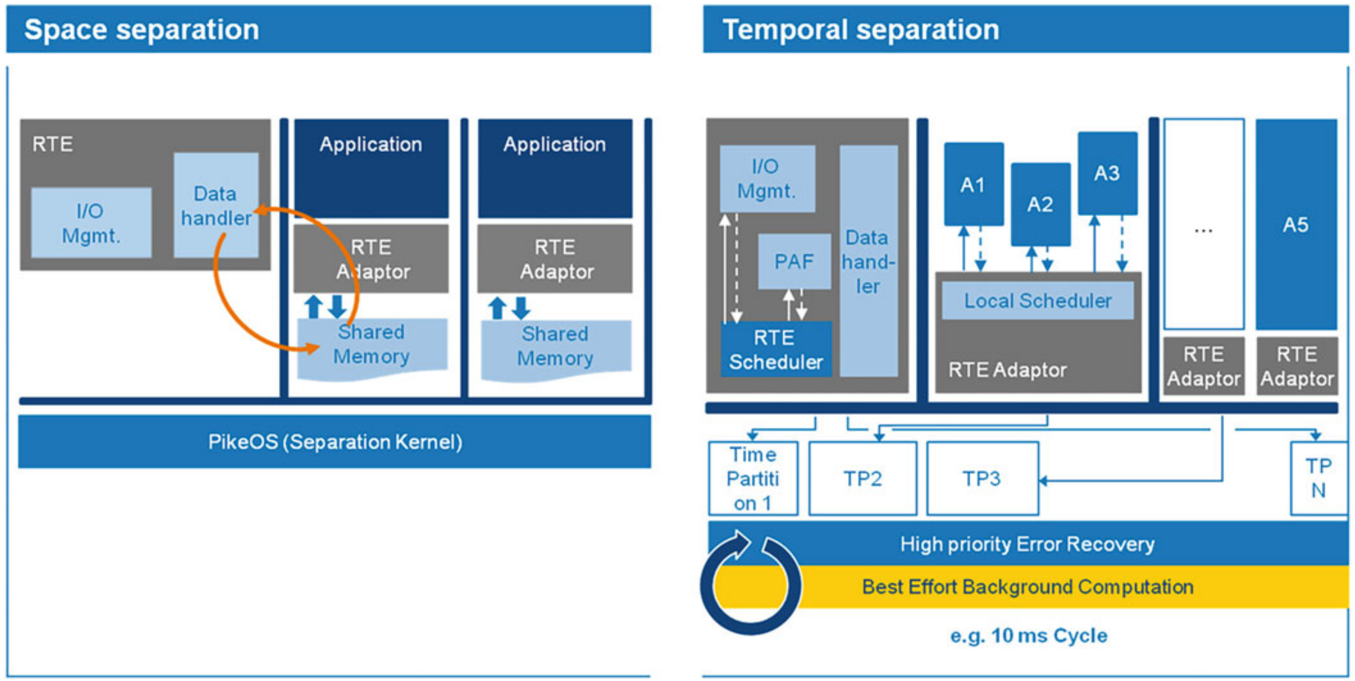
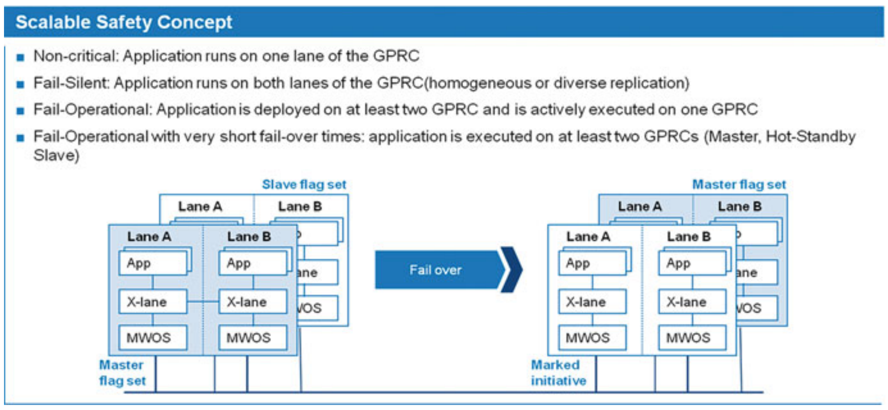


Fig. 4 RACE follows strict separation principles in space and time to ensure that applications can run independently—our approach to handling mixed criticality on one hardware platform. Currently, the underlying separation kernel is implemented in the PikeOS operating system, but other systems can be used as well


Scalable Safety Concept

- Non-critical: Application runs on one lane of the GPRC
- Fail-Silent: Application runs on both lanes of the GPRC(homogeneous or diverse replication)
- Fail-Operational: Application is deployed on at least two GPRC and is actively executed on one GPRC
- Fail-Operational with very short fail-over times: application is executed on at least two GPRCs (Master, Hot-Standby Slave)

Fig. 5 Illustration of the scalable safety concept. Depending on system configuration, applications can be run that are noncritical or applications with fail-silent/operational behavior—all in mixed mode on the same physical controllers

5 Implementation and Tooling

We believe that the most important aspect when a disruptive architecture is to be introduced is the compatibility with the current state of software engineering.

This will not only result in potential cost savings and a dramatic increase in productivity, but it will also enable the OEM to keep pace with the rapid developments in the consumer electronics world. Nevertheless, it is also of utmost importance to keep abreast and keep in sync with the developments in the field of mission critical systems in general and in autonomous systems in particular.

5.1 Information Flow

Figure 6 illustrates the information flow, which implements a "perception—cognition—action" cycle. This makes it not only possible to adapt paradigms from cognitive system theory and practice, but it also results in very logical layering of the processing functions at an easily comprehensible level of abstraction. At the lowest level, the data from (smart) sensors are generated and are routed into the system through a communication layer (middleware). Likewise, through this communication layer, the signals needed for behavior generation, i.e., the data for the (smart) actuators, are also distributed.

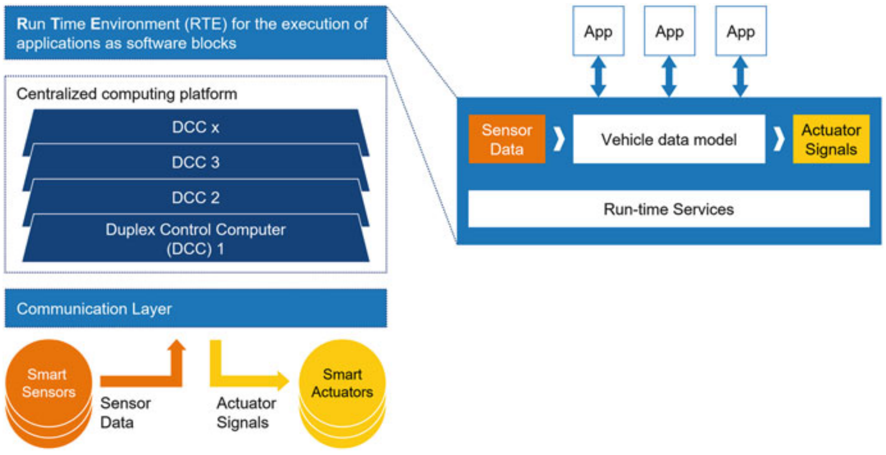


Fig. 6 Illustration of the basic implementation of a concrete car with four central Duplex Control Computers (DCCs). The general structure supports the construction of a “perception—cognition—action” architecture

The middleware subsystem used in RACE is based on CHROMOSOME [7]. In principle, other middleware systems could also be used, such as Data Distribution Service (DDS) [8]. One layer above the communication middleware is the layer of centralized processing through the control computers; these are the dual-lane processing elements described above. Depending on the processing power that is needed, there can be an indefinite number of DCCs.

The execution layer (run-time environment (RTE)) is responsible for the (virtual) connections between sensors and actuators. These connections can be dynamically generated, using the central data model of the vehicle, i.e., the abstraction of sensors and actuators. The RTE also provides the fail-operational services, as well as the “Plug & Play” management of all entities. The “Apps” correspond to the (retrofittable) automotive applications shown in Fig. 3.

Figure 7 shows an example of how this general architecture may be mapped to a topology in a real car: there are two DCCs, one for the drivetrain and energy source control, the other one for braking and steering. Hardware redundancy on the communication level is achieved by double ring structure: an inner ring for the direct DCC communications and an outer ring for sensors and actuators. Both rings are doubled so that the physical failure of one ring does not lead to a complete system failure. The communication protocol is a partial implementation of [6]; the DCCs can run mixed-criticality applications. There are also provisions for ingress/egress rate limiting to isolate faulty components, and there is hardware support for the precision time protocol (PTP).

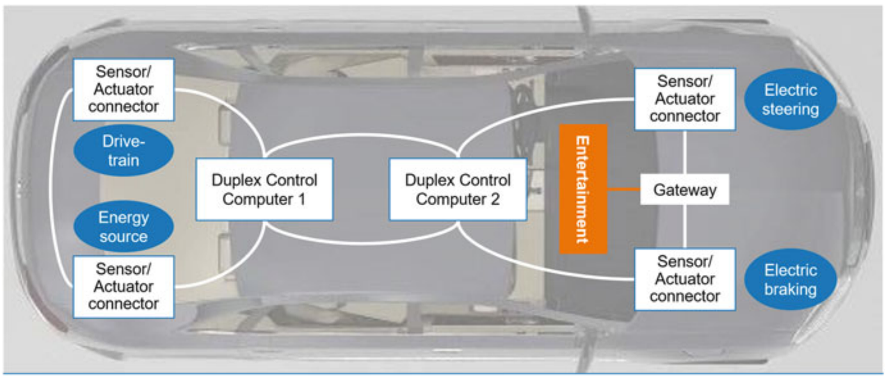

Sensor/ Actuator connector
Drive-train
Energy source
Sensor/ Actuator connector
Duplex Control Computer 1
Duplex Control Computer 2
Entertainment
Sensor/ Actuator connector
Gateway
Sensor/ Actuator connector
Electric steering
Electric braking

Fig. 7 Example of a topology of in a car with two central Duplex Control Computers. Inner glass-fiber ring (red) for DCC to DCC communication, outer ring (white) for sensor and actuator communications

5.2 *Software Design*

As mentioned above, one of the primary goals of RACE is to create a structural environment, i.e., the RTE, in which the latest developments in computer science, software engineering methods and tools, as well as insights from research into autonomous vehicles, can easily flow together. The resulting requirements on the RTE based on these project objectives are:

- The complexity of the software system should be reduced to the absolute essential minimum (see introductory section). This is realized by providing a modular development environment based on the decoupling of application modules that can establish dynamic communication links via virtual channels with guaranteed quality-of-service levels.
- Introduction of new complex functions at run time should be supported. This is realized by abstracting all automotive domain functions to the software level. New functions can hence be introduced simply by adding software modules. Clearly, this has not been done yet, but the foundations have been laid in RACE. This can be compared to the developments in the smartphone domain: they already combine a plethora of sensor modalities (vision, audition, touch, inertia . . .) in one single device, and hardly any new function needs a hardware accessory—it is all performed within the software. It is likely that at least in the sensor/communication domain, customers will expect the same from their cars.
- Plug-and-play capabilities are also mandatory. Customers will want to add new functions, or they want to attach new devices with added functionalities, such as additional entertainment equipment. If new complex functions are added (e.g., more sophisticated assistance and/or autonomy functions), this may even require the addition of one or more duplex control computers (DCCs), which means that the RTE has to be highly scalable.

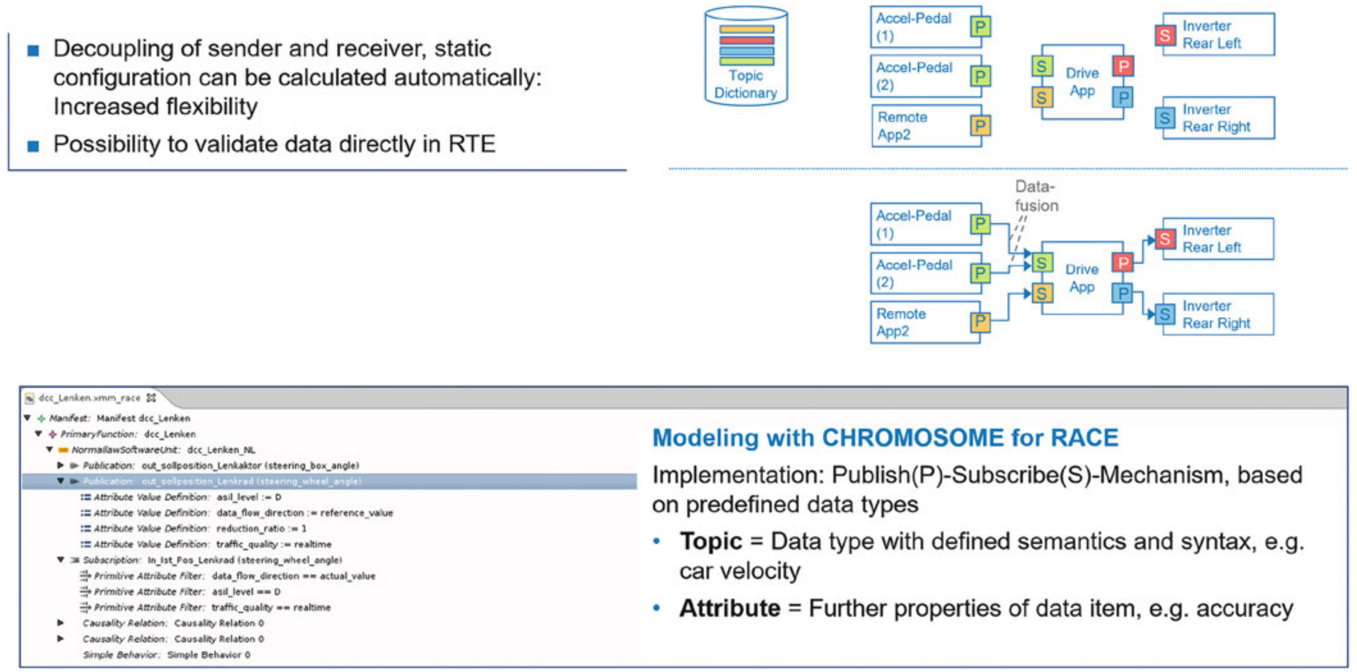
- Scalability with respect to functions and computation platform is also a must. This is approached by providing automatic configuration tools and functions wherever possible and therefore provide a maximum of scalability with respect to functions and resources.
- Certifiability is of the utmost importance. We believe that in the future, formal methods will have to be applied to ensure correct system behavior in all circumstances. However, until the underlying theory is powerful enough, we must provide all the generic mechanisms for safety and security, which will have to comply with the present standards for mission-critical systems.

Finally, there should also be a clear migration path from today's architectures to an environment like RACE.

Looking at the current state of run-time environments, and starting from that point, in RACE we have added the following features: (1) data-centric design, (2) segregation in time and space for running mixed-criticality applications on cost-effective hardware, (3) configurable safety and security mechanisms that enable tailored fault detection and recovery, (4) testing environment that can inject faults and trace them, and (5) automatic configuration functions at all levels.

It is beyond the scope of this article to elaborate on all of these topics. However, we describe one aspect of the RACE implementation: the data-centric communication design based on CHROMOSOME (see Fig. 8). The state of the art is to explicitly "wire together" senders and receivers via messages over a bus system. This results in strong coupling, an inflexible topology, and very often a redundant data acquisition and processing. The basic concept for data exchange is the decoupling of physical connections and logical communication relations. Moreover, data types can be associated not only with a syntax but also with semantics (why is it there? how is it used? etc.) and attributes (e.g., how precise is it?). These properties are stored in a central "topic dictionary," which lends itself to automatic configuration because dependencies, contradictions, redundancies, etc. can be checked automatically. All of these entities are supported by powerful tools, typically adapted from open-source tools, like Eclipse, that underpin rapid access to all system variables with all associated information, stored in one central repository.

Such automatic configuration—together with the modularity that enables the "plugging together" of predefined parameterized components—can be of great help in the system integration process. At the same time, the RTE registers and analyzes different error indications, which are accessible to all the DCCs in the system (see Fig. 9). There is a very fine-grained error handling available, with several reactions possible as a result of an error/failure indication. Errors inside the RTE can be handled by error masking (if there is redundant data). The application management can decide on a graceful degradation of the application and an adjustment of performance level, can change the master/slave role assignment, and/or activate/deactivate masters and slaves. Finally, the overall platform management can deactivate DCCs, and it can determine the whole platform mode, including an emergency shutdown after driving the system into a safe state.



- Decoupling of sender and receiver, static configuration can be calculated automatically: Increased flexibility
- Possibility to validate data directly in RTE

```
dcc_Lenken.xmm_race
▼ Manifest: Manifest dcc_Lenken
  ▼ PrimaryFunction: dcc_Lenken
    ▼ NormallawSoftwareUnit: dcc_Lenken_NL
      ▶ Publication: out_sollposition_Lenkaktor (steering_box_angle)
      ▼ Publication: out_sollposition_Lenkrad (steering_wheel_angle)
          Attribute Value Definition: asil_level := D
          Attribute Value Definition: data_flow_direction := reference_value
          Attribute Value Definition: reduction_ratio := 1
          Attribute Value Definition: traffic_quality := realtime
      ▼ Subscription: In_Ist_Pos_Lenkrad (steering_wheel_angle)
          Primitive Attribute Filter: data_flow_direction == actual_value
          Primitive Attribute Filter: asil_level == D
          Primitive Attribute Filter: traffic_quality == realtime
      ▶ Causality Relation: Causality Relation 0
      ▶ Causality Relation: Causality Relation 0
        Simple Behavior: Simple Behavior 0
```

Modeling with CHROMOSOME for RACE

Implementation: Publish(P)-Subscribe(S)-Mechanism, based on predefined data types

- **Topic** = Data type with defined semantics and syntax, e.g. car velocity
- **Attribute** = Further properties of data item, e.g. accuracy

Fig. 8 Data-centric design in RACE

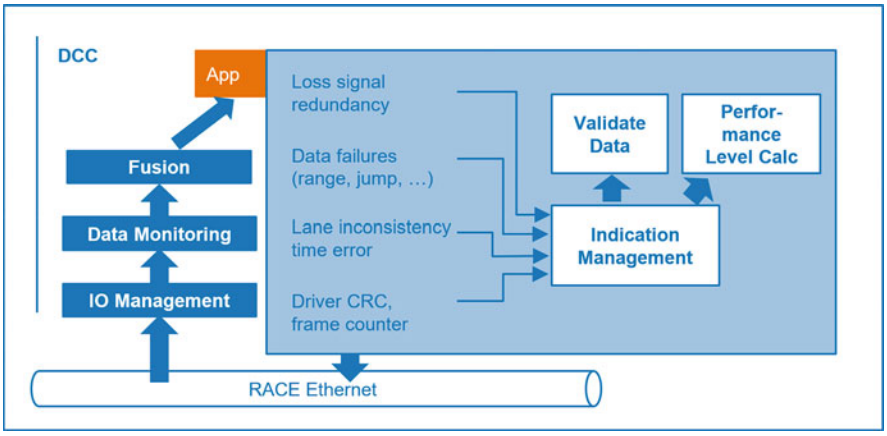


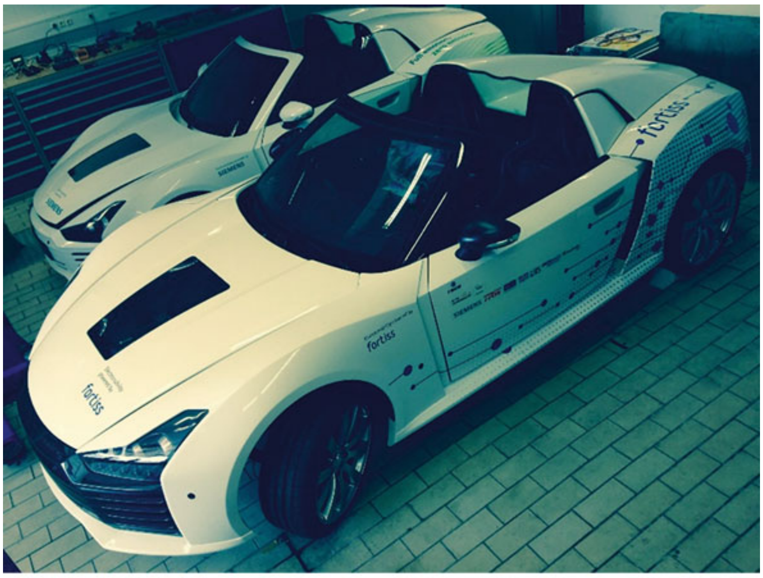
Fig. 9 The RTE registers and analyzes different error indications and makes them accessible system-wide

6 Realization on the Hardware Level

We briefly outline the design of the hardware that was developed for the prototypes. We have realized a number of successively complex testing configurations, equipped with prototypes of the hardware—starting from desktop "breadboards," by way of laboratory setups to complete cars. These cars (built by Roding GmbH, a manufacturer of small-series sports cars) were tested on a specially designed test rig (Fig. 10).

These cars have two wheel-hub motors and a steer-by-wire system without mechanical fallback, and they are a carbon/aluminum lightweight construction with a total car mass of 1250 kg. The braking system is a fully electric braking "future brake system" from TRW; the steer-by-wire subsystem was provided by Paravan GmbH. The E/E architecture is a redundant design based on RACE with an Ethernet ring structure, as depicted in Fig. 10, which uses of IOXPs for connecting sensors, actuators, and Human Machine Interface (HMI). The overall performance is 126 kW (up to 330 kW for a limited period of time); the overall torque is 1000 Nm (up to 2500 Nm for a limited period of time). Power electronics operate at a voltage level of 720 V; the battery capacity is 20 kWh.

Figure 11 shows the topology of the connections of the DCCs with all the units in the car, including the HMI and the steering wheel (an example of a redundant connection). The cars were built and programmed by a small project team. Several mission-critical functions were implemented and tested successfully. A complete set of functions, which would have enabled the cars to drive on a test site, was



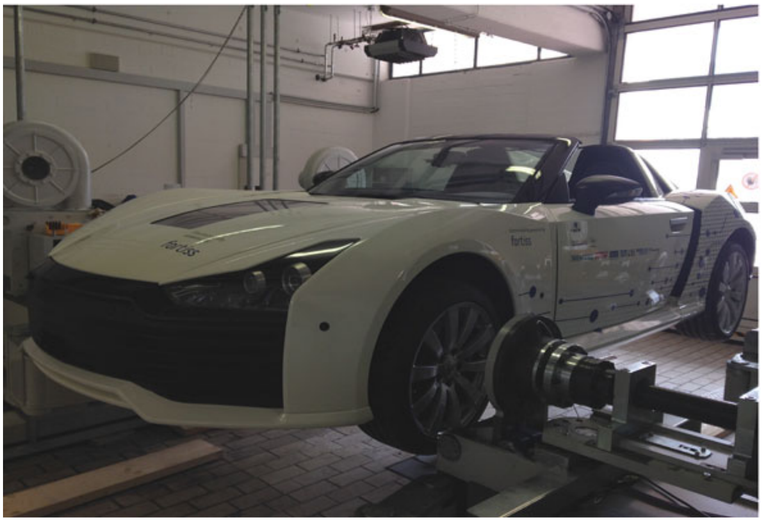


Fig. 10 The cars equipped with the RACE architecture. Top: physical appearance, bottom: car on test rig with direct coupling (all four wheels) to external electric motors that can induce very realistic driving dynamics scenarios

not implemented in RACE. As a project continuation, a parcel delivery vehicle was equipped for road testing by Siemens AG after the end of the RACE project (Fig. 12).

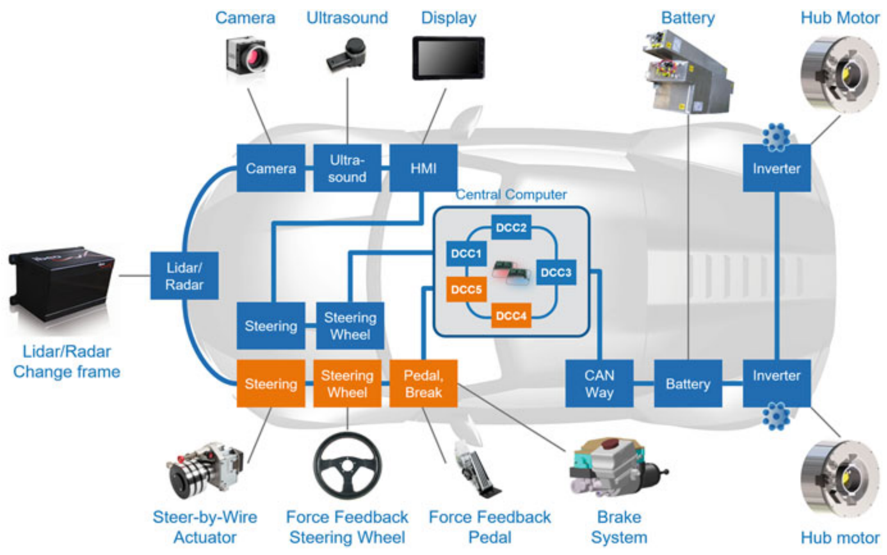


Fig. 11 The topology of the sensors, DCCs, and actuators in the prototype cars

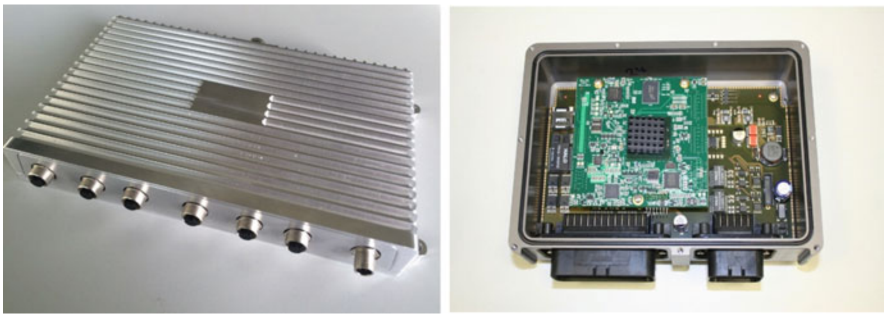


Fig. 12 The physical realization of the DCC (left) and the IOXP (right). The DCC has two ARM A9 CPUs and four RACE Ethernet connectors, as well as two test Ethernet. The IOXP features one CPU and several I/Os: CAN, LIN, digital, and analog. It also has two RACE Ethernet and one Test-Ethernet connector

7 Deployment and Business Opportunities

There has been an ongoing discussion about the commercial viability of alternative architectures, as outlined in the introductory section of this article. Clearly there are lots of arguments in favor of evolutionary approaches and certainly just as many arguments for disruptive changes. The problem for decision makers is that the question in general is virtually undecidable because there are so many factors that influence the decision. These range from nonrecurring costs by way of integration cost for every specialty of a car series, to the cost and risks resulting from customer desires for functional update over the lifetime, to the maintenance costs of (reusable) software, and to the replacement strategies of obsolete hardware.

The major challenge of automotive OEMs is the complexity of the E/E architecture conflicting with the required faster innovation cycles in the age of digitalization. This issue can be addressed with an approach like RACE because it reduces the number of controllers, minimizes the heterogeneity of network technologies, and offers several generic services that need to be covered during application development. Another complexity issue addressed by our approach is the undesired redundancy of data signals. With each new generation of cars, the number of data signals increases significantly.[6] The reason is mainly the difficult access to data across the different domains, controllers, and network technologies. A centralized approach offers a solution to this issue, and RACE alleviates this problem further by its data-centric design and the homogeneous hardware platform. Even if RACE is not used down to the implementation and hardware level, some of its concepts may still be used as a basis to redesign the function architecture with modularity and data centricity and be reused at the application level only.

Altogether, the approach pursued here enables OEMs to build up a common architecture across the different car platforms produced by the manufacturer, including the possibility to reuse the functions across the different cars. New and innovative suppliers can enter the automotive market and readily integrate their functions. While in the past new suppliers had to provide an integrated electronic control unit (ECU) with their function (software), which made it difficult for them to enter the market, they can now deliver just the software as a safety element out of context. This makes it also possible for the OEMs to reduce their dependency on tier-one suppliers. In addition, this approach also simplifies the in-sourcing of differentiating "brand-defining" functions by the OEM.

Finally, the approach using generic hardware and a high-integration platform offers the possibility to switch from a "bill of material" business model to a "revenue over lifetime" business model. Today, the electronics are individually optimized for concrete functions, which come with their own ECU. It is hard for developers to request additional resources to prepare for later software updates—or even new functions—because these resources would need to be stretched over a very high number (typically over 100) of ECUs. With the drastically reduced number of controllers that implement application-level functions, the software deployment, support, and update mechanisms can be changed in such a way that after-sales can be significantly increased. Therefore, a modular approach like this architecture lends itself to business developments along several lines:

- The RTE is the core offering, including a configuration tool.
- The full system is offered as a modular toolbox to be used (i.e., integrated and configured) by OEMs with focus on safety up to ASIL-D and fail-operational

[6]Some OEMs assume that the instantaneous speed of a car (i.e., one unambiguous variable) is identified by more than 20 different functions inside the car, e.g., by direct measurement, derivation from other sensor signals, or an estimation.

mode; it includes an operating system from a preferred supplier, communication system, and hardware development support.

- The application software development is delegated to a partner network. While RACE can be used as a "neutral" and open platform, third parties can integrate their functionality easily. It offers application engineering support up to Reference implementations.
- Fully integrated RACE controllers or support in HW design and manufacturing services can be provided.
- Integration services are not within the focus today, but customers can profit from the experience and expertise/Know-How in the RACE team.

The complete substitution of a proven architecture basis in large-volume car series without changing the structures of the producers' organizations is clearly infeasible. We therefore suggest collecting experience from the producers of smaller series, e.g., of special-purpose vehicles with low production volume ($\sim$3000 cars per year), which are individualized for specific customers. An interesting example is StreetScooter,[7] a company producing specialized parcel delivery vehicles. Due to the low volume of the market, this domain is not relevant for big OEMs and smaller companies can enter the market, testing the viability of their ideas. The major challenge of these manufacturers is exactly this low volume. It is not profitable to design specialized controllers for these cars, and therefore the manufacturers depend on already existing controllers from first-tier suppliers. However, due to the low volumes, required modifications are not given very high priority.

8 Summary

RACE is designed to fully support the requirements of upcoming automotive functions. With respect to automated driving, RACE offers a fail-operational platform, allows the integration of functions with different criticality levels on one controller, and offers high performance and safety simultaneously. Especially the latter is of importance as today's high-criticality solutions are based on processor technologies with low performance. Connected mobility is addressed by a high-bandwidth network based on [6], built-in interfaces to web services (prototype), and built-in security mechanism. RACE has developed several unique properties that will be required in the future:

- *No tradeoff between performance and safety*: while today's solutions rely typically on high-performance processors for functions with lower criticality levels (up to ASIL B) and low-performance processors for functions with high criticality levels, RACE has the potential to provide full processing power up to

[7] http://www.streetscooter.eu/

the highest safety levels. It is also a complete solution for functions with fail-operational requirements.

- *Hardware designed for use in series production*: standard suppliers optimize the controllers based on customer requirements. This is, however, only feasible for larger volumes. RACE provides controllers ready for application in car production. The low volumes of each single application are compensated by the general applicability across different cars.
- *Solution optimized for in-house configuration and integration*: today, integration of functions is typically done by the supplier. This leads to high costs and delays in development. RACE offers the possibility to configure the platform and to integrate software components by the OEM, shortening the development times. Configuration and integration tools automate many previously tedious tasks and reduce the risk of introducing errors. With RACE, the OEM has the choice to decide whether configuration and integration are done in-house or by a service provider.
- *Qualified tool chain and infrastructure for agile development*: RACE will provide a qualified tool chain and infrastructure for agile development, reducing the development time. Similar to prototyping platforms, RACE will offer a seamless integration of development tools. The result, however, will be serial code running on serial hardware.
- *Designed for testability*: RACE RTE was designed with testability in mind. All data flows can be monitored by a nonintrusive test system guaranteeing exactly the same behavior with and without the test system. Furthermore, a fault-injection infrastructure is available to provoke specific situations simplifying the verification systems significantly. This is made possible through a dedicated and fixed scheduling time slot for testing. This approach sacrifices some time and computational power, but we consider the ease of testing that results from it a good trade-off (see [9]).
- *Designed for updates*: RACE offers direct support for integrating new functions or updating existing functions via updates over the air. Via this mechanism, automotive functions become a freestanding product. The "Plug & Play capability" allows for new business models for the aftermarket.

Acknowledgments The development of the RACE platform was supported by the German Federal Ministry for Economic Affairs and Energy (http://bmwi.de/); see http://www.projekt-race.de/en/.

The authors wish to express their gratitude to the whole team who made the development of the concepts, tools, and the cars possible in record time. This project was only possible through a real concerted team effort and a lot of passion on all sides. Clearly, an overview paper like this can only describe results at a rather high level. The authors are happy to provide additional information on request.

References

1. Bernard M et al (2010) The software car: information and communication technology (ICT) as an engine for the electromobility of the future. A study for the German Federal Ministry of Economics and Technology. Published by fortiss GmbH. http://www.fortiss.org/ikt2030/
2. Watkins CB, Walter R (2007) Transitioning from federated avionics architectures to Integrated Modular Avionics. In: 2007 IEEE/AIAA 26th digital avionics systems conference, pp 2.A.1-1–2.A.1-10. https://doi.org/10.1109/DASC.2007.4391842
3. Quigley M, Gerkey B, Conley K, Faust J, Foote T, Leibs J, Berger E, Wheeler R, Ng A (2009) ROS: an open-source Robot Operating System. In: IEEE-ICRA workshop on open source software in robotics organized by Hirohisa Hirukawa and Alois Knoll, Kobe, Japan, May 2009
4. Sommer S, Camek A, Becker K, Buckl C, Zirkler A, Fiege L, Armbruster M, Spiegelberg G, Knoll A (2013) Race: a centralized platform computer based architecture for automotive applications. In: Vehicular electronics conference (VEC) and the international electric vehicle conference (IEVC) (VEC/IEVC 2013), IEEE, October 2013
5. Becker K, Frtunikj J, Felser M, Fiege L, Buckl C, Rothbauer S, Zhang L, Klein C (2015) Race RTE: a runtime environment for robust fault-tolerant vehicle functions. In: Proceedings of the CARS workshop, 11th European dependable computing conference – dependability in practice, 2015
6. http://www.ieee802.org/1/pages/tsn.html
7. Buckl C, Geisinger M, Gulati D, Ruiz-Bertol F, Knoll A (2014) CHROMOSOME: a run-time environment for plug&play-capable embedded real-time systems. In: Sixth international workshop on adaptive and reconfigurable embedded systems (APRES 2014), ACM, April 2014
8. http://www.omg.org/spec/DDS/
9. Fröhlich J, Frtunikj J, Rothbauer S, Stückjürgen C (2016) Testing safety properties of cyber-physical systems with non-intrusive fault injection – an industrial case study. Proceedings of the workshop on dependable embedded and cyber-physical systems and systems-of-systems (DECSoS). In: Skavhaug A et al (eds) Proceedings of the workshops international conference on computer safety, reliability, and security (SAFECOMP), vol 9923, Springer, LNCS, pp 105–107