AN INTEGRATED ARCHITECTURE FOR FUTURE CAR GENERATIONS

Real-Time Systems Journal, Volume 36, 2007, pages 101–133, Springer.

Roman Obermaisser
Vienna University of Technology
Real-Time Systems Group
romano@vmars.tuwien.ac.at

Philipp Peti
Vienna University of Technology
Real-Time Systems Group
peti@vmars.tuwien.ac.at

Fulvio Tagliabo
Centro Ricerche Fiat
Orbassano, Italy
fulvio.tagliabo@crf.it

Abstract The DECOS architecture is an integrated architecture that builds upon the validated services of a time-triggered network, which serves as a shared resource for the communication activities of more than one application subsystem. In addition, encapsulated partitions are used to share the computational resources of Electronic Control Units (ECUs) among software modules of multiple application subsystems. This paper investigates the benefits of the DECOS architecture as an electronic infrastructure for future car generations. The shift to an integrated architecture will result in quantifiable cost reductions in the areas of system hardware cost and system development. In the paper we present a current federated Fiat car E/E architecture and discuss a possible mapping to an integrated solution based on the DECOS architecture. The proposed architecture provides a foundation for mixed criticality integration with both safety-critical and non safety-critical subsystems. In particular, this architecture supports applications up to the highest criticality classes (10^{-9} failures per hour), thereby taking into account the emerging dependability requirements of by-wire functionality in the automotive industry.

Keywords: real-time systems, system architectures, automotive electronics, communication networks, legacy systems, dependability, component-based integration

1. Introduction

One can distinguish two classes of systems for distributed applications, namely *federated* and *integrated systems*. In a federated system, each application subsystem has its own dedicated computer system, while an integrated system is characterized by the integration of multiple application subsystems within a single distributed computer system. Federated systems have been preferred

for ultra-dependable applications due to the natural separation of application subsystems, which facilitates fault-isolation and complexity management.

Integrated systems, on the other hand, promise massive cost savings through the reduction of resource duplication. In addition, integrated systems permit an optimal interplay of application subsystems, reliability improvements with respect to wiring and connectors, and overcome limitations for spare components and redundancy management. An ideal future system architecture would thus *combine the complexity management advantages of the federated approach, but would also realize the functional integration and hardware benefits of an integrated system* [Ham03, p. 32]. The challenge is to devise an integrated architecture that provides a framework with generic architectural services for integrating multiple application subsystems within a single, distributed computer system, while retaining the error containment and complexity management benefits of federated systems.

The benefits of integrated architectures are becoming more and more important for the automotive domain, since there is a steady increase in electronics in automotive systems in order to meet the customer's expectation of a car's functionality [He04]. Cars are no longer simple means of transportation but rather need to convince customers with respect to design, performance, driving behavior, safety, infotainment, comfort, maintenance, and cost. In particular during the last decade, electronic systems have resulted in tremendous improvements in passive and active safety, fuel efficiency, comfort, and on-board entertainment. In combination with a "1 Function – 1 Electronic Control Unit (ECU)" design philosophy that is characteristic for federated architectures, these new functionalities have led to electronic systems with large numbers of ECUs and a heterogeneity of communication networks.

However, in order to satisfy the industrial demands on performance, dependability and cost with respect to a large variety of different car platforms, the current state-of-the-art system development methodology is heavily imposed to be reviewed, because of

- the strong competition among the carmakers;
- the requirement to continuously improve comfort functionality with stringent time-to-market constraints;
- the introduction of by-wire vehicle control and those functions introduced following normative pressure (e.g., fuel consumptions);
- a demand of greater versatility of the vehicle, conceived in a new view about modularity and standardization.

In particular, a low number of ECUs offers significant benefits with respect to architecture complexity, wiring, mounting, hardware cost and many others. Thus, a reduction of the number of ECUs is of great interest.

It is the objective of this paper to present the DECOS architecture for dependable embedded control systems for future automotive systems. This integrated architecture is based on a time-triggered core architecture and a set of high-level services that support the execution of newly developed and legacy applications across standardized technology-invariant interfaces. Rigorous encapsulation guarantees the independent development, seamless integration, and operation without unintended mutual interference of the different application subsystems. The integrated architecture offers an environment to combine both safety-critical and non safety-critical subsystems within a single distributed computer system. The architecture exploits the encapsulation

services to guarantee that software faults cannot propagate from non safety-critical subsystems into subsystems of higher criticality.

In this paper, we map a present automotive infrastructure onto the DECOS architecture and elaborate on the respective benefits, such as independent development, the assignment of integration responsibility, and an optimized use of resources through architectural gateway services. In order to maximize the economic impact, we propose a portable architecture that can be deployed in all segments of the car manufacturer (segment from A to E and also for luxury cars). Thereby a reduction of cost due to the increase in volume is expected. To achieve the required flexibility and portability, the architecture is be based on general purpose hardware and a modular software concept allowing to protect the intellectual property of vendors.

This paper is structured as follows. Section 2 investigates the advantages and disadvantages of federated and integrated architectures. Section 3 gives an overview of the DECOS integrated architecture, presenting the main underlying concepts. In Section 4, we discuss the current state-of-the-art of automotive architectures. The mapping to an integrated solution is the focus of Section 5. The discussion presented in Section 6 evaluates the integrated architecture based on prevalent E/E automotive trends. Section 2 is devoted to an overview of related work on integrated system architectures.

2. Federated vs. Integrated Architectures

One can distinguish two extreme classes of architectural paradigms for distributed applications, namely federated and integrated architectures. Real systems are often positioned between these extremes, leaning more to one or the other side. In a totally federated system, each application subsystem (e.g., multimedia domain in a car) has its own dedicated computer system, while an integrated system is characterized by the integration of multiple application subsystems within a single distributed computer system.

Advantages of Federated Systems

Although the federated system approach has significant deficiencies compared to the integrated systems approach, the federated system approach is superior with respect to complexity control, independent development of application subsystems, intellectual property protection, and exterior error containment.

Fault Containment. Based on the assumption that hardware faults effect an entire ECU, which is accepted for ultra-dependable systems [LH94, Kop03], federated systems offer advantages with respect to fault containment. When a hardware fault affects an ECU of a federated computer system, the fault impacts only a single application subsystem. Conversely, a hardware fault affecting an ECU of an integrated system can impact multiple application subsystems, because an ECU in an integrated system can be shared among software modules of multiple application subsystems. For developmental faults, better fault containment of a federated system in comparison with an integrated system is a consequence of the platform heterogeneity. The diversity of the employed hardware (e.g., processors, boards, etc.) and software platforms (e.g., operating systems) of the different application subsystems limits the regions of the immediate impact of developmental hardware and software faults.

Error Containment. Exterior error containment addresses error propagation between application subsystems. Since a federated system implements application subsystems via separate

computer systems, a computer system for an application subsystem remains functional despite the failure of other computer systems and the corresponding application subsystems. However, error containment within the application subsystem is no mere consequence of the federated system approach, but must be provided by the architecture or the application.

Independent Development. The ability for independent development follows from the near independence of federated systems. In federated systems, only a very limited level of interactions occurs via gateways, thus keeping the need for coordination between different vendors down to a minimum.

Complexity Control. The implementation of each application subsystem on a dedicated distributed computer system with controlled interactions across gateways also helps in managing complexity. The hiding of the internals of application subsystems enables a designer to understand the behavior of any particular application subsystem in isolation, i.e., without analyzing and fully understanding the rest of the system.

Advantages of Integrated Systems

Among the major advantages of integrated systems are hardware cost reduction and dependability improvements.

Hardware Cost Reduction. In contrast to federated systems, which require a dedicated computer system for each application subsystem, integrated systems facilitate the multiplexing of hardware resources. In the automotive area, the trend of federated architectures with increasing numbers of ECUs is coming to its limits, because systems are becoming too complex and too costly with the current practice of having each ECU dedicated to a single function.

Dependability Improvements due to Reductions of Wiring and Connectors. By tackling the "1 Function – 1 ECU" problem, the reduction in the overall number of ECUs also leads to increased reliability by minimizing the number of connectors and wires. Field data from automotive environments has shown that more than 30% of electrical failures are attributed to connector problems [SM99].

Fault-Tolerance. Replicated hardware is necessary to tolerate hardware faults in both federated and integrated systems. However, in a federated system these resources are available only to a single computer system out of the numerous loosely coupled ones. A computer system dedicated to a particular application subsystem can fail, although the overall number of spare ECUs would be sufficient to tolerate a given number of ECU failures. In an integrated system, resources are universally available among the different application subsystems.

3. The DECOS Integrated Architecture

The DECOS architecture [OPHS06] offers a framework for the design of large embedded real-time systems by physically integrating multiple application subsystems on a single distributed computer systems. The DECOS architecture distinguishes clearly between logical and physical structuring (top and bottom levels in Figure 1). Structuring rules guide the designer in the decomposition of the overall system at the logical level and support the transformation to the physical

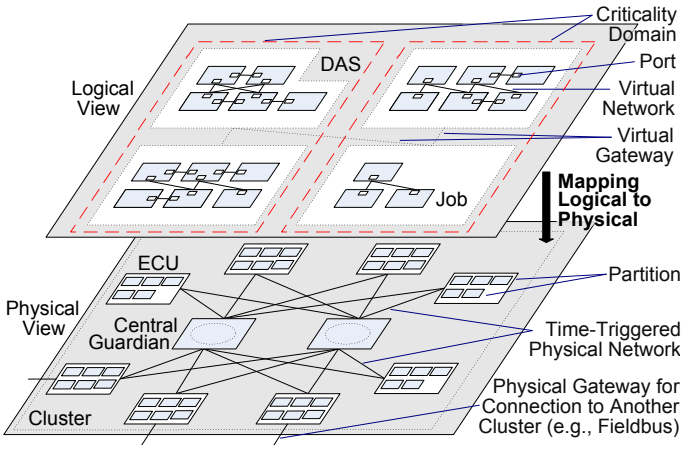

Figure 1. Structuring of a DECOS System – Logical and Physical View

level. Furthermore, the DECOS architecture offers to system designers generic architectural services, which provide a validated stable baseline for the development of applications.

Logical View of a DECOS System

A DECOS system (e.g., the complete on-board electronic system of a car) provides to its users (e.g., human operator) application services at the controlled object interface. By means of modularization, the DECOS system is decomposed into a set of nearly-independent *DASs*. Each DAS provides a subset of the overall application services, which is meaningful in the application context. An example for a DAS in the automotive domain would be a steer-by-wire subsystem [Hei03].

We further decompose each DAS into smaller units called *jobs*. A *job* is the basic unit of work and employs a *virtual network* in order to exchange messages with other jobs and work towards a common goal. The access point between a job and a virtual network is called a *port*.

Physical View of a DECOS System

From a physical point of view (see bottom level in Figure 1), a DECOS system encompasses a cluster containing a set of *integrated node computers* (also called ECUs), which are interconnected by a time-triggered physical network. The virtual networks introduced in the logical view are implemented on top of this time-triggered physical network. The use of a time-triggered physical network matches the predictability and fault-tolerance requirements of safety-critical applications [Rus01].

Every ECU provides one or more *partitions*, each hosting a corresponding job. A partition is an encapsulated execution space within an ECU with a priori assigned computational (e.g., CPU, memory, I/O) and communication resources (e.g., network bandwidth, latencies). By supporting the deployment of multiple jobs on one ECU, the DECOS architecture goes beyond the prevalent "1 Function – 1 ECU" design methodology [BS05, GFL+02].

In addition, a DECOS system can contain connections to external systems, such as a fieldbus network or a legacy cluster. For this purpose, ECUs implement so-called *physical gateways*.

Architectural Services

The DECOS architecture offers generic architectural services that separate the application functionality from the underlying platform technology. The architectural services facilitate reuse and reduce design complexity. The specification of the architectural services hides the details of the underlying platform, while providing all information required for ensuring the functional and meta-functional (dependability, timeliness) requirements in the design of a safety-critical real-time application. Applications build on an architectural service interface that can be established on top of numerous platform technologies.

In order to maximize the number of platforms and applications that can be covered, the DECOS architecture distinguishes a set of three *core services* and an open-ended number of *high-level services* that build on top of the core services. The core services are as follows:

1 **Deterministic and Timely Transport of Messages.** This service performs periodic time-triggered exchanges of state messages. TDMA controls the media access to the replicated communication channels and a communication schedule determines the global points in time of all message transmissions. Slots are statically assigned to ECUs in a way that allows each of them to send a message during a communication round.

2 **Fault-Tolerant Clock Synchronization.** In a distributed computer system, ECUs capture the progression of time with physical clocks containing a counter and an oscillation mechanism. An observer can record the current granule of the clock to establish the *timestamp* of an event. Since any two physical clocks will employ slightly different oscillators, the time-references generated by two clocks will drift apart. Clock synchronization is concerned with bringing the time of clocks in a distributed system into close relation with respect to each other.

3 **Strong Fault Isolation.** Although a Fault Containment Region (FCR) can demarcate the immediate impact of a fault, fault effects manifested as erroneous data must be prevented from propagating across FCR boundaries [LH94]. For this reason, the DECOS architecture provides error containment for failures recognized within the fault hypothesis [OP06]. In addition, DECOS offers a reliable distributed computing platform by ensuring that a message is either consistently received by all correct ECUs or detected as being faulty at all correct ECUs.

Any architecture with a time-triggered physical network that provides these core services (e.g., Time-Triggered Architecture (TTA) [KB03], FlexRay [Fle05], Time-Triggered Ethernet (TTE) [KAGS05]) can be used as a basis for the implementation of the DECOS integrated architecture. The small number of core services eases a thorough validation, which is crucial for preventing common mode failures as all high-level services and consequently all applications build on the core services.

Based on the core services, the DECOS integrated architecture realizes high-level architectural services, which are DAS-specific and constitute the interface for the jobs to the underlying platform. Among the high-level services are the virtual network services [OP05], the gateway services [OP05b], and the diagnostic services [PO06]. For example, the virtual network services establish on top of the core services the communication infrastructure of a DAS as an overlay network (i.e., denoted as a virtual network). On top of the time-triggered physical network, multiple virtual networks with different protocols can be established, such as a virtual network executing

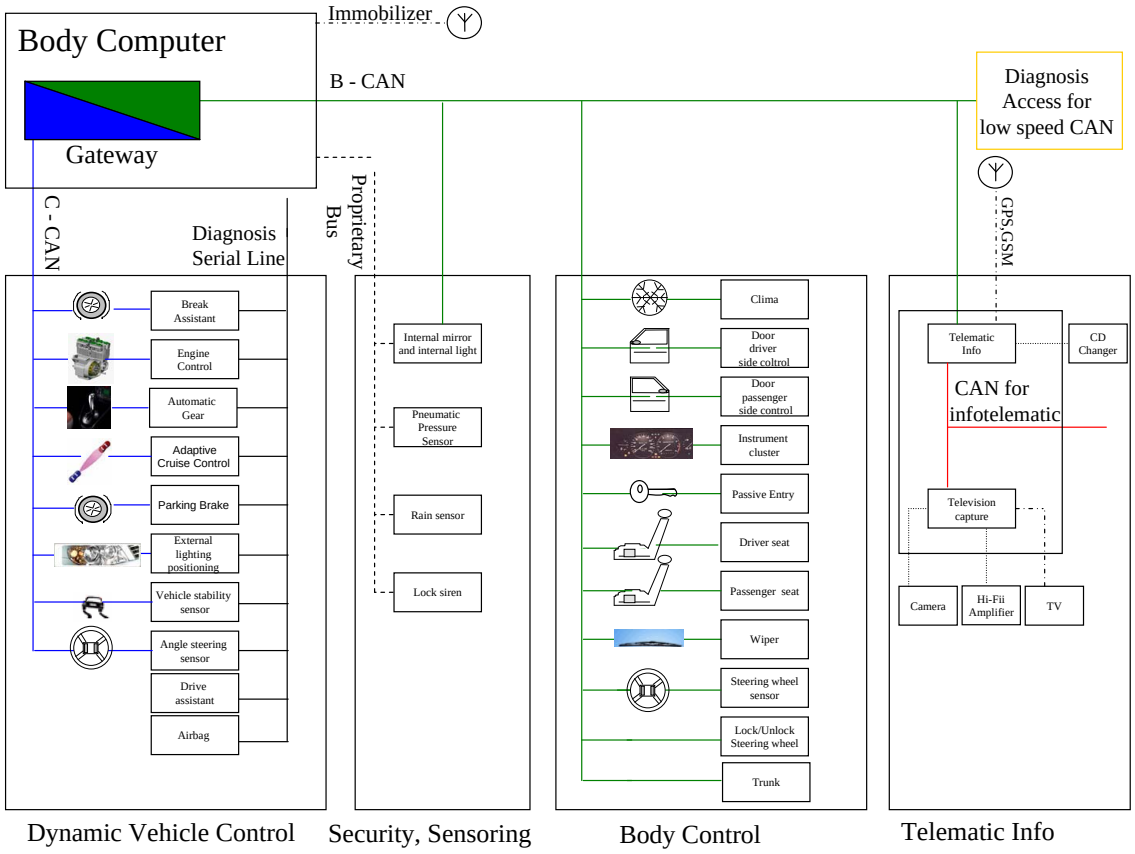

Figure 2. The Electronic Infrastructure of a Fiat Car

the CAN protocol for a non safety-critical DAS (e.g., comfort) or a time-triggered virtual network for a safety-critical DAS (e.g., X-by-wire).

4. Today's Automotive Architectures

To give an impression of the complexity and the amount of electronics in today's luxury cars take for example the electronic infrastructure of a Fiat car depicted in Figure 2. The distributed ECUs of each federated cluster of the car are interconnected via communication networks with different protocols (e.g., Controller Area Network (CAN) [Bos91], Local Interconnect Network (LIN) [LIN03]), physical layers, bandwidths (10 kbps–500 kbps), and dependability requirements.

The *Body Control* cluster as well as the *Telematic Info* cluster, are typically implemented via a low speed CAN bus (125 kbps), while the *Dynamic Vehicle Control* cluster is implemented via a high speed CAN bus (500 kbps). The *Infotelematic* system also uses a high speed CAN to exchange camera and video information. These multiple federated clusters are interconnected with a central gateway inside the *Body Computer*, allowing controlled data exchange between the *Dynamic Vehicle Control* cluster and the *Body Control* cluster, and access to the On-Board Diagnosis (OBD) system of each ECU. For on-board diagnosis either a dedicated serial line interconnects the ECUs or the diagnostic protocol is executed via the low speed CAN network.

Each of these clusters consists of ECUs that are typically dedicated to a single job. In combination with the need to adapt products to emerging trends and customer requests, manufactures are forced to steadily increase the number of deployed ECUs in order to improve the functionality of the car. For example, Fiat cars contain up to 40 ECUs. However, this trend of increasing the number of ECUs is coming to its limits, because of complexity, wiring, space and weight restrictions. For example, electrical connections are considered to be one of the most important failure causes in automotive systems. Field data from automotive environments has shown that more than 30% of electrical failures are attributed to connector problems [SM99]. With an average cost of 30-50 Euros per ECU this high number of ECUs also bears significant potential for cost reduction.

Development Process

The typical development process in the automotive industry follows the V-cycle model [Rei06, GFL+02]. The OEM acts as the overall system architect by defining the electronic architecture of the car. This includes the definition of the physical structure (e.g., ECUs, wiring, connectors, packaging) and the provided services (e.g., data dictionary for exchanged messages).

Based on this specification, the sourced suppliers (or in-house development teams) implement the respective ECUs. In general, the development of each ECU involves tight interactions between the suppliers and the OEM. Right from the beginning of the development process, the OEM monitors the progress of the suppliers in order to ensure correctness and timely delivery of the ECUs.

For the validation, both component and system integration tests are performed at the OEM. During the component test, each ECU is validated in isolation. For example, a CAN-based ECU is provided with messages from a rest bus simulator. For simulating the I/O, typically Hardware-in-the-Loop (HiL) is used. During system integration test, the interplay of multiple ECUs is tested (e.g., body domain HIL simulator). One focus of the system integration test is the evaluation of the emerging services of the ECUs, i.e., those services that are provided by more than one ECU (e.g., advanced parking assistant). Another focus is the validation whether the prior services of the ECUs are still correct after integration. For example, the temporal specification must not be violated (e.g., timeouts).

However, today during system integration significant efforts are caused by unanticipated interactions between subsystems provided by different vendors. The sharing of communication resources in today's cars across different subsystems (e.g., systems based on the CAN protocol) makes it hard to fully test the functionality of a subsystem in isolation as it will be integrated in the car. As a consequence, there is the need for a comprehensive integration test by the car manufacturer to determine possible mutual interference of subsystems. In contrast, a system architecture with rigorous operational interface specification [KS03] and error containment can avoid the introduction of mutual interference during system integration. Such a temporally composable architecture [KO02] exhibits the benefit of dramatically decreasing integration costs, because the validity of test certificates from suppliers is not invalidated during system integration.

Complexity Control

Each subsystem (e.g., engine control, brake assistant) possesses a functional complexity that is inherent to the application. The functional complexity of a subsystem when implemented on a target system is dramatically increased in case the architecture does not prevent unintended architecture-induced side effects at the communication system. Since federated systems employ a dedicated computer system for each subsystem, the complexity of the system is lower compared to the integrated systems approach. The absence of interactions and dependencies between sub-

systems reduces the cognitive complexity to a manageable level. In today's cars we do not find a totally federated architecture nor an integrated one. In fact, the economic pressure in the automotive industry requires system designers to utilize the available communication resources for more than one subsystem without protecting the resources from mutual interference. For a deeper understanding consider an exemplary scenario with two subsystems. If the two subsystems share a common CAN bus [Bos91], then both subsystems must be analyzed and understood in order to reason about the correct behavior of any of the two subsystems. Since the message transmissions of one subsystem can delay message transmission of the other DAS, arguments concerning the correct temporal behavior must be based on an analysis of both subsystems. In a totally federated system, on the other hand, unintended side effects are ruled out, because the two subsystems are assigned to separate computer system.

5. An Architecture for Future Car Generations

This section describes the proposed integrated architecture for future car generations. We start by discussing a hybrid top-down/bottom-up design strategy, improving the currently prevalent ECU-centric development process. The decomposing during the process results in a set of DASs. Thereafter, we elaborate on the DASs of a hypothetical future car and its constituting gateways. Finally, we show the physical structure of the integrated system.

Design Flow

The design flow of automotive distributed systems can be decomposed into three phases, the requirement analysis, the subsystem design, and the system integration phase [GFL+02] (see also Figure 3). As described in [RH04, SW04] an ECU-centric design process prevails in the automotive industry. Such a bottom up process, however, bears significant drawbacks such as resource duplications, local instead of global quality-of-service optimization, and exponential growth in terms of system integration costs. Furthermore, the number of the deployed ECUs steadily increases to satisfy recent market trends and the customer's demand for new functionality.

The DECOS architecture, by contrast, also supports a top-down design approach. During the requirement analysis the system integrator captures the requirements of the overall system (i.e., the car electronics) and decomposes the system into nearly-independent subsystems (i.e., DASs). The requirement analysis provides the foundation for all later design stages. Here, the overall functionality of the system is specified and subsystems are identified to enable an independent development of DASs. As depicted in Figure 3 the result of this design phase is a set of DASs that comprise the electronic infrastructure of the car.

The structuring of the overall application functionality into DASs is guided by the following principles:

1 **Functional Coherence.** A DAS should provide a meaningful application service (e.g., brake-by-wire service of a car) to its users at the controlled object interface. By associating with a DAS an application service that is relevant in the actual application context, the mental effort in understanding the various application services is reduced. An application service can be analyzed by solely considering the jobs of the DAS, the interactions to the controlled object and the gateways to other DASs (inter-DAS interfaces). In particular, it is not necessary to possess knowledge about the internal behavior of DASs, other than the one providing the application service that is of interest.

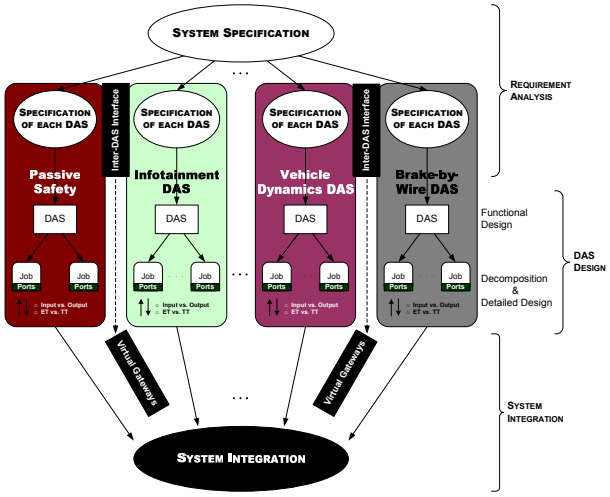

Figure 3. Design Flow

2 **Common Criticality.** In general, the realization of safety-critical services is fundamentally different from the design of non safety-critical services. While the first incorporate fault-tolerance functionality and focus on maximum simplicity to facilitate validation and certification, the latter are usually characterized by a larger amount of functionality and the requirement of flexibility and resource efficiency. The integrated architecture takes this difference into account by distinguishing between safety-critical and non safety-critical DASs along with dedicated architectural services.

3 **Infrastructure Requirements.** A DAS possesses common requirements for the underlying infrastructure. A single virtual network is employed for exchanging message within the DAS. Consequently, common requirements (e.g., with respect to dependability, bandwidth and latency requirements, flexibility) are a prerequisite for deciding on a particular virtual network protocol (e.g., time-triggered or event-triggered) and a corresponding configuration (e.g., bandwidth).

Whenever significant differences in the above aspects are present, such as missing functional coherence or differences with respect to the infrastructure requirements, a DAS is split into smaller DASs for resolving these mismatches.

This *divide and conquer* principle can only be realized if the dependencies between DASs are made explicit in order to avoid hidden interactions (e.g., via the controlled object) that may prevent a seamless system integration. These DASs are then assigned and independently developed by different vendors. In general, each vendor may also depend on subcontractors to deliver the subsystem.

In order to ensure correct system integration the specification of inter-DAS relationships is of high importance. Inter-DAS interfaces as indicated in Figure 3 are used to specify common information within DASs (e.g., sensor information), possible interrelationships via the controlled object, and meta-functional aspects. This way, resources can be shared among DASs, thus avoiding resource duplication by eliminating sensors or using redundant sensory information to improve dependability.

The DAS design is typically performed by different vendors with expert knowledge in particular application domains (e.g., infotainment, braking systems). Independent development of a DAS allows to adopt the benefits of the federated systems design approach to be incorporated into the integrated systems design approach.

Finally, the system integrator needs to unify the separately developed subsystems into the overall system. System integration unites the separately developed subsystems into the overall system. An integrated system approach must provide solutions that reduce integration time and efforts (and consequently reduce integration costs). Smooth system integration is only possible, if the inter-DAS interfaces have been precisely specified and all vendors have performed implementations adhering to these interface specifications. During system integrations three main tasks need to be performed by the system integrator: the physical allocation of the jobs (of all DASs) to partitions taking dependability and resource constraints into account, the configuration of the virtual communication networks, and the realization of the virtual and physical gateways in order to provide emerging services.

Integrated System Structure of Car

In this subsection, we map the introduced electronic infrastructure of today's cars onto the DECOS integrated architecture. In addition, we replace state-of-the art powertrain domain functionality with by-wire subsystems to emphasize the suitability of the proposed DECOS architecture for mixed criticality applications (i.e., safety-critical and non safety-critical applications). We split up the existing domains (e.g., powertrain, body) into smaller DASs. Smaller DASs are a key element to achieve the DECOS goals with respect to complexity management, independent development and error containment.

As described in Section 3, a DAS represents a nearly-independent subsystem [Sim96, chap. 8], because it can be understood independently from the detailed structure of other DASs, i.e., only based on the specification of the jobs of the DAS and the gateways to other DASs. The controlled export of information through gateways enables the designer to abstract from the jobs in other DASs, considering only the interface specification of the gateways [OP05b].

The DECOS architecture encapsulates DASs both at the level of the communication activities (through encapsulated virtual network services [OP05]) and at the level of the computational activities (partitions in application computers [OPHS06]). Therefore, a design fault in a DAS (e.g., a job with a babbling idiot failure [Kop97]) cannot affect the communication resources (e.g., bandwidth, guarantee of latencies) and computational resources (e.g., CPU time) available to other DASs. Naturally, the finer the subdivision into DASs, the more effective the encapsulation through the architecture becomes.

In a federated architecture, which assigns each DAS to its own dedicated computer system, a strategy with a large number of small DASs would not be feasible due to the cost resulting from increasing resource duplication (via separate networks and ECUs). However, in our proposed integrated architecture, the resulting larger number of DASs does not induce a larger number of physical networks and ECUs. Each DAS is provided as a virtual network on top of the physical time-triggered network of the DECOS architecture. Similarly, the virtual gateways (see Section 5) for coupling DASs do not induce any additional ECUs and connectors.

Based on this line of reasoning, we introduce a finer granularity of DASs compared to the domains of today's automotive architectures (see Figure 4):

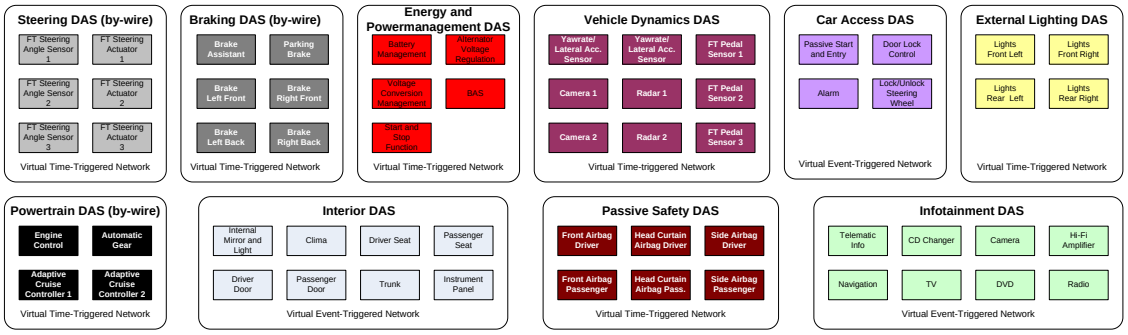

Figure 4. The Distributed Application Subsystems of the Integrated Automotive System

Steering DAS. With steer-by-wire the transmission of the wheel rotation to a steering movement of the front wheel is performed with the help of electronically controlled actuators at the front axle [Hei03]. The main advantages in comparison with conventional steering systems are improvements with respect to crashworthiness, weight, and interior design.

Powertrain DAS. The functionality of this DAS includes engine control, automatic gear control, and adaptive cruise control.

Braking DAS. The braking DAS comprises the brake-by-wire functionality [HB98]. Brake-by-wire systems remedy deficiencies of conventional hydraulic braking systems, such as aging of braking fluids, difficulties in routing of pipes, and the inconvenient feedback during ABS braking. Brake-by-wire systems incorporate brake power assist, vehicle stability enhancement control, parking brake control, and tunable pedal feel.

Vehicle Dynamics DAS. In the vehicle dynamics DAS all sensory information that is relevant for controlling the dynamics of the vehicle is captured. By exporting these real-time images to the other DASs of the system the problem of resource duplication can be significantly reduced. In addition, sensor-fusion algorithms [Elm02] can combine the measurements of different sensors to obtain more accurate real-time images.

Energy DAS. The main purpose of this DAS is the optimization of the power distribution (electrical energy and power management techniques) for conserving the power available in the vehicle.

Passive Safety DAS. The passive safety DAS intends to keep the passengers in the car and effectively decelerates the occupants in order to minimize harm in case of a crash.

Car Access DAS. The functionality of this DAS includes a passive keyless access and start system with theft alert. The driver carries an identification device to control the door lock and can start the vehicle by pressing a button.

Interior DAS. This DAS comprises the body electronics of the passenger compartment and accesses fieldbus network, such as those embedded in the doors and seats of the car. The functionality of this DAS includes the control of the doors (e.g., mirrors, window lifters), the seats

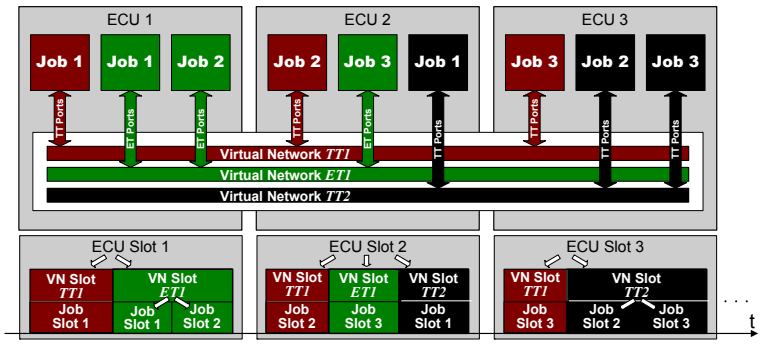

Figure 5. Hierarchic Subdivision of TDMA Slots

(e.g., seat adjustment, the position memory), the instrument panel, the climate control, and the lighting of the passenger compartment.

Infotainment DAS. Car drivers are no longer satisfied with cars being simple means of transportation. Today's luxury cars include GPS navigation systems, DVD players and high-end audio systems. In addition, voice control and hands-free speaker phones relieve the driver from concentrating on multimedia devices instead of traffic.

External Lighting DAS. This DAS encompasses jobs for controlling the rear lights (e.g., rear fog lamps, braking lights) and front lights (e.g., indicators, dipped beam, main beam, front fog lamps). In addition, the jobs of the external lighting DAS control the position of the adaptive forward lighting.

Virtual Networks

Each DAS is provided with a dedicated communication infrastructure that is realized as a *virtual network*. A virtual network is established as an overlay network on top of a time-triggered physical network [OP05].

The establishment of a virtual network occurs through a hierarchic temporal subdivision of the communication resources provided by the time-triggered communication protocol. The media access control strategy of the time-triggered communication protocol is TDMA. TDMA statically divides the channel capacity into a number of slots and controls access to the network solely by the progression of time. An ECU broadcasts messages during its ECU slot and receives messages during the slots of the other ECUs. A virtual network further subdivides the ECU slots by assigning to each job that is located on an ECU a respective job slot (see Figure 5). For transmitting the messages produced by a job, the virtual network service uses this job slot in the TDMA scheme.

Virtual networks follow rigorously a TDMA schedule, in which each job sends during an interval of time with a priori known start and end instants w.r.t. to a global time base. This static communication schedule has the benefit of enabling error containment with respect to the communication resources. The architecture can ensure that each job writes only into its own communication slots, e.g., by using guardian functionality with a fault-tolerant time-triggered network in conjunction with middleware as discussed in [OP05]. Thereby, a faulty job is prevented from affecting the data integrity and temporal properties (e.g., latency) of the messages sent by other jobs.

Using job slots in a TDMA scheme, virtual networks with higher protocols have been established on top of a time-triggered physical network. Examples are time-triggered virtual networks [OP05], virtual CAN networks [OP05a], a transport protocol for a hard-real time CORBA broker [SLO03], and virtual networks with TCP/IP [Nex03].

The selected protocol depends on the respective criticality and regularity of the communication activities. Time-triggered virtual networks are well-suited to handle the communication exchanges of all safety-critical and safety-relevant DASs. It has become widely accepted that safety-critical automotive applications employ the time-triggered control paradigm [Rus01], because this control paradigm permits to guarantee a deterministic behavior of all safety-related message transmissions even at peak-load. In addition to hard real-time performance time-triggered control also supports temporal composability and facilitates the realization of fault-tolerance mechanisms. For this reason the communication infrastructure for the steer-by-wire, the brake-by-wire, the powertrain, the vehicle dynamics, the passive safety and the external lighting DAS are *time-triggered virtual networks*.

Event-triggered virtual networks like CAN, on the other hand, are the communication infrastructure of choice for those DASs having less stringent dependability requirements. Here, the flexibility and the efficient use of resources is more important than the determinism provided by the time-triggered control paradigm. For this reason, the jobs of the interior, power management, car access, and infotainment DAS are interconnected by respective *event-triggered virtual networks* in the presented architecture.

Gateways

By splitting the overall functionality of the car into multiple DASs the need for a coupling of individual DASs emerges. The presented architecture supports gateways as a generic architectural services for the interconnection of DASs. Gateways have significant advantages with respect to the elimination of resource duplication and the tactic coordination of application subsystems. In a large automotive system, different application subsystems typically depend on the same or similar sensory inputs and computations. Gateways allow to exploit system-wide redundancy of sensor information in order to increase reliability or reduce resource duplication. In addition, gateways permit the coordination of DASs in order to improve quality of control.

In the DECOS integrated architecture, we sharply distinguish between architecture level and application level. Based on this differentiation, we can identify two choices for the construction of a gateway. A hidden gateway performs the interconnection of virtual networks at the architecture level. Generic architectural services – although parameterized by the application requirements – are transparent to the jobs at the application level. A visible gateway, on the other hand, performs the interconnection at the application level.

A *virtual gateway* [OP05] interconnects two virtual networks of two respective DASs by forwarding information contained in the messages received at the input ports of one virtual network onto the output ports towards the other virtual network.

In general, the semantic and operational properties of the input ports at one virtual network can be different to the semantic and operational properties of the output ports at the other virtual network. The resulting property mismatch [C. 02] is resolved by the gateway by performing transformations on the information passing through the gateway. For syntactic transformations, the gateway requires a description of the syntactic format (i.e., the data types) of the messages passing through the gateway and rules for transforming the different syntactic transformations into each other. If the DASs interconnected by the gateway exhibit different operational speci-

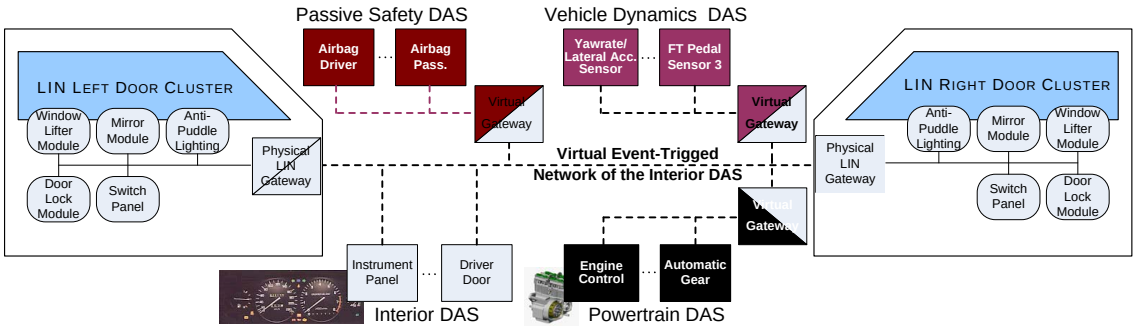

Figure 6. Physical and Virtual Gateways of the Interior DAS

fication styles [KS03], the gateway requires additional buffering functionality for the exchange of information between virtual networks with varying rigidity of temporal specifications. For example, such a scenario occurs, if one DAS operates time-triggered and the second DAS operates event-triggered.

In addition, virtual gateways ensure encapsulation by using a filtering specification in the value and temporal domain [OP05b] in order to restrict the redirection of messages through the gateway. In general, only a fraction of the information exchanged at one virtual network will be required by jobs connected to the virtual network at the other side of the gateway. By restricting the redirection through the gateway to the information actually required by the jobs of the other DAS, the gateway not only improves resource efficiency by saving bandwidth of unnecessary messages, but also facilitates complexity control.

In addition to the interconnection of virtual networks, the presented integrated architecture offers architectural gateway services for interacting with the environment via *physical gateways*. In general, the interaction of the computer system with the controlled object and the human operator can occur either via a direct connection to sensors and actuators or via a fieldbus network. The latter approach simplifies the installation – both from a logical and a physical point of view – at the expense of increased latency of sensory information and actuator control values.

Since the prevalent low-cost fieldbus protocol in the automotive domain is LIN [LIN03], the proposed architecture supports physical LIN gateways, each acting as a master for the slaves of the physical LIN bus. Figure 6, which exemplifies the role of hidden gateways in the logical structure of the future automotive architecture, depicts two of these LIN gateways for the interconnection of the interior DAS with the LIN fieldbusses located at the front doors. The driver door job and passenger door job exchange information with the actuators and sensors in the doors in order to control door locks, window lifters, mirrors, and anti-puddle lighting.

In addition, Figure 6 also contains virtual gateways for the interconnection of virtual networks. The interior DAS constructs real-time images capturing the state of the passenger compartment of the vehicle (e.g., status of the doors, seats, lighting, climate control) that are also important to other DASs. For example, the driver's weight as measured at a seat is an important parameter for the passive safety DAS in order adapt air bags to different passengers (e.g., children). The current temperature inside and outside the car, which is captured by the climate control subsystem, is another example for a real-time entity that is significant beyond the interior DAS. Temperature measurements are an essential input for physical models of sensors in other DASs (e.g., powertrain DAS) and permit to improve the precision and plausibility of sensory information.

Adversely, other DASs need to be able to control body electronics in the interior DAS. In hazardous situations, e.g., after the detection of a potential crash as indicated by yaw rate and lateral acceleration sensors (e.g., during skidding and emergency braking), the vehicle dynamics DAS causes the tensioning of seat-belts and realigning of seats to a safer positions.

In the following, we will discuss one of the virtual gateways in more detail, namely the gateway for the forwarding of engine information from the powertrain DAS to the interior DAS. Thereby, engine status information (such as engine speed, vehicle speed, fuel indication) is displayed on the instrument panel.

The *engine control job* is part of the powertrain domain. It accesses numerous actuators and sensors to the motor (e.g., cam shaft and crank shaft position). The engine control job receives messages from the vehicle dynamics DAS (e.g., message with engine torque request from ESP) and from the powertrain DAS (e.g., transmission control job). The output of the engine control job are messages with status information of the engine, such as:

$msg\ E1 : <$ *engine speed* (16bit, unit: RPM), $throttle\,position$ (8bit, unit: percent) $>$
$msg\ E2 : <$ $fuel\ consumption$ (16bit, unit: liters/hour), $engine\ oil\ temp.$ (8bit, unit: AřC) $>$

The *instrument panel job* receives messages from the virtual CAN network of the interior DAS and displays the status information contained in these messages on the instrument panel. Among the received messages are:

$msg\ IP1$: $<$ $engine\ speed$ (16bit, unit: RPM), $engine\ oil\ temperature$ (8bit, unit: °C) $>$
$msg\ IP2$: $<$ $vehicle\ speed$ (16bit, unit: km/hour), $odometer$ (16bit, unit: km) $>$
$msg\ IP3$: $<$ $door\ status$ (1bit per door, unit: open/closed) $>$

The virtual gateway between the powertrain DAS and the interior DAS needs to redirect information from the engine control job to the instrument panel job to be shown to the driver. Also, the gateway needs to resolve a property mismatch between the different communication paradigms of the DASs. In detail, the actions of the gateway are as follows:

Selective redirection. The virtual gateway redirects only those signals from the messages with the engine status that are actually required in the interior DAS. For example, the oil temperature and the engine speed in messages $E1$ and $E2$ are redirected via message *IP*1, while the throttle position in message $E1$ is discarded by the gateway.

Conversion of communication paradigms. The powertrain DAS employs a time-triggered virtual network, i.e., all messages are exchanged periodically at a priori specified points in time. The interior DAS, on the other hand, possesses an event-triggered virtual network with on-demand message transmissions (i.e., triggered by significant events). The gateway performs time-triggered receptions of engine status messages. The transmission of messages, however, occurs event-triggered. A message is transmitted on the interior DAS only in case the value of the real-time entity has changed. No message is sent in case the real-time entities (e.g., oil temperature) remain unchanged.

Traffic shaping. The message transmissions of the engine status information in the powertrain DAS occur at a higher frequency than can be used for updating the instrument panel in the interior DAS. For this reason, the virtual gateway enforces a minimum message interarrival time for the message transmissions on the event-triggered virtual network of the interior DAS.

More details on the formal specification of these actions using timed gateway automata and their implementation can be found in [OP05b].

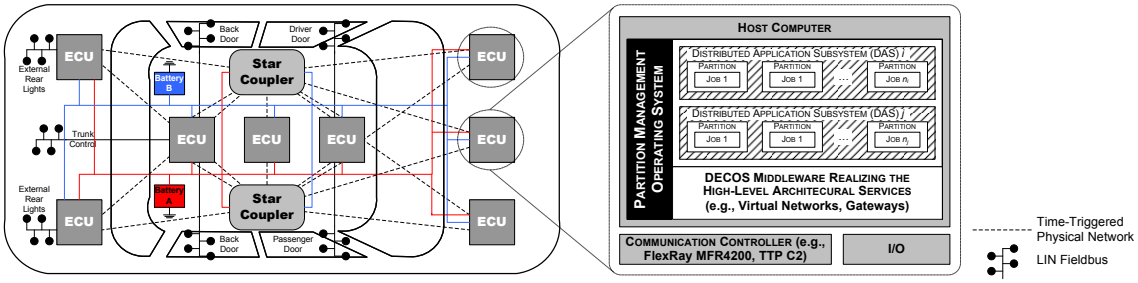

Figure 7. Physical System Structure

Physical System Structure

The mapping from the logical to the physical system structure needs to assign jobs to ECUs and virtual networks to the time-triggered physical network. In order to tolerate arbitrary single component failures it is mandatory to devise a mapping of jobs to ECUs in a way, that redundant jobs are not hosted on the same ECU. Similarly to the sharing of ECUs among jobs, the time-triggered physical network is the basis for multiple virtual networks. In order to achieve the dependability requirements of safety-critical DASs, the time-triggered physical network relies on two central guardians [BKS03] for achieving fault isolation even in case of arbitrary ECU failure modes. Fault injection experiments have shown that for ultra-dependable applications restrictions concerning the failure modes of ECUs are unjustified [ASBT03].

In order to support the collocation of multiple jobs and the establishment of virtual networks, an ECU incorporates the following structural elements as depicted in Figure 7:

Partition management operation system. The host computer of an ECU runs multiple jobs of different DASs. For this purpose, the host computer uses a *partition management operating system*, which establishes for each job a corresponding partition with guaranteed computational resources (CPU time, memory). The partition management operating system implements mechanisms for spatial and temporal partitioning in order to protect the computational resources of the individual partitions. The scheduling of partitions needs to ensure that a timing failure of a job, such as a worst-case execution time violation, does not affect the CPU time available to other partitions. The spatial partitioning mechanisms of the partition management operating system include memory protection between partitions (e.g., hardware-enforced with a Memory Management Unit (MMU)). Thereby, each partition emulates a "virtual ECU" that is dedicated to a single job only. An example for a partition management operating system is a time-triggered operating system based on Linux Real-Time Application Interface (RTAI) [HPOS05]. Further examples of suitable operating systems, which fit into the proposed architecture, are LynxOS-178 [Lyn06] and VxWorks [KWR04]. These two operating systems are certified operating systems that support time partitioning through a fixed-cyclic time-slice scheduler.

DECOS middleware. The host computer contains middleware for realizing the high-level architectural services. The purposes of the middleware include the management of the communication resources by means of virtual networks and gateways as previously described. The middleware provides a technology invariant interface to the jobs that abstracts from any hardware-specific implementation details.

Communication controller. The *communication controller* executes a time-triggered communication protocol (e.g., FlexRay [Fle05], Time-Triggered Protocol (TTP) [TTT02a]) to offer the deterministic and timely transport of messages, fault-tolerant clock synchronization, and strong fault isolation as described in Section 1.

I/O. The jobs hosted on an ECU exploit the input/output subsystem for interacting with the controlled object and the human operator. This interaction occurs either via a direct connection to sensors and actuators or via a fieldbus (e.g., LIN [LIN03]). The latter approach, which simplifies the installation from a logical and physical point of view at the expense of increased latency, is depicted in Figure 7. Physical LIN gateways connect physical LIN clusters to the integrated system, e.g., the LIN fieldbus within the doors of the cars are connected to the ECUs located in the center of the car.

6. Benefits of the Proposed Architecture

This section summarizes the main benefits of the proposed integrated automotive architecture and highlights the major design decisions. We will show that the proposed architecture is in line with prevalent architectural trends, such as reuse of functionality across different car segments and introduction of safety-critical applications.

Economic Benefits

Present day automotive systems follow a federated philosophy with different types of automotive networks [LH02]. The multitude of communication protocols is a result of the different requirements with respect to functionality, dependability, and performance of the automotive DASs, such as the infotainment DAS, the comfort DAS, or the powertrain DAS.

In contrast to these federated architectures, the integrated architecture for future automotive systems presented in this paper allows to share communication and computational resources among different DASs in order to evolve beyond a "1 Function – 1 ECU" strategy and achieve a significant reduction in the overall number of ECUs. With an average cost of 40 Euro per ECU in today's cars and 0.2 Euro per wire, the total hardware cost of a federated automotive system with 40 ECUs and 800 wires is about 1800 Euro. If we assume that the number of ECUs will be reduced to 30 ECUs and the number of wires to 500 in the integrated system, with an increased average cost per ECU of 50 Euro, then total hardware cost is reduced by 200 Euro per car.

Complexity

A minimal complexity, i.e., a minimal mental effort to analyze and understand a system, is one of the most important design drivers of the presented integrated architecture. Today, the unmanaged complexity of systems is the major cause of design faults. *Complexity increases the likelihood of serious, yet latent, design flaws* [JB92, p. 39]. In [Lev86, p. 131] it is stated that *complexity of software and hardware causes a non linear increase in human-error induced design faults*. High system complexity also prolongs development, which is detrimental to a company's economic success, because today's business realities demand a short time-to-market. In addition, complexity complicates validation and certification as state-of-the-art formal verification tools are limited in the size of a design that they can handle.

In order to manage complexity, the presented integrated architecture enables designers to proceed in such a way as if they were realizing DASs in a federated system. A DAS along with

the corresponding communication resources (virtual networks) and computational resources (partitions in ECUs) is encapsulated and interactions between DASs are limited to the exchange of messages via precisely specified hidden gateways. Similar to a federated architecture, the integrated architecture decouples each DAS from other DASs. Each DAS possesses a dedicated encapsulated virtual network. A virtual network is private for a DAS, i.e., other DASs cannot perceive or affect the exchanged messages other than those being explicitly exported via a gateway. By exercising this strict control over the interactions between DASs, only the behavior of the DAS's virtual network and the behavior of gateways is of relevance when reasoning about a DAS. The message transmissions on other virtual networks can be abstracted from. Similarly, each job executes in a corresponding partition, which forms an encapsulated execution environment with guaranteed communication and computational resources. The activities of jobs executing on the same ECU cannot affect the resources that are available to other jobs in the ECU.

By not only supporting error containment between DASs, but error containment between jobs within a DAS, the integrated architecture exceeds the error containment capabilities of most federated systems. Furthermore, the integrated architecture offers generic architectural services as a base line for application development, thus decreasing the functionality that must be realized at the application level. Such a slimmer application is easier to comprehend and leaves less room for software design faults. Only the interface between the architecture and the application is visible to the application developer, while the realization of the architectural services remains hidden.

Dependability and Mixed Criticality Integration

Carmakers are on the verge of deploying by-wire technology to improve the functionality of the vehicle that goes beyond traditional hydraulic and mechanic systems. This trend requires the deployed E/E architectures to meet the requirement for ultra-high dependability [SWH95] (a maximum failure rate of 10^{-9} critical failures per hour is demanded). In the proposed architecture, the support for applications up to the highest criticality classes (10^{-9} failures per hour) is based on the following four properties:

1. Replica determinism to support TMR. Since ECU failure rates are in the order of 10^{-5} to 10^{-6}, ultra-dependable applications require the system as a whole to be more reliable than any one of its ECUs. This can only be achieved by utilizing fault-tolerance strategies that enable the continued operation of the system in the presence of ECU failures [BJV91]. The consequence of a hardware fault will in general be a complete failure of an ECU with all the jobs located on the ECU [OP06]. For this reason, it is necessary to use replication (e.g., TMR) where the replicated jobs are assigned to different ECUs[1].

In order to support the implementation of TMR, the proposed architecture provides replica determinism [Pol94]. Replica determinism ensures that a majority decision with exact voting can be performed upon the outputs of three replicated jobs on different ECUs (without costly agreement protocols). Among the key mechanisms of the architecture for the establishment of replica determinism are the support for time-triggered control and static resource allocations. For example, the time-triggered virtual networks preclude race conditions in the access to the communication resources by design.

[1]This strategy does not preclude, however, the ability to integrated multiple jobs on a single ECU as long as the jobs are not part of the same TMR configuration.

2. Different classes of high-level architectural services. We distinguish safety-critical and non safety-critical DASs. The non safety-critical DASs possess only benign failure modes [Lap92], while safety-critical DASs exhibit critical failure modes that can lead to catastrophes and an endangerment of human life. Along with the distinction of safety-critical and non safety-critical DASs, the proposed architecture performs a likewise differentiation of the high-level architectural services. For the safety-critical DASs, the design of the high-level architectural services must be as simple as possible (e.g., only time-triggered virtual networks) in order to facilitate formal analysis and certification. For the non safety-critical DASs, on the other hand, high-level architectural services with additional functionality can be provided to ease the application development. For example, event-triggered virtual networks improve flexibility and resource efficiency through support for on-demand message exchanges. Furthermore, the integration of legacy applications without redevelopment efforts can require functionality for emulating the corresponding legacy platforms (e.g., emulation of CAN). This increased complexity is tolerable, when certification to the highest criticality levels (e.g., SIL4 in EN ISO/IEC 61508 [IEC99]) is not required.

3. Error containment between ECUs enforced by core architecture. The time-triggered core architecture uses a TDMA communication scheme, in which each communication slot possesses a unique sender ECU. The proposed architecture exploits bus guardians, which are available for several time-triggered networks (e.g., FlexRay, TTA), to protect these communication slots. The error containment between ECUs is significant to prevent common mode failures of the replicas in a TMR configuration.

4. Error containment within an ECU enforced by high-level architectural services. On top of the error containment mechanisms of the core architecture, the proposed architecture prevents an error of a faulty job to propagate to other jobs on the same ECU. The error containment within an ECU occurs in two ways:

- **Partition management operating system.** The partition management operating system (cf. Section 6) protects the computational resources by means of temporal and spatial partitioning. Temporal partitioning uses a scheduler that ensures each job gets its guaranteed CPU time, e.g., using a fixed-cyclic time-slice scheduler.

- **Virtual networks.** The virtual networks (cf. Section 4) protect the communication resources based on the hierarchic subdivision of the communication slots in the TDMA scheme. Each communication slot contains messages from a single job only. Thus, guardians can prevent a job from sending in a communication slot that belongs to another job.

This extended error containment is of particular importance for mixed criticality systems, in which jobs of safety-critical and non safety-critical DASs coexist on the same ECU. In general, such DASs will exhibit significant differences concerning the residue of design faults after deployment due to different development processes driven by economic constraints. In safety-critical application subsystems, the absence of design faults can no longer be shown by testing alone. The achievement of the reliability includes a rigorous development process, formal verification, and involvement of a certification agency. For non safety-critical application subsystems, certainty about the complete absence of design faults is usually economically infeasible. The level of rigidity in the development process of safety-critical applications would be too expensive.

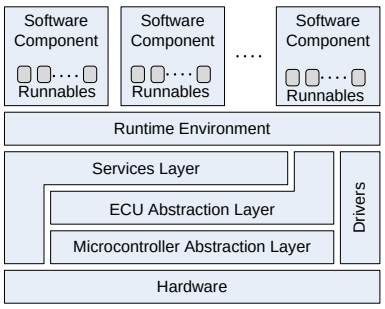

Figure 8. ECU in AUTOSAR

Flexibility

A newly developed E/E architectures is expected to be deployable on different car segments and models. Consequently, the reuse of applications in a modular way is of critical importance. This issue also has an impact on "Tier 1 system suppliers" that need to adapt their subsystem design according to this trend. The usage of validated and possibly certified application software in combination with standard physical ECUs on different models and segments of vehicles will lead to advantages in terms of increased volume, multi-supplier management, and multi-platform management. In addition, not only software modules but complete electronic subsystems can be made available for different vehicle platforms. This strategy also eases the design choices for the interior of the car, due to the freedom of decoupling application software from a particular ECU. The integrated automotive architecture provides a high degree of freedom in the allocation of jobs to ECUs and supports the migration of jobs between ECUs (constrained by dependability requirements).

7. Related Work

This section gives an overview of related work on integrated architectures (AUTOSAR and IMA). We focus on the sharing of the computational resources via ECUs with support for multiple software components. In addition, the sharing of communication resources is addressed. The section also discusses the relationship to the integrated architecture proposed in this paper.

AUTOSAR

The AUTOSAR [AUT06b] is a system architecture and development methodology for automotive electronic systems. Among the primary goals of AUTOSAR are the standardization of the basic software of an ECU (called standard core) and the ability to integrate software components from multiple suppliers. Another focus of AUTOSAR is the establishment of portability and location transparency for software components in order to accommodate future changes to the automotive electronic systems and facilitate reuse across product lines [Rol06].

Model of an ECU. An ECU in AUTOSAR supports the integration of software components from multiple suppliers (see Figure 8). A *software component* is a piece of software that can be mapped to an ECU. Internally, a software component contains threads of control called *runnables*. AUTOSAR distinguishes between event-triggered and communication-triggered runnables. The execution of event-triggered runnables is triggered by the occurrence of a significant event

(e.g., timeout). Communication-triggered runnables are activated by the arrival of messages from other software components.

The *runtime environment* decouples the software components from each other and from the hardware. It provides mechanisms for communication between the software components in the same ECU and on different ECUs.

Below the runtime environment in Figure 8, the AUTOSAR ECU contains the basic software encompassing layers with standardized functionality. These layers consist of software components that do not fulfill application functionality themselves, but interact with the application software components using the runtime environment.

- **Services layer.** This layer provides operating system services (e.g., priority-based scheduling of runnables, memory management), and communication services (e.g., communication stack for CAN or LIN).
- **ECU abstraction layer.** For accessing peripherals and devices, this layer establishes an API that abstracts from the underlying hardware. For example, the I/O hardware and the communication hardware (e.g., CAN controller) deployed on the ECU are hidden from the upper layers.
- **Microcontroller abstraction layer.** This layer provides access to internal and external peripherals of the ECU.
- **Drivers.** The drivers are application-dependent and interact with sensors and actuators. In order to satisfy tight timing requirements (e.g., injection control), the drivers can use interrupts and directly access the microcontroller and the peripherals.

Communication System. For interactions between software components, AUTOSAR provides a so-called *virtual function bus* with two communication patterns:

- The *sender-receiver communication* supports the transmission of signals with atomic data elements from a sending software component to one or more receiving software components. A sender-receiver communication is unidirectional, i.e., a reply involves another sender-receiver communication.
- The *client-server communication* enables a software component (acting as the client) to invoke a server function at another software component. The function invocation by the client can be synchronous (i.e., blocking until the result arrives from the server) or asynchronous (i.e., non-blocking).

The implementation of the virtual function bus occurs using the runtime environment, which acts as a communication switch and ensures location transparency. Communication between software components on same ECU is realized by passing arguments directly to the respective runnables. Communication between software components on different ECUs exploits the services layer of the ECU in order to establish a mapping to the communication network (e.g., CAN).

Relationship to the DECOS Architecture. Like the architecture proposed in this paper, AUTOSAR aims at evolving from today's ECU-centric approach to a development process that models application subsystems via software components. Both architectures support the integration of software components from multiple suppliers on shared ECUs, while providing architectural services as a baseline for the application development.

The contributions of the architecture proposed in this paper, which go beyond the features of AUTOSAR, include the following properties:

- **Rigorous encapsulation and error containment.** Based on static resource allocations at the communication network (i.e., TDMA scheme) and the operating system (i.e., fixed cyclic scheduler), the proposed architecture ensures that a job cannot affect the computational and communication resources that are available to other jobs. As discussed in Section 6, this property is of particular significance for the modular certification of mixed criticality systems, the implementation of active redundancy, and complexity management.

- **Predictability and determinism.** The rigorous encapsulation simplifies a timing analysis of application subsystems and jobs. When determining the temporal behavior of a job, a designer can abstract from the behavior of other jobs, because the architecture guarantees to each job predefined communication and computational resources.

  In contrast, the current version of the AUTOSAR runtime environment [AUT06a] supports priority-based mechanisms for communication and the scheduling of runnables. For example, a high-priority runnable of one software component could delay the execution of runnables of other software components. In general, it is thus necessary to analyze the execution behavior of all software components with higher priority on an ECU. Similar dependencies between software components can occur at the communication system when using a priority-based protocol such as CAN.

- **Time-triggered core architecture.** The proposed architecture builds on top of a time-triggered network (e.g., FlexRay), the services of which (e.g., fault isolation, time-triggered message transport) are the foundation for rigorous encapsulation and the achievement of sufficient dependability for safety-critical applications (e.g., X-by-wire). AUTOSAR, on the other hand, does not enforce this restriction. The standard core with its runtime environment supports event-triggered protocols such as CAN for coupling software components.

Nevertheless, these contributions do not contradict the AUTOSAR development methodology. They can also be used to improve AUTOSAR, e.g., through better complexity management and support for safety-critical applications. This includes the mechanisms for encapsulation and error containment, as well as the mapping to an underlying time-triggered core architecture with high predictability and determinism.

Integrated Modular Avionics

While present day automotive electronic systems adhere to the federated architectural paradigm, integrated architectures have already been successfully applied in the development of the electronic systems of commercial aircrafts (e.g., 777 [Wit96]). ARINC standard 651 [Aer91] is known as Integrated Modular Avionics (IMA) and addresses the design of integrated avionic systems. The construction of an IMA architecture relies on several other ARINC standards. For instance, the services of the avionic software environment are specified by ARINC 653 [Aer06], which is known as APplication EXecutive (APEX). APEX provides services for partition management, process management, time management, memory management, interpartition communication, intrapartition communication, and diagnosis. IMA systems are not restricted to a particular communication network. Common standards for the communication network include event-triggered communication systems (e.g., ARINC 664 [ARI03]), and time-triggered communication systems (e.g., ARINC 659 – SAFEbus [ARI93]).

Model of an ECU. ARINC specification 653 distinguishes between *core software* and *application software*. The core software is responsible for mapping the APEX API onto the underlying platform, e.g., implementing channels through the available transport mechanisms. If the core software employs fragmentation, sequencing, routing, or message redundancy, these mechanism have to be transparent to the application software.

Each end-system contains one or more *partitions*. A partition is a set of functionally separated tasks with their associated context and configuration data. The tasks of a partition and the required resources are statically defined. The scheduling of these tasks occurs via two-level scheduling. On the first level, partitions are scheduled by the progression of time. Partitions do not have priorities and are activated relative to a time frame. As the time frame can be synchronized to the underlying communication system, the activation of partitions can also be synchronized to the communication system. Tasks within partitions possess priorities, which are used for dynamic task scheduling. If a partition is active, the scheduler selects the ready task with the highest priority for execution – preempting task's with lower priorities.

Communication System. APEX defines *channels* for interpartition communication through the exchange of messages. The destination for messages exchanged through a channel is a partition, not a process. Communication activities are independent of the physical location of both source and destination partitions. A channel is configured by the system integrator and possesses exactly one sending port and one or more receiving ports. Each port is assigned a port name, which should refer to the data exchanged via the port rather than to the producer/consumer. A port can support either event or state semantics through operating in one of the following two transfer modes:

- **Sampling Mode:** Successive messages contain identical but updated data. Received messages overwrite old information, thus requiring no message queuing. Sampling mode assumes that applications are only interested in the most recent version of a message. A validity indicator denotes whether the age of the copied message is consistent with the required refresh rate defined for the port.
- **Queuing Mode:** Messages are assumed to contain uniquely different data, thus no message should be lost. Messages are buffered in queues, which are managed on a First-In/First-Out (FIFO) basis. The application software is responsible for handling overflowing queues.

The concept of channels, as used for interpartition communication according to APEX, is independent of the actual transport mechanism. While a time-triggered communication system simplifies the establishment of temporal validity in sampling mode, event-triggered communication systems natively support sporadic transmissions of event messages as required in queuing mode.

Relationship to the DECOS Architecture. The DECOS integrated architecture and IMA/APEX share the vision of a physically integrated computer system, in which communication resources as well as computational resources are available to multiple applications. In particular, both architectures emphasize the need for encapsulating communication resources and computational resources (CPU, memory) in order to prevent interference induced by physical integration. However, we can identify the following properties, in which IMA and the integrated DECOS architecture differ:

- **Domain-independence.** While DECOS aims at offering a domain-independent architecture, IMA focuses explicitly on the avionic domain. For example, the automotive domain

significantly differs with respect to life cycles, cost sensitivity of end products, and requirements with respect to flexibility. The automotive domain exhibits a higher number of variants and mass customization w.r.t. electronic systems plays an important role. The individualizing of a car to its customer has a high impact on the electronic systems.

- **System Structuring.** IMA/APEX adheres to a "partition-centric" point of view, while DECOS adopts an "application subsystem"-centric viewpoint. IMA/APEX does not support within the integrated system a set of "virtualized" clusters, as they would exist in a federated system. However, part of the success of the federated paradigm is the ability to structure the overall system into nearly-independent subsystems, each becoming the responsibility of a respective organizational entity (e.g., a supplier). Following this line of reasoning, DECOS supports within a physically integrated computer system a similar logical structuring as a federated computer system. This decoupling of the physical system structuring from the logical system structuring is the key element to achieve the benefits of federated and integrated computer systems. In adopting an application-subsystem centric viewpoint, in DECOS each DAS is provided with a corresponding set of architectural services (e.g., communication services, diagnostic services, time services, gateway services).
- **Communication Service.** IMA provides to partitions support for communication via queueing mode and sampling mode, a distinction which corresponds to the event-triggered and time-triggered control paradigms used in this paper. At the DAS-level, however, DECOS goes beyond the establishment of communication channels for event and state messages, but realizes virtual networks running different communication protocols as encapsulated overlay networks. The underlying design assumption is that at the level of a DAS, the developer is in a position to perform a meaningful decision on a particular communication protocol that best suits the requirements of the DAS (e.g., balanced compromise between conflicting properties, such as flexibility vs. predictability).
- **Generic Gateway Service.** In contrast to IMA, the DECOS architecture provides a generic architectural gateway service that can be parameterized to specific applications. Gateways support the selective redirection of messages between the virtual networks of different DASs. Gateways make dependencies between DASs explicit and provide a systematic solution for resolving property mismatches (e.g., differences w.r.t. syntax, protocols, or naming). Within a gateway, a real-time data base separates the DASs and stores temporally accurate real-time images.

8. Conclusion

Future car generations require computer architectures to accommodate the need for mixed criticality applications, i.e., supporting applications with ultra-high dependability requirements as well as applications where flexibility and resource efficiency is of primary concern (e.g., comfort electronics). The proposed architecture establishes such an infrastructure and also enables physical integration by combining multiple DASs and virtual networks within a single distributed real-time computer system. Thus, the architecture reduces the number of different networks and protocols.

The proposed integrated architecture exhibits flexibility and supports reuse of application software across different car segments. The key element for this flexibility, as well as for complexity management and the independent development of subsystems, are small DASs. Instead of the typical domain oriented system structure, we show that we can subdivide the overall functionality of a car into smaller DASs, each equipped with dedicated architectural services. By transforming

a today's automotive system onto the future E/E architecture, we have demonstrated the feasibility of the integrated architecture for a future automotive system.

Acknowledgments

This work has been supported by the European IST project DECOS under project No. IST-511764.

References

[ARI91] Aeronautical Radio, Inc., Annapolis, Maryland 21401. *ARINC Specification 651: Design Guide for Integrated Modular Avionics*, November 1991.

[ARI93] Aeronautical Radio, Inc., Annapolis, Maryland 21401. *ARINC Specification 659: Backplane Data Bus*, December 1993.

[ARI03] Aeronautical Radio, Inc., Annapolis, Maryland 21401. *ARINC Specification 664 (Draft): Aircraft Data Network Part 7 – Deterministic Networks*, May 2003.

[ARI06] Aeronautical Radio, Inc., Annapolis, Maryland 21401. *ARINC Specification 653: Avionics Application Software Standard Interface, Part 1 - Required Services*, March 2006.

[ASBT03] A. Ademaj, H. Sivencrona, G. Bauer, and J. Torin. Evaluation of fault handling of the time-triggered architecture with bus and star topology. In *Proc. of the 2003 Int. Conference on Dependable Systems and Networks*, pages 123–132, June 2003.

[AUT06a] AUTOSAR GbR. *AUTOSAR – Specification of RTE Software V1.0.1*, July 2006.

[AUT06b] AUTOSAR GbR. *AUTOSAR – Technical Overview V2.0.1*, June 2006.

[BJV91] R.W. Butler, J.L.ăCaldwell, and B.L.ăDi Vito. Design strategy for a formally verified reliable computing platform. In *Proc. of the 6th Annual Conference on Computer Assurance Systems*, pages 125–133, June 1991.

[BKS03] G. Bauer, H. Kopetz, and W. Steiner. The central guardian approach to enforce fault isolation in a time-triggered system. In *Proc. of the 6th Int. Symposium on Autonomous Decentralized Systems*, pages 37–44, April 2003.

[Bos91] Robert Bosch Gmbh, Stuttgart, Germany. *CAN Specification, Version 2.0*, 1991.

[BS05] B. Bouyssounouse and J. Sifakis, editors. *Embedded Systems Design*. Springer Verlag, 2005.

[C. 02] C. Jones et al. Final version of the DSoS conceptual model. *DSoS Project (IST-1999-11585)*, December 2002.

[Elm02] Wilfried Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Technische Universitat Wien, Institut fur Technische Informatik, 2002.

[Fle05] FlexRay Consortium. *FlexRay Communications System Protocol Specification Version 2.1*, May 2005.

[GFL+02] P. Giusto, A. Ferrari, L. Lavagno, J.-Y. Brunel, E. Fourgeau, and A. Sangiovanni-Vincentelli. Automotive virtual integration platforms: why's, what's, and how's. In *Proc. of the IEEE Int. Conference on Computer Design: VLSI in Computers and Processors*, pages 370–378, September 2002.

[Ham03] R. Hammett. Flight-critical distributed systems: design considerations [avionics]. *IEEE Aerospace and Electronic Systems Magazine*, 18(6):30–36, June 2003.

[HB98] B. Hedenetz and R. Belschner. Brake-by-wire without mechanical backup by using a TTP-communication network. In *Proceedings of SAE Congress*. Daimler-Benz AG, 1998.

[He04] H. Heinecke and et al. AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures. In *Proc. of the Convergence Int. Congress & Exposition On Transportation Electronics*, October 2004. 2004-21-0042.

[Hei03] H.D. Heitzer. Development of a fault-tolerant steer-by-wire steering system. *Auto Technology*, 4:56–60, April 2003.

[HPOS05] B. Huber, P. Peti, R. Obermaisser, and C. El Salloum. Using RTAI/LXRT for partitioning in a prototype implementation of the DECOS architecture. In *Proc. of the Third Int. Workshop on Intelligent Solutions in Embedded Systems*, May 2005.

[IEC99] IEC: Int. Electrotechnical Commission. *IEC 61508-7: Functional Safety of Electrical, Electronic, Programmable Electronic Safety-Related Systems – Part 7: Overview of Techniques and Measures*, 1999.

[JB92] S.C. Johnson and R.W. Butler. Design for validation. *IEEE Aerospace and Electronic Systems Magazine*, 7(1):38–43, January 1992.

[KAGS05] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer. The Time-Triggered Ethernet (TTE) design. *Proc. of 8th IEEE Int. Symposium on Object-oriented Real-time distributed Computing (ISORC)*, May 2005.

[KB03] H. Kopetz and G. Bauer. The time-triggered architecture. *IEEE Special Issue on Modeling and Design of Embedded Software*, January 2003.

[KO02] H. Kopetz and R. Obermaisser. Temporal composability. *Computing & Control Engineering Journal*, 13:156–162, August 2002.

[Kop97] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[Kop03] H. Kopetz. Fault containment and error detection in the time-triggered architecture. In *Proc. of the Sixth Int. Symposium on Autonomous Decentralized Systems*, April 2003.

[KS03] H. Kopetz and N. Suri. Compositional design of RT systems: A conceptual basis for specification of linking interfaces. In *Proc. of the 6th IEEE Int. Symposium on Object-Oriented Real-Time Distributed Computing*, pages 51–60, May 2003.

[KWR04] L. Kinnan, J. Wlad, and P. Rogers. Porting applications to an ARINC 653 compliant IMA platform using VxWorks as an example. In *Proc. of the 23rd Digital Avionics Systems Conference*, volume 2, pages 10.B.1–10.1–8, October 2004.

[Lap92] J.C. Laprie. Dependability: Basic concepts and terminology. In *Dependable Computing and Fault Tolerant Systems*, volume 5, pages 257–282. Springer Verlag, Vienna, 1992.

[Lev86] N.G. Leveson. Software safety: why, what, and how. *ACM Comput. Surv.*, 18(2):125–163, 1986.

[LH94] J.H. Lala and R.E. Harper. Architectural principles for safety-critical real-time applications. *Proc. of the IEEE*, 82:25–40, January 1994.

[LH02] G. Leen and D. Heffernan. Expanding automotive electronic systems. *Computer*, 35(1):88–93, January 2002.

[LIN03] LIN Consortium. *LIN Specification Package Revision 2.0*, September 2003.

[Lyn06] LynuxWorks. *LynxOS 4.0 User's Guide*, 2006.

[Nex03] NextTTA. Project deliverable D2.4. Emulation of CAN and TCP/IP, September 2003. IST-2001-32111. High Confidence Architecture for Distributed Control Applications.

[OP05a] R. Obermaisser and P. Peti. Comparison of the temporal performance of physical and virtual CAN networks. In *Proc. of the IEEE Int. Symposium on Industrial Electronics*, Dubrovnik, Croatia, June 2005.

[OP05b] R. Obermaisser and P. Peti. Realization of virtual networks in the DECOS integrated architecture. In *Proc. of the Workshop on Parallel and Distributed Real-Time Systems 2006 (WPDRTS)*. IEEE, April 2005.

[OP05c] R. Obermaisser and P. Peti. Specification and execution of gateways in integrated architectures. In *Proc. of the 10th IEEE Int. Conference on Emerging Technologies and Factory Automation*, Catania, Italy, September 2005. IEEE.

[OP06] R. Obermaisser and P. Peti. A fault hypothesis for integrated architectures. In *Proc. of the 4th Int. Workshop on Intelligent Solutions in Embedded Systems*, June 2006.

[OPHS06] R. Obermaisser, P. Peti, B. Huber, and C. El Salloum. DECOS: An integrated time-triggered architecture. *e&i journal (journal of the Austrian professional institution for electrical and information engineering)*, 3:83–95, March 2006.

[PO06] P. Peti and R. Obermaisser. A diagnostic framework for integrated time-triggered architectures. In *Proc. of the 9th IEEE Int. Symposium on Object-oriented Real-time distributed Computing*, April 2006.

[Pol94] S. Poledna. Replica determinism in distributed real-time systems: A brief survey. *Real-Time Systems*, 6:289–316, 1994.

[Rei06] K. Reif. *Automobilelektronik. Eine Einfuhrung fur Ingenieure (Broschiert)*. Vieweg, May 2006.

[RH04] G. Reichart and M. Haneberg. Key drivers for a future system architecture in vehicles. In *Convergence Int. Congress*. SAE, October 2004.

[Rol06] T. Rolina. Past, present, and future of real-time embedded automotive software: A close look at basic concepts of AUTOSAR. In *Proc. of SAE World Congress*, Detroit, Michigan, April 2006.

[Rus01] J. Rushby. Bus architectures for safety-critical embedded systems. In *Proc. of the 1st Workshop on Embedded Software*, volume 2211 of *Lecture Notes in Computer Science*, pages 306–323. Springer-Verlag, October 2001.

[Sim96] H.A. Simon. *The Sciences of the Artificial*. MIT Press, 1996.

[SLO03] M. Segarra, T. Losert, and R. Obermaisser. Hard real-time CORBA: TTP transport definition. Technical Report IST37652/067, Universidad Politecnica de Madrid, March 2003.

[SM99] J. Swingler and J.W. McBride. The degradation of road tested automotive connectors. In *Proc. of the 45th IEEE Holm Conference on Electrical Contacts*, pages 146–152, October 1999.

[SW04] A. Saad and U. Weinmann. Intelligent automotive system services: Requirements, architectures and implementation issues. In *Convergence Int. Congress*, Detroit, MI, USA, October 2004. SAE.

[SWH95] N. Suri, C.J. Walter, and M.M. Hugue. *Advances In Ultra-Dependable Distributed Systems*, chapter 1. IEEE Computer Society Press, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264, 1995.

[TTT02] TTTech Computertechnik AG. *Time-Triggered Protocol TTP/C – High Level Specification Document*, July 2002.

[Wit96] B. Witwer. Systems integration of the 777 airplane information management system (aims). *IEEE Aerospace and Electronic Systems Magazine*, 11(4):17–21, April 1996.