Integrating Automotive Applications using Overlay Networks on top of a Time-Triggered Protocol

Roman Obermaisser[1]

Vienna University of Technology, Austria
romano@vmars.tuwien.ac.at

Abstract. The integration of multiple automotive subsystems (e.g., powertrain, safety, comfort) on a single distributed computer system can significantly reduce the number of Electronic Control Units (ECUs) and networks for in-vehicle electronic systems. The benefits of this integration include reduced hardware cost and reliability improvements due to fewer connectors. However, a major challenge in such an integrated automotive architecture is the management of access to the shared communication resources (i.e., the common network). In order to support a seamless integration of application subsystems from different vendors and to permit the integration of application subsystems with different criticality levels, a fault in one application subsystem should not have an adverse affect on the resources that are available to other application subsystems. For this reason, we devise a solution for encapsulating the communication activities of application subsystems in this paper. Each application subsystem is provided with a dedicated overlay network on top of an underlying time-triggered network. Such an overlay network has predefined temporal properties (i.e., latencies, bandwidths), which are independent from the communication activities on the overlay networks of other application subsystems. An exemplary configuration of the overlay networks in a prototype implementation demonstrates that the encapsulated overlay networks can handle the communication load of a present day car with the additional time-triggered traffic of future X-by-wire subsystems.

1 Introduction

A steady increase in automotive electronics has occurred during the past years. While this trend has lead to significant improvements concerning safety and comfort, a side-effect has been a growth of the deployed in-vehicle hardware and software. In conjunction with the prevalent "1 Function – 1 ECU" strategy [1], present day luxury cars can contain more than 70 ECUs and multiple networks with different communication protocols (e.g., CAN, MOST, ByteFlight) [2]. Furthermore, the increase in automotive electronics is likely to continue its growth due to customers expectations. Cars are no longer simple means of transportation but rather need to convince customers with respect to design, performance, driving behavior, safety, infotainment, comfort, maintenance, and cost.

To circumvent the increase of ECUs and networks, the automotive industry is currently evolving towards integrated system architectures [3, 4]. Integrated system architectures promise a reduction of ECUs, networks, and connectors by using the ECUs and networks as shared resources for multiple application subsystems. Each ECUs can execute jobs from multiple application subsystems and different vendors. Likewise, a network supports message exchanges of more than one application subsystem.

The challenge in moving from today's federated automotive architectures to this new architectural paradigm is the management of the increasing complexity in the emerging integrated automotive systems. In order to manage this complexity, system architects are forced to follow a divide-and-conquer strategy that enables a reduction of the mental effort for understanding a large system by structuring the system into smaller subsystems that can be developed and analyzed in isolation.

Composability is a concept that refers to the stability of component properties across integration, thus enabling the correctness-by-construction of component-based systems [5]. A system is composable with respect to a particular property, if the integration does not invalidate the property when it has already been established at the subsystem level. For example, *temporal composability* is an instantiation of the general notion of composability. A system is temporally composable, if timeliness is not refuted by the system integration [6]. Temporal composability facilitates the construction of temporally predictable systems, because the temporal properties of subsystems can be analyzed in isolation. Temporal composability can be supported at the communication system by controlling the possible interactions of subsystems in order to prevent unintended side effects. An example for an unintended side effect is resource contention between subsystems on the communication system. Consider for example an exemplary scenario with two application subsystems. If the two application subsystems share a common CAN bus [7], then both application subsystems must be analyzed and understood in order to reason about the correct behavior of any of the two application subsystems. Since the message transmissions of one application subsystem can delay message transmissions of the other application subsystem, arguments concerning the correct temporal behavior must be based on an analysis of both application subsystems.

A federated architecture (e.g., as used in present day automotive systems [8]) trivially rules out unintended side effects by assigning to each application subsystem separate computational and communication resources, i.e., ECUs and networks for the exclusive use by the application subsystem. In an integrated architecture (without encapsulation mechanisms), however, the communication system could extend the inherent complexity of application subsystems with an additional accidental complexity [9] due to integration-induced interference at the communication system. Therefore, the communication system of an integrated architecture should ensure that interactions between subsystems occur only via the specified message-based interfaces.

Motivated by the need to avoid accidental complexity as a side-effect of integration, this paper presents a solution for providing to each application subsystem its own protected communication infrastructure that is free of interference with other application subsystems. The communication infrastructure of an application subsystem is realized as an encapsulated overlay network on top of a time-triggered communication protocol, thus exploiting the upcoming time-triggered communication networks that will be deployed in the automotive domain [10]. Based on the Time Division Multiple Access (TDMA) scheme of a time-triggered communication network (e.g., FlexRay [11], Time-Triggered Protocol (TTP) [12], or Time-Triggered Ethernet (TTE) [13]), temporal partitioning mechanisms ensure predefined temporal properties (i.e., latency, bandwidth) of each overlay network.

The paper is structured as follows. Section 2 describes the construction of time-triggered and event-triggered overlay networks on top of a time-triggered communication network. A prototype implementation based on a TDMA-controlled Ethernet network is presented in Section 3. Section 4 shows an exemplary configuration of the prototype implementation for handling the communication requirements of a future X-by-wire car. An overview of related work is the content of Section 5. The paper concludes with a discussion in Section 6.

2 Overlay Networks on top of a Time-Triggered Physical Network

Based on the requirements and functional coherence of automotive applications, the functionality of the electronic systems aboard a car can be structured into a set of application subsystems (e.g., powertrain subsystem, comfort subsystem, passive safety subsystem, etc.). On its behalf, each application subsystem consists of smaller functional elements called jobs (e.g., in the powertrain subsystem: engine control job, automatic gear job, etc.).

Today, the implementation of automotive electronic systems typically follows the “1 Function – 1 ECU” philosophy [1], where each ECU exclusively hosts a single job from a respective application subsystem. In contrast, we focus on integrated system architectures, where each ECU supports the coexistence of multiple jobs from one or more application subsystems.

In order to provide the communication infrastructure for the exchange of messages between the jobs of an application subsystem, this section describes the construction of event-triggered and time-triggered overlay networks on top of a time-triggered physical network. After defining the required services of the underlying time-triggered communication network, the allocation of communication resources based on a hierarchic subdivision of TDMA slots is explained. Finally, this section explains the exploitation of the communication resources on the time-triggered communication network for the exchange of state and event messages.

2.1 Time-Triggered Physical Network

Time-triggered networks (e.g., SafeBus [14], the Time-Triggered Protocol [12], FlexRay [11]) have become generally preferred for safety-critical systems [15, 16]. For example, in the automotive industry a time-triggered network will provide the ability to handle the communication needs of by-wire cars [17]. In addition to hard real-time performance, time-triggered networks help in managing the complexity of fault-tolerance and corresponding formal dependability models, as required for the establishment of ultra-high reliability (failure rates in the order of 10^{-9} failures/hour). The predetermined points in time of the periodic message transmissions allow error detection and establishing of membership information. Redundancy can be established transparently to applications [18], i.e., without any modification of the function and timing of application systems. A time-triggered network also supports replica determinism [19], which is essential for establishing fault-tolerance through active redundancy.

Since the presented system architecture with its overlay networks targets mixed-criticality applications with application subsystems up to the highest considered criticality class (e.g., level A in RTCA DO-178B [20] or SIL4 in EN ISO/IEC 61508 [21]), we use a time-triggered physical network as the basis for the establishment of the encapsulated overlay networks. The time-triggered physical network provides a *clock synchronization service* in order to establish a global time base. In addition, the time-triggered network offers a *time-triggered message transport service* for the periodic exchange of state message at predefined instants with respect to the global time base. At each ECU the communication controller (e.g., TTP controller C2 [22], FlexRay Controller MFR4200 [23]) provides a memory element with outgoing state messages that are written by the application and read by the communication controller prior to broadcasting them on the time-triggered network. In addition, the memory element contains incoming state messages that are read by the application and updated by the communication controller with state messages read from the time-triggered network (i.e., information broadcast by other ECUs). This memory element, which is denoted Communication Network Interface (CNI) in TTP and Controller Host Interface (CHI) in FlexRay, is provided by most time-triggered networks with syntactic differences of state messages (e.g., header format) and protocol-specific constraints (e.g., only one message sent by an ECU per communication round in TTP [12], same size for all state messages in FlexRay [11]).

2.2 Hierarchic Subdivision of Communication Slots

For the realization of overlay networks, we use the time-triggered physical network and perform a hierarchic temporal subdivision of the communication resources (see Figure 1). The media access control strategy of the time-triggered physical network is TDMA. TDMA statically divides the channel capacity into a number of slots and controls access to the network solely by the progression of time. Each ECU is assigned a unique ECU slot that periodically recurs at a priori specified global points in time. An ECU sends messages during its ECU

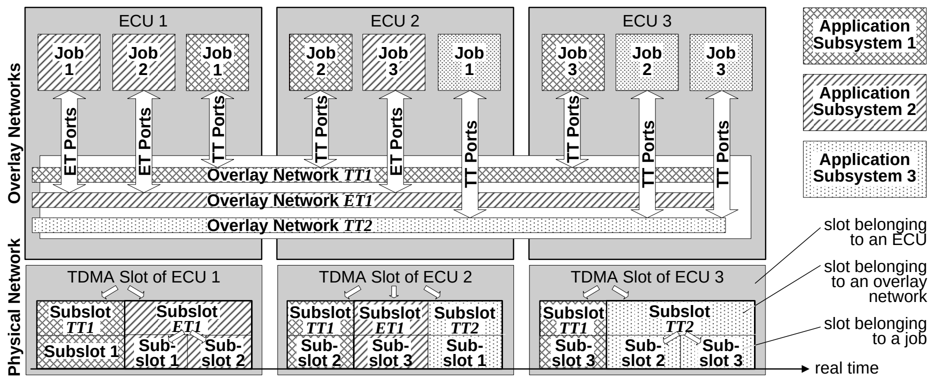

Fig. 1. Hierarchic Subdivision of Communication Resources

slot and receives messages during the ECU slots of other ECUs. A sequence of ECU slots, which allows every ECU in an ensemble of ECUs to send exactly once, is called a TDMA round.

We further subdivide each ECU slot in correspondence to the functional structuring of the overall system. In a first step, the ECU's slot is subdivided into subslots for the overlay networks. Such a subslot contains those messages that are produced by the jobs in the ECU that are connected to a particular overlay network. By using a one-to-one mapping between overlay networks and application subsystems, these subslots are also specific to a particular application subsystem. On its part, a slot belonging to an overlay network consists of smaller subslots belonging to individual jobs. When an ECU hosts multiple jobs of an application subsystem that send messages to the overlay network, then each of these jobs is assigned a corresponding slot carrying the messages sent by that job.

The assignment of the slots within a TDMA round to ECUs, as well as the further subdivision into slots for overlay networks and jobs is fixed at design time. This static allocation ensures that the network resources are predictably available to jobs. In particular, this static strategy facilitates complexity management, because for understanding the behavior of a job, the consumption of network resources by other jobs need not be considered.

2.3 Time-Triggered and Event-Triggered Overlay Networks

An overlay network is a network which is built on top of another network. For the Internet, several solutions for overlay networks have been designed in the past. For example, Virtual Private Networks (VPNs) [24] have been realized in order to improve security. Another example of overlay networks is the resilient overlay network described in [25], which aims at improving robustness of Internet applications in the presence of path outages and periods of degraded performance.

In this paper, we build overlay networks for encapsulating the communication activities of application subsystems in an integrated embedded system. When moving from a federated to an integrated architecture, each overlay network serves as a substitute for a respective physical network.

We distinguish between two fundamentally different types of overlay networks: event-triggered and time-triggered overlay networks. A *time-triggered overlay network* is designed for the periodic exchange of state messages. The access point between a time-triggered overlay network and a job is a memory element denoted as a *time-triggered port*. As depicted in Figure 2, the sender job acts according to the information push paradigm [26] and writes information into

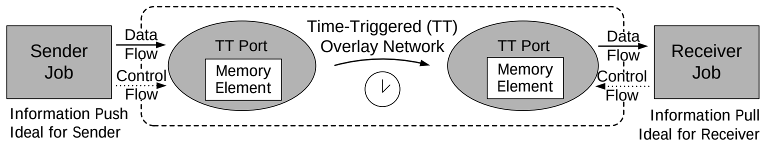

Fig. 2. Message Exchange between two Jobs through a Time-Triggered Overlay Network

the memory element at its output port (update-in-place). The receiver job must pull information out of the input port by reading the memory element in the input port. Using the job slot of the sender, the time-triggered overlay network autonomously carries the state information from the memory element of the sender to the memory element(s) of the receiver(s) at a priori determined global points in time. Since no control signals cross the ports, temporal fault propagation is prevented by design. Time-triggered overlay networks employ implicit flow control [27]. A job's ability for handling received messages can be ensured at design time, i.e., without acknowledgment messages. Implicit flow control makes time-triggered overlay networks well-suited for multicast communication relationships, because ports offer elementary interfaces [28], i.e., a unidirectional data flow involves only a unidirectional control flow.

Event-triggered overlay networks are designed for the sporadic exchange of event messages, combining event semantics with external control [27]. In order to support exactly-once processing of event messages, the access point between an event-triggered overlay network and a job is a message queue (denoted as an *event-triggered port*). Thereby, the overlay networks provides bandwidth elasticity. Due to the queues at the output ports, a job can pass more message to the overlay network than can be transmitted in a single TDMA round using the underlying time-triggered network. Overlay networks 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.

The interactions between a sender and a receiver via an event-triggered overlay network are depicted in Figure 3. At the sender side, event messages are

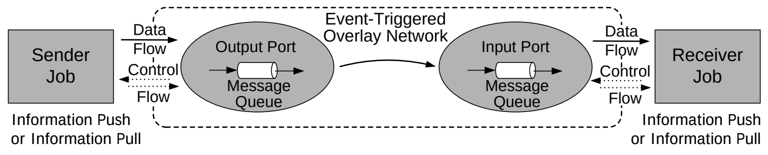

Fig. 3. Message Exchange between two Jobs through an Event-Triggered Overlay Network

inserted into the message queue at the output port via an explicit transmission request from the job (information push with external control) or as a result of the reception of a request message (information pull, e.g., a client/server interaction). The event-triggered overlay network exploits the bandwidth available via the sender's job slot for transporting the event messages to the message queue at the receiver. At the receiver side, the job either fetches the incoming message from the input port (information pull via polling for messages) or the event-triggered overlay network presses received messages into the job (information push via interrupt mechanism).

2.4 Encapsulation of Overlay Networks

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 [29, 30]. 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) [31]), 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.

Providing a dedicated port for each overlay network at all receivers and the reservation of dedicated slots in the underlying TDMA scheme are the two key elements for temporal partitioning of overlay networks.

In case of event ports, *separate ports* and thus separate queues ensure that the queuing delays for messages received from one 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 [32] 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 job.

In addition to providing separate ports, we also need to prevent interference between messages from different senders prior to the arrival at the respective

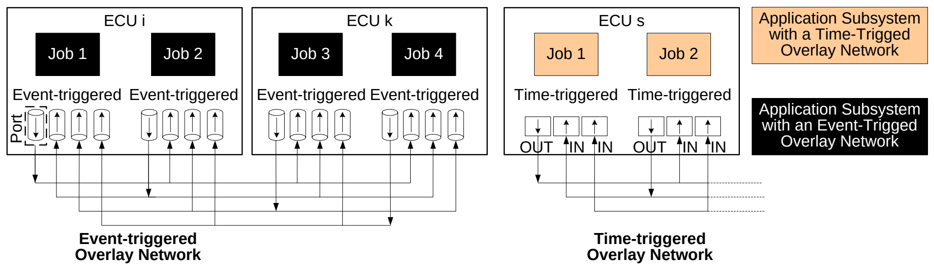
ECU i
Job 1
Job 2
Event-triggered
Event-triggered
Port
ECU k
Job 3
Job 4
Event-triggered
Event-triggered
ECU s
Job 1
Job 2
Time-triggered
Time-triggered
OUT
IN
IN
OUT
IN
IN
Application Subsystem with a Time-Trigged Overlay Network
Application Subsystem with an Event-Trigged Overlay Network
Event-triggered Overlay Network
Time-triggered Overlay Network

Fig. 4. Encapsulation of Overlay Networks

ports. For this purpose, the overlay network performs a separation of application subsystems and jobs via statically reserved slots in the underlying TDMA scheme. 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 (i.e., application subsystems and jobs). While several solutions for the protection of ECU slots on a time-triggered network are already available (e.g., [33–35]), middleware services for the protection of the subslots for application subsystems, and jobs have been realized in the scope of this work.

Due to encapsulation, developers need not look at all possible interactions between jobs and application subsystems in order to understand the temporal behavior of an overlay network. In particular, upon the occurrence of faults of individual jobs and application subsystems, the encapsulation of overlay networks preserves the modularization of the overall system. The primary purpose of encapsulation is the prevention of adverse effects on the message exchanges of a particular overlay network induced by the message exchanges on overlay networks of other application subsystems. In addition, overlay networks are designed for encapsulation at the level of jobs by preventing adverse effects on the message exchanges of a job induced by the message exchanges of other jobs in the same application subsystem.

3 Communication Middleware for the Realization of Overlay Networks

This section describes a realization of encapsulated time-triggered and event-triggered overlay networks on top of an exemplary time-triggered physical network. The time-triggered physical network is a TDMA-controlled Ethernet network that interconnects a set of five ECUs (see Figure 5). Each ECU is implemented on a Soekris net4801 [36] embedded single-board computer, which contains the 586 class processor SC1100 clocked at 266 MHz. The software within an ECU encompasses a real-time operating system, a real-time Ethernet driver, communication middleware for the establishment of overlay networks, and multiple jobs of one or more application subsystems.

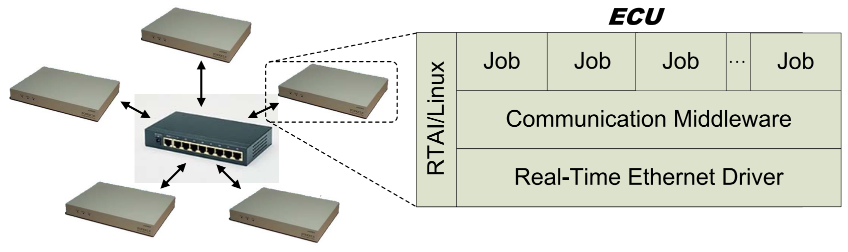

Fig. 5. Setup for Implementation of Encapsulated Overlay Networks

Although the prototype implementation uses a TDMA-controlled Ethernet network, the presented solution of a communication middleware for the establishment of overlay network is also compatible with other time-triggered communication protocols. For example, the ECUs can be deployed with FlexRay or TTP communication controllers and the real-time Ethernet driver can be replaced with a driver for interfacing FlexRay or TTP. Except for protocol-specific constraints (e.g., maximum message size), these time-triggered communication protocols offer the same type of interface to the host: a memory element that contains state messages that are sent or received by the communication system at predefined global points in time.

3.1 Operating System

The real-time Linux variant RTAI/LXRT [37] serves as the operating system of the ECUs. Each job executes as a Linux Real-Time (LXRT) user mode task and the Memory Management Unit (MMU) of the SC1100 processor provides memory protection. Furthermore, a time-triggered task scheduler ensures that jobs cannot delay other jobs by blocking the CPU. A static round-robin schedule uses predefined time intervals for the execution of all jobs on an ECU.

3.2 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 [13]. For the implementation of overlay networks, we have employed a software-based implementation of a TDMA-controlled Ethernet network. In each ECU a real-time Ethernet driver performs a master/slave clock synchronization and periodic time-triggered transmissions and receptions of state messages. The real-time Ethernet offers to the higher layers (i.e., communication middleware and application) a global time base with a precision of 50 μs. In addition, the real-time Ethernet driver provides a memory region with five state messages. One state message can be written by the communication middleware and is broadcast on the TDMA-controlled Ethernet network. The other four state messages contain information received from the TDMA-controlled Ethernet network and can be read by the communication middleware.

```
$S = \langle O[] \rangle$                                        // system: the overall system provides multiple overlay networks
$O = \langle M[], id_{network}, \text{paradigm}: <tt \mid et> \rangle$        // overlay network: comm. infrastructure for the jobs of an appl. subsystem
$M = \langle P_{out}, P_{in}[], id_{job} \rangle$                          // job: associated with input and output ports
$P_{out} = \langle \text{buffer}: \langle \text{variable} \mid \text{queue} \rangle, msg_{partial} \rangle$        // output port: access point of an overlay network for a job to send msgs.
$P_{in} = \langle \text{buffer}: \langle \text{variable} \mid \text{queue} \rangle, sender_{job}, msg_{partial} \rangle$  // input port: access point of an overlay network for a job to receive msgs.
$N = \langle msg_{Eth}[] \rangle$                                  // time-triggered physical network: provides a msg. for each node
$msg_{Eth} = \langle segment_{overlay-network}[] \rangle$                // msg. on time-triggered physical network: data segments for overlay networks
$segment_{overlay-network} = \langle segment_{job}[], id_{network} \rangle$  // overlay network segment: data segment belonging to a specific overlay network
$segment_{job} = \langle \text{data}, id_{job} \rangle$                     // job segment: data segment that belongs to a specific job

// pre-transmission invocation: transfer msgs. from output ports to data segments
$pretransmit(id_{node})$:
$\forall x \in msg_{eth}[id_{node}].segment_{overlay-network}, \ \forall y \in x.segment_{job}$:
    if ($x$.paradigm = TT)                                 // time-triggered overlay network
      $y$.data = $S.O[x.id_{network}].M[x.id_{job}].P_{out}$.buffer      // update-in-place of data segment with data from port
    else                                                   // event-triggered overlay network
      $y$.data = empty msg.                                // dequeue as many msgs. from output ports as fit into the segment
      while (space available in $y$.data)
        if ($msg_{partial} = empty$) $msg_{partial}$ = dequeue($S.O[x.id_{network}].M[x.id_{job}].P_{out}$)
        extract packet $p$ from $msg_{partial}$
        $y$.data | $p$
      end
    end

// post-reception invocation: transfer received msgs. into input ports
$postreception(id_{node})$:
$\forall x \in msg_{eth}[id_{node}].segment_{overlay-network}, \ \forall y \in x.segment_{job}$:
    if ($x$.paradigm = TT)                                                        // time-triggered overlay network
      $\forall z \in S.A[x.id_{network}].M[].P_{in}$ with $z.sender_{job} = id_{job}$ : $z.buffer = y$.data   // update-in-place of port with data from segment
    else                                                                          // event-triggered overlay network
      $\forall z \in S.A[x.id_{network}].M[].P_{in}$ with $z.sender_{job} = id_{job}$ :                  // assemble msgs. and enqueue in input ports
        $\forall p \in y$.data:
          $z.msg_{partial} \mid p$
          if ($msg_{partial}$ completely assembled)
            $z$.data $\leftarrow msg_{partial}$
            $msg_{partial}$ = empty message
          end
    end
```

Fig. 6. Pseudo Code Description of Communication Middleware

We have used a homogeneous configuration with a communication round consisting of five communication slots. The communication round allows each of the five ECUs to send exactly once and has a duration of 2 ms. The size of a state message that is broadcast within a communication slot has been set to 1500 bytes. Note, however, that the size of a message on the time-triggered physical is configurable, i.e., the size of 1500 bytes is only an example configuration. Using the hierarchical subdivision of the communication resources (as described in Section 2.2), the 1500 bytes per message are used by the communication middleware for disseminating information from multiple jobs attached to one or more overlay networks.

3.3 Communication Middleware

The communication middleware is a Linux kernel module that establishes the ports, which enable the jobs to access the overlay network of the respective application subsystem. As introduced in Section 2, a port is either a memory element storing a single state message (i.e., with updates-in-place) or a queue for multiple event messages. The ports of each job are located in a corresponding shared memory between the communication middleware in Linux kernel mode and the jobs in user mode. Each shared memory has a unique shared memory identifier. A job possesses only knowledge of the identifier of the shared memory containing the job's own ports. In contrast to the jobs, the communication middleware possesses all shared memory identifiers, because a single communication middleware on each ECU serves all jobs on the ECU.

Figure 6 describes the behavior of the communication middleware in pseudo code. Lines 1 to 9 capture the configuration information that parameterizes the communication middleware according to a specific application. The overall system S provides a set of overlay networks O, each serving as the communication infrastructure of a respective application subsystem. An overlay network exhibits a corresponding control paradigm (i.e., time-triggered or event-triggered) and is connected to a set of jobs via output and input ports (Lines 3ff). Both types of ports contain a buffer, which is either a message queue for an event-triggered overlay network or a state variable with update-in-place semantics for a time-triggered overlay network. Ports, which are attached to an event-triggered overlay network, also contain a partially transmitted or received message msg_{partial} for message fragmentation. In addition, an input port stores an identification of the sending job, i.e., the job from which messages are stored in the input port.

The structure of the messages that are exchange on the TDMA-controlled Ethernet work is defined in Lines 6–9. An Ethernet message exchanged on the TDMA-controlled Ethernet network is a compound structure, which results from the hierarchic subdivision of the TDMA slots. At the finest granularity in Line 9, each data segment is uniquely associated with an overlay network and a sending job.

After the configuration data structures, the reception and transmission handlers follow in Figure 6. The activation of the communication middleware occurs time-triggered, either prior to the periodic transmission of an Ethernet message (Line 10) or after the a priori known receive instants of Ethernet messages (Line 23). Both handlers go through the segments of all overlay networks and all jobs (Lines 12 and 24). The processing of such a data segment depends on the control paradigm of the overlay network.

- **Data segment of a time-triggered overlay network.** Prior to the periodic transmission of an Ethernet message, the communication middleware overwrites the data segment in the Ethernet message with the contents of the state variable at a job's port (Line 14). Consequently, the communication middleware samples the current value of the port and causes the dissemination of the value at the sampling point.
  After the reception of a periodic Ethernet message, each data segment belonging to a time-triggered overlay network is used for an update-in-place in the ports of the jobs. In order to determine which input ports need to be updated, the communication middleware exploits the a priori knowledge

concerning the identity of the sender that is uniquely associated with each data segment in the Ethernet message. For each job, which is located on the ECU and possesses an input port that accepts messages from the sender of the data segment, the communication middleware overwrites the state variable at the port with the contents of the data segment (Line 27).

- **Data segment of an event-triggered overlay network**. Prior to the periodic transmission of an Ethernet message, the communication middleware fills each data segment with event messages retrieved from the queue at the ports of the respective job (Lines 16ff). To efficiently use the bandwidth, the communication middleware not only transfers complete messages, but also fragments messages into slices so they fit into the data segment. Depending on the size of the event messages relative to the size of the data segment, it can be possible for multiple event messages to fit into the data segment or it may be necessary to fragment an event message into packets to be sent during multiple consecutive rounds.
  After the reception of a periodic Ethernet message, the communication middleware reads event messages from the data segment and forwards them to corresponding ports of the jobs (Lines 29ff). For each job, which is located on the ECU and possesses an input port that accepts messages from the sending job of the data segment, the communication middleware inserts the event messages from the data segment into the port's message queue. If messages are fragmented over multiple rounds, the communication middleware reassembles the event messages out of the parts retrieved from the data segment.

4 Exemplary Overlay Networks based on Automotive Communication Requirements

This section describes an exemplary configuration of overlay networks for the prototype implementation, which is based on the requirements of present day automotive network infrastructures. In addition, the configuration provides a time-triggered overlay network, e.g., for X-by-wire functionality [38, 39].

Network Class	Exemplary Protocols	Bandwidth	Exemplary Application Domains
Class A	LIN	< 10 kbps	sensor/actuator access
Class B	CAN	10kbps-125kbps	comfort domain
Class C	CAN	125kbps-1Mbps	powertrain domain
Class D	FlexRay, Byteflight	> 1 Mbps	multimedia, X-by-wire

Table 1. SAE Network Classes

4.1 Present Day Network Infrastructure

Based on the performance four classes of in-vehicle networks can be distinguished (see Table 1) according to the Society of Automotive Engineers (SAE) [40]. In

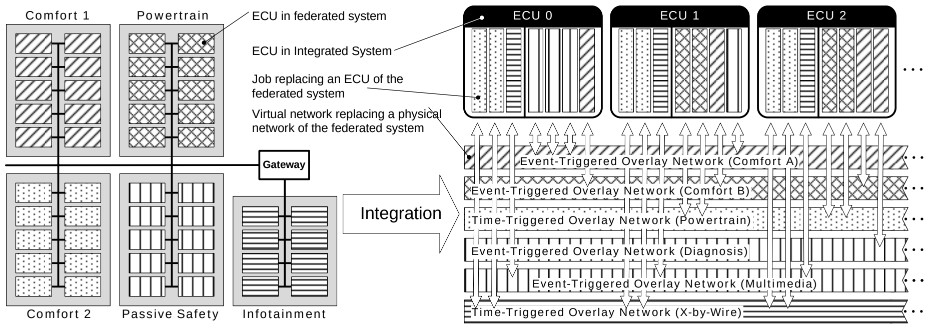
Comfort 1
Powertrain
ECU in federated system
ECU in Integrated System
Job replacing an ECU of the federated system
Virtual network replacing a physical network of the federated system
Gateway
Comfort 2
Passive Safety
Infotainment
Integration
ECU 0
ECU 1
ECU 2
Event-Triggered Overlay Network (Comfort A)
Event-Triggered Overlay Network (Comfort B)
Time-Triggered Overlay Network (Powertrain)
Event-Triggered Overlay Network (Diagnosis)
Event-Triggered Overlay Network (Multimedia)
Time-Triggered Overlay Network (X-by-Wire)

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

present-day luxury cars, networks belonging to all four classes can be found. For example, in the BMW 7 series [2] two class B networks (peripheral CAN and body CAN) interconnect the ECUs 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 (MOST [41]) and safety functions (Byteflight [42]). LIN fieldbus networks [43] 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 [44].

4.2 Exemplary Configuration of Overlay Networks

The exemplary overlay networks are based on the communication architecture of a typical present day automotive system. The overlay network configuration, which is depicted in Figure 7, enables the transition from a collection of physical networks towards an integrated architecture with overlay networks. Each overlay network serves as a substitute for a respective physical network.

The overlay network configuration supports two class B networks, one class C network, and two class D networks (see Table 2). Two overlay networks with a bandwidth of 117 kbps (named comfort A and comfort B) support on-demand event message exchanges, e.g., as deployed as medium-speed CAN networks for the comfort subsystem in a car. An overlay network with a bandwidth of 492 kbps 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 overlay networks exceed the net bandwidths of physical CAN networks with a raw bandwidth of 125 kbps or 500 kbps respectively. A further difference to a physical CAN network is that the 117 kbps or 492 kbps are simultaneously available to all jobs. Effectively, the entire overlay

Overlay Network	Node 0	Node 1	Node 2	Node 3	Node 4
Comfort A	Job 0 (117kbps)	Job 1 (117kbps)	Job 2 (117kbps)	Job 3 (117kbps)	Job 4 (117kbps)
	Job 5 (117kbps)	Job 6 (117kbps)	Job 7 (117kbps)	Job 8 (117kbps)	Job 9 (117kbps)
Comfort B	Job 0 (117kbps)	Job 1 (117kbps)	Job 2 (117kbps)	Job 3 (117kbps)	Job 4 (117kbps)
	Job 5 (117kbps)	Job 6 (117kbps)	Job 7 (117kbps)	Job 8 (117kbps)	Job 9 (117kbps)
Powertrain	Job 0 (492kbps)	Job 1 (492kbps)	Job 2 (492kbps)	Job 3 (492kbps)	Job 4 (492kbps)
	Job 5 (492kbps)	Job 6 (492kbps)	Job 7 (492kbps)	Job 8 (492kbps)	Job 9 (492kbps)
Diagnosis	Job 0 (70kbps)	Job 1 (70kbps)	Job 2 (70kbps)	Job 3 (70kbps)	Job 4 (70kbps)
Multimedia	Job 0 (1496kbps)	Job 1 (1496kbps)	Job 2 (492kbps)	Job 3 (492kbps)	Job 4 (492kbps)
X-by-Wire	Job 0 (617kbps)	Job 1 (617kbps)	Job 2 (617kbps)	Job 3 (617kbps)	Job 4 (617kbps)
	Job 5 (617kbps)	Job 6 (617kbps)	Job 7 (3117kbps)	Job 8 (3117kbps)	Job 9 (3117kbps)

Table 2. Overlay Network Configuration

network has the ten-fold bandwidth (1170 kbps or 4920 kbps) in case of the 10 jobs in the overlay network configuration. The reason for this bandwidth multiplication is that in a physical CAN network, the total system-wide bandwidth is available to a single ECU. In an overlay network, however, the available bandwidth via a job slot is not shared and only available to a single ECU. However, in case of a priori knowledge of the fraction of the overall bandwidth used by a job, this overhead can be significantly reduced.

An overlay (named diagnosis) with a bandwidth of 70 kbps serves for the exchange of event messages carrying diagnostic information. Such a diagnostic overlay network is required for the online analysis of observed errors, which is a promising strategy for reducing the numbers of cannot duplicate failures in future car generations [45].

The multimedia cluster is frequently based on a protocol with support for streaming audio and video (e.g., MOST [41]). For this purpose, we have provided an overlay network (named multimedia) 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 an overlay network (named X-by-wire) for the time-triggered exchange of state messages as required for safety-critical application subsystems.

No overlay networks are provided for class A networks, because low-cost fieldbus networks (e.g., LIN) are assumed to remain as separate physical networks despite the shift to an integrated architecture.

5 Related Work

This section describes related work on integrated architectures in the automotive domain. In addition, we point out the differences of the proposed overlay networks compared to other solutions for the integration of event-triggered and time-triggered communication.

5.1 Integration of Automotive Application Subsystems

The Automotive Open System Architecture (AUTOSAR) [46] is a system architecture and development methodology for automotive electronic systems, which also addresses the sharing of communication resources among application subsystems and software components. For interactions between software components, AUTOSAR provides a so-called *Virtual Function Bus (VFB)* with support for unidirectional message exchanges (called sender-receiver communication) and bidirectional interactions (called client-server communication).

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

In contrast to the proposed solution of overlay networks on top of a time-triggered physical network, AUTOSAR currently does not specify mechanisms for encapsulation of the communication activities of application subsystems and software components.

Nevertheless, the presented overlay networks for an integrated architecture can also serve for improving the communication system in an AUTOSAR system. Encapsulated overlay networks could provide the communication infrastructure of AUTOSAR software components. To accomplish this goal, future work will have to address the mapping of the proposed overlay network to the VFB interface that is part of the AUTOSAR runtime environment.

5.2 Integration of Control Paradigms

The proposed solution of layering event-triggered and time-triggered overlay networks on top of an underlying time-triggered physical network significantly differs from other solutions that combine these two communication paradigms. For example, event-triggered and time-triggered communication have been integrated at the MAC layer using two types of periodically recurring time intervals in the protocols FlexRay [11], TTCAN [48], and FTT-Ethernet [47]. These protocols employ a MAC layer that supports both event-triggered and time-triggered message transmissions. The start and end instants of the periodic time-triggered message transmissions, as well as the sending ECUs are specified at design time. For this class of messages, contention is resolved statically. Within each cycle, the time that is not reserved for time-triggered message exchanges is available for event-triggered communication. Consequently, time is divided into two types of intervals: event-triggered and time-triggered intervals. In event-triggered intervals, message exchanges depend on external control and the start instants of message transmissions can vary. This difference with respect to the start instants of event-triggered and time-triggered intervals is depicted in Figure 8 (arrows mark the start instants of the message transmissions in Figure 8). Furthermore,

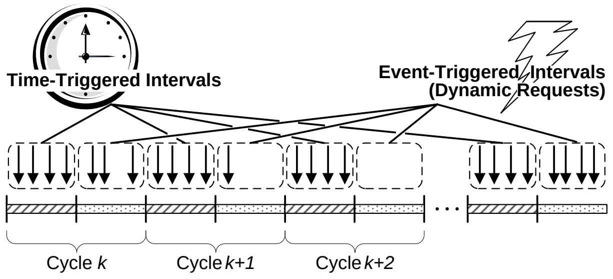

Fig. 8. Integration of Control Paradigms at MAC Layer: Time Intervals for Event-Triggered and Time-Triggered Communication

event-triggered intervals can be assigned to multiple (or all) ECUs of the system. For this reason, the MAC layer needs to support the dynamic resolving of contention when more than one ECU intends to transmit a message. During event-triggered intervals a sub-protocol (e.g., CSMA/CA, CSMA/CD) takes over that is not required during time-triggered intervals in which contention is prevented by design.

The main benefit of the two types of intervals at the MAC layer is the ability to combine temporal predictability in the time-triggered intervals with resource efficiency and flexiblity in the event-triggered intervals. Within the latter type of interval, ECUs can dynamically share communication resources since bandwidth that is not used by one ECU becomes available to other ECUs.

On the negative side, the event-triggered intervals at the MAC layer can introduce dependencies in the temporal domain between ECUs. The message transmission latencies at an ECU can no longer be computed in isolation. Upon system integration, the addition of ECUs will lead to more messages competing for transmission in the event-triggered intervals, thus increasing the transmission latencies. In addition, at run-time the latencies of the messages transmitted by a ECU can vary depending on the transmission behavior of the other ECUs. This variability of the transmission latency is disadvantageous for jitter-sensitive applications, e.g., in control loops where jitter impairs the quality of control.

In contrast to these MAC layer solutions, the presented overlay networks exhibit an invariant temporal behavior (i.e., bandwidth, latency, latency jitter) during system integration. The layering of event-triggered and time-triggered overlay networks on top of an underlying time-triggered physical network ensures that the temporal properties of independently developed components (i.e., jobs) remain invariant upon integration, thus enabling a seamless system integration without unintended side effects (i.e., temporal composability [6]).

Furthermore, the use of an underlying time-triggered physical network facilitates replica determinism [19] by preventing the occurrence of race conditions between jobs at the communication system.

Finally, the static schedule of the time-triggered physical network can be exploited for error detection and error containment, e.g., using bus guardians [33–

35]. Thereby, the communication system improves the robustness of the overall system by providing error containment for the consequences of physical faults (e.g., single event upsets, single event transients) and software faults (e.g., design fault of a job).

On the negative side, this rigid design with a static communication schedule results in lower resource efficiency. Firstly, large communication loads that dynamically vary between ECUs can lead to an inefficient use of the overall bandwidth. The presented overlay networks support no global sharing of bandwidth, i.e., communication slots that are not used by one ECU do not become available to the other ECUs. Secondly, the static resource allocation decreases flexibility w.r.t. extensions and modifications of the communication system. For the addition of messages or ECUs, either free slots need to be reserved at design time or a new time-triggered communication schedule needs to be computed and programmed into the ECUs.

6 Discussion

This paper has shown that a time-triggered physical network is an effective foundation for establishing multiple overlay networks, each tailored to a respective application subsystem via its control paradigm (event-triggered vs. time-triggered). Overlay networks exhibit predefined temporal properties for the messages transmitted by a job, independently from the transmission behavior of other jobs and other application subsystems. A prototype implementation has yielded evidence that the inherent mechanisms for encapsulation do not prevent the overlay networks from meeting the bandwidth requirements imposed by present-day automotive applications and those envisioned for the future (e.g., X-by-wire).

Encapsulation is particularly important in the context of the increasing complexity of embedded systems. System architects are 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 application subsystems or jobs, 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 are not 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.

Overlay networks 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 (e.g., FlexRay [11]) 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 application subsystems. In conjunction with integrated ECUs, i.e., ECUs for the execution of application software from different application subsystems, the sharing of communication and computational resources not only reduces the number of ECUs, but also results in fewer connectors and wires. Fewer connectors and wires not only decrease hardware cost, but also lead to improved reliability.

Acknowledgments

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. B. Bouyssounouse and J. Sifakis, editors. *Embedded Systems Design.* Springer Verlag, 2005.
2. 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.
3. H. Heinecke, K.-P. Schnelle, H. Fennel, J. Bortolazzi, L. Lundh, J. Leflour, J.-L. Maté, K. Nishikawa, and T. Scharnhorst. AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures. In *Proceedings of the Convergence Int. Congress & Exposition On Transportation Electronics*, Detroit, MI, USA, October 2004. SAE. 2004-21-0042.
4. R. Obermaisser, P. Peti, B. Huber, and C. El Salloum. DECOS: An integrated time-triggered architecture. *e&i journal (journal of the Austrian professional institution for electrical and information engineering)*, 3:83–95, March 2006. Available at http://www.springerlink.com.
5. 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.
6. H. Kopetz and R. Obermaisser. Temporal composability. *Computing & Control Engineering Journal*, 13:156–162, August 2002.
7. Robert Bosch Gmbh, Stuttgart, Germany. *CAN Specification, Version 2.0*, 1991.
8. G. Leen, D. Heffernan, and A. Dunne. Digital networks in the automotive vehicle. *Computing & Control Engineering Journal*, 10(6):257–266, December 1999.
9. F.P. Brooks. *The Mythical Man-Month.* Addison Wesley, 1975.
10. Analysis of the European automotive in-vehicle network architecture markets. Technical report, Frost & Sullivan, October 2004.
11. FlexRay Consortium. BMW AG, DaimlerChrysler AG, General Motors Corporation, Freescale GmbH, Philips GmbH, Robert Bosch GmbH, and Volkswagen AG. *FlexRay Communications System Protocol Specification Version 2.1*, May 2005.

12. TTTech Computertechnik AG, Schönbrunner Strasse 7, A-1040 Vienna, Austria. *Time-Triggered Protocol TTP/C – High Level Specification Document*, July 2002.
13. 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.
14. K. Hoyme and K. Driscoll. SAFEbus. *IEEE Aerospace and Electronic Systems Magazine*, 8:34–39, March 1993.
15. J. Rushby. Bus architectures for safety-critical embedded systems. In Tom Henzinger and Christoph Kirsch, editors, *Proc. of the First Workshop on Embedded Software (EMSOFT 2001)*, volume 2211 of *Lecture Notes in Computer Science*, pages 306–323, Lake Tahoe, CA, October 2001. Springer-Verlag.
16. H. Kopetz. Why time-triggered architectures will succeed in large hard real-time systems. In *Proc. of the 5th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, Cheju Island, Korea, August 1995.
17. E. Bretz. By-wire cars turn the corner. *IEEE Spectrum*, 38(4):68–73, April 2001.
18. G. Bauer and H. Kopetz. Transparent redundancy in the time-triggered architecture. In *Proc. of the Int. Conference on Dependable Systems and Networks (DSN 2000), NY, USA*, pages 5–13, June 2000.
19. S. Poledna. *Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism.* Kluwer Academic Publishers, November 1995.
20. Radio Technical Commission for Aeronautics, Inc. (RTCA), Washington, DC. *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, December 1992.
21. 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.
22. TTTech Computertechnik AG, Schönbrunner Strasse 7, A-1040 Vienna, Austria. *TTP/C Controller C2 Controller-Host Interface Description Document, Protocol Version 2.1*, November 2002.
23. Freescale Semiconductor. MFR4200 datasheet FlexRay communication controllers. Technical report, August 2005.
24. R. Venkateswaran. Virtual private networks. *IEEE Potentials*, 20(1):11–15, February 2001.
25. D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 131–145, New York, NY, USA, 2001. ACM Press.
26. R. Deline. *Resolving Packaging Mismatch.* PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, June 1999.
27. H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.
28. H. Kopetz. Elementary versus composite interfaces in distributed real-time systems. In *Proc. of the 4th Int. Symposium on Autonomous Decentralized Systems*, Tokyo, Japan, March 1999.
29. F. Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, 1991.
30. C. Temple. Avoiding the babbling-idiot failure in a time-triggered communication system. In *Proc. of 28th Int. Symposium on Fault-Tolerant Computing*, pages pp. 218–227, Munich, Germany, 1998.
31. B. Randell, P. Lee, and P. C. Treleaven. Reliability issues in computing system design. *ACM Computing Surveys*, 10(2):123–165, 1978.

32. L. Kleinrock. *Queuing Systems Volume I: Theory.* John Wiley and Sons, New York, 1975.
33. 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.
34. 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, Pisa, Italy, April 2003.
35. 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.
36. Soekris Engineering. net4801 series boards and systems. Technical report, April 2004. `www.soekris.com`.
37. D. Beal, E. Bianchi, L. Dozio, S. Hughes, P. Mantegazza, and S. Papacharalambous. RTAI: Real-Time Application Interface. *Linux Journal*, April 2000.
38. B. Hedenetz and R. Belschner. Brake-by-wire without mechanical backup by using a TTP-communication network. In *Proceedings of SAE Congress*. Daimler-Benz AG, 1998.
39. H.D. Heitzer. Development of a fault-tolerant steer-by-wire steering system. *Auto Technology*, 4:56–60, April 2003.
40. G. Leen and D. Heffernan. Expanding automotive electronic systems. *Computer*, 35(1):88–93, January 2002.
41. MOST Cooperation, Karlsruhe, Germany. *MOST Specification Version 2.2*, November 2002.
42. 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.
43. 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.
44. J. Leohold. Communication requirements for automotive systems. In *Keynote Automotive Communication – 5th IEEE Workshop on Factory Communication Systems*, Vienna, Austria, September 2004.
45. P. Peti and R. Obermaisser. A diagnostic framework for integrated time-triggered architectures. In *Proc. of the 9th IEEE Int. Symposium on Object-oriented Real-time distributed Computing*, April 2006.
46. AUTOSAR GbR. *AUTOSAR – Technical Overview V2.0.1*, June 2006.
47. P. Pedreirasand P. Gai and L. Almeidaand G.C. Buttazzo. Ftt-ethernet: a flexible real-time communication protocol that supports dynamic qos management on ethernet-based systems. *IEEE Transactions on Industrial Informatics*, 1(3):162–172, August 2005.
48. T. Führer, B. Müller, W. Dieterle, F. Hartwich, and R. Hugel. Time-triggered CAN - TTCAN: Time-triggered communication on CAN. In *Proc. of 6th Int. CAN Conference (ICC6)*, Torino, Italy, 2000.