RACE: A Centralized Platform Computer Based Architecture for Automotive Applications

Stephan Sommer*, Alexander Camek*, Klaus Becker*, Christian Buckl*,
Andreas Zirkler‡ Ludger Fiege‡, Michael Armbruster‡, Gernot Spiegelberg‡, Alois Knoll†
*fortiss GmbH, Guerickestraße 25, D-80805 München, Germany, {sommer, camek, becker, buckl}@fortiss.org
†Technische Universität München, Boltzmannstraße 3, D-85748 Garching, Germany, knoll@in.tum.de
‡Siemens AG, Corporate Research and Technologies, Otto-Hahn-Ring 6, 81730 München, Germany,
{andreas.zirkler,ludger.fiege,michael.armbruster, gernot.spiegelberg}@siemens.com

***Abstract*—In the last couple of years software functionality of modern cars increased dramatically. This growing functionality leads directly to a higher complexity of development and configuration. Current studies show that the amount of software will continue to grow. Additionally, advanced driver assistance systems (ADAS) and autonomous functionality, such as highly and fully automated driving or parking, will be introduced. Many of these new functions require access to different communication domains within the car, which increases system complexity. AUTOSAR, the software architecture established as a standard in the automotive domain, provides no methodologies to reduce this kind of complexity and to master new challenges. One solution for these evolving systems is developed in the RACE project. Here, a centralized platform computer (CPC) is introduced, which is inspired by the well-established approach used in other domains like avionics and automation. The CPC establishes a generic safety-critical execution environment for applications, providing interfaces for test and verification as well as a reliable communication infrastructure to smart sensors and actuators. A centralized platform also significantly reduces the complexity of integration and verification of new applications, and enables the support for Plug&Play.**

I. Introduction

Over the past 30 years, information and communication technology (ICT) has made significant innovations in automotive construction possible: from the anti-lock braking system in 1978 to electronic stability control in 1995 and emergency brake assist in 2010. Accordingly, ICT, and especially its software, has expanded significantly, from about 100 lines of code (LOC) in the 1970s to as much as ten million LOC.

The increase in size and complexity of the software is driven by the increasing number of software functions which emerge from single stand-alone application to a tightly interacting distributed system. The high complexity and the distribution of software functions over many different ECUs and bus systems leads to high cost for developing and testing functionality. This complexity is additionally increased by the introduction of technology like X-by-wire driven by the demands to realize highly and fully automated driving functionality.

One approach to reduce system complexity is to partition applications and their interactions into different domains / cluster as depicted in Figure 1, which is only sufficient as long as these domains / clusters only rarely interact and as long as the number of functions inside a cluster stay manageable.

In a recent study [1], we investigated the future use of software in electric vehicles and how the automotive ICT architecture will evolve in the next 20 years. The study was focused on electric cars, but its results can be mapped to cars with internal combustion engines. The study identified several architectural changes to satisfy the three essential capabilities: zero accidents, Plug&Play and always-on. For zero accidents like driver assistance up to autonomous driving require access to many functions located in different domains so the approach to partition these functions into domains is not sufficient anymore to keep the complexity low.

The complexity will even increase by the need for fail-operational behavior which will be required by most of the highly and fully automated driving functionality. To provide Plug&Play in the field even for safety critical applications, the system architecture needs to provide capabilities to guarantee extra functional properties for resource utilization, safety and security. In addition, methods need to be provided by the system for testing single functions by separating them from the remaining system.

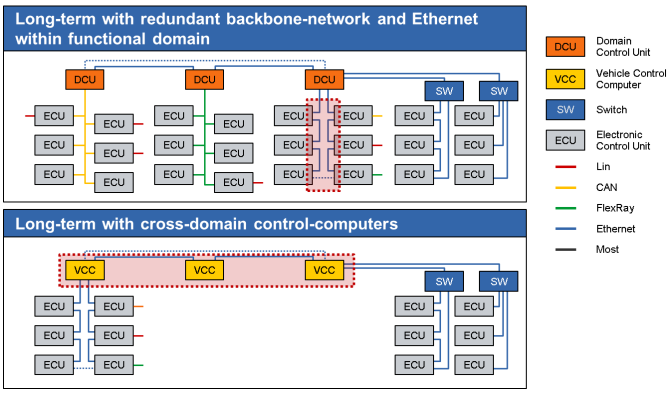
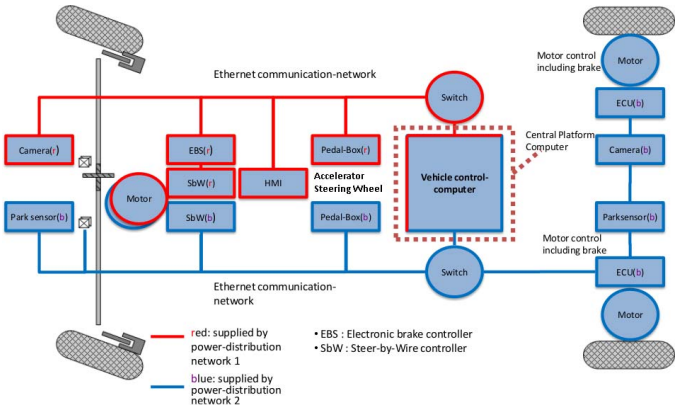


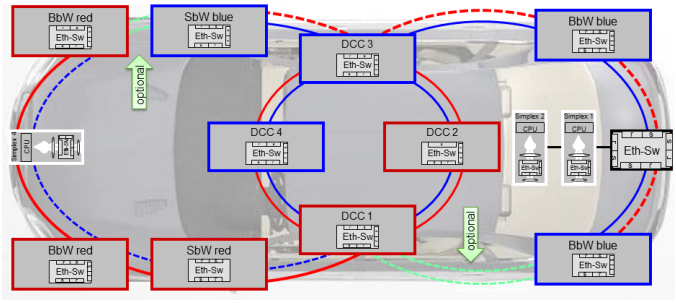
Fig. 1. Transition from functional domains and Ethernet backbone to a Vehicle Control Computer

Based on these challenges, the RACE [1] platform is developed consisting of a centralized platform computer inspired by well-established concepts from the aerospace domain like IMA and ARINC 653. The RACE platform consists of a redundant and reliable communication infrastructure based on Ethernet, and a runtime environment (RTE). The RTE provides

[1] http://www.projekt-race.de/



(a) Fail-operational design with vehicle control computer and smart sensors and actuators, e.g., for drive by wire



(b) Communication network topology based on Ethernet

Fig. 2. RACE system architecture

generic safety mechanisms as well as an execution platform with time and space partitioning for applications running on the same HW and having different ASIL classifications. The communication between applications is decoupled using the publisher subscriber communication pattern [2].

The remainder of this paper is structured as follows: In Section II challenges for a reliable system architecture are elaborated and a system design is derived. According to the system design, a suitable software architecture is presented in Section III. A selection of related work is discussed in Section IV. The paper is concluded in Section V by a summary of the results and the future work showing the new options provided by this approach.

II. Proposed Vehicular Architecture

As discussed in the introduction, new automotive architecture must support an increasing amount of new driver assistance functionality, such as predictive advanced driver assistance systems (ADAS) up to highly and fully automated driving, parking and charging. These changes in society drive the high level requirements for a new system architecture of modern cars.

This will force the replacement of existing combination of functionality and sensors by a centralized platform computer (CPC), which executes all high-level functionality. Sensors and actuators will become smart but still responsible for the low-level control tasks. These sensors and actuators are executing high-level control commands calculated by applications deployed on vehicle control-computers (VCCs), the ECU-items of the central platform computer. The interconnection of these smart components and the VCCs is done by a high-bandwidth communication backbone, such as Ethernet.

The main goal of the architecture is to support fail-operational behavior in combination with a flexible assignment of software components to computers enabling easy integration of additional software functions. According to ISO 26262 [3], ASIL-D including fail-operational safety requirements must be fulfilled. This requires a redundant power supply (blue and red lines in Figure 2a), a redundant communication infrastructure and redundant controllers.

A. Hardware components

In addition to cables, battery system and electric motors, the system is setup by a variety of subassemblies, which can be sender and receiver of data at the same time (smart sensors / actuators), as shown in Figure 2. These subassemblies are our smart components directly connected to the communication backbone. Subassemblies can directly process raw data, react on stimuli, conduct automatic-monitoring, or put local control loops into action. They are also flexible in their configuration and provide standardized interfaces, which allow an optimal adaption to specific tasks. Subassemblies can be built-up by commercial-of-the-shelf or problem-specific components, such as ASICS, DSPs, or FPGAs.

These subassemblies are interconnected with a centralized platform computer (CPC), which describes a computing element that consists of multiple VCCs (Vehicle Control Computers), see Figure 3.

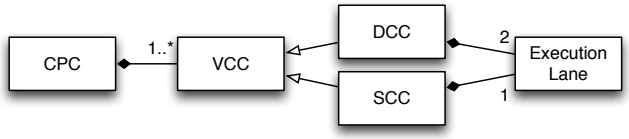


Fig. 3. Meta Model of the Central Platform Computer (CPC)

One type of control computer is a duplex control-computer and therefore considered using the abbreviation DCC. The main task of the DCC is to execute control functionality. To guarantee fail-safe, a DCC has two execution lanes and both execution lanes monitor input and output data mutually. When an error occurred their results will be discarded. To guarantee fail-operational behavior, at least a second DCC is required to take over the control tasks after the first one failed. The DCCs are built-up with high-performance multi-core systems and are interconnected with an Ethernet backbone [4]. The other type of control computer is a simplex control-computer (SCC), which has only one execution lane and is not capable to ensure fail-safe features.

The proposed vehicular architecture uses a variety of aggregates (smart sensors / actuators), which share their data and

control specific commands collected through the CPC. To collect and distribute the information in the system, a communication infrastructure is needed. For safety-critical functionality, particularly X-by-wire, this communication infrastructure must provide fail-operational behavior, thus a network infrastructure coming with redundant data paths as well as guaranteed latency and bandwidth.

In parallel, new types of sensors, e.g., stereo cameras, and infotainment systems like on-board Internet, TV, or a top-view parking assistant increase the required network bandwidth. At the same time, unit costs and cabling weight play an important role in automotive applications. For data rate intensive systems (e.g., camera based ADAS) costs of Ethernet are extreme low compared to the provided data rate, as shown in [5]. For redundancy, the RACE network uses a single Ethernet-based ring topology, as shown in Figure 2b. The Ethernet-switches are integrated in the VCCs and aggregates. Both directions of the ring are used to realize disjoint paths, thus a parallel redundant network is not necessary.

As there is no real physical independence between the two directions on the same ring, *babbling idiots* meaning computers that send arbitrary messages could trouble both paths. Novel "network fuses" solve that issue. "Slightly out of specification" errors are implicitly solved by the switched Ethernet architecture, which does not use a shared communication medium, but individual links between nodes. To reduce cabling length, a combination of one "inner ring", which incorporates the VCCs to a central platform computer, and one or more "outer rings" to connect the aggregates can be installed. The inner ring is needed to allow a cross check between all VCCs.

Each network node is connected with its neighbors by two communication links, resulting in ring architecture. The information is sent through both directions in the same ring (red and blue lines are representing the clockwise and counterclockwise direction in Figure 2b), which avoids a loss of information if one link in the ring is broken and thus guarantees availability. At the same time, safety critical messages are sent in a normal and an inverted characteristic in both directions to guarantee consistency amongst the DCCs to a very high degree. A message is valid if and only if both messages received over one link are identical in their representation, i.e., the consistent message matches its inversion. This enables a receiver to validate the message according to correct transmission as CRC methods do not detect errors with residual error probability. If no message or no valid pair of messages was received from one direction, the other redundant direction cares for the availability of data.

The outer ring does not necessarily need to be a closed ring by design, for assemblies without fail-operational requirements the ring can be left open and thus will become a simple daisy-chain (compare green "optional" links in Figure 2).

Unlike the heavy and expensive Ethernet cables and plugs known from office-environments, the RACE-Ethernet uses the BroadR-Reach [6] technology standardized by the OPEN Alliance SIG, which uses single twisted pair cables. The mixed-criticality communication system supports two entirely different traffic classes, which are realized on standard Ethernet frames.

The first traffic class is designed for cyclic communication of sensor and control data, especially for safety critical data. To guarantee the Quality of Service, the frames are sent redundantly as described above. To reach a lower transport delay, the frame size is limited to 250 bytes. Within this first traffic class different priorities, e.g., for CPC-internal and aggregate-frames, can be established.

The second traffic class is used for bulk traffic, which uses the whole Ethernet frame length and is sent on demand. These acyclic messages are used for updates of system firmware or other software components, streaming data (e.g., camera data), or bigger control and sensor data like navigation data. For those frames, no guarantee for latency is given and no redundant frames are sent, but therefore they can be sent without knowing anything about their traffic patterns in advance, just using the bandwidth not occupied by safety-critical frames. Within this second class, different priorities, e.g., for video-streams, can be established, too.

The impact of acyclic data on cyclic data is limited through prioritization and the preemption mechanism standardized in the IEEE 802.1 TSN (Time sensitive networks, formerly known as AVB generation 2) Group [7]. For high-end requirements, the Time-aware shaper will be available. The required synchronization is met with IEEE 1588v2 [8] transparent clocks. The benefit is that all required features can be covered with off-the-shelf standard hardware, empowered by the RACE RTE. When the requirements are raised in future, the network can - partly or completely - be shifted to Gigabit Ethernet.

B. Middleware-based Approach

The proposed software architecture is designed with different components, which provide automotive specific services, such as plausibility checks or frame unpacking. This design is influenced by decoupling and analytic decisions, as given in [3]. A middleware, which defines the Runtime Environment (RTE) together with the operating system, drivers, and components providing services (as depicted in Figure 4), interconnects all system components. A full version of the RTE runs on all VCCs of the central computer, but can also be used on smart sensors and actuators in a tailored version to provide rudimentary functionality.

III. Software Architecture

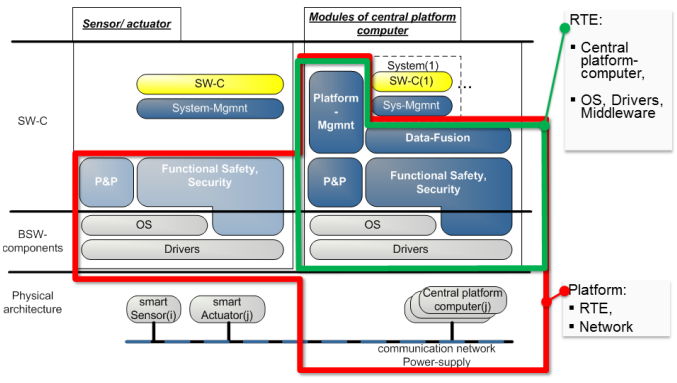


Fig. 4. RACE Runtime Environment (RTE) Layout

The middleware uses a data-centric approach. Thereby, a combination of current existing data elements represents the active vehicle state. Data is not directly delivered from a sender to a receiver, instead a publisher provides the data and subscriber can register to the data (Publish-Subscribe [9]). The data-centric approach enables a decoupling of applications from the infrastructure components, e.g., the network or the ECU. To represent various information of a vehicle, e.g., velocity or exterior temperatures, different data types, so called *topics* are pre-defined. These topics are used for the communication among software components over hardware boundaries. Wire format and data conversion are encapsulated by middleware services. A fail-operational system requires deterministic execution or termination time of a calculation. Thus, the applications are executed in cyclic manner with a defined duration, e.g., 10ms. At the beginning of each cycle data values are copied from the middleware to the application and before a cycle ends all results are copied back to the middleware.

A. Application Design

Utilizing the middleware as a basic execution, communication and safety layer, the complexity of developing applications is significantly reduced. In the RACE context an application is usually realized by one or more application components. Each application component contributes to realize a functional feature of the automotive application. The bundle of functions required to realize an application is called automotive function.

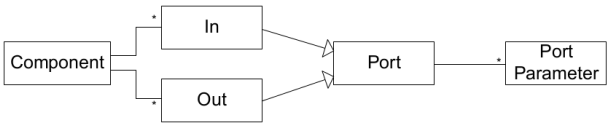


Fig. 5. Application component

Application components are black-boxes for the RACE Runtime Environment having ports for communication similar to the actor oriented design [10] depicted in Figure 5. To support layering, these black-boxes might however have nested invisible sub-components.

To provide advanced tooling support, e.g., for deployment during system assembly and for Plug&Play, applications are described using models [11], so called *Manifests*. These Manifests contain information about the application components and their external communication by defining the required or provided data at each port by specifying the published respectively subscribed topics. In addition to the topics, attributes are defined that describe the provided or assumed data instances in detail.

In order to decide the composability of applications and to obtain valid configurations of the RTE, additional pieces of information are required, such as worst case execution time (WCET) or automotive safety integrity level (ASIL). This information is also part of the Manifest. As an automotive application may be realized by multiple components, the Manifest contains information about each specific component as well as higher level information about the whole application.

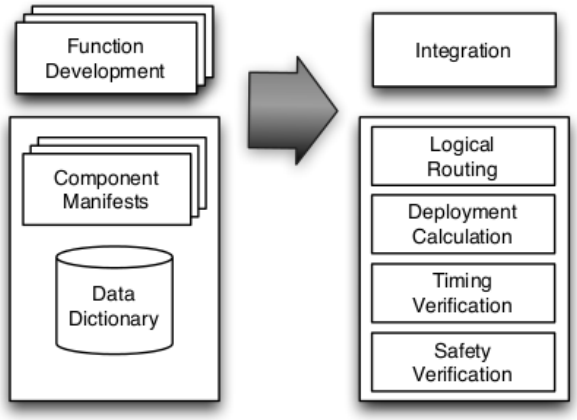


Fig. 6. Function Development and Integration

Based on the information stored in the Manifests during Function development, a suitable deployment of the applications onto the platform is generated as depicted in Figure 6.

Part of the deployment is the location of the applications on the platform as well as the interconnection of components, the so called wiring [12] or logical routing. The wiring is a matching based on the specified topics and attributes of the component ports. In addition a safety and timing verification is performed.

IV. Related Work

A. Communication

1) AFDX: AFDX (Avionics Full-Duplex Switched Ethernet) [13], ARINC 664 standard network, is an Ethernet-based communication system designed for safety-critical applications, particularly in aircrafts. It uses a time-based scheduler which can be compared to ATM networks or PROFINET IRT. The physical topology of AFDX is based on a central switch, thus called a star architecture. The central switch has a point-to-point connection to every network node and monitors timing and bandwidth of those. To achieve fail-operational behavior, AFDX uses two redundant independent networks with two network interfaces per node, two central switches, and two parallel network links to every node, one from each central switch. AFDX is established in avionics.

2) Flexray: Flexray [14] is a communication bus designed for automotive applications. The Flexray Consortium disbanded in 2009, but the specification is still available and currently being transferred into an ISO standard. It provides data rates up to 10 Mbit/s. The communication protocol is similar to the Time Triggered Protocol. Basically it is a TDMA[2] schedule, which offers an optional dynamic segment to transport event-triggered data. Flexray is a communication bus with a shared medium, while the topology can be line or star. To achieve redundancy, two parallel buses are required. Flexray is developed for and established in automotive applications.

[2]Time division multiple access

3) TTEthernet: TTEthernet [15] is one approach to extend Ethernet with real-time capabilities by adding a time triggered communication scheme on top. Due to its costs and the usage of specialized hardware components TTEthernet is not used in the authors' approach.

4) AVB: AVB [16] is a general term for a set of protocols which are standardized within the IEEE 802.1 AV Audio/Video Bridging Task Group. AVB generation 1 is an already finalized standard, which allows time-synchronized low-latency streaming through Ethernet networks. While AVB is aiming on audio and video streams, the idea of a data transport with guaranteed bandwidth and latency via standardized components immediately sparked interest in industrial automation and automotive applications. As AVB gen 1 does not yet ideally fulfill the requirements for control data, the currently developed AVB gen 2 standard, which is officially called TSN (Time Sensitive Networks), is expected to do. The preemption mechanism, which will be used in RACE, provides low latency for high priority packets, without the need for precise synchronization of the nodes and without the need for a complex traffic planning.
Nevertheless, precise synchronization is available using the 1588v2 [17] protocol with one-step transparent clock mechanism, which allows precise time-stamping in hardware and forwarding the sync messages without burdening the CPU with interrupts. The additional Time Aware Shaper will provide similar performance as other TDMA schedulers. A layer 2 routing mechanism, which allows for disjoint paths through arbitrary network topologies, is under development as well. As AVB and TSN are standardized in an IEEE task group, staffed with members of the most important switch hardware suppliers, all this will be available in off-the-shelf standard Ethernet fabrics for mass-market prices.

B. System architecture alternatives

1) IMA: The Integrated Modular Avionics (IMA) [18] represents real-time computer network airborne system, which consists of a number of computing modules of supporting numerous applications of different criticality levels. This concept replaces numerous separate processors and line replaceable units (LRU) with fewer, more centralized processing units. It also simplifies the development process of avionics software by its modular design. The structure of the network modules and API is standardized for accessing hardware and network resources. This improves the integration of new developed hardware and software components into existing or newly designed systems. Application developers can focus on the Application layer and need not to handle lower-level software layers, which in turn reduces the risk of faults occurring in these lower-levels.
The resulting complexity, which is added to the system by mixed-criticality at hardware and software resource level, requires a new system. Partitioning is the solution used by ARINC to segregate mixed-criticality applications. Our approach is similar to this one, but differs in the usage of Ethernet instead of the high costly AFDX communication network.

2) AUTOSAR: The AUTOSAR standard [19] describes a platform, which allows implementing future vehicle applications and minimizes the current barriers between functional domains. It will be possible to map functions and functional networks to different control nodes in the system, almost independently from the associated hardware. Therefore, AUTOSAR introduced a Run-Time Environment (RTE), which implements together with the operation system, AUTOSAR COM, and other Basic Software Modules the concept of Virtual Functional Bus (VFB). The VFB interfaces realizes the communication between AUTOSAR software components, thus the RTE encompasses both the variable elements of the system infrastructure as well as standardized services. Communication in AUTOSAR can be categorized by sender-receiver, message passing facility, or client-server, which provides function invocation. Communication is provided not only on a task basis (intra-task and inter-task) of the same partition, but also as a paradigm to exchange data between partitions and ECUs. Partitioning was introduced in version 4.1 of AUTOSAR [19] in order to enhance safety and support ISO 26262 [3]. AUTOSAR applications and RTE is configured during design phase and then deployed on the ECU as it is.
In contrast to AUTOSAR where the whole system must be replaced when new functionality is integrated, the RACE system supports the installation of new functionality without replacing the whole RTE.

3) ARINC 653: ARINC 653 [20] is the baseline operating environment for application software used within IMA and traditional ARINC 700-series avionics. It specifies a general-purpose APEX (APplication/EXecutive) interface between the Operating System (O/S) of an avionics computer resource and the application software. The basic philosophy of ARINC 653 consists of partitioning, which executes applications independent for fault containment by functional separation, and software decomposition. Communication between partitions is realized independent of the location of either source or destination partition. Thereby portability is ensured. Message exchange between two applications is identical regardless of whether the applications reside on the same module, or on different modules. Therefore, the standard uses a logical connection of channels and ports to describe the interconnections of applications in the same partition, in different partition, or on different hardware components. Additional mechanisms, such as buffers, blackboards, semaphores, and events, are used to establish intra-partition communication.
Similar to AUTOSAR, ARINC 653 uses partitioning to guarantee safety support and provides a fault containment, which separates faulty applications from good ones in the same partition without interference. ARINC 653 does not provide any possibility to update the system with new functionality after deployment similar to AUTOSAR.

4) SIMATIC: SIMATIC [21], Siemens automation standard, integrates automation solutions based on programmable logic controller (PLC) into a business environment. A central data storage embedded in a Master Control Station collects all configurations and other system specific information communicated by PROFINET and IWLAN networks. SIMATIC systems allow integration of high-performance or complex applications, such as closed-loop control, cam control, or motion control. All these functionality is supported by fast but reliable operations based on a fail-safe communication link and a redundant system design. Correct system behavior of safety modules and standard components is monitored by diagnostic functionality, which can be activated without programming the specific module under test.

The SIMATIC architecture relies on standardized communication protocols and compatible HW, which is not in common use within vehicles. For these reasons we re-use some of the successful concepts (like cyclic data images, manifests, central processing concepts), but not off-the-shelf hardware.

V. Conclusion and Future work

In this paper we showed the inadequateness of current automotive ICT architectures for future requirements driven by new, complex advanced driver assistance systems (ADAS), functions for autonomous driving and parking as well as by after-market extendibility. To cope with these challenges, the RACE ICT architecture, a system architecture based on a centralized platform computer in combination with smart sensors and actuators was proposed and elaborated in context of relevant related work like IMA and ARINC 653.

The key aspect of the RACE system and software architecture is a middleware based approach providing a reconfigurable, QoS-aware real-time system with means of safety, security and reliability build on top of a suitable duo-duplex hardware with an Ethernet communication ring. The configurability provided by the RACE RTE supports methodologies to simply reconfigure a deployed system to handle faults as well as to install new, even safety critical applications by the user.

Finally, the resulting RACE RTE will be deployed to a prototypical electric car to verify and prove the applicability within real world scenarios.

Acknowledgments

This work is partially funded by the German Federal Ministry of Economics and Technology under grant no. 01ME12009 through the project RACE (http://www.projekt-race.de/).

References

[1] Bernhard, M., Buckl, C., DO Richt, V., Fehling, M., Fiege, L., von Grolman, H., Ivandic, N., Janelle, C., Klein, C., Kuhn, K.-J., Patzlaff, C., Riedl, B., Sch"'atz, B., Stanke, C, *The Software Car: Information and Communication Technology (ICT) as an Engine for the Electromobility of the Future, Summary of results of the "eCar ICT System Architecture for Electromobility" research project sponsored by the Federal Ministry of Economics and Technology.* ForTISS GmbH, March 2011.

[2] OMG, "Data Distribution Service for Real-time Systems Version 1.2," http://www.omg.org/spec/DDS, 2007.

[3] "ISO/DIS 26262-1 - Road vehicles Functional safety Part 1 Glossary," Geneva, Switzerland, Tech. Rep., November 2011.

[4] M. Armbruster, L. Fiege, G. Freitag, T. Schmid, G. Spiegelberg, and A. Zirkler, "Ethernet-based and function-independent vehicle control-platform: Motivation, idea and technical concept fulfilling quantitative safety-requirements from iso 26262," *Advanced Microsystems for Automotive Applications (AMAA)*, pp. 91–107, 2012.

[5] A. Camek, C. Buckl, P. Correia, and A. Knoll, "An automotive side-view system based on ethernet and ip," in *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, 2012, pp. 238–243.

[6] Broadcom Corporation, *BCM89810 – BroadR-Reach Single-Port Automotive Ethernet Transceiver*, October 2008.

[7] Audio/Video Bridging Task Group, "Audio Video Bridging Systems," *IEEE Std 802.1BA*, vol. Draft 2.4, pp. 1 –, 2010.

[8] J. Eidson and K. Lee, "Ieee 1588 standard for a precision clock synchronization protocol for networked measurement and control systems," in *Sensors for Industry Conference, 2002. 2nd ISA/IEEE*. IEEE, 2002, pp. 98–105.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Elements of reusable object-oriented software," 1994.

[10] E. Lee, S. Neuendorffer, and M. Wirthlin, "Actor-oriented design of embedded hardware and software systems," *JOURNAL OF CIRCUITS SYSTEMS AND COMPUTERS*, vol. 12, pp. 231–260, 2003.

[11] A. Kavimandan and A. Gokhale, "Automated middleware qos configuration techniques using model transformations," in *Proceedings of the 14 th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2008), St. Louis, MO, USA*, 2008.

[12] A. Scholz, S. Sommer, C. Buckl, G. Kainz, A. Kemper, A. Knoll, J. Heuer, and A. Schmitt, "Towards and adaptive execution of applications in heterogeneous embedded networks," in *Software Engineering for Sensor Network Applications (SESENA 2010)*. ACM/IEEE, 2010.

[13] R. L. Alena, J. Ossenfort, K. I. Laws, A. Goforth, and F. Figueroa, "Communications for integrated modular avionics," in *Aerospace Conference, 2007 IEEE*. IEEE, 2007, pp. 1–18.

[14] R. Makowitz and C. Temple, "Flexray-a communication network for automotive control systems," in *Factory Communication Systems, 2006 IEEE International Workshop on*. IEEE, 2006, pp. 207–212.

[15] W. Steiner, "Ttethernet: Time-triggered services for ethernet networks," in *Digital Avionics Systems Conference, 2009. DASC '09. IEEE/AIAA 28th*, 2009, pp. 1.B.4–1–1.B.4–1.

[16] G. M. Garner, F. Feng, K. den Hollander, H. Jeong, B. Kim, B.-J. Lee, T.-C. Jung, and J. Joung, "Ieee 802.1 avb and its application in carrier-grade ethernet [standards topics]," *Communications Magazine, IEEE*, vol. 45, no. 12, pp. 126–134, 2007.

[17] J. C. Eidson, *Measurement, control, and communication using IEEE 1588*. Springer Publishing Company, Incorporated, 2010.

[18] "DO-297 Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations."

[19] AUTOSAR Group, "AUTomotive Open System ARchitecture (AUTOSAR) Release 4.1," AUTOSAR, 2013.

[20] A. E. E. Committee, *Avionics Application Software Standard Interface: ARINC Specification 653P1-2*. Aeronautical Radio, 2007.

[21] H. Berger, *Automating with SIMATIC: integrated automation with SIMATIC S7-300/400 : controllers, software, programming, data communication, operator control and process monitoring*. Publicis Corporate, 2006. [Online]. Available: http://books.google.de/books?id=3c5SAAAAMAAJ