Temporal Partitioning of Communication Resources in an Integrated Architecture

Roman Obermaisser
Vienna University of Technology, Austria
email: romano@vmars.tuwien.ac.at

***Abstract*— Integrated architectures in the automotive and avionic domain promise improved resource utilization and enable a better coordination of application subsystems compared to federated systems. An integrated architecture shares the system's communication resources by using a single physical network for exchanging messages of multiple application subsystems. Similarly, the computational resources (e.g., memory, CPU time) of each node computer are available to multiple software components. In order to support a seamless system integration without unintended side effects in such an integrated architecture, it is important to ensure that the software components do not interfere through the use of these shared resources. For this reason, the DECOS integrated architecture encapsulates application subsystems and their constituting software components. At the level of the communication system, virtual networks on top of an underlying time-triggered physical network exhibit predefined temporal properties (i.e., bandwidth, latency, latency jitter). Due to encapsulation the temporal properties of messages sent by a software component are independent from the behavior of other software components, in particular from those within other application subsystems. This paper presents the mechanisms for temporal partitioning of communication resources in the DECOS integrated architecture. Furthermore, experimental evidence is provided in order to demonstrate that the messages sent by one software component do not affect the temporal properties of messages exchanged by other software components. Rigid temporal partitioning is achievable, while at the same time meeting the performance requirements imposed by present-day automotive applications and those envisioned for the future (e.g., X-by-wire). For this purpose, we use an experimental framework with an implementation of virtual networks on top of a TDMA-controlled Ethernet network.**

***Index Terms*— System architectures, real-time and embedded systems, system integration and implementation, fault-tolerance, computer network performance, distributed architectures, infrastructure protection**

I. Introduction

In application domains with large distributed embedded computer systems, such as the avionic or automotive domains, integrated architectures are viewed as superior compared to federated architectures. While federated architectures provide each application subsystem (e.g., powertrain subsystem in a car) with a dedicated set of nodes interconnected by a respective network, integrated architectures permit the coexistence of multiple application subsystems within a single distributed computer system. Integrated architectures support the sharing of computational resources (i.e., node computers with corresponding CPU time, memory, I/O) and communication resources (i.e., network bandwidth) among multiple software components belonging to different application subsystems. Integrated Modular Avionics (IMA) [1], Automotive Open System Architecture (AUTOSAR) [2], and the cross-domain Dependable Embedded Components and Systems (DECOS) architecture [3] are examples of integrated architectures that are already partly deployed (IMA) or still in development (AUTOSAR, DECOS). Especially in the automotive domain, integrated architectures are a solution that promises to stop the continuous increase in the number of node computers, which has been a side-effect of the increasing functionality realized by electronic systems in the past years. While significant advances w.r.t. safety and comfort can be attributed to electronics, the high number of node computers and networks has resulted in high hardware cost, weight, and power consumption. For example, with an average cost of 40 Euro per node computer in today's cars and 0.2 Euro per wire, the total hardware cost of a federated automotive system with 40 nodes and 800 wires is about 1800 Euro [4]. In addition, increased wiring can affect the reliability of automotive electronics. Field data from automotive environments has shown that more than 30% of electrical failures are attributed to connector problems [5]. Even so, the functionality of electronic systems is expected to continue its rapid growth. Automotive electronics are estimated to enable 90% of all innovations in cars [6][p. 15].

Despite the importance of deploying integrated architectures, it is important to recognize the benefits that have resulted from the natural separation of application subsystems in federated architectures. Each application subsystem possesses a functional complexity that is inherent to the application. The functional complexity of an application subsystem when implemented on a target system is dramatically increased by the so-called accidental complexity [7] in case the architecture does not prevent unintended architecture-induced side effects in the communication system. For example, consider a scenario with two application subsystems. If the two application subsystems share a common CAN bus [8], then both application subsystems must be analyzed and understood in order to reason about the correct behavior of either of the two application subsystems. Since the message transmissions of one application subsystem can delay message transmission of the other application subsystem, arguments concerning the correct temporal behavior must be based on an analysis of both application subsystems.

The challenge for the communication infrastructure of an integrated architecture is therefore the prevention of accidental complexity through architecture-induced side effects in the communication system. In analogy to a federated architecture, the communication system of an integrated architecture must support the division of the overall functionality into separate application subsystems with guaranteed communication resources. When constructing mixed criticality systems, which are composed of application subsystems with different criticality levels, it is also important to preserve these guarantees in the presence of design

faults. Otherwise, error propagation to subsystems with higher criticality could occur through the communication system.

In previous work, we have developed the conceptual foundation for the communication infrastructure of the DECOS architecture along with a model-based development process [9]. Each application subsystem is provided with a dedicated communication infrastructure that is realized as an encapsulated *Virtual Network (VN)*, i.e., an overlay network on top of a time-triggered physical network. Each VN supports a corresponding communication paradigm (i.e., time-triggered or event-triggered control [10]) and is tailored to the requirements of the respective application subsystem via its temporal properties (e.g., latencies, bandwidth) and its communication topology (e.g., broadcast, multicast, point-to-point).

The *encapsulation mechanisms* of VNs, which will be explained in this paper, address the challenge of preventing architecture-induced side effects in the communication system. By supporting temporal partitioning [11] both between and within application subsystems, VNs not only help developers to focus on the inherent application complexity, but also contribute to the correctness-by-construction of component-based systems (i.e., composability [12]). The temporal properties of a VN remain invariant throughout the system integration. This property called *temporal composability* [13] is an important baseline for a constructive system development that prevents unintended interference between independently developed software components. Furthermore, encapsulation also addresses the integration of mixed criticality systems. Encapsulation ensures that a failure of a software component belonging to a non safety-critical application subsystem does not affect the exchange of messages between software components of safety-critical application subsystems.

The paper is a substantial extension of previous work in the context of the DECOS architecture w.r.t. virtual networks [14], [9] and operating systems [15]. While [15] has focused on the temporal partitioning (i.e., CPU time) and spatial partitioning (i.e., memory protection) for the *computational resources* [15], the present paper describes the partitioning for the *communication resources* (i.e., communication bandwidth, latency, jitter). In extension to previous work on virtual networks [14], [9], we introduce mechanisms for temporal partitioning of overlay networks on top of a time-triggered network and perform an experimental evaluation. We provide experimental evidence for the effectiveness of temporal partitioning in a prototype implementation with multiple event-triggered and time-triggered overlay networks. Despite rigid temporal partitioning, the measurements also show that the overlay networks can handle the performance requirements of today's automotive electronic systems and those of future X-by-wire cars.

The main contributions of this paper include:

- **Mechanisms for temporal partitioning in the communication system of an integrated architecture.** We present a conceptual model of an integrated computer system that distinguishes clearly between logical and physical structuring. Based on this model, we use the communication slots of a time-triggered physical network and subdivide them hierarchically for the structural entities of the logical and physical system structuring. Software mechanisms (e.g., communication middleware) in conjunction with hardware mechanism (e.g., bus guardians) protect these communication slots down to the level of individual software components, which can be collocated on shared integrated node computers.
- **Communication infrastructure for heterogeneous application subsystems.** The presented communication system supports both time-triggered and event-triggered communication activities and the coexistence of application subsystems with mixed criticality levels.
- **Experimental assessment of temporal partitioning.** In this paper, the invariance of the temporal properties of a communication system comprising multiple VNs is subject to comprehensive tests. We provide experimental evidence for the guaranteed temporal properties of the message exchanges. Two experimental campaigns systematically explore different scenarios for the behavior of software components at the communication system. We also assess the effects of faulty software components (e.g., babbling idiot failures).
- **Experimental assessment of performance.** By comparing the observed performance with the bandwidth and latency requirements of present day and upcoming automotive applications, we demonstrate that a communication system with rigid temporal partitioning can also support a competitive temporal performance.

II. Basic Concepts and Related Work

Partitioning is concerned with ensuring that a failure in one fault-containment region does not propagate to cause a failure in another fault-containment region. Effective partitioning mechanisms are thus important in order to preserve the independence of fault-containment regions and prevent common mode failures. Temporal partitioning [11] is an instantiation of the general notion of partitioning and ensures that a fault-containment region cannot affect the ability of other fault-containment regions to access shared resources, such as the common network or a shared CPU. This includes the temporal behavior of the services provided by resources (e.g., latency, jitter, duration of availability during a scheduled access).

In integrated architectures, one can distinguish two types of fault-containment regions and two respective types of partitioning [3]. For physical faults, a complete node computer can be regarded as a fault-containment region. It is usually not justified to assume independent fault-containment regions on a single node computer due to the shared physical resources (e.g., power supply, oscillator, physical proximity). For design faults, on the other hand, each of the software components (that are collocated on a node computer) can be regarded as an individual fault-containment region. The reason for this finer structuring w.r.t. design faults in an integrated architecture is that software components on a node computer are typically self-contained functional elements, which are developed by different organizations.

Along with this discrimination of fault-containment regions, one can distinguish partitioning between nodes (i.e., *inter-node partitioning*) and partitioning between software components within a node (i.e., *inner-node partitioning*). In the following, we will give an overview of related work for these different types of temporal partitioning with a focus on time-triggered solutions. Also, solutions for mediating data flows between different levels of criticality will be addressed.

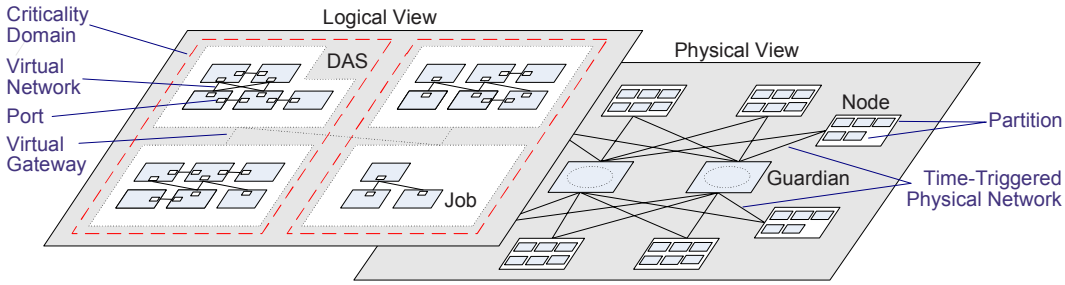


Fig. 1. Modularization and Abstraction – Logical and Physical View

A. Inter-Node Partitioning

For federated architectures that are deployed on time-triggered networks, temporal partitioning between nodes has been investigated extensively in previous work. The a priori knowledge about the delimiting points in time of a node's slot in the time-triggered communication scheme has been exploited by guardians that manage access to the underlying network in order to prevent a node from sending in the slot of another node. For this purpose, node-local and centralized guardians have been developed and validated for different time-triggered communication protocols (e.g., [16], [17]).

B. Inner-Node Partitioning for Computational Resources.

At the level of the computational resources, the primary source of temporal fault propagation within a node is a task that delays other tasks by holding a shared resource (e.g., the processor). Based on static scheduling of the computational resources, solutions for partitioning have been developed in the context of IMA. For example, LynxOS-178[1] is a certifiable operating system that supports temporal partitioning through a fixed-cyclic time-slice scheduler. Fixed-cyclic time-slice schedulers have also been combined with other scheduling policies using two-level hierarchical scheduling. For instance, [18] proposes a microkernel with a cyclic/priority driven scheduler pair that guarantees temporal partitioning. Another example of a solution for temporal partitioning at the level of the computational resources is the formally verified DEOS scheduler kernel [19], which enforces time partitioning using a rate monotonic scheduling policy.

C. Inner-Node Partitioning for Communication Resources.

In avionics, the need for temporal partitioning of communication resources within a line-replacable unit has gained high recognition with the introduction of IMA. When discussing the use of SAFEbus in IMA [20], it is stated that the execution environment of each function in a cabinet should be as much like the environment in the discrete line-replacable unit. Therefore, SAFEbus has been designed as a table-driven protocol, which enforces strict deterministic control for temporal partitioning between the line-replacable modules in a cabinet. However, line-replacable modules represent separate hardware elements equipped with dedicated hosts and bus interface units [20], although they typically share power supply and I/O units. In contrast, this paper addresses temporal partitioning w.r.t. communication resources at the level of software components within a node computer.

[1]http://www.lynuxworks.com/

D. Mediation of Data Flow between Different Levels of Criticality

The virtual networks presented in this paper provide temporal partitioning w.r.t. the communication resources. The only way in which a faulty job can affect other jobs is by providing to the other jobs faulty inputs. The elimination of interference in the use of communication resources is an important baseline for partitioning mechanisms at higher levels. In particular, higher levels can focus on the mediation of data flows between different levels of criticality. For example, many safety-critical control applications depend on sensory information from unreliable sources [21]. This criticality differential can be resolved by using fault-tolerance mechanisms and exploiting redundancy. For example, the GUARDS architecture proposes an information flow integrity policy [21] in which validation objects take low integrity inputs and run fault-tolerance mechanisms to produce high integrity outputs.

In the DECOS architecture, the mediation between data flows of different criticality is the purpose of so-called *virtual gateways* [22]. Virtual gateways interconnect virtual networks, while enforcing error containment and resolving property mismatches. In addition, virtual gateways can recombine messages out of their constituting parts. For example, data contained in multiple messages from unreliable sources can be combined in order to send a message to a safety-critical application subsystem.

III. System Model

Modularization and *abstraction* are two fundamental principles used for managing complexity [23]. Modularization deals with the meaningful decomposition of a system into smaller subsystems, while abstraction is concerned with the level of detail in the representation of a system or subsystem. This section discusses modularization and abstraction for a DECOS system, i.e., a system developed in compliance with the DECOS architecure, at both the logical and physical level (top and bottom levels in Figure 1). Furthermore, we introduce a naming scheme for identifying structural elements of the physical and logical level.

A. Logical View

At the top-most level of abstraction, 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 structured into a set of nearly-decomposable *Distributed Application Subsystems (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 [24]. The DASs can

be grouped into criticality domains, e.g., based on a common Safety Integrity Level (SIL) [25] or critical failure modes [26].

In analogy to the structuring of the overall system, we further decompose each DAS into smaller units called *jobs*. A *job* is the basic unit of work and employs a *Virtual Network (VN)* [9] with predefined temporal properties in order to exchange messages with other jobs of the DAS. The interface between a job and a VN is denoted as a *port*.

The two-level structuring into DASs and jobs is motivated by the insight that large distributed real-time systems typically consists of a set of DASs with high inner connectivity and looser interactions in-between. These different orders of magnitude in the interactions between subsystems correspond to the notion of near-decomposability in hierarchic systems as introduced in [27, chap. 8]. For example, in the automotive domain a two-level structuring is applied to manage the complexity of the in-vehicle electronic system. In many present-day cars, the electronic system consists of several DASs (e.g., powertrain, comfort, passive safety subsystem), each of which is implemented with a distributed computer system. On its behalf, such a DAS is further structured into smaller logical elements that are implemented by electronic control units [28], [29].

The DECOS integrated architecture facilitates the management of the complexity of large distributed real-time systems by enabling designers to perform a logical structuring of systems in the same way as if they were realizing DASs in a federated architecture. Although the DASs are physically integrated and the hardware (i.e., network, node computers) is shared, the DECOS integrated architecture encapsulates the elements of the logical system structuring (i.e., DASs, jobs) along with their communication resources (i.e., the VNs). Other DASs cannot perceive or affect exchanged messages (in the temporal or value domain) other than those being explicitly exported via a *virtual gateway* [22]. By exercising this strict control over the interactions between DASs, only the behavior of the DAS's VN and the behavior of gateways is relevant when reasoning about a DAS.

B. Physical View

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* (nodes for short), which are interconnected by a time-triggered physical network. The VNs 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 [30]. Every node provides one or more *partitions*, each hosting a corresponding job. A partition is an encapsulated execution space within a node 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 node, the DECOS architecture goes beyond the prevalent "1 Function – 1 Electronic Control Unit" design principle [6].

C. Architectural Namespace

The namespace of the integrated architecture, which has been introduced in [14], consists of two parts, one reflecting the physical structure and one reflecting the logical structure of the system:

$$\underbrace{id_{\text{node}}}_{\text{physical structure}} \quad : \quad \underbrace{id_{\text{criticality}}.id_{\text{DAS}}.id_{\text{job}}}_{\text{logical structure}}$$

The physical part identifies the node (id_{node}) of the integrated system, while the logical part following the colon identifies the criticality domain ($id_{\text{criticality}}$), the DAS (id_{DAS}), and the job (id_{job}), where id is the numerical identification of a structuring element. Since each DAS possesses exactly one VN, the DAS identifier also designates its corresponding VN.

For convenience in writing a name, it is possible to omit either the physical or the logical part if not needed by the specification. For example, by describing only logical aspects, one can abstract from the physical allocation (i.e., specifying solely $:id_{\text{criticality}}.id_{\text{DAS}}.id_{\text{job}}$).

IV. Virtual Networks

An overlay network is a computer network which is built on top of another network. The DECOS architecture provides overlay networks, which are denoted as *Virtual Networks (VNs)*, on top of a time-triggered physical network. Each VN handles the message exchanges of a corresponding DAS and provides encapsulation for the jobs by preventing jobs from affecting the temporal properties of messages sent by other jobs.

This section describes how the time-triggered physical network provides dedicated communication resources for the different structural elements introduced in the logical (i.e., criticality domains, DASs, jobs) and physical view-points (i.e., nodes). These communication resources are used to construct VNs for the exchange of event and state messages. In addition, we discuss the encapsulation of VNs based on the static resource allocation on the physical network and the design of the interfaces between VNs and jobs.

A. Communication Resources provided by the Time-Triggered Physical Network

For the realization of the communication services in the integrated DECOS architecture, we employ a time-triggered physical network and perform a hierarchic temporal subdivision of the communication resources (see Figure 2). The media access control strategy of the time-triggered physical network is Time Division Multiple Access (TDMA). TDMA statically divides the channel capacity into a number of slots and controls access to the network solely by the progression of time. Each node is assigned a unique *node slot* that periodically recurs at a priori specified global points in time. A node sends messages during its node slot and receives messages during the slots of the other nodes. A

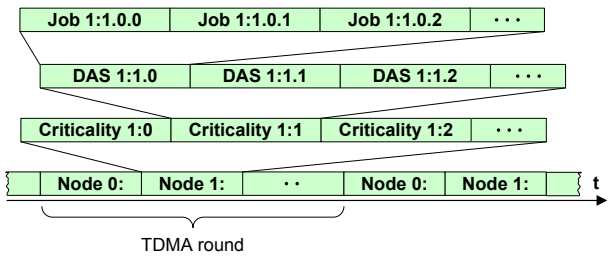


Fig. 2. Hierarchic Subdivision of TDMA Slots

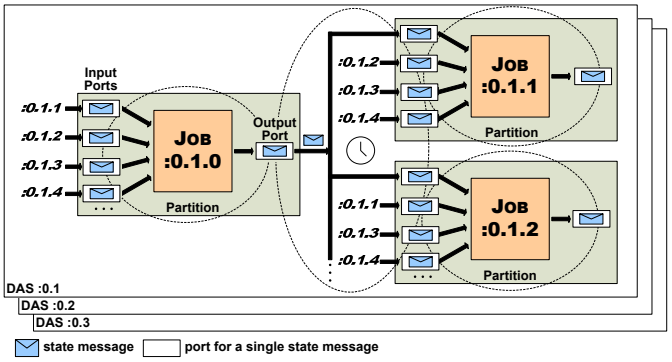


Fig. 3. Virtual Network for Exchange of State Messages

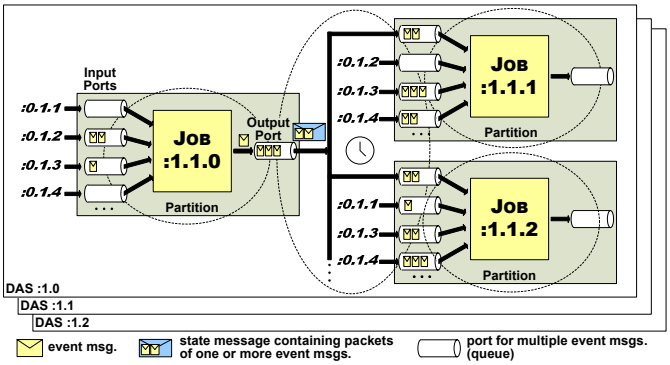


Fig. 4. Virtual Network for Exchange of Event Messages

sequence of node slots, which allows every node in an ensemble of nodes to send exactly once, is called a TDMA round.

We further subdivide each node slot (bottom-most layer in Figure 2) in correspondence to the logical structuring of a DECOS system. In a first step, the node slot is subdivided into subslots for the criticality domains. Thereafter, a subdivision into subslots for the DASs takes place. Such a *DAS slot* contains those messages that are produced by the jobs in the node that belong to a particular DAS. On its part, a DAS slot consists of smaller subslots denoted as *job slots* (top-most level in Figure 2). Each job slot provides the communication resources for the establishment of a *communication channel*, which serves the transport of messages produced by a particular sender job. A communication channel connects one output port (of the sender job) with multiple input ports (of the receiver jobs). The output port of a communication channel maps to exactly one job slot, namely the slot of the job possessing the output port. In this job slot the messages produced at the output port are sent on the underlying time-triggered network. The number of the input ports depends on the communication topology. For example, a single input port establishes a point-to-point topology, while a broadcast topology requires associated input ports for all other jobs of the DAS.

The composition of the communication channels for all jobs of a DAS results in a *VN*. In general, a VN comprises job slots that belong to jobs located on different nodes. In order to construct the VN of a DAS with multiple jobs that need to send messages, the VN requires a number of communication channels that is equal to the number of jobs in the DAS.

The assignment of the slots within a TDMA round to nodes, as well as the further subdivision into criticality-domain slots, DAS slots, and job slots is fixed at design time. This static allocation ensures that the network resources are predictably available to jobs. The definition of the TDMA scheme is an important design activity that determines the temporal properties of the resulting communication channels. Firstly, the period of the job slot in the TDMA scheme determines the latency for message transmissions that are not synchronized to the TDMA-scheme. In the worst case, a job writes a message into its output port after the occurrence of its job slot, thus incurring a delay of a complete TDMA round. Secondly, the size of the job slots in conjunction with the period of the job slots determines the available bandwidth.

B. Virtual Networks for State and Event Messages

One can distinguish between state messages and event messages [10, p. 31] depending on the temporal behavior and the information semantics. *State messages* are periodic messages with state information (e.g., temperature is 20°), while *event messages* are sporadic and contain event information (e.g., temperature increase by 2°). Based on this distinction, there are two types of ports (event ports and state ports) as the interface between the application and the communication system. For any given communication channel, a particular port type must be used consistently.

*a) **State ports**:* A communication channel for the transport of state messages employs state ports, each containing a memory element that is overwritten with a new value whenever a state message arrives from the VN (in case of an input port) or the job (in case of an output port).

For example, Figure 3 depicts a VN for DAS :0.1 with communication channels for the exchange of state messages with a broadcast topology. Each of the jobs possesses a communication channel, which is interfaced by an output port with a memory element storing a state message. In addition, a job has an input port for every communication channel the job receives messages from. A message arriving via a communication channel causes an update of the memory element in the respective input port, but does not affect the contents of other input ports.

*b) **Event ports**:* An event port supports event messages containing information that is associated with a particular event. In order to reconstruct the current state of a real-time entity [10, p. 98] from event messages, it is essential to process every message exactly once. Consequently, a VN must support the transmission of event messages with exactly-once delivery semantics and provide message queues at input and output ports. For the transport via the underlying time-triggered network, the communication channel packs event messages into state messages for the dissemination via the periodically reoccurring job slot.

Figure 4 depicts a VN for the exchange of event messages. In analogy to the previous example, this VN also establishes a broadcast topology. However, the interface between the jobs and the VN is provided with event ports.

Each port separates *two spheres of control*, namely the time-triggered control of a communication channel and the event-triggered control of a job (see dashed ellipses in Figures 3 and 4). At the communication channel, the transport of messages between the (one) output port and the input ports is periodically triggered by the occurrence of the sender's job slot in the TDMA scheme. Consequently, the communication channels employ strict time-triggered control. A job, on the other hand, can perform event-triggered access operations, i.e., at points in time that are not

a priori known. A job acts according to the *information push paradigm* [31] when updating a state message in the output state port or inserting an event message into an output event port. A job does not delegate control outside its sphere of control, but determines autonomously the point in time of an access operation. This job autonomy simplifies a timing analysis of the job and limits control error propagation. Likewise, the points in time for reading the memory element of an input port are chosen by the job, thus supporting message receptions according to the *information pull paradigm* [31].

The design decision of supporting both state and event messages is motivated by the insight that both message types exhibit respective advantages. An integrated system can combine heterogeneous DASs with different requirements concerning the communication systems. While state messages are well-suited for safety-critical DASs (e.g., control applications), event messages are typically favorable in non safety-critical DASs with emphasis on flexibility. In the proposed architecture, event messages provide the following benefits:

- **Reuse of event-triggered legacy applications.** Event ports support the reuse of legacy applications that were designed to perform sporadic exchanges of event messages. For example, many CAN-based applications [8] in today's cars send messages as a consequence of significant events in the environment (e.g., input from driver).
- **Message size is decoupled from size of job slots.** By using a packet service, communication channels for event messages decouple the size of transferred messages from the size of a job slot. For example, multiple small messages can be transmitted collectively within a single job slot or a large message can be fragmented over multiple slots within consecutive rounds.
- **Bandwidth elasticity.** The event ports of a communication channel for the exchange of event messages provide bandwidth elasticity. Due to the queues at the output ports, a job can pass more messages to the communication system than can be transmitted in a single TDMA round on the underlying time-triggered network. VNs can handle such a burst as long as the average bandwidth consumption can be bounded to dimension the job slots in the TDMA scheme, and the maximum message load of a burst is known in order to dimension the queue sizes.
- **Variability of a job's message service time.** The queue of an input port supports a variability of the message service time (i.e., the time between successive retrievals of messages through the job). However, in order to prevent queue overflows at the input port, the average service time must be smaller than the average interarrival time of messages from the VN. In addition, a priori knowledge concerning message bursts is required to dimension the queue size.
- **Exactly-once semantics.** The connection between the output port at the sender and the input port at the receiver occurs with a communication channel that exhibits both unidirectional data flow and unidirectional control flow. Thereby, we prevent by design the introduction of an error propagation path from the receiver to the sender. While this design decision supports rigid temporal partitioning, it also implies restrictions for the exactly-once semantics. Exactly-once semantics only hold for a known message timing model, e.g., minimum message interarrival times with bursts of bounded duration and load. Such a message timing model can be used to dimension the queue lengths at input and output ports, as well as the size of the job slots.

For guaranteeing exactly-once semantics without sufficient a priori knowledge concerning the message timing, end-to-end control can be realized using higher protocols on top of a VN. For example, in [32] such an end-to-end control scheme has been realized on top of a VN with support for event messages. However, the bidirectional control flow can introduce paths for error propagation from the receivers to the sender. Therefore, (at the basic level) the presented VNs adhere to the design decision of using only unidirectional communication channels, leaving the optional establishment of bidirectional control flows to higher protocols (e.g., using TCP/IP as in [32]).

C. Encapsulation of Virtual Networks

Due to encapsulation, developers need not look at all possible interactions between jobs and DASs in order to understand the temporal behavior of a VN. In particular, upon the occurrence of faults covered in the fault hypothesis [3], the encapsulation of VNs preserves the modularization of the overall system into DASs and jobs as introduced in the logical system structuring. The primary purpose of encapsulation is the prevention of adverse effects on the message exchanges of a particular VN induced by the message exchanges on VNs of other DASs. In addition to this encapsulation at the DAS-level, VNs are designed for encapsulation at the job-level. Encapsulation at the job-level encompasses the prevention of adverse effects on the message exchanges of a job induced by the message exchanges of other jobs in the same DAS.

Encapsulation confines the effects of a job failure that results in the transmissions of incorrect messages. In case of such a job failure, one can distinguish between message timing and message value failures. A message sent at an unspecified time is denoted as a *message timing failure*. Examples for specific message timing failures are crash/omission failures and babbling idiot failures [33]. A *message value failure* occurs in case the contents of a transmitted message do not comply with the interface specification. In general, the detection of message value failure requires application-specific knowledge either through a priori knowledge or redundant computations. An example for the latter case is active redundancy (e.g., Triple Modular Redundancy (TMR)), which supports the detection and masking of message value failures by majority voting. In the scope of this work, we focus on the encapsulation in the time domain by means of temporal partitioning.

For describing the temporal partitioning, we take on a *sender-centric view*. This means that we look at the non interference of the message transmissions between sender jobs, while abstracting over interference between message transmissions from the same sender job. The reason for this view is the fault hypothesis of the DECOS architecture [3], which regards each job as a distinct fault-containment region w.r.t. to software faults.

Providing a dedicated input port for each communication channel at all receivers and the reservation of dedicated slots in the underlying TDMA scheme are the two key elements for temporal partitioning of VNs.

In case of event ports, *separate input ports* and thus separate queues ensure that the queuing delays for messages received from one sender job do not depend on the communication activities of other jobs (see Figure 4). In addition, separate message queues prevent a sender job that violates its message interarrival time specification [34] from causing the loss of messages sent by other

jobs. A message omission failure caused by a queue overflow at an input port only affects the messages sent by a single sender job.

In addition to providing separate input ports, we also need to prevent interference between messages on different communication channels prior to the arrival at the respective input ports. For this purpose, the DECOS architecture performs a separation of communication channels via statically reserved slots in the underlying TDMA scheme (cf. Section IV-A). Thereby, guardians can protect the access to these slots based on the a priori knowledge concerning the delimiting points in time of TDMA slots and the associations between TDMA slots and the structural elements of the integrated system. At all four levels identified in the hierarchic subdivision of TDMA slots (i.e., nodes, criticality domains, DASs, and jobs), protection of the respective slot is enabled. While several solutions for the protection of node slots on a time-triggered network are already available (e.g., [35], [16], [17]), middleware services for the protection of the subslots for criticality domains, DASs, and jobs have been realized in the scope of this work.

1) Encapsulation at node-level by protecting node slots: According to the DECOS fault hypothesis [3], a node is the fault-containment region for hardware faults. Since the justification for building ultra-reliable systems from replicated nodes rests on the assumption of failure independence among redundant units, the independence of fault-containment regions is of critical importance [36]. Although a fault-containment region can restrict the immediate impact of a fault, error containment must ensure that errors cannot propagate across the boundaries of fault-containment regions. For hardware faults, an important goal of error containment is the protection of the node slots at the shared time-triggered network, i.e., preventing a faulty node from sending in the slot of another node.

2) Encapsulation of criticality domains by protecting criticality-domain slots: The DECOS architecture supports mixed criticality integration by sharing the nodes and the network among jobs of different criticality levels. In general, safety-critical and non safety-critical 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 DASs with reliability requirements of 10^{-7} failures/hour or better (i.e., SIL 3/SIL 4 according to IEC 61508 [25]), the absence of design faults can no longer be shown by testing alone. The achievement of reliability includes a rigorous development process, formal verification, and involvement of a certification agency. For this purpose correctness-by-construction methods have also gained more and more momentum in recent years for the development of safety-critical software [37]. For non safety-critical DASs, certainty about the complete absence of design faults is usually economically infeasible. For this reason, the integrated architecture needs to prevent error propagation between different criticality domains to prevent a design fault in a non safety-critical DASs from affecting safety-critical ones.

3) Encapsulation of DASs by protecting DAS slots: Even within a particular criticality domain, encapsulation of DASs is beneficial in order to simplify the system integration and to improve robustness. If DASs are developed by different organizations (e.g., different suppliers), then encapsulation eases the task of tracing back a fault to the responsible organization.

4) Encapsulation of jobs by protecting job slots: Encapsulation at the job-level further improves the system integration and robustness benefits that can be gained by encapsulation at the DAS-level. In particular, if individual jobs are within the responsibility of different organizations or development teams, system integration and diagnosis efforts are reduced.

V. Experimental Setup with Prototype Implementation of Virtual Networks

VNs have been realized in a prototype implementation that serves as an experimental setup for the evaluation of the temporal performance and encapsulation properties. The experimental setup depicted in Figure 5 contains a distributed computer system with five nodes that are interconnected by a TDMA-controlled Ethernet network. The TDMA-controlled Ethernet network handles the message exchanges between the nodes hosting jobs of one or more DASs. Each DAS is provided with guaranteed communication resources via a dedicated VN realized on top of the TDMA-controlled Ethernet network.

A. Nodes

Each node is shared among a number of jobs in order to overcome the prevalent “1 Function – 1 Electronic Control Unit” limitation of present-day electronic systems [6]. The so-called *VN middleware* realizes multiple VNs on top of the TDMA-controlled Ethernet network and provides a set of ports for the exchange of state or event messages to each of the jobs on the node.

1) Hardware and Operating System: Each node is implemented on a Soekris net4801[2] embedded single-board computer. A real-time Linux variant extended with a time-triggered task scheduler [15] is used in the construction of encapsulated partitions for the execution of the jobs. The operating system provides execution slots with a duration of 400 μs for the execution of jobs and the virtual network middleware. Also, it monitors task execution times in order to detect worst-case execution time violations.

2) Real-Time Ethernet Driver: Time-triggered communication on top of Ethernet is a solution capable of ensuring predictable real-time behavior, while employing Commercial-Off-The-Shelf (COTS) hardware [38]. For the implementation of VNs, we have employed a software-based implementation of a TDMA-controlled Ethernet network. In each node a real-time Ethernet driver performs clock synchronization and periodic time-triggered transmissions and receptions of state messages. The real-time Ethernet driver provides access to these state messages via a set of state ports at the interface to the VN middleware.

The state ports include one output port for sending a state message on the TDMA-controlled Ethernet network and four input ports for receiving state messages from the other four nodes of the prototype cluster. We have used a homogeneous configuration with a single state message with a size of 1500 bytes for each node in the distributed system. The receptions and transmissions on the TDMA-controlled Ethernet network are strictly time-triggered, i.e., controlled by the progression of the global time.

[2]http://www.soekris.com

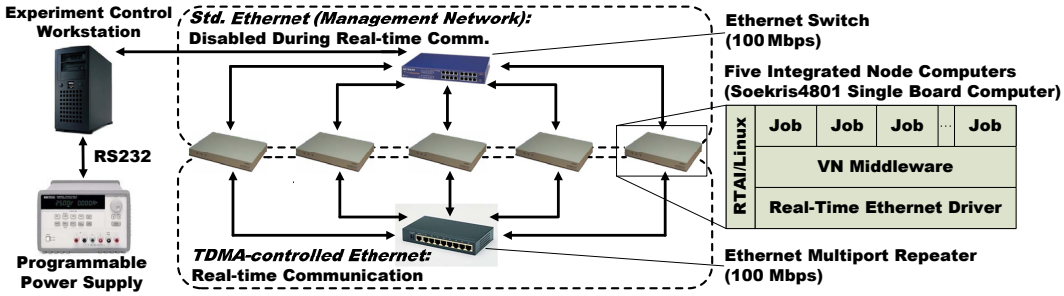


Fig. 5. Implementation of Virtual Networks and Experimental Setup

3) VN Middleware: The VN middleware is a Linux kernel module that translates between the ports of the jobs and the ports towards the TDMA-controlled Ethernet network. The ports of the jobs include input and output ports, each storing either a state message (in case of a state port) or event messages (in case of an event port) as introduced in Section IV.

The ports towards the TDMA-controlled Ethernet network contain state messages with a compound structure, which represents the entities of the logical system view: criticality domains, DASs, and jobs. This compound structure of the state messages results from the hierarchic subdivision of the TDMA slots. At the finest granularity (i.e., at job-level), each data segment is uniquely associated with a specific sender job. Prior to a message transmissions on the TDMA-controlled Ethernet network, the VN middleware goes through all output ports at the interface to the jobs and constructs the job-specific data segments in the Ethernet message to be broadcast.

In addition, the VN middleware is activated after the (time-triggered) receptions of messages by the real-time Ethernet driver. In this case, the VN middleware processes all job-specific data segments in the Ethernet message and forwards the information to the input ports of the jobs.

The design decision of realizing the VNs in middleware on top of a time-triggered communication controller is motivated by the compatibility to existing time-triggered communication protocols and the preservation of existing validation results:

- **Compatibility with different time-triggered communication protocols.** Any time-triggered communication protocol, which provides a global time base and periodic exchanges of state messages, can be used as a basis for the realization of VNs. Examples for suitable protocols are TTP [39], SAFEbus [20], Time-Triggered Ethernet [38], and FlexRay [40].
- **Preservation of validation results.** Since the time-triggered communication controller remains unchanged, the evidence for the correctness of the core algorithms of time-triggered communication protocols (e.g., for clock synchronization) is preserved.

4) Application Software – Probe Jobs: The jobs in the experimental setup are named *probe jobs*, because they stress the communication system in a controlled manner with predefined message send patterns and observe the resulting message receptions. The transmission of messages follows a *message send pattern*, which denotes for each output port and each round, the number and size of the messages that have to be transmitted. A probe job includes in each sent message a global timestamp of the send instant and a sequence number.

In addition, a probe job stores significant parameters of received messages (e.g., send and receive instant w.r.t. the global time base, sequence number), thus enabling an off-line analysis and an evaluation of the temporal properties of the VNs under different message patterns. A probe job polls upon each invocation all input ports for received messages. For each message read from an input port, the probe jobs constructs a *measurement record*:

$\langle$send instant, msg. sequ. number, receive instant, job id.$\rangle$

The timestamp with the send instant and the message sequence number are part of the message content. The timestamp of the receive instant is determined by the probe job upon the reception of the message. The job identification denotes at which input port the message has been acquired. The job identification is expressed using the architecture-level namespace (n:$s.v.j$) and denotes the sender job along with the respective VN and node hosting the sender job.

B. Experiment Control Workstation and Management Network

The *experiment control workstation* (PC with Linux as its operating system) controls the experiments and imposes the following experimental sequence:

- *Reset of cluster.* The workstation resets the cluster every 90 sec. using a programmable power supply (HP E3631A) in order to start with defined initial state in each test run.
- *Generation of message patterns.* At the beginning of each test run, the workstation runs a *test pattern generator* that constructs control structures containing test patterns for the probe jobs. This tool uses the index of the test run and the identification of the campaign as parameters in order to create test patterns with a specific bandwidth consumption. The message model behind the test patterns is described in Section VI-C.
- *Transfer of message patterns.* The generated test patterns are transferred via Network File System (NFS) to the nodes. This transfer occurs via the *management network*, which is a standard Ethernet network (i.e., non real-time). On each node, the probe jobs load the test patterns of the current test run during startup.
- *Acquisition and storage of measurement records.* The probe jobs store the acquired measurement records in a file located in a directory of the workstation mounted via NFS. This back-transfer of the collected measurement records to the experiment control workstation uses the management network.

VI. Hypotheses and Experiments

The experiments serve the purpose of evaluating the encapsulation of VNs in the temporal domain and the meeting of

performance requirements for the targeted application-domain. These properties are key elements in order to enable the shift from federated to integrated architectures, while preserving fault isolation, complexity management, and system integration benefits of a federated solution in the integrated architecture.

Encapsulation is investigated w.r.t. different behaviors of jobs. We determine whether the application software comprising a job can (via its message transmissions) affect the temporal properties of messages exchanged by other jobs. In a first step, we are interested in the temporal effects under correct job behaviors. In general, the transmission behavior of a job transmitting event messages will adhere to an envelope that constrains the minimum and maximum interarrival times of messages. This envelope also constrains the job's load on the communication system.

In a second step, we look at the temporal effects of faulty job behaviors, e.g., a job that transmits messages with a total bandwidth consumption exceeding the specified bandwidth. According to the DECOS fault hypothesis [3], a job forms a fault-containment region w.r.t. software faults. Thus, a particular software fault is restricted in its immediate impact to a single job. In the context of mixed-criticality systems, which are a major target domain of the DECOS architecture, it is vital to avoid error propagation via the shared communication system to other jobs.

We focus on the encapsulation of message transmissions in the VN middleware, which manages the communication resources provided by the underlying time-triggered communication network. The evaluation of the encapsulation for hardware faults, such as a node failure or a failure of the underlying time-triggered network (e.g., due to EMI) is not part of the experiments, because the handling of hardware faults through time-triggered communication protocols has already been evaluated extensively in previous work [41]. The experiments performed in this work are orthogonal to these hardware fault injection activities. In conjunction with such a dependable time-triggered network (e.g., TTP [39], FlexRay [40], fault-tolerant Time-Triggered Ethernet [38]), the presented VNs also provide resilience against hardware faults.

The analysis of the performance of the VNs aims at quantifying the effect of temporal partitioning on the bandwidth and latencies of exchanged messages. As part of the experiments, we want to answer the question whether rigid temporal partitioning is achievable, while meeting the performance requirements imposed by present-day automotive applications and those envisioned for the future (e.g., X-by-wire).

A. Hypotheses

The following two hypotheses have been evaluated in the experiments:

a) ***Hypothesis 1 (Temporal Partitioning)****:* A VN provides predefined temporal properties (bandwidth, latencies). Hypothesis 1 consists of the following two subclaims:

Subclaim 1.1 (Temporal partitioning between DASs): The message transmissions of the jobs in the VN of one DAS do not affect the temporal properties (bandwidth, latencies, variability of latencies) of messages exchanged in VNs of other DASs.

Subclaim 1.2 (Temporal partitioning within the DAS): The temporal properties (bandwidth, latencies, variability of latencies) of messages transmitted by a job are independent from the message transmissions of other jobs in the same DAS.

Network Class	Examples of Protocols	Bandwidth	Typical Latencies	Application Domains
Class A	LIN	< 10 kbps	10-100ms	sensor/actuator
Class B	CAN	10kbps-125kbps	10-100ms	comfort domain
Class C	CAN	125kbps-1Mbps	5ms	powertrain domain
Class D	Byteflight	> 1 Mbps	5ms	multimedia, X-by-wire

Fig. 6. SAE Network Classes

b) ***Hypothesis 2 (Performance)****:* Multiple VNs can be established on top of a time-triggered physical network to meet the performance requirements (i.e., bandwidth, latencies) of present day automotive networks.

Based on the performance four classes of in-vehicle networks can be distinguished (see Figure 6) according to the Society of Automotive Engineers (SAE) [42]. In present-day luxury cars, networks belonging to all four classes can be found. For example, in the BMW 7 series [28] two class B networks (peripheral CAN and body CAN) interconnect the nodes of the comfort domain. A class C network (powertrain CAN with 500 kbps) serves as the communication infrastructure of the powertrain domain. In addition, the BMW 7 series is equipped with class D networks for multimedia (Media Oriented Systems Transport [43]) and safety functions (Byteflight [44]). LIN fieldbus networks [45] for accessing low-cost sensors/actuators belong to SAE class A. Similarly, the communication architecture of the Volkswagen Phaeton comprises LIN fieldbus networks, two class B CAN networks for the comfort domain, a class C CAN network for drivetrain, and a class D network for multimedia [29].

Hypothesis 2 consists of the following two subclaims, which are specified w.r.t. to the four network classes in Figure 6:

Subclaim 2.1 (Minimum Bandwidth): This subclaim states that the VNs on top of the underlying time-triggered physical network can provide the bandwidth of two class B networks (e.g., for comfort domain), one class C network (e.g., for powertrain domain), and two class D networks (e.g., for multimedia and X-by-wire). Class A networks are not addressed in the claim, because low-cost fieldbus networks (e.g., LIN) are assumed to remain as separate physical networks despite the shift to an integrated architecture [4].

Subclaim 2.2 (Maximum Latencies): The maximum latencies of the VNs must be 10 ms to 100 ms in the body domain (class B networks) and in the order of ms in the chassis domain (class C networks) [29]. For safety functions realized with class D networks, a reaction time of 5 ms is required [28].

B. Virtual Network Configuration

The VN configuration, which has been used in the experimental evaluation, is based on the federated communication architecture of a typical present day automotive system. The VN configuration, which is depicted in Figure 7, enables the transition from a federated to an integrated architecture with multiple VNs. Each VN serves as a substitute for a respective physical network of a federated system. The VNs belong to five DASs, which are named according to the introduced architecture-level namespace with criticality domain 0 (non safety-critical) and criticality domain 1 (safety-critical).

For meeting the bandwidth requirements identified in hypothesis 2, the VN configuration supports two class B networks, one class C network, and two class D networks (see Figure 8).

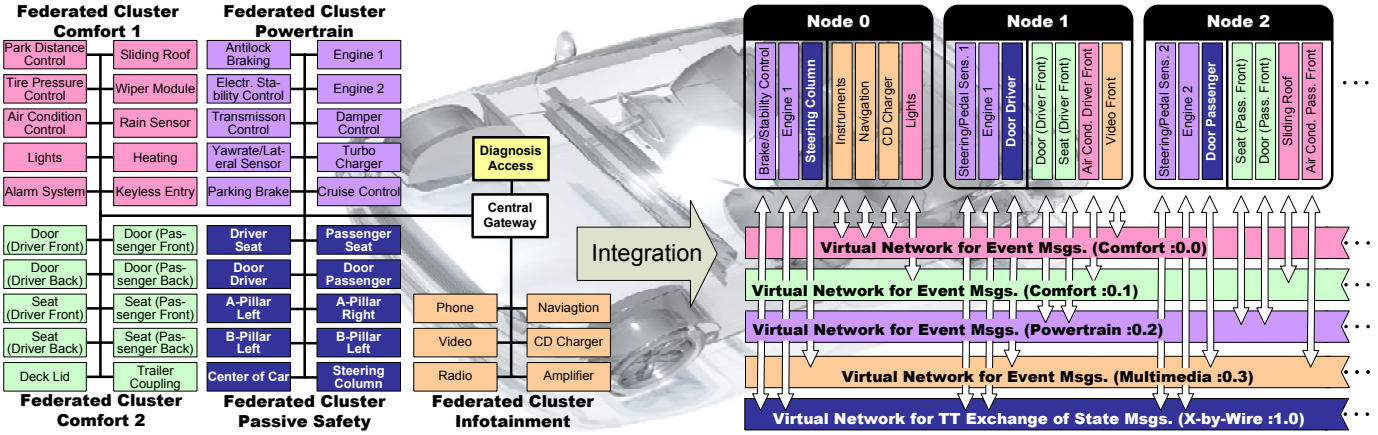


Fig. 7. Mapping Physical Networks to Virtual Networks in the Integrated Architecture

Virtual Network	Node 0:	Node 1:	Node 2:	Node 3:	Node 4:
Comfort :0.0	:0.0.0 (117kbps)	:0.0.1 (117kbps)	:0.0.2 (117kbps)	:0.0.3 (117kbps)	:0.0.4 (117kbps)
	:0.0.5 (117kbps)	:0.0.6 (117kbps)	:0.0.7 (117kbps)	:0.0.8 (117kbps)	:0.0.9 (117kbps)
Comfort :0.1	:0.1.0 (117kbps)	:0.1.1 (117kbps)	:0.1.2 (117kbps)	:0.1.3 (117kbps)	:0.1.4 (117kbps)
	:0.1.5 (117kbps)	:0.1.6 (117kbps)	:0.1.7 (117kbps)	:0.1.8 (117kbps)	:0.1.9 (117kbps)
Powertrain :0.2	:0.2.0 (492kbps)	:0.2.1 (492kbps)	:0.2.2 (492kbps)	:0.2.3 (492kbps)	:0.2.4 (492kbps)
	:0.2.5 (492kbps)	:0.2.6 (492kbps)	:0.2.7 (492kbps)	:0.2.8 (492kbps)	:0.2.9 (492kbps)
Multimedia :0.3	:0.3.0 (1496kbps)	:0.3.1 (1496kbps)	:0.3.2 (492kbps)	:0.3.3 (492kbps)	:0.3.4 (492kbps)
X-by-Wire :1.0	:1.0.0 (617kbps)	:1.0.0 (617kbps)	:1.0.0 (617kbps)	:1.0.0 (617kbps)	:1.0.0 (617kbps)
	:1.0.5 (617kbps)	:1.0.5 (617kbps)	:1.0.5 (3117kbps)	:1.0.5 (3117kbps)	:1.0.5 (3117kbps)

Fig. 8. Virtual Network Configuration

Two VNs with a bandwidth of 117 kbps (named comfort :0.0 and comfort :0.1 after the respective DASs) support on-demand event message exchanges, e.g., like the medium-speed CAN networks deployed in the comfort subsystem of a car. A VN with a bandwidth of 492 kbps (named powertrain :0.2) provides the communication infrastructure of the powertrain subsystem of a car, thus replacing a high-speed CAN network.

Since CRC checks are handled by the underlying time-triggered communication protocol, the net bandwidths of the VNs exceed the net bandwidths of physical CAN networks with a raw bandwidth of 125 kbps or 500 kbps respectively. A further difference is that in a physical CAN network, the total bandwidth is potentially available to every node, but must be shared between them. The VN, on the other hand, assigns TDMA slots with their corresponding bandwidth exclusively to the jobs. In order to provide to each individual job the same maximum bandwidth in the prototype implementation, the entire VN is allocated the tenfold bandwidth (1170 kbps or 4920 kbps) in case of the 10 jobs in the VN configuration. Consequently, the 117 kbps or 492 kbps are simultaneously available to all jobs. This allocation is possible, because the underlying physical Ethernet network provides a higher raw bandwidth (i.e., 100 Mbps) compared to the physical CAN networks. Due to bit arbitration, a physical CAN network is limited to 1 Mbps at a length of 40m [8].

However, if the bandwidth is never simultaneously required by all jobs, sharing of the bandwidth between the jobs as in a physical CAN network would be more efficient. From this point of view, the higher bandwidth usage on the physical Ethernet network represents an overhead. In order to reduce this overhead, a priori knowledge concerning the communication behavior of the jobs can be used to assign to each job only the required fraction of the overall bandwidth. This means that the size of the job slots in the TDMA scheme would be reduced.

In addition to the VNs that serve as replacements for physical CAN networks, the configuration also contains a VN (multimedia :0.3) for the multimedia domain with a bandwidth of 492 kbps or 1496 kbps (depending on the job). A non-uniform bandwidth allocation is chosen, since some jobs may only transmit audio information, while other jobs also transmit audio and video information. Finally, the configuration includes a VN (X-by-wire :1.0) for the time-triggered exchange of state messages as required for safety-critical application subsystems.

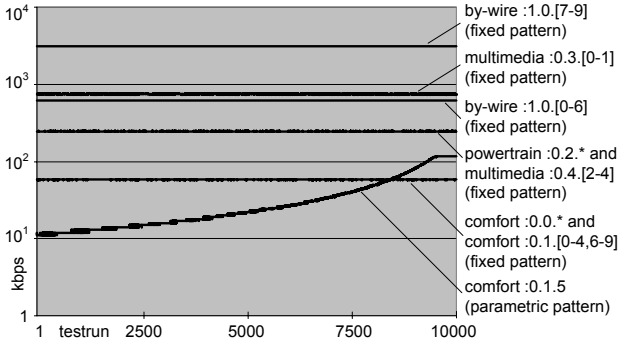
C. Experiments

Test patterns define the transmission behavior of the probe jobs w.r.t. a corresponding message model. In the following, test patterns for sporadic and periodic messages will be explained.

1) Test Patterns: Test patterns control the transmission requests issued by the probe jobs as described in Section V-A.4. Each port is provided with a test pattern that controls the message transmissions performed by the probe job possessing the port. We distinguish between parametric and fixed test patterns:

- *Fixed test pattern.* The fixed test pattern exhibits the same predefined bandwidth utilization in all test runs.
- *Parametric test pattern.* A parametric test pattern exhibits a different bandwidth utilization in each test run. Within any specific test run, the bandwidth utilization in the parametric test pattern is constant. From the point of view of the entire experimental evaluation (i.e., all test runs), a parametric test pattern employs a variable bandwidth utilization, which is constrained by upper and lower bandwidth limits. The index of the current test run determines the bandwidth utilization. For example, starting with a lower bandwidth in the first test run, the bandwidth utilization is increased until reaching the upper limit for the bandwidth consumption in the last test run.

In the experiments, we have assigned a parametric test pattern to job :0.1.5, i.e., one of the jobs of DAS :0.1 (comfort). All other jobs have been assigned fixed test patterns. Job :0.1.5 with the parametric test pattern allows us to explore the effects of message transmissions with varying message interarrival times on the temporal behavior of VNs, both at the same port (i.e., the port with the parametric test pattern) and at other ports (i.e., ports with fixed test patterns).



(a) Observed Net Bandwidth

Virtual Network	Latency in Sporadic and Periodic Campaigns at Ports with Fixed Test Patterns (in ms)									
	Node 0:		Node 1:		Node 2:		Node 3:		Node 4:	
Comfort :0.0	:0.0.0	-	:0.0.1	4ms	:0.0.2	3ms	:0.0.3	3ms	:0.0.4	2ms
	:0.0.5	4ms	:0.0.6	4ms	:0.0.7	3ms	:0.0.8	3ms	:0.0.9	2ms
Comfort :0.1	:0.1.0	-	:0.1.1	4ms	:0.1.2	3ms	:0.1.3	3ms	:0.1.4	2ms
	:0.1.5	6ms	:0.1.6	4ms	:0.1.7	3ms	:0.1.8	3ms	:0.1.9	2ms
Powertrain :0.2	:0.2.0	-	:0.2.1	2ms	:0.2.2	2ms	:0.2.3	2ms	:0.2.4	2ms
	:0.2.5	4ms	:0.2.6	2ms	:0.2.7	2ms	:0.2.8	2ms	:0.2.9	2ms
Multimedia :0.3	:0.3.0	-	:0.3.1	1ms	:0.3.2	3ms	:0.3.3	2ms	:0.3.4	2ms
X-by-Wire :1.0	:1.0.0	-	:1.0.0	2ms	:1.0.0	2ms	:1.0.0	3ms	:1.0.0	3ms
	:1.0.5	2ms	:1.0.5	2ms	:1.0.5	2ms	:1.0.5	3ms	:1.0.5	3ms

(b) Maximum Latencies

Fig. 9. Observed Net Bandwidth and Maximum Latencies in Sporadic Campaign

2) Message Timing Model for Test Patterns: In the experiments, each job is assigned one or more messages that are repeatedly sent by the job. A job requests the dissemination of a message m at points in time $\rho_{k,m} \in \mathbb{R}^+$ $(k \in \mathbb{N})$, where $\rho_{k,m}$ are stochastic variables. The parameter m identifies the message as well as the corresponding job, because each message is sent exclusively by a single job. The parameter k counts the instances of the message at that job. Every message m is characterized by two parameters, a minimum interarrival time $d_m \in \mathbb{R}^+$ and a random interval offset $\delta_{k,m} \in \mathbb{R}^+$.

$$\forall k \in \mathbb{N}: \quad \rho_{k+1,m} - \rho_{k,m} = d_m + \delta_{k,m}$$

d_m specifies an a priori known minimum interval of time between two transmission requests of m. The stochastic variables $\delta_{k,m}$ cover the random part in the time interval between two transmission requests of m. For a particular message m, we assume that all stochastic variables $\delta_{k,m}$ possess a uniform distribution $U(0, u_m)$.

This message model allows different message types to be distinguished. For a *sporadic message* the transmission request instants are not known, but it is known that a minimum time interval exists between successive transmission requests ($\forall k\ \delta_{k,m} \sim U(0, u_m)$, $d_m \in \mathbb{R}^+$). A *periodic message* has a constant time interval between successive message transmission requests ($\forall k\ \delta_{k,m} = 0$, $d_m \in \mathbb{R}^+$).

3) Campaigns: The experiments consist of two campaigns each comprising 10,000 test runs. Campaign I employs a test pattern consisting of sporadic messages with varying minimum interarrival times, while campaign II explores the effects of different frequencies in periodic message exchanges. The reason for using two distinct campaigns is the evaluation of the effectiveness of the partitioning mechanisms for different types of messages (i.e., sporadic messages with random behavior and periodic messages with high predictability). Each of the two campaigns covers both correct and faulty job behaviors. The correct job behavior comprises message transmissions with a bandwidth consumption below the specified bandwidth limit of the VN (cf. Figure 8). An incorrect job behavior occurs when a job transmits more messages than permitted by the specified bandwidth constraints. The latter case represents a design fault, which is either a message timing failure of the job or a misconfiguration of the VN.

a) Campaign I – Sporadic Messages: Campaign I of the experiments comprises sporadic messages. One test pattern is parametric, while all other test patterns are fixed. In the fixed test pattern, d_m and u_m are selected for a 50% utilization of the bandwidth assigned to the port. In the parametric test pattern, d_m varies between 166 μs and 3.33 ms, while u_m varies between 333 μs and 6.66 ms. 10,000 sporadic messages perform an equidistant division of both parameter ranges $(0 \leq n < 10000)$:

$$d_m = 3330\mu s - (n/9999) \cdot (3330\mu s - 166\mu s)$$
$$u_m = 6660\mu s - (n/9999) \cdot (6660\mu s - 333\mu s)$$

b) Campaign II – Periodic Messages: Periodic messages are used in campaign II of the experiments. In analogy to the previous campaign, all but one test pattern are fixed with d_m selected for a 50% utilization of the bandwidth assigned to the port. The parametric test pattern varies the fixed delay between 333 μs and 6.66 ms, while the random delay always equals 0:

$$d_m = 6660\mu s - (n/9999) \cdot (6660\mu s - 333\mu s)\ ,\ 0 \leq n < 10^4$$

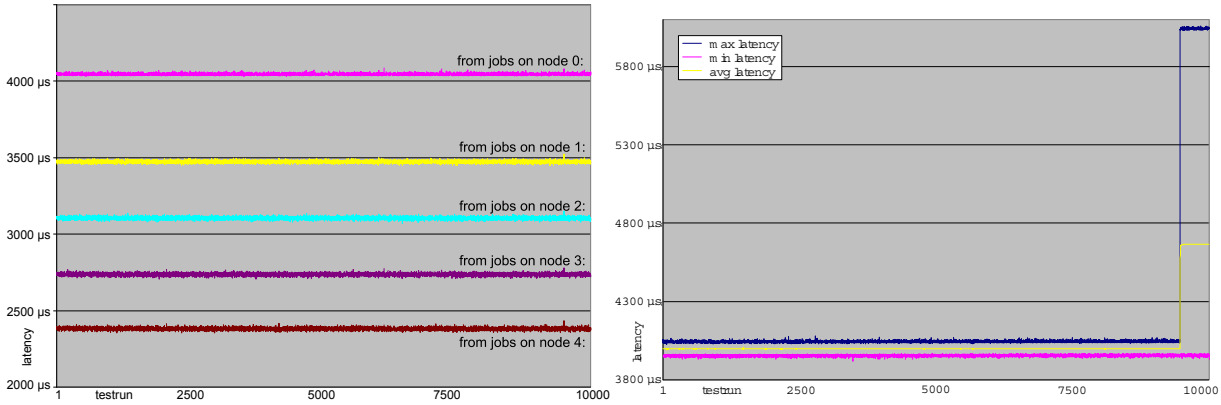
VII. Results

This section describes the results of the experiments. The observed bandwidths and transmission latencies for the parametric and fixed test patterns are presented. In addition, information concerning message omissions in the experiments is provided.

A. Bandwidth

Figure 9(a) depicts the observed net bandwidth in the test runs of campaign I. Along the horizontal axis of the diagram, the test runs are distinguished. Along the vertical axes, the net bandwidth of a test run is depicted (averaged over the duration of the complete test run). For all ports with fixed test patterns, the random interarrival times of the sporadic messages result in a variability ($< 5\%$) of the observed net bandwidth in different test runs. However, this variability results only from the local differences in the random intervals between the sporadic message transmissions. As shown in Figure 9(a), the bandwidth variability does not change as the bandwidth consumption of job $:0.1.5$ increases. The initial bandwidth consumption of job $:0.1.5$ in test run 1 is 11.7 kbps. In test run 10,000, job $:0.1.5$ would require 234.6 kbps, but is confined to the pre-configured bandwidth of 117 kbps.

For the second campaign with periodic test patterns, the observed net bandwidths have exhibited the same characteristic progression as for the sporadic test patterns. However, in the second campaign no bandwidth variability has occurred at the ports with the fixed test patterns due to the absence of random interarrival times.



(a) Ports with Fixed Test Patterns: *max. latencies observed by job* 0:0.0.0

(b) Port with Parametric Test Pattern: *max., avg., and min. latencies observed by job* 0:0.1.0

Fig. 10. Latencies (y-axis denotes latency, x-axis denotes test run)

B. Latencies

Figure 9(b) gives an overview about the observed maximum (end-to-end) latencies of the messages exchanged at each port with a fixed test pattern. The maximum latencies have been computed over all 10,000 test runs and were identical for campaigns I and II. The major reason for the identical maximum latencies in both campaigns is that except for the parametric test patterns, the messages produced by a job could always be transmitted using the subsequent job slot. Also, the periods of the periodic messages were not multiples or factors of the TDMA round length, thus leading to variable delays (like for the sporadic messages) between a job's message transmission requests and the dissemination through the underlying time-triggered network.

Important parameters resulting in these message transmission latencies on the VNs are the time-triggered activation times of the probe jobs in conjunction with the statically predefined slots for the exchange of messages on the underlying time-triggered network. The communication schedule employed in the prototype implementation consists of five slots for the five nodes. Each probe job is scheduled in a communication slot, i.e., synchronized with the communication schedule. Therefore, we can use a slot index ($0 \ldots 4$) from the beginning of the round start to capture the following three parameters. The latencies depend on the phase shift relationship between the slot at which the sending probe job is scheduled (n^s_{pb}), the slot at which the receiving probe job is scheduled (n^r_{pb}), and the sender's slot n^s_{slot} in the TDMA scheme.

Based on these parameters, we can define two delays (d_{snd}, d_{rcv}) that contribute to the latency of a message exchanged on a VN. At the sending node, the *sender delay* d_{snd} results from the delay between the activation of the probe job and the start instant of the respective node's slot in the TDMA scheme. The receiver delay d_{rcv} is the delay between the start instant of the sending node's slot in the TDMA scheme and the activation of the receiving probe job. In the chosen VN configuration, each node sends exactly once in each TDMA round, thus a message transmission of a probe job is delayed until the following slot of the node that hosts the probe job. Formally, the sender and receiver delays are as follows:

$$d_{\text{snd}} = (1+((n^s_{\text{slot}} - n^s_{\text{pb}} - 1 + 5) \mod 5)) \cdot d_{\text{slot}}$$
$$d_{\text{rcv}} = (1+((n^r_{\text{pb}} - n^s_{\text{slot}} - 1 + 5) \mod 5)) \cdot d_{\text{slot}}$$

In the above formulas, the summand -1 (inside the modulo expression) results from the need to sample the exchanged state messages at the end of the slot before the actual sending slot or the execution slot of the probe job. The summand 1 represents the delay for execution of the probe job. d_{slot} is the duration of a TDMA slot (i.e., 400 μs in the implementation).

In order to exemplify the sender and receiver delays in the implementation, Figure 11 depicts the execution slots for the probe jobs of selected DASs (:0.0, :0.2, and :0.3). For example, the transmission of messages on the VN of DAS :0.0 from node 0.1: to node 0.0: results in the parameters $n^s_{\text{pb}} = 1$, $n^s_{\text{slot}} = 1$, and $n^r_{\text{pb}} = 0$. The resulting sum of the sender and receiver delay is 3.6 ms, which matches the value observed in the implementation. Likewise, the transmission of messages on the VN of DAS :0.2 from node 2: to node 0: results in the parameters $n^s_{\text{pb}} = 2$, $n^s_{\text{slot}} = 0$, and $n^r_{\text{pb}} = 0$. In this case, the sum of the sender and receiver delay is 2 ms.

In addition, we have captured the progression of the latencies as the bandwidth consumption of the port with the parametric test pattern increases. Figure 10(a) depicts the latencies of one of the VNs in the 10,000 test runs for campaign I. As the bandwidth consumption of the port with the parametric test pattern at job :0.1.5 increases from 11.7 kbps to 234.6 kbps, no change in the maximum message transmission latencies at the ports with fixed test patterns is observable. The interval of observed message transmission latencies depends on the sender, with $[4033\,\mu s, 4086\,\mu s]$ for node 0, $[3449\,\mu s, 3522\,\mu s]$ for node 1, $[3070\,\mu s, 3151\,\mu s]$ for node 1, $[2706\,\mu s, 2777\,\mu s]$ for node 2, and $[2355\,\mu s, 2435\,\mu s]$ for node 3. No correlation has been recognized between the bandwidth consumption of job :0.1.5 and the observed latencies at other jobs.

For the port with parametric test patterns, on the other hand, the bandwidth consumption beyond the specified bandwidth limit of 117 kbps in test runs 9,471 to 10,000 results in a variability of the latencies over the test runs. Maximum latencies of 6.1 ms have been observed by the jobs of DAS :0.1 in test runs with message transmissions exceeding the bandwidth of the VN (see Figure 10(b)). In these test runs, messages accumulate in message queues, thus requiring multiple communication rounds for the exchange on the VN. The maximum latency is 4 ms before test run 9,471, when the bandwidth consumption is below 117 kbps.

	Slot 0	Slot 1	Slot 2	Slot 3	Slot 4
Node 0:	:0.0				
Node 1:		:0.0			
Node 2:			:0.0		
Node 3:				:0.0	
Node 4:					:0.0

	Slot 0	Slot 1	Slot 2	Slot 3	Slot 4
Node 0:	:0.2				
Node 1:	:0.2				
Node 2:	:0.2				
Node 3:	:0.2				
Node 4:					:0.2

	Slot 0	Slot 1	Slot 2	Slot 3	Slot 4
Node 0:			:0.3		
Node 1:					:0.3
Node 2:		:0.3			
Node 3:				:0.3	
Node 4:				:0.3	

Fig. 11. Time-Triggered Scheduling of Probe Jobs

C. Message Omissions

For all ports with fixed test patterns no message omissions as indicated by the sequence numbers included in the messages have been observed. At the ports with parametric test patterns, message omissions were observed in case of transmission requests exceeding the pre-configured VN bandwidth (i.e., test runs 9,471 to 10,000). These test runs represent message timing failures, because job :0.1.5 does not comply to its specified bandwidth limit. No message omissions have been observed for the parametric ports before test run 9,471, i.e., when the bandwidth consumption is below the pre-configured 117 kbps.

VIII. Discussion of Results

The measurement results demonstrate that the communication system for an integrated architecture can be built with rigid temporal partitioning, while also providing competitive temporal performance (bandwidth, latency). The measurements have indicated that varying message loads created by a job do not affect the temporal properties of messages exchanged by other jobs.

A. Hypothesis 1 – Temporal Partitioning

The experiments have demonstrated that the maximum and average transmission latencies of the ports with the fixed test patterns are independent from the behavior at the ports controlled by parametric test patterns. Smaller periods of periodic messages or smaller minimum interarrival times in case of sporadic messages have not resulted in the observation of increased transmission latencies at other ports. Consequently, both subclaims of hypothesis 1 hold. Interference in the transmission latencies of different ports was neither observed within a DAS nor between DASs.

The foundation for the effective temporal partitioning is the design decision of statically subdividing the communication slots according to the physical and logical structuring of the DECOS system (cf. Section III). Based on this static resource allocation, the time-triggered communication protocol can protect the communication resources at the node-level, while the VN middleware protects the communication resources at the level of VNs and jobs.

A drawback of this design decision is a decreased flexibility w.r.t. extensions and modifications of the communication system. For example, in order to support the addition of nodes, either free slots need to be reserved at design time or a new time-triggered schedule needs to be computed and programmed into the nodes. Consequently, system designers need to assess the reduction of flexibility against the advantages of the static resource allocation (e.g., temporal composability, better complexity management, error containment for consequences of software faults).

B. Hypothesis 2 – Performance

The experiments have shown that VNs on top of an underlying time-triggered network provide sufficient performance to support the communication requirements of present day automotive applications (see Section VI-A), as well as future safety-critical time-triggered DASs.

This result is particularly interesting as it demonstrates that temporal partitioning by the complete temporal isolation of DASs and jobs does not preclude competitive performance. Of course, there is a fundamental tradeoff between encapsulation and resource efficiency. Communication protocols, such as CAN, support the global multiplexing of bandwidth between all senders. In contrast to the proposed static resource allocation, global bandwidth multiplexing leads to a more efficient use of the overall bandwidth in case of large communication loads that dynamically vary between senders. However, bandwidth utilization limits (e.g., 50% [46]) are introduced in order to control latencies (probabilistically since the utilization determines only the average load). Furthermore, bit arbitration constrains the maximum bandwidth [8]. VNs do not exhibit these constraints. Neither is the raw bit rate constrained by bit arbitration, nor do bandwidth utilization constraints apply in order to guarantee message latencies.

1) Bandwidth: The configuration in the experimental setup supports two VNs that are equivalent to low-speed CAN networks, a VN equivalent to a high-speed CAN network, a VN for multimedia, and a VN for time-triggered communication. For the VN equivalent to low-speed CAN, each job is provided with a net bandwidth of 117 kbps. Since CRC checks are handled by the underlying time-triggered communication protocol, the net bandwidth of the VN exceeds the net bandwidth of a physical CAN network with a raw bandwidth of 125 kbps. For the VN equivalent to a high-speed CAN network, each job is provided with a net bandwidth of 492 kbps. In analogy to the low-speed VN, the realization of CRCs by the underlying time-triggered communication protocol makes the net bandwidth superior to a physical CAN network with a raw bandwidth of 500 kbps. The net bandwidth of the VN for multimedia (1496 kbps or 492 kbps) enables the playback of video or audio information. In the time-triggered VN, a net bandwidth of 3117 kbps or 617 kbps is available for safety-critical application subsystems.

2) Latencies: Depending on the scheduling of jobs, maximum latencies between 1.5 ms and 4.1 ms have been observed in the test cases with the fixed test patterns. These latencies meet the requirements of typical CAN-based automotive applications (e.g., order of ms in chassis domain, 10 ms to 100 ms in body domain [29]). In particular, these latencies have been unaffected by job :0.5.0 executing the parametric test patterns. The latencies for the fixed test patterns have remained invariant even in case of a timing message failure of job :0.5.0, i.e., a message load exceeding the configured bandwidth. As a consequence of this message timing failure, only the faulty job :0.5.0 itself has experienced increased latencies of 6.1 ms and message omissions.

For the time-triggered message exchanges, a maximum latency of 2.8 ms has been observed, which is suitable for the imple-

mentation of safety-critical X-by-wire applications. For example, see [47] for a discussion of the maximum delays for ensuring the safety in steer-by-wire systems.

IX. Conclusion

This paper has shown that a time-triggered physical network is an effective foundation for establishing multiple virtual networks, each tailored to a respective application subsystem via its control paradigm (event message vs. state messages) and its temporal properties (e.g., bandwidth). The experimental assessment has yielded evidence that the realized VNs exhibit predefined temporal properties for the messages transmitted by a job, independently from the transmission behavior of other jobs and other application subsystems. In particular, rigid temporal partitioning is achievable, while at the same time meeting the performance requirements imposed by present-day automotive applications and those envisioned for the future (e.g., X-by-wire).

These results are particularly important in the context of the increasing complexity of embedded systems. System architects become forced to follow divide-and-conquer strategies that permit a reduction of the mental effort for developing and understanding a large system by partitioning the system into smaller subsystems that can be developed and analyzed in isolation.

The temporal encapsulation of the communication resources belonging to subsystems, such as DASs or jobs in the DECOS architecture, is a key requirement for the constructive integration of integrated computer systems. By ensuring guaranteed temporal properties (e.g., bandwidth, latencies) for the messages transmitted by each job, prior services cannot be invalidated by the behavior of newly integrated jobs at the communication system. This quality of an architecture, which is denoted as temporal composability, relates to the ease of building systems out of subsystems. A system, i.e., a composition of subsystems, is considered temporally composable if the temporal correctness is not invalidated by the integration provided that temporal correctness has been established at the subsystem level.

VNs on top of a time-triggered network support temporal composability by ensuring that temporal properties at the communication system are not invalidated upon system integration. Furthermore, in the context of upcoming time-triggered technology in the automotive domain, the availability of a time-triggered communication network with high bandwidth enables the elimination of some of the physical networks deployed in present day cars. The communication resources of a single time-triggered network can be shared among different DASs. In conjunction with nodes for the execution of application software from different DASs, this integration not only reduces the number of node computers, but also results in fewer connectors and wires.

For future work, additional experiments are suggested as part of the development path towards the exploitation of VNs in ultra-dependable systems such as a drive-by-wire car. The experiments presented in this paper have focused on a single probe job within a selected VN. Interesting scenarios for future experimental evaluations include test cases with multiple probe jobs, which can exhibit simultaneous timing failures and are located in different VNs. Thereby, additional experiments can further increase the confidence in the presented hypotheses w.r.t. fault isolation and performance.

Acknowledgment

This work has been supported in part by the European IST project ARTIST2 under project No. IST-004527 and the European IST project DECOS under project No. IST-511764.

References

[1] Aeronautical Radio, Inc., 2551 Riva Road, Annapolis, Maryland 21401. *ARINC Specification 651: Design Guide for Integrated Modular Avionics*, November 1991.

[2] H. Heinecke 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*. SAE, October 2004. 2004-21-0042.

[3] 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.

[4] P. Peti, R. Obermaisser, F. Tagliabo, A. Marino, and S. Cerchio. An integrated architecture for future car generations. In *Proc. of the 8th IEEE Int. Symposium on Object-oriented Real-time distributed Computing*, May 2005.

[5] 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.

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

[7] F.P. Brooks. No silver bullet: Essence and accidents of software engineering. *Computer*, April 1987.

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

[9] R. Obermaisser and B. Huber. Model-based design of the communication system in an integrated architecture. In *Proc. of the 18th International Conference on Parallel and Distributed Computing and Systems*, pages 96–107, 2006.

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

[11] J. Rushby. Partitioning for avionics architectures: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999. Also to be issued by the FAA.

[12] J. Sifakis. A framework for component-based construction. In *Proc. of 3rd IEEE Int. Conference on Software Engineering and Formal Methods (SEFM05)*, pages 293–300, September 2005.

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

[14] R. Obermaisser, P. Peti, and H. Kopetz. Virtual networks in an integrated time-triggered architecture. In *Proc. of the 10th IEEE Int. Workshop on Object-oriented Real-time Dependable Systems (WORDS2005)*, 2005.

[15] 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.

[16] 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 (ISADS 2003)*, pages 37–44, 2003.

[17] FlexRay Consortium. BMW AG, DaimlerChrysler AG, General Motors Corporation, Freescale GmbH, Philips GmbH, Robert Bosch GmbH, and Volkswagen AG. *Node-Local Bus Guardian Specification Version 2.0.9*, December 2005.

[18] D. Kim, Y.-H. Lee, and M. Younis. SPIRIT – μKernel for strongly partitioned real-time systems. In *Proc. of the 7th Int. Conference on Real-Time Computing Systems and Applications*, 2000.

[19] J. Penix et al. Verification of time partitioning in the DEOS scheduler kernel. In *Proc. of the 22nd International Conference on Software Engineering*, pages 488–497, 2000.

[20] K. Hoyme and K. Driscoll. SAFEbus. *IEEE Aerospace and Electronic Systems Magazine*, 8:34–39, March 1993.

[21] E. Totel, J. P. Blanquart, Y. Deswarte, and D. Powell. Supporting multiple levels of criticality. In *Proc. of the The 28th Annual International Symposium on Fault-Tolerant Computing*, page 70, Washington, DC, USA, 1998.

[22] 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 (ETFA)*, Catania, Italy, September 2005. IEEE.
[23] *Software Fundamentals: Collected Papers by David L. Parnas*. Addison-Wesley, April 2001.
[24] H.D. Heitzer. Development of a fault-tolerant steer-by-wire steering system. *Auto Technology*, 4:56–60, April 2003.
[25] 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.
[26] Radio Technical Commission for Aeronautics, Inc. (RTCA), Washington, DC. *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, December 1992.
[27] H.A. Simon. *The Sciences of the Artificial*. MIT Press, 1996.
[28] A. Deicke. The electrical/electronic diagnostic concept of the new 7 series. In *Convergence Int. Congress & Exposition On Transportation Electronics*, Detroit, MI, USA, October 2002. SAE.
[29] J. Leohold. Communication requirements for automotive systems. In *Keynote Automotive Communication – 5th IEEE Workshop on Factory Communication Systems*, Vienna, Austria, September 2004.
[30] J. Rushby. A comparison of bus architectures for safety-critical embedded systems. Technical report, SRI Intern., 2001.
[31] R. DeLine. *Resolving Packaging Mismatch*. PhD thesis, Carnegie Mellon University, Pittsburgh, June 1999.
[32] DECOS. Project deliverable D2.2.3 – Virtual communication links and gateways. Technical report, 2006.
[33] F. Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, 1991.
[34] L. Kleinrock. *Queuing Systems Volume I: Theory*. John Wiley and Sons, New York, 1975.
[35] J. Ferreira, P. Pedreiras, L. Almeida, and J. Fonseca. Achieving fault tolerance in FTT-CAN. In *Proc. of the 4th IEEE Int. Workshop on Factory Communication Systems*, 2002.
[36] R.W. Butler, J.L. Caldwell, and B.L. Di Vito. Design strategy for a formally verified reliable computing platform. In *Proc. of the 6th Annual Conference on Systems Integrity, Software Safety and Process Security*, pages 125–133, June 1991.
[37] B. Dion. Correct-by-construction methods for the development of safety-critical applications. In *SAE 2004 World Congress & Exhibition*, Detroit, MI, USA, March 2004. SAE.
[38] 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.
[39] H. Kopetz and G. Grünsteidl. TTP – A protocol for fault-tolerant real-time systems. *Computer*, 27(1):14–23, 1994.
[40] FlexRay Consortium. *FlexRay Communications System Protocol Specification Version 2.1*, May 2005.
[41] 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.
[42] G. Leen and D. Heffernan. Expanding automotive electronic systems. *Computer*, 35(1):88–93, January 2002.
[43] MOST Cooperation, Karlsruhe, Germany. *MOST Specification Version 2.2*, November 2002.
[44] J. Berwanger and M. Peller R. Griessbach. Byteflight a new protocol for safety critical applications. In *Proc. of the FISITA World Automotive Congress*, Seoul, 2000.
[45] Audi AG, BMW AG, DaimlerChrysler AG, Motorola Inc., Volcano Communication Technologies AB, Volkswagen AG, and Volvo Car Corporation. LIN specification and LIN press announcement. *SAE World Congress Detroit*, 1999.
[46] A. Albert and W. Gerth. Evaluation and comparison of the real-time performance of CAN and TTCAN. In *Proc. of 9th Int. CAN Conference*, Munich, 2003.
[47] C. Wilwert, Y. Song, F. Simonot-Lion, and T. Clement. Evaluating quality of service and behavioral reliability of steer-by-wire systems. In *Proc. of 9th IEEE Int. Conference on Emerging Technologies and Factory Automation*, 2003.

Roman Obermaisser received the Dipl. degree and the Ph.D. degree in computer science from the Vienna University of Technology, Vienna, Austria. He is currently assistant professor in the Real-Time Systems Group at the Vienna University of Technology. His research interests include system architectures for distributed embedded real-time systems, communication protocols, and automotive electronic systems. His current research work focuses on the design, implementation, and validation of a time-triggered network-on-a-chip.