Virtual Networks in an Integrated Time-Triggered Architecture

R. Obermaisser, P. Peti, H. Kopetz
Vienna University of Technology, Austria
email: {ro,php,hk}@vmars.tuwien.ac.at

Abstract

Depending on the physical structuring of large distributed safety-critical real-time systems, one can distinguish federated and integrated system architectures. This paper investigates the communication services of an integrated system architecture, which combines the complexity management advantages of federated systems with the functional integration and hardware benefits of an integrated approach. A major challenge is the need to accommodate the communication services to the different types of integrated application subsystems that range from ultra-dependable control applications (e.g., an x-by-wire system) to non safety-critical applications such as multimedia or comfort systems. In particular, the encapsulation of the communication activities of different application subsystems is required not only to prevent error propagation from non safety-critical application subsystems to higher levels of criticality, but also to facilitate complexity management and permit independent development activities.

This paper introduces virtual networks as the encapsulated communication infrastructure of an application subsystem in the integrated DECOS architecture. Virtual networks are constructed as overlay networks on top of the time-triggered communication system of a base architecture. Each virtual network runs a corresponding communication protocol that is determined either by a legacy platform or selected to meet the requirements of the application subsystem. Encapsulation mechanisms ensure that the temporal properties of each virtual network are known a priori and independent from the communication activities in other virtual networks. By assigning to each application subsystem a dedicated virtual network and by ensuring that the virtual network abstractions hold also in the case of faults, the integrated architecture supports the benefits of a federated system, such as fault isolation, complexity management, independent development, and intellectual property protection. In addition, virtual networks promise massive cost savings through the reduction of physical networks and reliability improvements with respect to wiring and connectors.

1 Introduction

Until now, the introduction of *structure and hierarchical relationships* represents the only promising approach for understanding complex systems with large numbers of parts and interactions between these parts [34, chap. 8]. This insight applies to all technical systems and in particular to the mastering of large, complex real-time computer systems. The complexity of a large real-time computer system can only be managed, if the overall system can be decomposed into nearly-independent application subsystems with linking interfaces that are precisely specified in the value and time domain [19]. We denote such an application subsystem that provides a part of the computer system's overall functionality as a *Distributed Application Subsystem (DAS)* (e.g., steer-by-wire DAS in the automotive domain, cabin pressurization DAS in the avionic domain), because the implementation of the corresponding functionality will most likely involve multiple components that are interconnected by an underlying communication system.

Based on the allocation of DASs to the available hardware resources, one can distinguish *federated* and *integrated architectures*. In a federated architecture, each DAS has its own dedicated computer system, while an integrated architecture is characterized by the integration of multiple DASs within a single distributed computer system. Federated architectures have been preferred for ultra-dependable systems due to the natural separation of DASs. Based on the widely accepted assumption that a hardware fault effects an entire component [20, 15], federated architectures excel with respect to fault containment. Since each component is dedicated to a single function, only one function is affected by a hardware fault. Only a very limited level of interaction between DASs occurs via gateways (e.g., On-Board Diagnosis (OBD) systems [4]), which results in error containment between DASs and facilitates an independent development of DASs through different vendors. In addition, federated architectures solve the problem of intellectual property protection trivially. Since the whole DAS is typically developed by one vendor, no internals need to be revealed to the system integrator or other vendors. The hiding of the internals of DASs also helps in managing the complexity of a system. The absence of interactions and dependencies between DASs allows to understand a DAS, while abstracting from the detailed behavior of other DASs.

The independently developed computer systems can also

be optimized to the specific requirements of the respective DASs. The communication system of a distributed computing platform constitutes a primary target for these optimizations. The selection of the communication protocol, the temporal properties (e.g., bandwidth), and the fault-tolerance mechanisms can be matched to the designed DAS. Thus, a balanced compromise between contradicting requirements is achievable (e.g., flexibility vs. predictability).

For this purpose, present day automotive systems follow the federated philosophy with different types of automotive networks [21]. The distributed Electronic Control Units (ECUs) are interconnected via specialized communication networks with different protocols (e.g., LIN [22], CAN [28], MOST [23]), physical layers (e.g., fibre, twisted pair), bandwidths (10 kbps–25 Mbps), and dependability. The multitude of communication protocols is a result of the different requirements with respect to functionality, dependability, and performance of the automotive DASs, such as the infotainment DAS, the comfort DAS, or the powertrain DAS.

In contrast to federated architectures, integrated architectures share network and component resources among different DASs in order to evolve beyond a "1 Function – 1 ECU" strategy. In the automotive area the trend of steadily increasing numbers of ECUs along with the addition of functionality has lead to luxury cars with up to 75 components [4]. Integrated architectures not only permit a dramatic reduction in the overall number of ECUs through tackling this "1 Function – 1 ECU" problem, but also offer increased reliability by minimizing the number of connectors and wires. Field data from automotive environments has shown that more than 30% of electrical failures are attributed to connector problems [36].

An ideal future system architecture would thus *combine the complexity management advantages of the federated approach, but would also realize the functional integration and hardware benefits of an integrated system* [6, p. 32]. This paper looks at the role of the *communication system* in bringing to an integrated architecture the error containment and complexity management benefits of federated architectures. We argue that the communication system should support a similar structuring of the overall system as in a federated architecture. By dividing the overall functionality hierarchically into a set of DASs, each with dedicated architectural communication services, developers can act as if they were building an application for a federated architecture. The key requirements for the communication services are as follows:

- *Support for Ultra-Dependable Control Applications.* Achievement of control stability in real-time applications depends on the completion of activities in bounded time [13]. For example, in drive-by-wire applications, the dynamics for steered wheels in closed control loops enforce computer delays of less than 2 ms [7]. Hard real-time systems ensure guaranteed response even in the case of peak load and fault scenarios.

- *Encapsulation.* Encapsulation must ensure that a DAS along with the corresponding communication resources and computational resources is encapsulated and interactions between DASs are limited to the exchange of messages via precisely specified gateways. By supporting temporal and spatial partitioning [30] both between and within DASs, an integrated architecture exceeds the error containment capabilities of many federated architectures. Through the prevention of hidden interactions between DASs, encapsulation also reduces the mental effort for understanding DASs and eliminates interference between different developers, which is a prerequisite for an independent development of DASs.

- *Service Optimization.* Different DASs have different requirements with respect to communication services. For supporting different, possibly contradicting requirements from different application subsystems, the integrated architecture has to provide not a single but multiple communication services, each tailored to the needs of the respective subsystem.

  It has been recognized that communication protocols fall into two general categories with corresponding strengths and deficiencies: event-triggered and time-triggered control. Event-triggered protocols (e.g., CAN [28], TCP/IP [27], Ethernet [9]) offer flexibility and resource efficiency. Time-triggered protocols (e.g., TTP [37], SafeBus [8]) excel with respect to predictability, composability, error detection and error containment. Consequently, the communication service of an integrated architecture should support both control paradigms.

- *Legacy Integration Support.* A legacy system is an information system that resists evolution [3], thereby restricting the opportunities for extensions of the system. Nevertheless, legacy systems often represent major investments and it can be too costly to replace an entire legacy system in a single step. The resulting desire to reuse legacy subsystems in an integrated architecture requires a communication system that can establish legacy protocols, such as the CAN protocol that is widely used in the automotive domain.

As part of the integrated architecture that is developed in the Dependable Embedded Components and Systems (DECOS) EU Framework Programme 6 [18], we introduce a solution to these requirements. We provide each DAS with a dedicated communication infrastructure that is realized as an encapsulated *virtual network*, i.e. an overlay network on top of a time-triggered physical network. Each virtual network supports a corresponding communication protocol and is tailored to the requirements of the respective DAS via its functionality (provided services), its operational properties (syntactic and temporal), and its meta-level properties (dependability, semantics of interface data structures).

The paper is structured as follows. Section 2 gives a short overview of the DECOS integrated architecture that

employs virtual networks as the communication infrastructure of DASs. The construction of virtual networks on top of a time-triggered core network is the focus of Section 3. Section 4 discusses the establishment of virtual networks at the component level. Components consists of distinct hardware elements that realize a hierarchic subdivision of network resources. We sketch the application of virtual networks in Section 5 as part of an exemplary automotive system. The paper finishes with a conclusion in Section 6.

2 DECOS Integrated Architecture

The DECOS architecture [18] offers a framework for the development of distributed embedded real-time systems integrating multiple DASs with different levels of criticality and different requirements concerning the underlying platform. Structuring rules guide the designer in the decomposition of the overall system both at a functional level and for the transformation to the physical level. In addition, the DECOS integrated architecture aims at offering to system designers generic architectural services, which provide a validated stable baseline for the development of applications.

2.1 Functional System Structuring

For the provision of application services at the controlled object interface, the overall system is divided into a set of nearly-independent DASs. Each DAS is further decomposed into smaller units called *jobs*. A *job* is the basic unit of work and accesses the communication system via *ports* in order to exchange information with other jobs. Depending on the data direction, one can distinguish input ports and output ports.

2.2 Physical System Structuring

During the development of an integrated system the functional elements must be mapped to the physical building blocks of the platform. These building blocks are *clusters*, *physical networks*, *components* and *partitions*. A *cluster* is a distributed computer system that consists of a set of components interconnected by a *physical network*. A *component* is a self-contained computational element with its own hardware (processor, memory, communication interface, and interface to the controlled object) and software (application programs, operating system) [19], which interacts with its environment by exchanging messages across Linking Interfaces (LIFs). The behavior of a component can be specified in the domains of value and time. Components are the target of job allocation and provide encapsulated execution environments denoted as *partitions* for jobs. Each partition prevents temporal interference (e.g., stealing processor time) and spatial interference [30] (e.g., overwriting data structures) between jobs. In the DECOS architecture, a component can host multiple partitions and host jobs that can belong to different DASs.

2.3 Architectural Services

Generic architectural services separate the application functionality from the underlying platform technology in order to facilitate reuse and reduce design complexity. This strategy corresponds to the concept of platform-based design [33], which proposes the introduction of abstraction layers, which facilitate refinements into subsequent abstraction layers in the design flow.

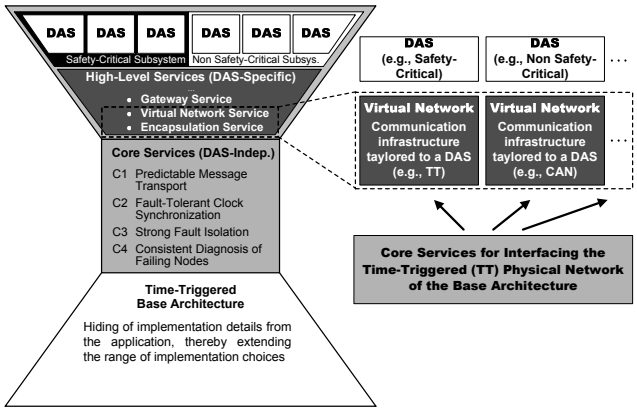

Figure 1. DECOS Integrated Architecture

The DECOS architectural services depicted in Figure 1 are such an abstraction layer. The specification of the architectural services hides the details of the underlying platform, while providing all information required for ensuring the functional and meta-functional (dependability, timeliness) requirements in the design of a safety-critical real-time application. The architectural services serve as a validated stable baseline that reduces application development efforts and facilitates reuse, because applications build on an architectural service interface that can be established on top of numerous platform technologies.

In order to maximize the number of platforms and applications that can be covered, the DECOS architectural service interface distinguishes a minimal set of *core services* and an open-ended number of *high-level services* that build on top of the core services. The core services include predictable time-triggered message transport, fault tolerant clock synchronization, strong fault isolation, and consistent diagnosis of failing components through a membership service. The small number of core services eases a thorough validation (e.g., permitting a formal verification), which is crucial for preventing common mode failures as all high-level services and consequently all applications build on the core services. Any architecture that provides these core services can be used as a base architecture [32] for the DECOS integrated distributed architecture. An example of a suitable base architecture is the Time-Triggered Architecture (TTA) [16].

Based on the core services, the DECOS integrated architecture realizes high-level architectural services, which are DAS-specific and constitute the interface for the jobs to the

underlying platform. Among the high-level services are virtual network services and encapsulation services. On top of the time-triggered physical network, different kinds of virtual networks can be established and each type of virtual network can exhibit multiple instantiations (see Figure 1). The encapsulation services ensure spatial and temporal partitioning for virtual networks in order to obtain error containment and control the visibility of exchanged messages.

2.4 Fault Hypothesis

In the integrated system architecture, we perform a differentiation of Fault Containment Regions (FCRs) for hardware and software faults [1]. For hardware faults, we regard a complete component as a FCR, because a component contains shared physical resources (e.g., processor, power supply). The failure mode of a hardware FCR is assumed to be arbitrary. The failure frequency in case of permanent hardware failures is in the order of 100 FIT [25]. In case of transient failures a significantly higher failure frequency in the order of 1000-10000 hours is assumed.

For software faults, we regard a job as a FCR. The failure mode of a job is a violation of the port specification in either the time or value domain. In case of a failure in the value domain, the content of a message does not conform to its specification, while in case of a timing failure, the send instant of the message is incorrect.

3 Virtual Network Services

A *virtual network* is an overlay network that is established on top of a physical network. In the DECOS integrated system architecture, we provide virtual networks on top of the time-triggered core communication service of the base architecture. To achieve the advantages of the federated approach in an integrated architecture, we propose the provision of a dedicated virtual network for each DAS in order to exchange messages between the jobs of the DAS. Each virtual network is tailored to the requirements of the respective DAS via the provided functionality, the operational properties, the namespace, and the dependability properties. Furthermore, each virtual network is encapsulated so that communication activities in other virtual networks are neither visible nor have any effect (e.g., performance penalty) on the exchange of messages in the virtual network. Consequently, virtual networks extrapolate the idea of component-level error containment to the network-level. The fine-grained multiplexing of components for multiple protected jobs not only requires error containment for component resources (processor and memory resources), but also error containment for a component's network resources.

3.1 Characterization of Virtual Networks

As the communication infrastructure for a DAS, a virtual network can be characterized by the following properties:

- **Functionality:** The obligatory functionality of a virtual network is the transport of messages between senders and receivers. For this transport of messages, one can distinguish different *communication topologies* that define which jobs receive a sent message, e.g., point-to-point and broadcast communication topologies. In addition, a virtual network can provide further services such as flow control.

- **Operational Properties:** The operational properties of a virtual network denote the characteristics of exchanged messages in the value and temporal domain. The syntactic properties define the structure of messages. The temporal properties include information about the employed control paradigm (time-triggered or event-triggered), the control flow (information push or information pull [5]), the message ordering, the available bandwidth, and the transmission latencies.

- **Namespace:** Message names are defined according to the namespace of the respective DAS and used for addressing and the identification of communicated information. A message name can either be part of the syntactic structure of a message or be defined implicitly (e.g., via the point in time of the message transmission as in [37]). Each virtual network encompasses a *separate namespace*, which is a prerequisite for the independent development of DASs and the ability of preserving existing namespaces in legacy applications.

- **Dependability:** Each virtual network is characterized by dependability properties [1] such as the reliability of the message transport service. The dependability of a virtual network results from the properties of the underlying physical network (e.g., message failure rate, media redundancy) and the realization of the high-level architectural services. A virtual network can implement additional fault-tolerance mechanisms such as retransmissions in case of transmission failures (timing redundancy).

3.2 Time-Triggered Virtual Networks

The virtual networks for safety-critical DASs are strictly time-triggered, because of the respective advantages of this control paradigm [12, 31] (e.g., with respect to predictability, error detection, fault-tolerance, replica determinism). A time-triggered virtual network is designed for the periodic exchange of state messages. The self-contained nature and idempotence of state messages eases the establishment of state synchronization, which does not depend on exactly-once processing guarantees. Since applications are often only interested in the most recent value of a real-time object, old state values can be overwritten with newer state values.

The temporal firewall concept [17] is used as the interface between a job and a time-triggered virtual network (see Figure 2). The sender acts according to the information

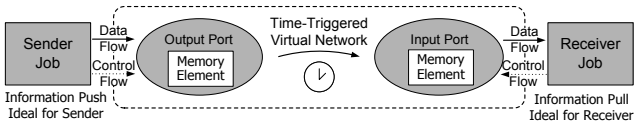

Figure 2. Temporal Firewall Interface Provided by a Time-Triggered Virtual Network

push paradigm [5] and writes information into the memory element at its output port (update in place). The receiver must pull information out of the input port by reading the memory element in the input port. The time-triggered virtual 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 virtual networks employ *implicit flow control* [13]. 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 virtual networks well-suited for multicast communication relationships, because ports offer *elementary interfaces* [14], i.e. a unidirectional data flow involves only a unidirectional control flow.

3.3 Event-Triggered Virtual Networks

Event-triggered virtual networks can be deployed to interconnect the jobs of non safety-critical DASs. An event-triggered virtual network is designed for the sporadic exchange of event messages, combining event semantics with external control. The interactions between a sender and a receiver via an event-triggered virtual network are depicted in Figure 3. At the sender side, event messages are passed to the event-triggered virtual network 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). At the receiver side, the job either fetches the incoming message from the input port (polling for messages via information pull) or the event-triggered virtual network presses received messages into the job (interrupt mechanism via information push).

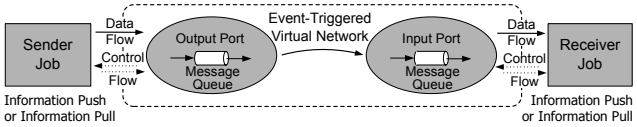

Figure 3. Message Exchange between two Jobs through an Event-Triggered Virtual Network

Messages exchanged via an event-triggered virtual network can exhibit event semantics by denoting the change in value of a real-time entity associated with a particular event (e.g., temperature increase by 2 degrees). In order to reconstruct the current state of a real-time entity from messages with event semantics, it is essential to process every message exactly once. The loss of a single message with event information can affect state synchronization between a sender and a receiver. Consequently, an event-triggered virtual network must support the transmission of event messages with exactly-once delivery semantics and provide message queues at input and output ports.

In contrast to time-triggered virtual networks, an event-triggered virtual network can support implicit and explicit flow control. *Implicit flow control* typically occurs through the specification of message interarrival and service times [11]. For *explicit flow control* the receiver exerts back pressure on the sender by sending acknowledgment messages. In this case, ports represent *composite interfaces* [14], because a unidirectional data flow involves a bidirectional control flow. The transmission of a message through the sender depends on a control flow in the opposite direction, i.e. from the receiver to the sender. While a composite interface prevents senders from overloading receivers, correctness of a sender depends on correctness of receivers, which can constitute a problem with respect to error propagation. Furthermore, explicit flow control cannot be applied for events occurring in the natural environment, because it is usually impossible to exert back pressure on the natural environment.

3.4 Hierarchic Subdivision of the Network Resources

For the realization of virtual networks at the physical level, we employ the time-triggered core communication service and perform a hierarchic temporal subdivision of the communication resources (see Figure 4). The media access control strategy of the time-triggered core communication service 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 component is assigned a unique *component slot* that periodically recurs at a priori specified global points in time. A component sends messages during its component slot and receives messages during the component slots of other components. A sequence of component slots, which allows every component in an ensemble of n components to send exactly once, is called a TDMA round. The sequence of the different TDMA rounds forms the cluster cycle and determines the periodicity of the time-triggered communication.

A component slot is further subdivided into subslots called *subsystem slots*, each being dedicated to a component subsystem. A component subsystem represents a particular criticality level (e.g., level A according to DO-178B [29]). An example for a subsystem classification is the bivalent distinction between safety-critical and a non

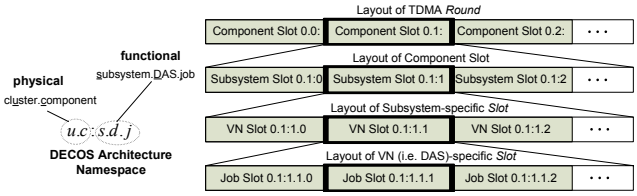

Figure 4. Hierarchic Structure of TDMA Slots: *TDMA rounds are incrementally subdivided to finally reach the subslots, in which messages produced by a particular job are being disseminated*

Notation	Meaning
:0.3.4	job 4 of DAS 3 of subsystem 0 (abstracting from physical structure)
0.2:0.3.4	job 4 of DAS 3 of subsystem 0, in component 2 of cluster 0
:0.1	DAS 1 of subsystem 0 (abstracting from physical structure)
:0.*.*	all jobs of all DAS of subsystem 0 (abstracting from physical structure)
0.*:	all components of cluster 0 (abstracting from functional structure)

Figure 5. Architecture Namespace Examples

safety-critical subsystem. All safety-critical DASs belong to safety-critical subsystem, while non safety-critical DASs form the latter subsystem. In this case, a component slot can consist of one or two subsystem slots. The component slot contains only one subsystem slot, if the component contains only jobs of DASs with the same criticality. More than one subsystem slot is necessary for mixed criticality components.

Again, subsystem slots are further subdivided into subslots. The subslots of a subsystem slot are denoted as *virtual network slots*, because each subslot contains those messages that are produced by jobs in the component that are connected to a particular virtual network. Since there is a one-to-one mapping between virtual networks and DASs, the virtual network slots are DAS-specific and the jobs that produce messages for this virtual network belong to the same DAS. On its part, a virtual network slot consists of subslots denoted as *job slots*. When a component hosts multiple jobs of a DAS that send messages to the virtual network of the DAS, then each of these jobs is assigned a job slot carrying the messages sent by that job.

The assignment of the slots within a TDMA round to components, as well as the further subdivision into subsystem slots, virtual network slots, and job slots 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.

In order to explain the realization of virtual networks, we use an architecture namespace to identify the constituting elements of an integrated DECOS system. (This architecture namespace is, however, different to the DAS-specific namespaces employed at the different virtual networks for the exchange of messages.) The architecture namespace consists of a part reflecting the physical structure (cluster, component) and a part reflecting the functional structure of the system (subsystem, DAS, job) as depicted in Figure 4. Table 5 contains examples of the introduced notation, in which a colon separates the numerical identification of the structural elements belonging to the functional and physical part.

For convenience, it is possible to omit either the physical or the functional part if not needed by the specification. For example, by describing only functional aspects of the safety-critical subsystem, one can abstract from the physical allocation (i.e. writing only $:s.d.j$). Wildcards (*) are used to express that not a particular element is of interest, but all elements are referred to. For example, $:s.d.*$ addresses all jobs of a particular DAS of a particular subsystem. Based on this architecture namespace, we also introduce a source-based namespace for messages. $m(u.c{:}s.d.j)$ identifies the message that is produced by job j of DAS $:s.d$ in component $u.c:$.

4 Realization of Virtual Networks at the Component Level

A component in the DECOS integrated architecture is a multiprocessor node with multiple hardware elements. As depicted in Figure 6, a component is horizontally structured into a communication controller, connector units, and application computers. The *communication controller* implements the core architectural services described in Section 2. *Connector units* realize the high-level architectural services that are provided to the jobs running in the *application computers*. In particular, the connector units perform the allocation of the network resources by incrementally dividing the capacity of the physical network. A *basic connector unit* establishes a safety-critical and a non safety-critical subsystem by means of *spatial* and *temporal* inner-component partitioning [30]. The *safety-critical subsystem* is an encapsulated execution environment for ultra-dependable applications, while the *non safety-critical subsystem* offers an environment for those applications having less stringent dependability requirements. Secondary connector units, which are named *safety-critical connector units* in the safety-critical subsystem and *complex connector units* in the non safety-critical subsystem, perform a further subdivision of the network resources.

The component structure with dedicated hardware elements facilitates a modular certification of the constituting elements. In particular, subsystems with different criticality can be independently certified to their respective criticality levels. In contrast to the safety-critical connector unit and basic connector unit, the complex connector unit and the non safety-critical subsystems of a component need not be certified to the highest criticality levels. Since the secondary connector units do not directly access the communication controller, the complex connector unit is not involved in the fault isolation and error containment between the safety-critical and non safety-critical subsystems. The realization of virtual networks exploits this aspect for providing in the

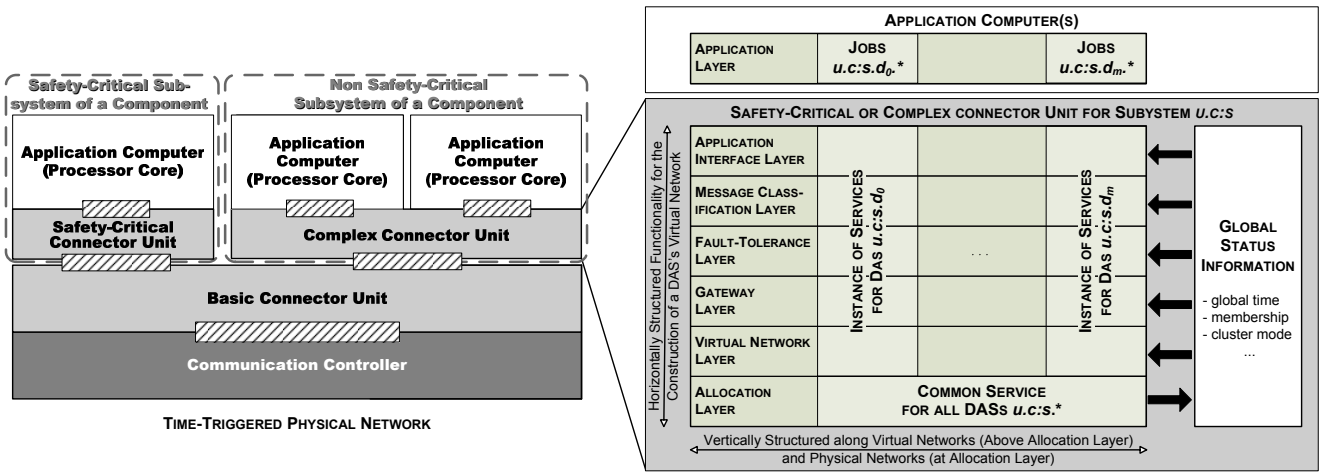

Figure 6. Component Structure: Layers perform a stepwise abstractions of the underlying platform.

complex connector unit increased functionality at the cost of increased complexity. The safety-critical connector unit must, however, aim at maximum simplicity to control certification efforts.

Both secondary connector units possess the same inner structure that consists of six layers (see Figure 6). However, the realization of these layers in the two types of connector units can differ and is determined by the need for low complexity and certification in the safety-critical connector unit and increased functionality in the complex connector unit. In addition, each secondary connector unit is vertically structured for supporting multiple DASs and multiple physical networks. The *allocation layer* manages the bandwidth of the physical network, which is passed by the basic connector unit to the secondary connector unit. For each physical network that is support by the connector unit, there is a dedicated instance of the allocation layer. The allocation layer splits up the component slots of the different virtual networks employed in the component (third level of the hierarchic subdivision of TDMA slots in Figure 4). For this purpose, the allocation layer forwards a statically fixed part of each TDMA slot to the upper layer, the *virtual network layer*. The virtual network layer performs the construction of overlay networks, each with defined operational properties. For each virtual network, the connector unit contains a dedicated instance of the virtual network layer and all higher layers. Such an instance of virtual network layer executes a particular communication protocol and maps this protocol to the underlying state message interface of the allocation layer. The virtual network layer is also responsible for subdividing slots to the job-level (see Figure 4). The *gateway layer* performs selective redirection and property transformation of messages exchanged between physical and/or virtual network segments. The *fault-tolerance layer* is part of the architectural support for active redundancy and votes on the redundant messages produced by replica deterministic jobs. The subsequent layer is the *message classification layer*, which applies input and output assertions on exchanged message in order to collect diagnostic information for the evaluation of out-of-norm assertions [26]. The last layer of a secondary connector unit is the *Application Programming Interface (API) layer*. This layer provides an abstraction that hides the technological details of the architectural services and establishes the programming interfaces required by existing and newly developed applications.

4.1 Separation of Bandwidth with Respect to Subsystems

The basic connector unit performs the primary allocation of network resources, as required for the separation of the safety-critical and non safety-critical subsystems of a component. For this purpose, the basic connector unit transforms a single state message interface into two state message interfaces. The single state message interface is provided via an input port and an output port (denoted as outer ports) towards the communication controller. In addition, for each of the two state message interfaces towards a secondary connector unit there is an input port and an output port (denoted as inner ports). Associated with each port is a memory element that holds a single state message. Whenever the subsystem attached to an input port accesses the port, it reads the content of this memory element. In analogy, each access at an output port results in the writing of a new value into the memory element.

The operation of a basic connector unit comprises a periodic sequence of copy operations for the transfer of state messages between memory elements associated with inner and outer ports. A static schedule specifies the global points in time at which each copy operation is performed. The static schedule must be designed in such a way that the copy operations of the basic connector unit are phase-aligned with the time-triggered updates and disseminations of the memory elements at the outer ports towards the basic, time-triggered communication system. The creation of the static schedule of the basic connector unit is part of a two-level design and occurs after the construction of the static schedule of the underlying time-triggered communi-

cation system. The described operation of a basic connector unit ensures spatial and temporal partitioning [30], which is fundamental to the separation of the safety-critical and non safety-critical subsystems of a component:

- **Temporal partitioning.** Temporal interference via ports of different subsystems is prevented by the strict separation of the execution of the basic connector unit's copy operations from the behavior at ports. The statically defined schedule of the copy operations between input and output ports is independent of the behavior of the two subsystems at the respective inner ports.
- **Spatial partitioning.** The basic connector unit transfers messages autonomously between the memory elements at inner and outer ports. In particular, no explicit message names are employed to control the copying of messages, which could cause an ill-named message of a subsystem to be transferred into a wrong memory element, thereby overwriting messages of another subsystem.

Effective partitioning between the safety-critical and non safety-critical subsystems is a prerequisite for the independent certification of the safety-critical subsystem. If credible evidence for the basic connector unit's ability to prevent interference in the access of network resources (e.g., in case of a failure of a job in the non safety-critical subsystem) is available, then the non safety-critical subsystem need not be certified to the same level as required for the safety-critical subsystem (e.g., class A in DO-178B [29]).

Since the reasoning about the basic connector unit plays such a major role in the certification process, it is of utmost importance to limit the basic connector unit's complexity, i.e. the mental effort for understanding its operation. Therefore, the basic connector unit aims at a minimal functionality for the allocation of network resources. Furthermore, the basic connector unit enforces uniformity of inner and outer port types by imposing the restriction of state messages on all ports. The self-contained nature and idempotence of state messages relieves the designer from the need to devise flow control mechanisms for ensuring exactly-once processing guarantees and thus state synchronization.

4.2 Allocation Layer

The allocation layer transforms the network resources from subsystem granularity to virtual network granularity. The allocation layer is located between the virtual network layer of the secondary connector unit and the outer ports of the basic connector unit. The interface to the basic connector unit are time-triggered ports with state messages. The outer port of the secondary connector unit is connected to the inner port of a basic connector unit. At this port, the secondary connector unit acquires state messages, each being part of the TDMA slot of a component. We denote the state message of component $u.c$: as $m(u.c:)$ and the part of state message $m(u.c:)$ that originates from jobs of component subsystem $:s$ as $m(u.c:s)$. In case of a component with a safety-critical and a non safety-critical subsystem, the identification s identifies one of these two subsystems ($s \in \{0, 1\}$).

The data direction of every message at each component is statically fixed. A state message $m(u.c:)$ and thus all parts $m(u.c:*.*.*)$ of $m(u.c:)$ are written by jobs in component $u.c:$, while jobs at all other components $u.c_r:$ ($c_r \neq c$) read message $m(u.c:)$ and the parts $m(u.c:*.*.*)$ of it (see Figure 7).

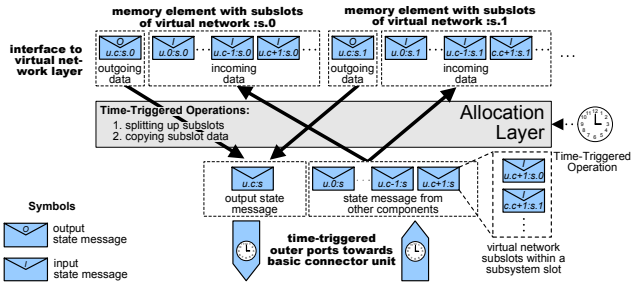

Figure 7. Operation of Allocation Layer

The interface to the virtual network layer are memory elements with state messages that contain incoming (from the basic connector unit) or outgoing data (destined to the basic connector unit) in virtual network granularity (cf. Figure 4). For memory elements with incoming data, there is one message per sender component and virtual network. In addition, there is one memory element with the outgoing data from the component to the virtual network.

The operation of the allocation layer comprises periodic, time-triggered copy operations between the memory elements associated with virtual networks and the memory elements at the outer ports (see Figure 7). These copy operations are controlled by a static schedule, which denotes the global point in time of each operation. In addition, the schedule encodes the involved memory elements, the data direction (from a virtual network's memory element to an outer port or vice versa), and which part of a memory element is read or written.

A state message at an outer port is a compound message, i.e. it contains data from one or more virtual networks. For outer input ports, the allocation layer splits up each compound message into parts belonging to the different virtual networks and copies each resulting part to the respective memory element. The structure of the compound messages is fixed at design time and described by the schedule. The allocation layer uses this description of the syntactic format to control its copy operations. Adversely, the allocation layer reads from memory elements with outgoing state messages at the interface to the virtual network layer and copies these state messages into the compound state message at the time-triggered outer output port.

A properly designed static schedule of copy operations in combination with the the separation of memory elements is the key element for the error containment between virtual networks. Since there is a separate memory element and consequently a dedicated subslot in the underlying TDMA

scheme for each virtual network, every virtual network is assigned a dedicated bandwidth at the underlying time-triggered communication system. A proper schedule at the allocation layer ensures that this bandwidth is made available only to a single virtual network and thus protected by spatial and temporal partitioning form the communication activities of other virtual networks in the component.

4.3 Virtual Network Layer

While the allocation layer is shared between all jobs of the component subsystem, regardless of the DAS these jobs belong to, the virtual network layer and all higher layers comprise a dedicated instance of architectural services for each DAS with jobs in the component subsystem.

The interface of the virtual network layer to the upper layers are message buffers. The realization of these message buffers depends on the employed control paradigm. For a time-triggered virtual network, the upper interface are memory elements with state messages that are overwritten whenever a more recent version arrives. For an event-triggered virtual network, the upper layer interface are event message queues for supporting exactly-once processing of messages with event information. The queues allow to tolerate limited intervals of time during which message service times exceed message interarrival times. For example, application jobs can temporarily request message disseminations at a rate beyond the available bandwidth at the job's subslot. In analogy, a queue allows to tolerate intervals of time during which a job retrieves messages at a lower rate than messages are arriving from other jobs attached to the virtual network. Of course, if no information about the maximum duration of such intervals is available, then message queues cannot be dimensioned or it is necessary to implement explicit flow control.

In order to realize the encapsulation service and establish error containment between jobs within a DAS, there is a dedicated message buffer for each sender job and each receiver job. Each message buffer is exclusively accessible by a single job (e.g., enforced through a memory protection unit), thus ensuring spatial partitioning between jobs. For event-triggered virtual networks, the separate message queues also prevent messages from correct jobs from getting lost, in case a faulty job produces a message load that leads to a queue overflow. Temporal partitioning between jobs is achieved via a static allocation of the bandwidth from the virtual network subslot to job subslots. Such a static allocation decouples the network resources that are available to a job from the behavior [10] of other jobs.

However, for event-triggered virtual networks, the static allocation scheme does not preclude the multiplexing of network resources. A single event message queue can be shared among different types of event messages produced by a job. If a better resource utilization is required for an event-triggered virtual network due to economic reasons, then the static allocation from a virtual network subslot to job subslots can be replaced with a dynamic allocation scheme. Such a solution represents a trade-off between the independence of jobs within a DAS and the ability for multiplexing bandwidth among jobs. Nevertheless, independence between virtual networks is not compromised, because the allocation of subsystem slots to virtual network subslots is always static.

Although the operation of the virtual network layer depends on the type of virtual network that is constructed, we can identify three major categories of activities:

- **Conversion of Control Paradigms.** For event-triggered virtual networks, the virtual network layer performs a conversion of control paradigms by mapping the state message interface of the allocation layer to an event message interface.

- **Establishment of Temporal Properties.** The virtual network layer controls message visibility to ensure that only permanent [13] messages are transferred to upper layers. In addition, the virtual network layer can ensure a predefined message order (e.g., the messages order on a physical CAN bus [24]). The establishment of temporal properties is important for the ability to integrate legacy applications and simplifies the construction of new applications.

- **Switching of Messages** When a component contains multiple jobs of a DAS, the virtual network layer acts as a virtual switch. The virtual network layer controls the flow of messages between jobs, which can reside on the same component (local) and on other components (remote). The virtual network layer accepts messages that are produced by local jobs (arriving from the gateway layer) and remote jobs (arriving from the allocation layer). Similarly, the virtual network layer passes messages to local jobs (via the gateway layer) and to remote jobs (via the allocation layer). The passing of a message to a local job occurs by copying the message to the job's memory element (either a single state message or a message queue) at the interface to the gateway layer. For passing a message to a remote job, the virtual network layer transfers the state message to the memory element (a single state message) of the allocation layer.

  The switching functionality of the virtual network layer is determined by the communication topology of the virtual network, which determines to which jobs an arriving message will be passed. Two common communication topologies are a broadcast communication relationship (e.g., time-triggered virtual network, CAN [28] network) and a point-to-point communication relationship (e.g., TCP/IP). In the first case, the message is transferred to the memory elements of all jobs in the DAS. In the latter case, only a single memory element is subject to the transfer.

4.3.1 Time-Triggered Virtual Network

A time-triggered virtual network layer performs the periodic exchange of state messages at a priori specified global

points in time. A conversion of control paradigms is not required, because the underlying core communication service is also time-triggered. Consequently, the functionality required for the realization of a time-triggered virtual network is kept low, which is important for limiting the certification efforts of safety-critical connector units in which time-triggered virtual networks have to be realized at the component level.

The establishment of temporal properties comprises the determination of message permanence. A message is only forwarded to the upper layers after the core membership services indicates that the sender is operational and all correct receivers have acquired this message. For this purpose, the time-triggered virtual network layer must delay messages for a duration of time that depends on the latency of the membership service.

The switching of messages between jobs is determined by the fact that a time-triggered virtual network always exhibits a broadcast communication relationship. The virtual network layer of a DAS $:s.d$ in a component $u.c$: performs three basic actions (A_1, A_2, and A_3) on the messages from remote and local jobs:

***A1*($s.d.j$). Message transfer from local to remote jobs.** The virtual network layer reads the state message $m(u.c{:}s.d.j)$ produced by job $u.c{:}s.d.j$. The virtual network layer packs this state message into a compound state message, which contains in addition to $m(u.c{:}s.d.j)$ the state messages produced by all other jobs $u.c{:}s.d.*$ of DAS $:s.d$ in component $u.c$:. This compound message forms the virtual network layer's output $m(u.c{:}s.d)$ towards the allocation layer.

***A2*($s.d.j$). Message transfer between local jobs.** The virtual network layer copies the state message $m(u.c{:}s.d.j)$ produced by job $u.c{:}s.d.j$ to all other local jobs $u.c{:}s.d.*$ of DAS $:s.d$ in component $u.c$:.

***A3*($\mathbf{u_r.c_r}$). Message transfer from remote to local jobs.** The virtual network layer takes the state message $m(u_r.c_r{:}s.d)$ originating from DAS $:s.d$ jobs in a remote component $u_r.c_r$: and copies this state message to the memory element associated with each job $u.c{:}s.d.*$ of DAS $:s.d$ in component $u.c$:.

Figure 8 visualizes these three actions. The operation of the virtual network layer is a periodically recurring sequence of these actions. In component $u.c$: the action sequence of a virtual network layer for a DAS $:s.d$ includes for each local job $u.c{:}s.d.j \in \{u.c{:}s.d.*\}$ the actions $A_1(u.c{:}s.d.j)$ and $A2(u.c{:}s.d.j)$. In addition, the action sequence contains action $A_3(u_r.c_r{:})$ for all components $u_r.c_r$: $(u \neq u_r, c \neq c_r)$. The execution of these actions is solely controlled by the progression of time and fixed at design time. In order to avoid a variability in the delay imposed by the communication system, both the ordering of the actions in the sequence and the points in time of their execution should be synchronized with the allocation and gateway layers.

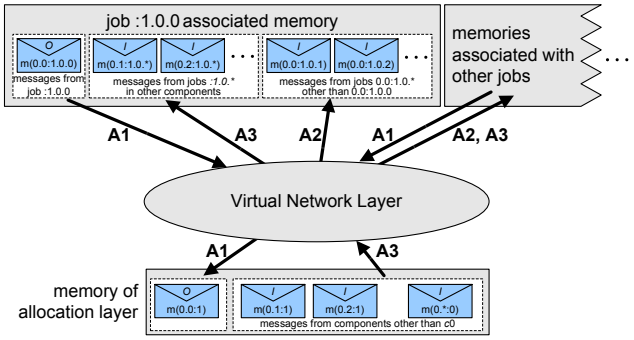

Figure 8. Switching Messages through Virtual Network Layer: *The virtual network layer operates as a virtual switch that transfers messages between local jobs and the allocation layer.*

4.3.2 Event-Triggered Virtual Network

Event-triggered virtual networks are only used in complex connector units, which need not be certified to the highest criticality levels and can thus provide increased functionality at the cost of increased complexity. The event-triggered paradigm facilitates the establishment of cost-effective solutions for the non safety-critical subsystems. In this domain, the high flexibility and resource efficiency of event messages outweigh the lower predictability and increased complexity resulting from the event-triggered control paradigm with its imprecise temporal specifications (e.g., probabilistic queuing models).

For each job the virtual network layer provides a dedicated set of queues to the upper layer (gateway layer). In a DAS with n jobs, each job possesses one queue with outgoing event messages and $n-1$ queues with incoming event messages.

As for time-triggered virtual networks, the interface towards the allocation layer is formed by a state message interface (one state message $m(*.*{:}s.d)$ per component with jobs of the DAS). For an event-triggered virtual network, the virtual network layer employs an *event service* [24] for mapping each state message $m(*.*{:}s.d)$ to a corresponding event message queue. For an outgoing state message $m(u.c{:}s.d)$ at component $u.c$:, the event service periodically reads from the corresponding queue and slices the event messages into packets that can be placed into the state message interface $m(u.c{:}s.d)$. For an incoming state message $m(u_r.c_r{:}s.d)$ $(c_r \neq c \vee u_r \neq u)$, the event service periodically reads from the state message interface $m(u_r.c_r{:}s.d)$ and assembles event messages. Whenever a complete event message has been constructed, the event services inserts this message into the event message queue.

Above the event service, there is an event-triggered switching service. This service is responsible for transferring messages between event message queues – either queues towards the upper layer or queues towards the event service. The operation of the event-triggered switching ser-

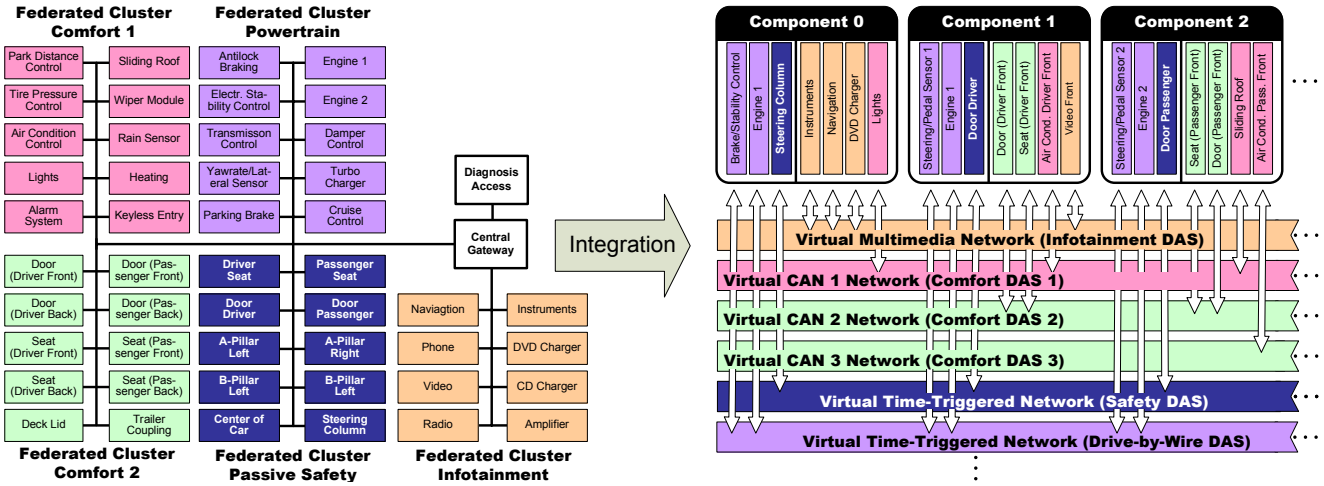

Figure 9. Typical Network of State-of-the-Art Car and Integrated Approach

vice is similar to the time-triggered operation and also consists of three types of transfer activities:

A₁$(s.d.j)$. **Message transfer from local to remote jobs.** The event-triggered virtual network layer reads a message from the event message queue $u.c{:}s.d.j$ and forwards this message to the outgoing message queue leading to the event service. This operation must be performed for each job $:s.d.j$ in component $u.c:$.

A₂$(s.d.j)$. **Message transfer between local jobs.** A message that is transferred to a message queue leading to the event service (action A_1) is also inserted into a dedicated message queue at each job of the DAS in the respective component. A necessary precondition for the insertion of the message into a job's queue is that the job is a receiver of the message according to the communication topology of the virtual network protocol.

A₃$(\mathbf{u_r.c_r})$. **Message transfer from remote to local jobs.** The event-triggered virtual network layer retrieves an incoming message from the queue towards the event service (containing event messages constructed out of $m(u_r.c_r{:}s.d)$) and inserts this message into a dedicated message queue at each local job. In analogy to A_2, a necessary precondition for the insertion of the message into a job's queue is that the job is a receiver of the message according to the communication topology of the virtual network protocol.

The event-triggered virtual network layer operates in a round robin style. In case a job does not have a message in its queue, it is possible to either take the next job or to discard the bandwidth. The first strategy is similar to the proportional share resource allocation described in [35]. The duration of a round robin cycle is variable and depends on the state of the job queues. The maximum duration occurs, in case all job queues are non empty. While the advantage of this strategy is the better bandwidth utilization and the ability to multiplex bandwidth between jobs in a component, a dependency between the jobs of a particular DAS within a component is introduced.

In order to reuse legacy applications without redevelopment efforts, the establishment of the temporal properties that are induced by a particular legacy protocol can be necessary. An example for such a property is the message ordering that results from a specific media access protocol. By taking the messages from both local and remote jobs as an input to an online simulation of the media access protocol, messages of the virtual network can be revealed to upper layers in the exact same order as if the messages would have been transferred via a physical network natively running the legacy protocol. In [24] an algorithm is devised that performs such a simulation and determines the message ordering that would result from the CSMA/CA media access control strategy of the CAN protocol.

5 Virtual Networks in an Automotive Example Application

A typical automotive network is depicted in Figure 9. Multiple federated clusters are connected via a central gateway allowing data exchange and access to the OBD systems of each ECU. The comfort clusters as well as the powertrain cluster are typically implemented via the CAN protocol. The multimedia cluster is frequently based on a protocol with support for streaming audio and video (e.g., MOST), while the passive safety clusters either use CAN or vendor specific communication protocols such as byteflight [2]. Each of these clusters consists of components designed according to the "one function – per ECU" principle in order to simplify system integration and ensure intellectual property protection. Consequently, additional ECUs need to be added to the clusters in order to improve the functionality of the car. However, this trend of increasing the number of ECUs is coming to its limits, because systems are becoming too complex and too costly with the current practice of having each ECU dedicated to a single function.

Figure 9 also depicts a mapping of the above described electronic infrastructure onto the integrated architecture. In order to emphasize the suitability of the proposed DECOS

architecture for safety-critical applications, we exchanged the powertrain DAS with a drive-by-wire DAS. As depicted in Figure 9 the federated clusters are mapped onto corresponding DASs in the integrated system with dedicated virtual networks that are tailored to the particular requirements of the DASs. Each of the three comfort DASs is assigned a corresponding virtual CAN network. Two time-triggered virtual networks are deployed for the safety DAS and the by-wire DAS. A virtual TCP network supports the multimedia applications.

Driven by the need to adapt products to emerging trends and customer requests, manufactures are forced to design new products in ever decreasing design time in order to stand up to competitors. The introduced integrated solution supports this economic requirement by offering the ability to parallelize the development through an independent development of different DASs. From a development perspective, the integrated system architecture permits a much finer subdivision of the overall functionality than a conventional federated system. The subdivision of DASs can occur in such a fine granularity that each DAS becomes finally the responsibility of a single vendor only. When implemented as virtual networks, a higher number of separate communication infrastructures involves only a minor resource overhead. In a physically federated system, however, such a strategy would lead to an explosion of the number of components and wires and thus not only to enormous hardware cost, but also to reliability problems. This condition leads to the paradox situation that although the resulting system is physically integrated, functionally the system is even more federated than before (e.g., on the right hand side in Figure 9 the two comfort DASs have transformed into three DASs).

However, a key point for the independent development are the provided encapsulation services in combination with a diagnostic architecture that allows to trace for faulty jobs (i.e. software faults) in order to precisely assign integration responsibilities to both the system integrator, as well as the vendors, that have independently developed jobs or even a complete DAS to be integrated into the final system. The proposed architecture precludes vendors from transferring integration responsibilities for their jobs to the system integrator, since encapsulation services (both at network and component level) allow to trace integration problems back to the vendor of the respective subsystem which is responsible for the experienced integration problems.

Besides the economic and reliability benefits resulting from the reduction of components, wiring, and contact points, the integrated architecture supports the next step in the evolution of automotive electronics: the introduction of ultra-dependable x-by-wire application subsystems. The integrated architecture supports x-by-wire applications that require a reliability of communication services exceeding the one of the present day physical networks in the federated automotive systems (e.g., CAN). The time-triggered communication technology, which is employed by the base architecture, has become widely accepted to close this reliability gap [32, 7].

Through the provision of virtual CAN and multimedia networks, legacy applications from a previous federated system can be reused to leverage investments. In addition, reliability benefits result form the fault-tolerance mechanisms of the underlying time-triggered base architecture that is employed as a basis for the realization of virtual networks. Examples for these mechanisms are the fault isolation capabilities of the base architecture [15] and media redundancy of the time-triggered physical network.

6 Conclusion

The DECOS integrated architecture aims at combining the advantages of integrated and federated architectures. Functional structuring occurs similar to a federated system by decomposing the overall functionality into a set of nearly independent Distributed Application Subsystems (DASs). The communication services described in this paper enable the interconnection of the jobs in each DAS by a dedicated virtual network with predefined operational properties. All virtual networks share a common physical network, thus reducing the number of wires and connectors. This not only leads to savings with respect to hardware and maintenance cost, but also improves dependability due to the significant proportion of connector and wiring related failures in distributed embedded systems.

Despite a single underlying physical network, we preserve the abstractions of independent network resources through encapsulated virtual networks. A virtual network is private for a DAS, i.e. other DASs cannot perceive or effect the exchanged messages other than those being explicitly exported via a gateway. By exercising this strict control over the interactions between DASs, only the behavior of the DAS's virtual network and the behavior of gateways is of relevance when reasoning about a DAS. The message transmissions on other virtual networks can be abstracted from. This solution reduces the mental effort to analyze and understand a system and allows an independent development of DASs through different vendors.

The encapsulation of virtual networks results from hierarchically subdividing state messages that are transported by the time-triggered communication service of a base architecture. On the one hand, this hierarchic subdivision simplifies the comprehension of the encapsulation process by being able to examine each layer individually and proceed incrementally from a component's state message to the network resources made available to a job. In addition, the different layers of the subdivision process facilitate a modular certification of an integrated system. Based on the argument that lower layers (namely the basic connector unit) prevent any effect onto an ultra-dependable application subsystem, layers employed only for non safety-critical application subsystems (complex connector unit and jobs that build on top) can be omitted from certification to the highest criticality classes.

Virtual networks as described in this paper also permit an optimization of the communication services to different application subsystems, because each application subsystem possesses a dedicated virtual network with a corre-

sponding communication protocol. The operational properties (e.g., event-triggered or time-triggered control) and meta-level properties (e.g., fault-tolerance mechanisms) are tailored to the requirements of the corresponding application subsystem. This strategy permits support for contradicting requirements through separate virtual networks and offers flexibility, because the system integrator can experimentally evaluate different configurations and communication topologies without changing the physical structure of the deployed system.

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] A. Avizienis, J. Laprie, and B. Randell. Fundamental concepts of dependability. Research Report 01-145, LAAS-CNRS, Apr. 2001.

[2] J. Berwanger and M. P. R. Griessbach. Byteflight a new protocol for safety critical applications. In *Proceedings of the FISITA World Automotive Congress*, Seoul, 2000.

[3] M. Brodie and M. Stonebreaker. *Migrating legacy systems*. Morgan Kaufmann, 1995.

[4] A. Deicke. The electrical/electronic diagnostic concept of the new 7 series. In *Convergence International Congress & Exposition On Transportation Electronics*. SAE, 2002.

[5] R. Deline. *Resolving Packaging Mismatch*. PhD thesis, Carnegie Mellon University, Pittsburgh, June 1999.

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

[7] G. Heiner and T. Thurner. Time-triggered architecture for safety-related distributed real-time systems in transportation systems. In *Proc. of the 28th Symposium on Fault-Tolerant Computing*, June 1998.

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

[9] IEEE, New York. *ANSI/IEEE Standard 802.3 – CSMA/CD Access Method and Physical Layer Specifications*, 1985.

[10] C. Jones and et. al. DSoS conceptual model. *DSoS Project (IST-1999-11585) Deliverable CSDA1*, 2002.

[11] L. Kleinrock. *Queuing Systems Volume I: Theory*. John Wiley and Sons, New York, 1975.

[12] H. Kopetz. Why time-triggered architectures will succeed in large hard real-time systems. In *Proc. of the 5th IEEE Workshop on Future Trends of Distributed Computing Systems*, Cheju Island, Korea, Aug. 1995.

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

[14] H. Kopetz. Elementary vs. composite interfaces in distributed real-time systems. In *Proc. of the International Symposium on Autonomous Decentralized Systems*, 1999.

[15] H. Kopetz. Fault containment and error detection in the time-triggered architecture. In *Proc. of the 6th International Symposium on Autonomous Decentralized Systems*, Apr. 2003.

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

[17] H. Kopetz and R. Nossal. Temporal firewalls in large distributed realtime systems. In *Proc. of IEEE Workshop on Future Trends in Distributed Computing*, Tunis, Tunesia, 1997. IEEE Press.

[18] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri. From a federated to an integrated architecture for dependable embedded real-time systems. Technical Report 22, TU Vienna, Real-Time Systems Group, 2004.

[19] H. Kopetz and N. Suri. Compositional design of RT systems: A conceptual basis for specification of linking interfaces. In *Proc. of 6th Symposium on Object-Oriented Real-Time Distributed Computing*, 2003.

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

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

[22] LIN Consortium. LIN specification 2.0. Sept. 2003.

[23] MOST Cooperation, Karlsruhe, Germany. *MOST Specification Version 2.2*, Nov. 2002.

[24] R. Obermaisser. *Event-Triggered and Time-Triggered Control Paradigms: An Integrated Architecture*, volume 22 of *Real-Time Systems Series*. Kluwer Academic Publishers, Oct. 2004.

[25] B. Pauli, A. Meyna, and P. Heitmann. Reliability of electronic components and control units in motor vehicle applications. In *VDI Berichte 1415, Electronic Systems for Vehicles*, pages 1009–1024, 1998.

[26] P. Peti, R. Obermaisser, and H. Kopetz. Out-of-norm assertions. In *Proc. of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2005.

[27] J. Postel. RFC 793: Transmission control protocol, Sept. 1981.

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

[29] RTCA. *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, 1992.

[30] J. Rushby. Partitioning for avionics architectures: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999.

[31] J. Rushby. Bus architectures for safety-critical embedded systems. In *Proc. of the 1st Workshop on Embedded Software*, pages 306–323, Lake Tahoe, CA, Oct. 2001. Springer-Verlag.

[32] J. Rushby. A comparison of bus architectures for safety-critical embedded systems. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, Sept. 2001.

[33] A. Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EETimes*, February 2002.

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

[35] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and C. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proc. of the 17th IEEE Real-Time Systems Symposium*, pages 288–299, Dec. 1996.

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

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