DECOS: An Integrated Time-Triggered Architecture

R. Obermaisser, P. Peti, B. Huber, C. El Salloum

Technische Universität Wien / Institut für Technische Informatik
Treitlstrasse 3, A-1040 Vienna (Austria)

Abstract – *Depending on the physical structuring of large distributed safety-critical real-time systems, one can distinguish federated and integrated system architectures. This paper describes an integrated system architecture, which combines the complexity management advantages of federated systems with the functional integration and hardware benefits of an integrated approach. In order to control complexity, the overall functionality is divided into a set of application subsystems, each with dedicated architectural communication services, allowing developers to act as if they were building an application for a federated architecture. The introduced architecture builds upon the validated services of a time-triggered core architecture, which provides a physical network as a shared resource for the communication activities of more than one application subsystem. The communication resources are encapsulated and multiplexed between application subsystems. In analogy, encapsulated partitions are used to share node computers among software modules of multiple application subsystems. Architectural encapsulation mechanisms ensure that the assumptions and abstractions performed in the functional system structuring also hold after combining the different subsystems on the target platform.*

Keywords: *system architectures, integrated systems, federated systems, time-triggered protocols*

DECOS: Eine integrierte zeitgesteuerte Architektur

Zusammenfassung – *In Abhängigkeit der physikalischen Strukturierung von großen verteilten sicherheitskritischen Echtzeitsystemen können föderierte und integrierte Systemarchitekturen unterschieden werden. Diese Arbeit beschreibt eine integrierte Systemarchitektur, welche die Vorteile föderierter Architekturen in Bezug auf Komplexitätsmanagement mit den Vorteilen eines integrierten Ansatzes (d.h. bessere funktionale Integration und Ressourcenauslastung) vereint. Um die Komplexität des Gesamtsystems zu beherrschen, erfolgt eine Unterteilung in Applikationssubsysteme, die zudem mit spezifischen Architekturdiensten ausgestattet sind. Insbesondere werden die Kommunikationsdienste in deren Funktionalität und Zeitverhalten an die jeweiligen Applikationsanforderungen angepasst. Designer können das System daher in einer Weise entwickeln, wie dies eine föderierte Architektur gestatten würde. Die vorgestellte integrierte Systemarchitektur basiert auf den validierten Diensten einer zeitgesteuerten Kernarchitektur, wobei das physikalische Netzwerk eines einzelnen, verteilten zeitgesteuerten Computersystems als gemeinsame Ressource für die Kommunikationsaktivitäten mehrere Applikationssubsysteme dient. Die Kommunikationsressourcen werden enkapsuliert und zwischen Applikationssubsystemen gemultiplext. Ebenso dienen enkapsulierte Partitionen innerhalb von Komponenten der Aufteilung von Komponentenressourcen (z.B. Prozessorzeit und Speicher) zwischen Softwaremodulen verschiedener Applikationssubsysteme. Die Enkapsulierungsmechanismen der Architektur auf Netzwerk- und Komponentenebene stellen sicher, dass die im Rahmen der funktionalen Systemstrukturierung getroffenen Annahmen und Abstraktionen auch nach der Integration der verschiedenen Subsysteme auf der Zielplattform halten.*

Stichworte*: Systemarchitekturen, integrierte Systeme, föderierte Systeme, zeitgesteuerte Protokolle*

1. Introduction

The progress of the semiconductor industry makes it economically feasible to replace mechanical and hydraulic control systems by embedded computer systems and to increase their functionality far beyond the capabilities of the systems they replace. The automotive industry shares the view that in the next 10 years 80% of its expected innovations will be based on novel IT applications within the car and its environment [Hansen, 2002]. For example, vision-based driver assistance systems that guide the driver in routine traffic scenarios and assist the driver in critical situations can significantly reduce the number and impact of traffic accidents [Leohold and Schmidt, 2004]. In the aerospace industry the deployment of fly-by-wire systems has led to a safer and more dependable air-transport system. A key challenge in next-generation aircraft designs is the reduction of the development and recurring costs of aircraft electronics. If mass-produced Commercial-Off-The-Shelf (COTS) components from the automotive industry that meet the reliability requirements of the aerospace industry were available, this challenge could be tackled. Other control applications (e.g., in railway, industrial, and medical domain) can greatly benefit from the availability of COTS components for the design of dependable control systems. Given the mass-market, cost-constrained design needs of the automotive sector, we believe that this domain offers the dominant applications to drive integrated design approaches. Using this automotive domain as an illustrative case in this paper, we address the major obstacles that hinder the deployment of more advanced embedded systems. We have identified four key obstacles that interlink economic and technical considerations:

- **Electronic Hardware Cost.** Hardware costs are recurring costs that are decisive for the economic success in a mass market. Since in today's system a new node (e.g., Electronic Control Unit (ECU) in the automotive industry) is required for every major new function, the hardware costs escalate significantly with every additional electronic function.
- **Dependability.** The replacement of mechanical subsystems by electronics requires a level of electronics dependability that is higher than the dependability of the mechanical subsystems that are replaced. Such a high dependability can only be achieved by the provision of fault-tolerance.
- **Development Cost.** The unintended side effects between different application subsystems increase significantly the development and integration efforts. Changes in one subsystem often affect the operation of another subsystem. The unproductive glue code needed to support the communication of ECUs located in different networks is substantial and growing. At present the interfaces of many subsystems are ill-specified in the tem-

poral domain [Kopetz and Suri, 2003]. As a consequence modular certification, reuse of software components and the ease of integration are severely compromised.

- **Intellectual Property (IP) Protection.** Suppliers in the automotive supply chain are reluctant to open their IP which is contained in a compiled software module to Original Equipment Manufacturers (OEMs) or other suppliers in order to maintain their competitive edge.

At present the design of the electronic control system within a car is guided by the principle of a separate ECU for every major function (frequently denoted as "1 Function – 1 ECU principle"), which is characteristic for a federated distributed architecture [Swanson, 1998]. It is the objective of this paper to present an integrated architecture for dependable embedded control systems. This integrated architecture is based on a time-triggered core architecture and a set of high-level services that support the execution of already existing legacy applications across standardized technology-invariant interfaces. Such an integrated architecture is being developed in the course of the DECOS Integrated Project as part of the Sixth European Framework Programme.

This paper is structured as follows. Section 2 investigates the advantages and disadvantages of federated and integrated architectures. Section 3 presents the system structuring defined by the integrated DECOS architecture. Through functional federation and physical integration, the DECOS architecture aims at unifying the benefits of both architectural paradigms. Section 4 presents the generic services of the DECOS architecture for the coexistence of multiple application subsystems with different criticalities within a single, distributed computer system. An implementation based on the Time-Triggered Architecture is the focus of Section 5 in order to show the feasibility of the introduced concepts. A discussion on the benefits achieved by the introduced integrated architecture is contained in Section 6.

2. Federated vs. Integrated Architectures

One can distinguish two extreme classes of architectural paradigms for distributed applications, namely *federated* and *integrated architectures*. Real systems are often positioned between these extremes, leaning more to one or the other side. In a totally federated system, each application subsystem (e.g., primary flight control, multimedia domain in a car) has its own dedicated computer system (possibly with internal redundancy), while an integrated system is characterized by the integration of multiple application subsystems within a single distributed computer system. Federated systems have been preferred for ultra-dependable applications due to the natural separation of application functions, which facilitates fault-isolation and complexity management.

Integrated systems, on the other hand, promise massive cost savings through the reduction of resource duplication. In addition, integrated systems permit an optimal interplay of application functions, reliability improvements with respect to wiring and connectors, and overcome limitations for spare node computers and redundancy management. An ideal future system architecture would *combine the complexity management advantages of the federated approach, but would also realize the functional integration and hardware benefits of an integrated system* [Hammett, p. 32]. The challenge is to devise an integrated architecture that provides a framework with generic architectural services for integrating multiple application subsystems within a single, distributed computer system, while retaining the error containment and complexity management benefits of federated systems. Thus, an application subsystem developer can proceed in such a way, as if he would be working in a "classic" federated system.

2.1 Advantages of Federated Systems

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

- **Fault Containment.** Based on the assumption that hardware faults effect an entire node computer, which is accepted for ultra-dependable systems [Lala and Harper 1994, Kopetz 2003], federated systems offer advantages with respect to fault containment. When a hardware fault hits a node computer of a federated computer system, the fault impacts only a single application subsystem. Conversely, a hardware fault hitting a node computer of an integrated system can impact multiple application subsystems, because a node computer in an integrated system can be shared among software modules of multiple application subsystems. In analogy, a hardware fault hitting a physical channel of a federated system affects only a single application subsystem. For developmental faults, better fault containment of a federated system in comparison with an integrated system is a consequence of the platform heterogeneity. The diversity of the employed hardware (e.g., processors, boards, etc.) and software platforms (e.g., operating systems) of the different application subsystems limits the regions of the immediate impact of developmental hardware and software faults.

- **Error Containment.** Exterior error containment addresses error propagation between application subsystems. Since a federated system implements application subsystems via separate computer systems, a computer system for an application subsystem remains functional despite the failure of other computer systems and the corresponding application subsystems. However, error containment within the application subsystem is no mere consequence of the federated system approach, but must be provided by the architecture or the application.
- **Independent Development.** The ability for independent development follows from the near independence of federated systems. In federated systems, only a very limited level of interactions occurs via gateways, thus keeping the need for coordination between different vendors down to a minimum.
- **Complexity Control.** Each application subsystem possesses a functional complexity that is inherent to the application. The functional complexity of an application subsystem when implemented on a target system is dramatically increased in case the architecture does not prevent unintended architecture-induced side effects at the communication system. Since federated systems employ a dedicated computer system for each application subsystem, the complexity of the system is lower compared to the integrated systems approach. The absence of interactions and dependencies between application subsystems reduces the cognitive complexity.

  Consider for example an exemplary scenario with two application subsystems. A merging of application subsystems is common in present day automotive systems in order to restrain wiring-induced cost by multiplexing communication busses. However, if the two application subsystems share a common CAN bus [Bosch, 1991], 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 transmission of the other application subsystem, arguments concerning the correct temporal behavior must be based on an analysis of both application subsystems. In a federated system, on the other hand, unintended side effects are ruled out, because the two application subsystems are assigned to separate computer system.

2.2 Advantages of Integrated Systems

Among the major advantages of integrated systems are hardware cost reduction, dependability improvements, improved coordination between application subsystems.

- **Hardware Cost Reduction.** In contrast to federated systems, which require a dedicated computer system for each application subsystem, integrated systems facilitate the multiplexing of hardware resources. In the automotive area, modern luxury cars contain almost a hundred computer nodes. For example, the BMW 7 series cars contain up to 70 ECUs [Deicke, 2002]. The distributed ECUs are interconnected via communication networks with different protocols (e.g., CAN [Bosch, 1991], MOST [MOST Cooperation, 2002]), physical layers, bandwidths (10 kbps – 25 Mbps), and dependability requirements. 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.
- **Dependability Improvements due to Reductions of Wiring and Connectors.** By tackling the "1 Function – 1 ECU" problem, the reduction in the overall number of ECUs also leads to increased reliability by minimizing the number of connectors and wires. Field data from automotive environments has shown that more than 30% of electrical failures are attributed to connector problems [Swingler and McBride, 1998].
- **Fault-Tolerance.** Replicated hardware is necessary to tolerate hardware faults in both federated and integrated systems. However, in a federated system these resources are available only to a single computer system out of the numerous loosely coupled ones. A computer system dedicated to a particular application subsystem can fail, although the overall number of spare node computers would be sufficient to tolerate a given number of node computer failures. In an integrated system, resources are universally available among the different application subsystems.
- **Improved coordination of application subsystems.** Technical innovations often require the tight coupling of subsystems to realize emerging service beyond domain borders. In contrast to federated systems, integrated systems permit a tactic coordination of tightly coupled control activities. For example hybrid cars are gaining more and more momentum in the automotive industry due to the acceptance of the customers as an environmental friendly and economically feasible alternative. Hybrid technology requires an interplay of the electric motor controller, the gasoline engine controller, the battery management system, the brake system controller, and the power split device controller [Toyota, 2004]. Consider for instance integrated regenerative braking as an example for the complex coordination between the components. Here, the standard brake system (e.g., disc braking systems) needs to be tactically coordinated with the braking recovery system that sends electrical energy to the rechargeable batteries.

Another example from the automotive industry is the coordination of different application subsystems for improving the quality of service with respect to passenger safety. The passive safety mechanism (Pre-Safe) of the Mercedes S-class [Birch, 2003] tensions seat-belts, realigns seats to safer positions, and closes an open sun roof when sensors detect possibly hazardous situations. The system correlates information of existing car dynamics sensors in order to determine hazardous situations such as skidding, emergency braking, or avoidance maneuvers.

3. System Structure of a DECOS System

This section describes the structure of a distributed embedded real-time system that is based on the DECOS integrated architecture. We describe both the functional and physical structuring of the overall system into nearly-decomposable parts, which possess a high inner connectivity and weak connections in between. 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 [Simon, 1996, 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 subsystems with linking interfaces that are precisely specified in the value and time domain [Kopetz and Suri, 2003]. Near-independence is the ability of a subsystem to serve its purpose independently from the detailed structure of other subsystems, i.e. only based on the specification of the linking interfaces of the subsystems.

The system structuring in DECOS is guided by the ideas of the Model Driven Architecture (MDA) [OMG, 2001] introduced by the Object Management Group (OMG). The MDA proposes platform-independent and platform-specific viewpoints – denoted as Platform Independent Model (PIM) and Platform Specific Model (PSM) – in order to separate the application logic from the underlying platform technology. The PIM captures the functional system structure of the application subsystems, while abstracting away implementation details. The PSM denotes the physical system structuring and is expressed in terms of the specification model of the target platform.

3.1 Functional System Structure

The PIM is a formal specification of the structure and function of a system that abstracts away technical details [OMG, 2001]. The PIM is a hierarchical model, the construction of which is supported by the integrated architecture through the definition of basic building blocks and rules for the interaction between these building blocks.

For the provision of application services at the controlled object interface, the real-time computer system is divided into a set of nearly-independent subsystems, each providing a part of the computer system's overall functionality. We denote such a subsystem as a *Distributed Application Subsystem (DAS)*, since the implementation of the corresponding functionality will most likely involve multiple node computers that are interconnected by an underlying communication system. The implementation as a distributed system is a prerequisite for establishing fault-tolerance by redundantly performing computations at spatially separated node computers that fail independently. Furthermore, a distributed solution becomes a necessity, when the resource requirements of the application providing the subsystem's functionality exceed the available resources of a single node computer.

The identification of DASs is guided by functional coherence and common criticality of subsystems. Each DAS should provide a meaningful application service (e.g., brake-by-wire service of a car) to its users at the controlled object interface. By associating with a DAS an application service that is relevant in the actual application context, the mental effort in understanding the various application services is reduced.

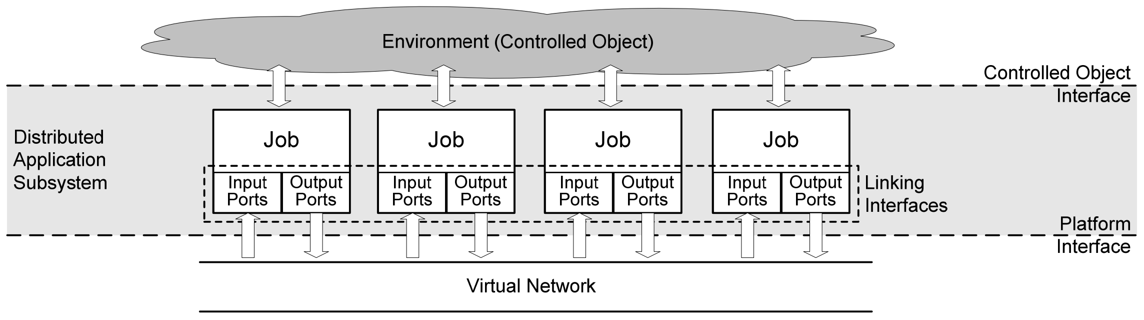
Environment (Controlled Object)
Controlled Object Interface
Distributed Application Subsystem
Job
Job
Job
Job
Input Ports
Output Ports
Linking Interfaces
Platform Interface
Virtual Network

Figure 1: Distributed Application Subsystem

In analogy to the structuring of the overall system, we further decompose each DAS into smaller units called *jobs* (see Figure 1). A job is the basic unit of work that employs a *virtual network* for exchanging information with the other jobs of the DAS, thus working towards a collective goal. A virtual network is an overlay network on top of a time-triggered physical network and will be discussed in Section 4.1.

A major focus of the PIM is also the specification of the linking interfaces of the jobs within a DAS, as well as the specification of the interfaces between DASs. A linking interface consists of one or more ports through which jobs access virtual networks. The linking interface specification captures operational properties (syntax, temporal constraints, interface state) and meta-level properties (e.g., dependability) of the messages exchanged via the port. The linking interface specification enables the independent development of jobs through separate vendors, because it provides each job developer with the required information about obligations and available services at the interfaces to jobs. Thus, a job developer knows which services must be provided and which services can be relied on in order to provide the job's own services. In addition, a precise linking interface specification is also a prerequisite for reuse of jobs in different systems.

3.2 Physical System Structure

The PSM extends the specifications in the PIM with the details that define how the system uses the available platform resources. During the PIM-to-PSM transformation, the functional elements, i.e. the building blocks of the PIM (DASs, jobs, virtual networks), are mapped to the physical building blocks of the platform. These building blocks include clusters, node computers, partitions, and (physical) networks. Each DAS is assigned to one or more clusters, each job of a DAS is allocated to a partition within a node computer of these clusters and each virtual network is allocated to a physical network of a cluster. This allocation process for the generation of the PSM out of a PIM is constrained by dependability requirements, constraints with respect to hardware resources, and constraints with respect to network resources.

A *cluster* is a distributed computer system that consists of a set of node computers interconnected by a network. A cluster supports one or more DASs and each of these DASs receives a share of the communication and computational resources of the cluster. The allocation of communication resources occurs via virtual networks. A virtual network exploits a specific part of the bandwidth from the physical network in order to construct a private communication channel for the DAS. The computational resources are available via protected execution environments within node computers.

A *node computer* 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), which interacts with its environment by exchanging messages. The behavior of a node can be specified in the domains of value and time. A node contains one or more partitions. A *partition* is an encapsulated execution space within a node computer with a priori assigned computational (e.g., CPU, memory, I/O) and network resources (e.g., network bandwidth) that can host a job. Partitions are the target of job allocation and each job is always assigned in its entirety onto a partition, i.e. a job is never fragmented onto multiple partitions.

3.3 Physical Integration and Functional Federation

The DECOS architecture allows the decoupling of the functional structuring from the physical structuring. In contrast to a federated system, an integrated system assigns its *n* DASs to less than *n* distributed computer systems, ideally to a single one (i.e. n=1) [Peti et al., 2005]. Consequently, the functional system structure itself does not enforce a certain physical system structure. Physical collocation (e.g., application software on a particular node computer) does not imply a close functional relationship. Adversely, physically separate parts of an integrated architecture can represent common functionality. The system structure, as proposed above, leads to an overall system that is functionally federated, but physically integrated.

On the one hand, the node computers are shared among the jobs of more than one DAS, while the physical network is shared between the virtual networks of the DASs allocated to the cluster. Thereby, the presented system architecture establishes the strengths of integrated systems, such as reduced hardware cost through the sharing of node computers and physical networks.

On the other hand, the overall system is functionally structured like a conventional federated architecture. The overall system contains multiple DASs each with a dedicated communication infrastructure (provided via a virtual network) that is tailored to the requirements of the DAS via its functionality (e.g., communication topology), temporal performance (e.g., bandwidth, latencies), and dependability (e.g., redundancy degree). In order to preserve the assumptions and abstractions preformed in the functional system structuring, the DECOS integrated architecture encapsulates the communication and computational resources:

- **Encapsulation of communication resources preserves abstraction of having a dedicated network for a DAS.** In case there would only be a single DAS accessing a physical network – as in a federated system – encapsulation of the communication resources would follow naturally. In the integrated architecture we preserve this abstraction through the encapsulation of the communication resources, which are provided by virtual networks on top of a time-triggered physical network. Message exchanges are only visible to the jobs of the respective DAS and each virtual network exhibits predefined temporal properties that are independent from the communication activities at other virtual networks. The realization of the encapsulation of the communication resources will be the focus of Sections 4 and 5.
- **Encapsulation of computational resources preserves abstraction of having only one job per ECU.** The execution environment within an integrated node computer establishes for each job allocated to a node computer a corresponding partition. According to the fault hypothesis of the DECOS architecture [Kopetz et al., 2004], a partition forms a fault

containment region with respect to software faults. The realization of the encapsulation of the computational resources will also be discussed in Sections 4 and 5.

In conjunction with these architectural encapsulation services for protecting the communication and computational resources of different functional entities (e.g., DASs, jobs), the strategy of functional federation and physical integration aims at bringing to the integrated system the benefits of the federated paradigm. Functionally, a DECOS system can even be more federated than present day federated systems. For example, in the automotive industry the system is typically structured into large domains, such as the powertrain, comfort, and multimedia domains. By exploiting the services of the DECOS architecture, these domains can be split into more manageable parts, such as a steering DAS, a braking DAS, a gearbox DAS, and an engine control DAS in case of the powertrain domain.

Since virtual networks inhibit no additional wiring or connectors and higher numbers of DASs do not require additional clusters or node computers, designers are free to devise small, manageable DASs. Since smaller DASs are easier to understand and yield a finer granularity of the protection mechanisms for the computational and communication resources, the benefits of federated systems are even amplified in the presented system architecture.

4. Generic Architectural Services

The DECOS architecture offers a framework for the development of distributed embedded real-time systems integrating multiple DASs with different levels of criticality. The DECOS integrated architecture aims at offering to system designers generic architectural services, which provide a validated stable baseline for the development of applications. It is based on a time-triggered core architecture that supports the safety requirements up to the highest considered criticality class by providing the following core services:

- **Time-triggered communication service:** The time-triggered communication service is available via the temporal firewall interface [Kopetz and Nossal, 1997], which eliminates control error propagation by design and minimizes coupling between node computers. The purpose of this service is the transport of state messages from the Communication Network Interface (CNI) of the sending node computer to the CNIs of the receiving node computers.
- **Fault tolerant clock synchronization:** Due to clock drifts, the clock times in an ensemble of clocks will drift apart, if clocks are not periodically resynchronized. Clock synchronization is concerned with bringing the values of clocks in close relation with respect to each other. The clock synchronization is a fundamental service in a time-triggered sys-

tem, since all activities are controlled by the progression of time [Kopetz and Ochsenreiter, 1987].

- **Error containment:** Although a Fault Containment Region (FCR) can demarcate the immediate impact of a fault, fault effects manifested as erroneous data can propagate across FCR boundaries. For this reason, the system must also provide error containment [Lala and Harper, 1994]. To avoid error propagation by the flow of erroneous messages the error detection mechanisms must be part of different Fault Containment Regions (FCRs) than the message sender. Otherwise, the error detection service can be affected by the same fault that caused the message failure.
- **Consistent diagnosis of failing nodes:** The membership service provides consistent information about the operational state (correct or faulty) of nodes [Cristian, 1991]. The membership service is based on the a priori knowledge about the points in time of the time-triggered message exchanges. Every receiver knows a priori when a message of a sender is supposed to arrive, and interprets the arrival of the message as a life sign at the membership point of the sender. From the arrival of the expected messages at two consecutive membership points, it can be concluded that a node was operational during the interval delimited by these membership points.

Any architecture that provides these core services can be used as a core architecture [Rushby, 2001] for an integrated distributed architecture. It has been demonstrated by formal analysis [Rushby, 2002] and experiments [Hexel, 1999] that the Time-Triggered Architecture (TTA) [Kopetz and Bauer, 2003] is appropriate for the implementation of applications of the highest criticality class in the aerospace domain according to RTCA DO-178B [RTCA, 1992] and consequently meets the requirements of safety-critical embedded systems. It is thus possible to integrate, within the Time-Triggered Architecture (TTA), a number of nearly autonomous DASs of differing criticality up to the highest criticality class.

Based on the core services, the DECOS integrated architecture realizes high-level architectural services, which are DAS-specific and constitute the interface for the jobs to the underlying platform. Among the high-level services are the virtual network service and the gateway service.

4.1 Virtual Networks

A *virtual network* is an overlay network that is established on top of a physical network [Obermaisser et al., 2005]. In the DECOS integrated system architecture, we provide virtual networks on top of the time-triggered core communication service of the core architecture. To achieve the complexity management and fault isolation 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 (e.g., broadcast service, request/reply messages), the operational properties (e.g., bandwidth, latencies), and the namespace (e.g., CAN identifiers). Furthermore, each virtual network is encapsulated, thus 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 node-level error containment to the network-level. The multiplexing of node computers for multiple jobs not only requires error containment with respect to the computational resources (e.g., processor and memory resources), but also error containment for a node computer's network resources.

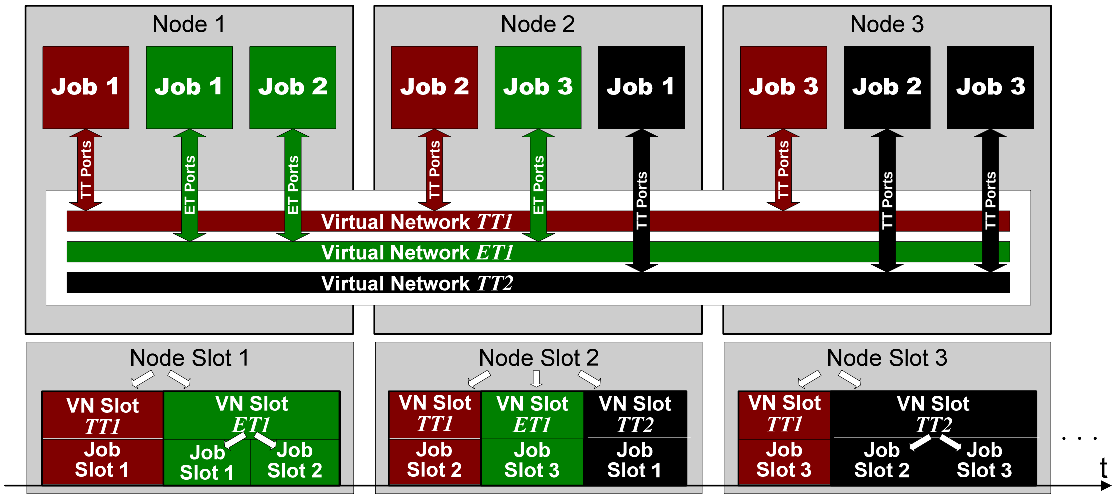

Figure 2: Hierarchic Subdivision of Communication Resources

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 2). 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 node computer is assigned a unique *node slot* that periodically recurs at a priori specified global points in time. A node computer sends messages during its node slot and receives messages during the node slots of other node computers. A sequence of node slots, which allows every node computer in an ensemble of node computers to send exactly once, is called a TDMA round.

We further subdivide each node slot in correspondence to the functional structuring of a DECOS system. In a first step, the node slot is subdivided into subslots for the virtual networks. Such a *virtual network slot* contains those messages that are produced by the jobs in the node computer 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 smaller subslots denoted as *job slots*. When a node computer 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 node computers, 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.

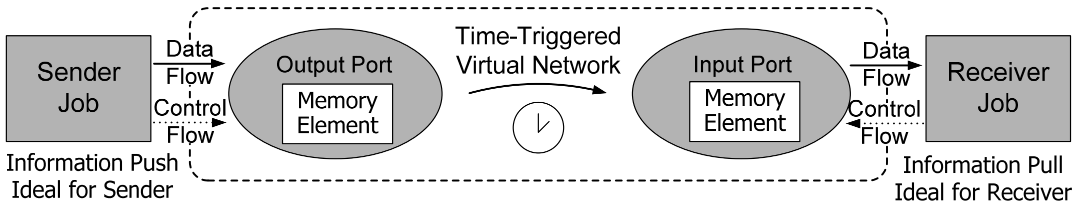

Figure 3: Temporal Firewall Interface Provided by a Time-Triggered Virtual Network

We distinguish between two fundamentally different types of virtual networks: event-triggered and time-triggered virtual networks. A *time-triggered virtual network* is designed for the periodic exchange of state messages. The temporal firewall concept [Kopetz and Nossal, 1997] is used as the interface between a job and a time-triggered virtual network (see Figure 3). The sender acts according to the information push paradigm [Deline, 1999] 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* [Kopetz, 1997]. 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* [Kopetz, 1999], i.e. a unidirectional data flow involves only a unidirectional control flow.

Event-triggered virtual networks are 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 [Kopetz, 1997]. The interactions between a sender and a receiver via an event-triggered virtual network are depicted in Figure 4. 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 (information pull via polling for messages) or the event-triggered virtual network presses received messages into the job (information push via interrupt mechanism).

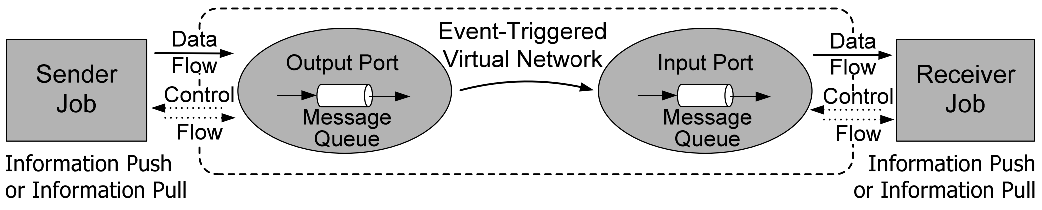

Figure 4: Message Exchange between two Jobs through an Event-Triggered Virtual Network

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.

4.2 Gateways

In the DECOS architecture, gateways are a generic architectural service for the controlled coupling of virtual and physical networks by selectively forwarding information contained in the messages received at the input ports of one network onto the output ports towards the other network (and vice versa in case of a bidirectional gateway). Gateways give the designer the ability to coordinate application services and exploit redundancy (e.g., common sensors) in a system to either improve reliability or reduce resource duplication. Gateways are transparent to the jobs at the application level, thus avoiding any complication of the application software. Functionality is moved to the architecture level, where it is developed and validated once and for all.

Central to gateways in the DECOS architecture is the concept of the gateway repository, which is a real-time database that is maintained by the gateway and contains state variables and event queues updated by the contents of received messages [Obermaisser et al., 2005b]. For the gateway specification, we employ an extension of deterministic timed automata with corresponding execution semantics [Obermaisser and Peti, 2005]. Thereby, timed automata can form the input to automatic code generation and be executed as part of an architectural service, which offers a generic framework for the realization of virtual gateways.

The need for gateways in the DECOS architecture arises either through the functional or physical system structuring. In case the coordination of the behavior of two DASs requires an information exchange between them, the functional structuring of the system into DASs induces so-called *virtual gateways*. The purpose of introducing this new architectural element instead of directly connecting all jobs through a single virtual network is twofold:

1. **Encapsulation of virtual networks.** A particular message is only redirected from one virtual network to a second one, if the virtual gateway has been explicitly parameterized to forward this message. Hence, jobs perceive only those messages from another DAS, which are required for realizing the DAS's application service. This strategy minimizes the mental effort for understanding a DAS and its constituting jobs, because the designer can abstract from all messages in other DASs that are not explicitly imported through a virtual gateway. In addition, the provision of a separate encapsulated virtual network for each DAS establishes clear error propagation boundaries. Each virtual network has its own guaranteed communication resources, which are assigned through the static subdivision of the communication slots. A faulty job is thus unable to affect the temporal properties (e.g., bandwidth, latencies) of messages exchanges at the virtual networks of other DASs. The virtual gateway represents the only potential path for error propagation, but employs error detection mechanisms to block the redirection of faulty messages [Obermaisser and Peti, 2005].
2. **Resolving of property mismatches.** Since each virtual network is an overlay network that can have its own communication protocol (e.g., CAN, TCP/IP) and name space, virtual networks cannot simply be directly interconnected. The coupling of virtual networks requires a mediating entity – the virtual gateway – that resolves the property mismatches. Examples for common property mismatches are naming incoherences (e.g., use of the same name for different messages at two virtual networks) and different temporal specifications (e.g., time-triggered vs. event-triggered control).

In addition to virtual gateways, the physical structuring of the overall system into a set of clusters, each with a separate physical network, induces so-called *physical gateways*. A physical gateway is realized by a node computer with physical connections to two physical networks. A physical gateway is needed for the information exchange between jobs, if the jobs of a DAS are allocated to more than one cluster. For example, consider an existing automotive application subsystem realized by a cluster of node computers with a CAN network. A designer may decide to perform a gradual evolution by leaving a part of the cluster as it is, while integrating the software of some of the CAN-based legacy nodes as jobs into integrated node computers. The latter part of the legacy cluster is "virtualized" by replacing the CAN nodes with jobs interconnected by a virtual CAN network. However, since the designer decides to also reuse complete CAN nodes (i.e. hardware and software), this part of the legacy cluster is interconnected by a physical CAN network.

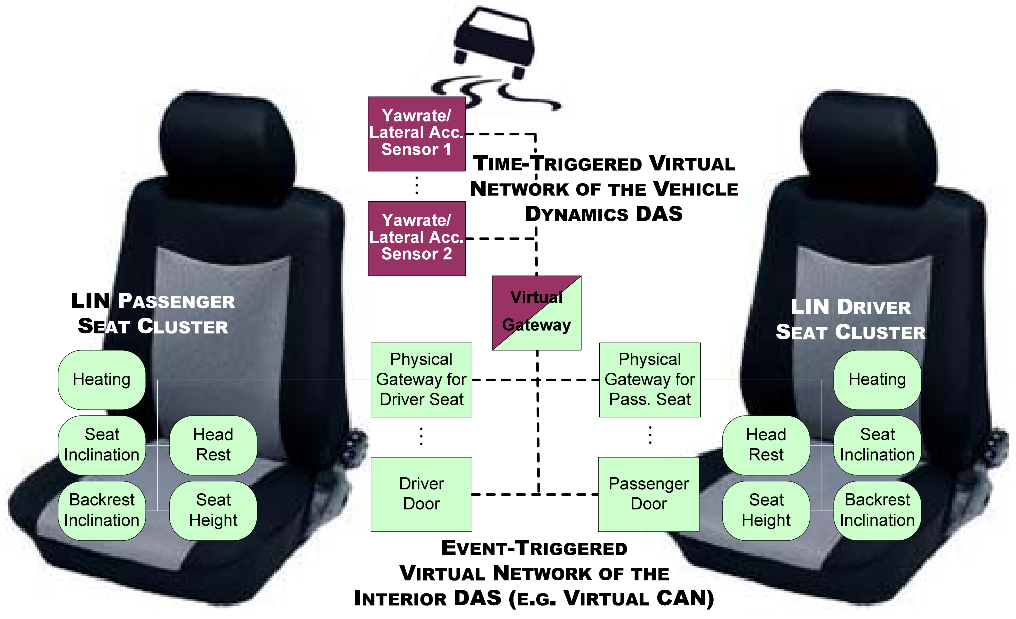
Yawrate/ Lateral Acc. Sensor 1
Yawrate/ Lateral Acc. Sensor 2
Time-Triggered Virtual Network of the Vehicle Dynamics DAS
Virtual Gateway
LIN Passenger Seat Cluster
LIN Driver Seat Cluster
Heating
Seat Inclination
Head Rest
Backrest Inclination
Seat Height
Physical Gateway for Driver Seat
Driver Door
Physical Gateway for Pass. Seat
Passenger Door
Head Rest
Seat Inclination
Seat Height
Backrest Inclination
Heating
Event-Triggered Virtual Network of the Interior DAS (e.g. Virtual CAN)

Figure 5: Exemplary Virtual and Physical Gateways

Figure 5 shows an automotive example employing both virtual and physical gateways. The interior DAS in this example comprises the body electronics of the passenger compartment. The functionality of this DAS includes the control of the doors (e.g., mirrors, window lifters), the sliding roof, the trunk lid, the climate control, and the lighting of the passenger compartment. The interior DAS uses an event-triggered virtual network as its communication infrastructure. This virtual network is connected via physical gateways to the field bus networks embedded in the seats. In addition, the exemplary automotive system employs a virtual gateway for coupling the interior DAS with the vehicle dynamics DAS. The virtual gateway between the vehicle dynamics DAS and the interior DAS permits to improve quality-of-control and realize pre-crash functionality, e.g., by tensioning seat-belts and realigning seats as described in [Birch, 2003].

5. Prototype Implementation of a DECOS Cluster

In the following we describe our implementation setup. We will explain the constituting integrated DECOS node computers, which are multiprocessor nodes with dedicated hardware elements for core architectural services and application software. Emphasis is placed on the encapsulation of computational and communication resources by means of temporal and spatial partitioning.

5.1 HW/SW Platform for the Implementation of an Integrated Node Computer

A DECOS node computer is a multiprocessor node as depicted in Figure 6. It consists of a *Basic Connector Unit* (BCU) and two application computers, namely one for each of the two node subsystems (safety-critical and non safety-critical). Each application computer hosts the application software (i.e. jobs) in conjunction with the corresponding high-level architectural services. The purpose of the BCU is the primary allocation of network resources in order to enable an effective partitioning between the safety-critical and the non safety-critical subsystem of a DECOS node computer. This design facilitates modular certification [Rushby 2001b, Kopetz et al., 2004], because applications with different criticality levels are assigned to separate hardware elements. The BCU, which also contains a time-triggered communication controller, is realized using a single hardware element – the TTP MPC855 single board computer. This single board computer is equipped with an MPC855 PowerPC from Freescale clocked with 80 MHz and provides a C2 (AS8202) TTP communication controller.

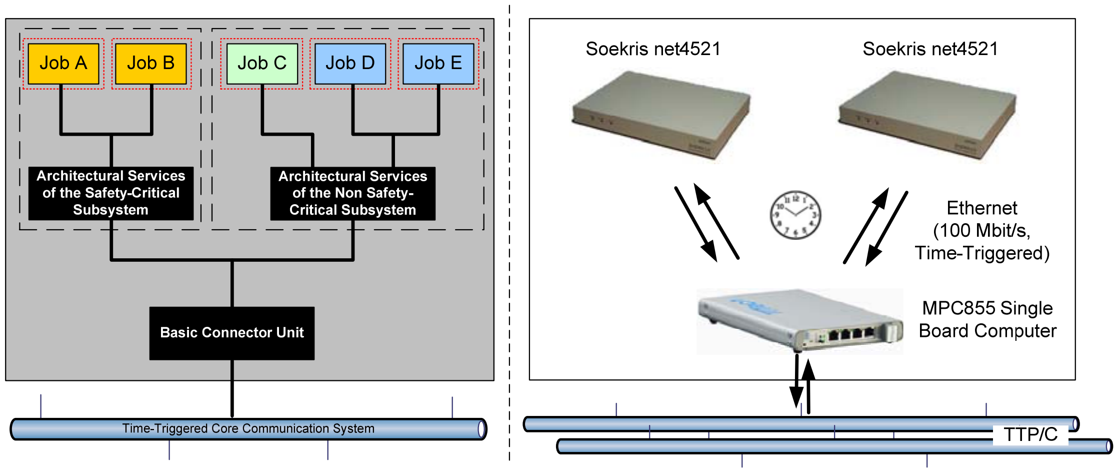
Job A
Job B
Job C
Job D
Job E
Architectural Services of the Safety-Critical Subsystem
Architectural Services of the Non Safety-Critical Subsystem
Basic Connector Unit
Time-Triggered Core Communication System
Soekris net4521
Soekris net4521
Ethernet (100 Mbit/s, Time-Triggered)
MPC855 Single Board Computer
TTP/C

Figure 6: Realization of a DECOS Node

Each application computer is implemented on a *Soekris net4521* embedded computer from Soekris Engineering, which is based on a 133 MHz 486 class ElanSC520 processor from

AMD. The interconnection between the application computers and the BCU is performed using time-triggered Ethernet (100 Mbps).

We deploy on all embedded computer nodes the real-time Linux variant LXRT/RTAI [Beal, 2000] as the operating system and execution platform. The operating system in conjunction with the architectural services provides encapsulated partitions for the execution of the jobs of the respective node subsystem. Since according to the fault hypothesis of DECOS [Kopetz et al., 2004], a job is treated as FCR for software faults, these partitions have to ensure that a faulty job cannot interfere with any other job. This includes the computational as well as the communication resources.

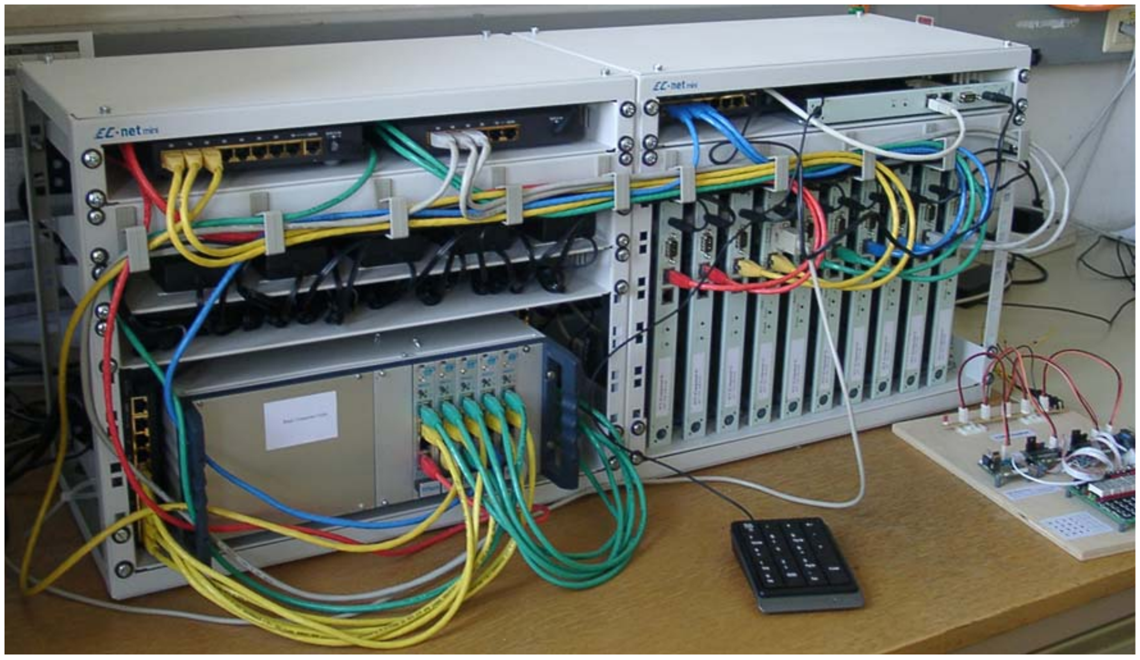

Figure 7: Prototype Cluster

The complete prototype setup as depicted in Figure 7 comprises a cluster with five integrated node computers using TTP [Kopetz, 1999b] as the time-triggered core network. TTP provides the four mandatory core services as stated in Section 4. The resulting prototype setup contains 15 single board computers, 5 Ethernet switches of the node-internal networks, and 2 Ethernet multi-port repeaters of the time-triggered core network (TTP with Ethernet as physical layer via Media-Independent Interface (MII)).

5.2 Partitioning of the Computational Resources

Since we physically separate safety-critical and non safety-critical subsystems in our prototype implementation, a disruption of a safety-critical job via computational resources caused by a failure of a job in the non safety-critical subsystem is prevented by design. However, jobs and high-level services of the same node subsystem share the same computational resources. Therefore, an execution environment on the application computer is required that enables the partitioning of the computational resources and ensures that fault isolation holds in the case of software failures of jobs.

Partitioning addresses two dimensions: *spatial* and *temporal partitioning* [Rushby 1999]. Spatial partitioning deals with preventing jobs from overwriting memory elements of other jobs and preventing jobs in interfering in the access of devices [Rushby 1999]. In our prototype implementation we utilize the Memory Management Unit (MMU) of the AMD Elan CPU for establishing spatial partitioning. MMUs normally distinguish between an unrestricted access mode to the memory, denoted as *supervisor mode*, and a more restrictive *user mode*. Unlike many real-time operating systems, RTAI Linux permits the execution of real-time tasks in user mode (using the Linux Real-Time (LXRT) extension of RTAI), which is a mandatory requirement for establishing spatial partitioning, because the execution of all jobs and high-level services in supervisor mode (e.g., as Kernel modules in Linux) would bypass the protection mechanisms of the MMU. Additionally, since jobs in DECOS have exclusive access to I/O, explicit synchronization mechanisms for protecting devices are not required.

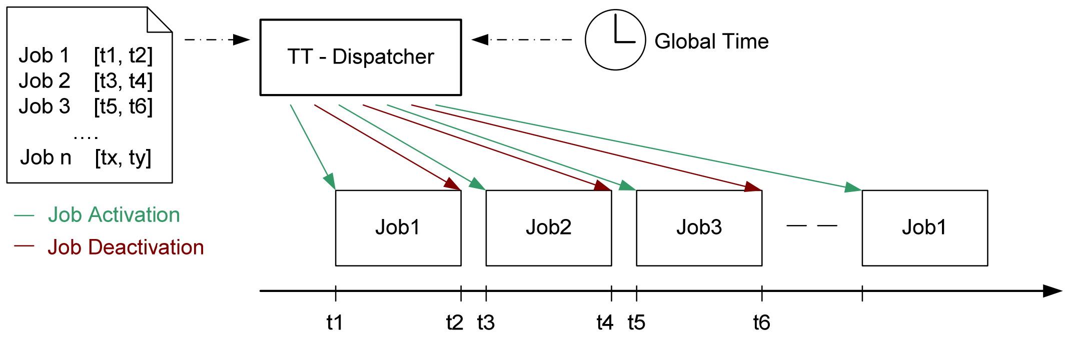

Figure 8: Temporal Partitioning in the DECOS Prototype Implementation

Temporal partitioning on the other hand ensures that one job cannot delay another job by holding a shared resource (e.g. the processor). Our prototype implementation employs a static round-robin schedule to guarantee that the temporal requirements of all jobs are fulfilled (see Figure 8). This off-line generated schedule defines time budgets for the execution of all jobs and high-level services on the application computer.

Although RTAI offers mechanisms for periodical time-triggered scheduling of real-time processes, it lacks of the possibility to specify a deadline or a maximum execution time until a real-time process has to have finished its execution. Therefore, we have developed a time-triggered dispatcher extending the RTAI/LXRT functionality that activates high-level services and jobs according to a static timetable and ensures that their assigned execution times are not exceeded [Huber et al., 2005]. The time-triggered dispatcher is realized as an RTAI Linux Kernel module (note that jobs and high-level services are LXRT tasks and executed in user space) and therefore runs with a higher priority as all of the LXRT tasks. According to the static schedule, the dispatcher activates the respective jobs or high-level services and releases the CPU. After the pre-specified time-budget has elapsed, the dispatcher is invoked again and forces the jobs or high-level services to release the CPU again. Due to this enforced preemption by RTAI and the dispatcher task, the LXRT real-time jobs do not need to act cooperatively and release the processor. Thus, it is prevented by our execution environment that a faulty job delays other jobs by holding the processor longer than specified.

5.3 Partitioning of the Communication Resources

Virtual networks are established through the static subdivision of the communication resources (see Section 4.1). In the implementation we perform this subdivision incrementally in two steps. Firstly, the BCU splits the bandwidth between the two application computers implementing the safety-critical and non safety-critical node subsystems. The messages produced by each application computer are packed into one of the subslots of the node computer's communication slot at the underlying time-triggered communication system. Secondly, the virtual network service, which is realized as middleware in the application computer, further subdivides the subslot of an application computer into subslots for the jobs. Whenever a job requests a message transmission at the virtual network service, the message is packed into the subslot belonging to this job.

The interface between a job and the virtual network middleware consists of one or more state or event ports realized in shared memory regions protected by digital signatures. A state port contains a state variable that is accessed by the virtual network middleware at a priori specified global points in time. The state variable is either updated by the virtual network service (data copied from CNI memory region to the input state port) or read by the virtual network service (data copied from output state port to the CNI memory region). An event port, on the other hand, contains a message queue into which messages that are read from the CNI memory region are inserted by the virtual network service in case of an input event port. For an output event port, the virtual network service retrieves messages from the queue and writes them into the CNI memory region.

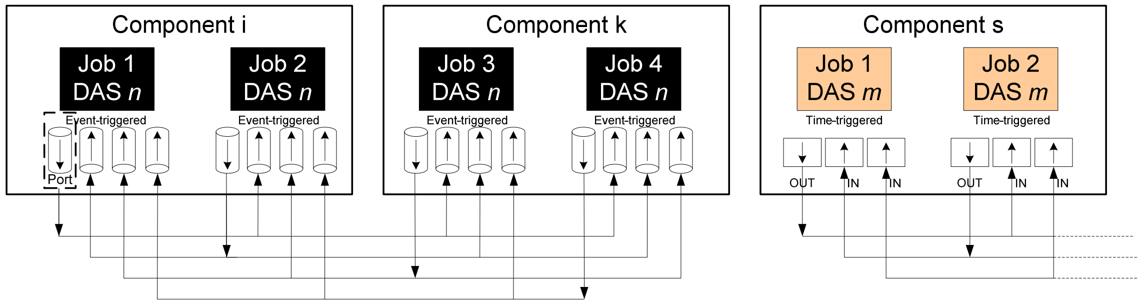

Figure 9: Implementation of Partitioning at Virtual Network Level

As depicted in Figure 9, a job contains a dedicated input port for each other job of the DAS that sends messages to the job (for more detail see also [DECOS, 2005]). Providing a dedicated input port for each sender at all receivers is a key element for error containment within the DAS:

1. **Spatial partitioning by handling masquerading failures.** Masquerading is defined as the sending or receiving of messages using the identity of another sender without authority [Coulouris et al., 1994]. Systems that rely on an explicit name stored in a message to identify the transported message are vulnerable to masquerading failures. Such a failure results in the possibility that a single faulty node computer can masquerade other node computers, without the receiver having a chance to detect the fault. For example, if a bit in the name of a message to-be-sent that is stored in the sending node computer is incorrect, this message could, after arrival at its destination, overwrite correct messages at correct receivers [Kopetz 2003, Driscoll and Hoyme 1993]. Consequently, even an external observer cannot identify the faulty sender. Such a forged message identifier can have a significant impact on the application.

   The virtual network addresses masquerading failures through a static association between jobs and sending slots both at the time-triggered inner-node network and at the time-triggered core network. In addition, each of these sending slots is associated with a corresponding input port and thus a corresponding message buffer. Jobs can exploit the provision of separate input ports for the detection of masquerading failures. Each input port exclusively stores messages from a single sender job only. Although a sender job can transmit a message with an incorrect message name (e.g., a CAN identifier that is reserved for another job), such a message is exchanged within the sending slot of the sender job and reaches the sender job's input port at the receiver job. Hence, the presence of the message at the wrong input port enables the receiver job to detect such a masquerading failure.
2. **Temporal partitioning between jobs.** Temporal partitioning requires the communication latencies and communication jitter for messages sent by one job to be independent from the communication activities of other jobs. In case of time-triggered ports, the prototype implementation ensures temporal partitioning by performing the updates of real-time images at a priori fixed global points in time. In case of event-triggered ports, separate input ports and thus separate queues ensure that the queuing delays for messages received from one sender job do not depend on the communication activities of other jobs. In addition, separate message queues prevent a sender job that violates its message inter-arrival time specification from causing the loss of messages sent by other jobs. A message omission failure

caused by queue overflow at an input port only affects the messages sent by a single sender job.

6. DISCUSSION

This section summarizes the main benefits of the DECOS integrated time-triggered architecture and highlights the major design decisions. While the part on economic benefits tries to quantitatively express the cost savings, the remainder of this section focuses on qualitative statements.

6.1 Economic Benefits

The shift to an integrated architecture will result in quantifiable cost reductions in the areas of system hardware cost and maintenance. In order to quantify these savings, we analyze the development and deployment of a typical embedded system onboard a high-end car.

Let us assume that the integration of diverse DASs within a car into a single integrated architecture will result in a reduction of 20% of the hardware units and a corresponding reduction in the number of wiring points of a car. With an average cost of 40 Euro per ECU in today's cars and 0.2 Euro per wire [Peti et al., 2005], the total hardware cost of a federated automotive system with 40 ECUs and 800 wires is about 1800 Euro. If we assume that the number of ECUs will be reduced to 30 nodes and the number of wires to 500 in the integrated system, with an increased average cost per ECU of 50 Euro, then total hardware cost is reduced by 200 Euro per car.

6.2 Reduction of Complexity

A minimal complexity, i.e. a minimal mental effort to analyze and understand a system, is one of the most important design drivers of the presented architecture. Today, the unmanaged complexity of systems is the major cause of design faults. Complexity increases the likelihood of serious, yet latent, design flaws [Johnson and Butler, 1992, p. 39]. In [Leveson, 1986, p. 131] it is stated that complexity of software and hardware causes a non linear increase in human-error induced design faults. High system complexity also prolongs development, which is detrimental to a company's economic success, because today's business realities demand a short time-to-market. In addition, complexity complicates validation and certification as state-of-the-art formal verification tools are limited in the size of a design that they can handle.

In order to manage complexity, the presented integrated architecture enables designers to proceed in such a way as if they were realizing DASs in a federated system. A DAS along with the corresponding communication resources (virtual networks) and computational resources (partitions in node computers) is encapsulated and interactions between DASs are limited to the exchange of messages via precisely specified hidden gateways. By not only supporting error containment between DASs, but error containment between jobs within a DAS, the integrated architecture exceeds the error containment capabilities of most federated systems. Furthermore, the integrated architecture offers generic architectural services as a base line for application development and provides designers with design patterns and rules for the structuring of the overall system. The resulting complexity management benefits of the presented integrated architecture are as follows:

- **Near-independence at DAS level and job level.** Similar to a federated architecture, the integrated architecture decouples each DAS from other DASs. Each DAS possesses a dedicated encapsulated virtual network. A virtual network is private for a DAS, i.e. other DASs cannot perceive or 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 and the behavior of gateways is of relevance when reasoning about a DAS. Message transmissions on other virtual networks can be abstracted from. Similarly, each job executes in a corresponding partition, which forms an encapsulated execution environment with guaranteed computational and network resources. The activities of jobs executing on the same node computer cannot affect the computational and network resources that are available to other jobs in the node computer.
- **Generic architectural services.** The presented integrated architecture offers generic architectural services, thus decreasing the functionality that must be realized at the application level. Less functionality makes the application easier to comprehend and leaves less room for software design faults. Only the interface between the architecture and the application is visible to the application developer, while the realization of the architectural services remains hidden.

6.3 Improved Dependability

The integrated architecture supports ultra-high dependability [Walter et al., 1995] with failure rates in the order of 10^{-9} failures/h (115,000 years), e.g., as required for the implementation of automotive drive-by-wire applications. Since today's technology cannot support the manufacturing of electronic devices with failure rates low enough to meet the reliability requirements, the reliability of an ultra-dependable system must be higher than the reliability of each of its node computers. This can only be achieved by utilizing fault-tolerant strategies that enable the continued operation of the system in the presence of node computer failures [Butler et al., 1991]. The integrated architecture builds on top of a time-triggered architecture, which offers a consistent distributed computing base [Poledna et al., 2002] with a consistent distributed state induced by the sparse time. The consistent distributed state is necessary for fault-tolerance through active redundancy.

Furthermore, the integrated architecture exhibits dependability improvements with respect to reliability. In the integrated architecture, multiple jobs can execute within each node computer. The resulting reduction in the number of node computers leads to a lower probability for a node computer failure in the overall system. In addition, the jobs executing within a node computer share the node computer's connection to the physical network by accessing virtual networks. Virtual networks improve the reliability of the system significantly by reducing wiring and connectors. Several studies document the significant proportion of connector and wiring failures in distributed embedded systems. Field data provided by Swingler indicates that more than 30% of electrical failures are attributed to connection problems [Swingler and McBride, 1998]. Considering that a luxury car can have up to several hundred connectors this number underpins the potential for reliability improvements through the use of the introduced integrated architecture.

6.4 Simplified Certification

Certification is a significant cost factor in the development of ultra-dependable systems, e.g., in the avionic domain [Hayhurst et al., 1999]. Today, between 60% and 70% of the cost of an avionics node is verification and certification cost [Langley, 2003]. Depending on the level of criticality, certification according to DO-178B increases the cost of developing software by 300%–500% [Atkinson, 2002]. Consequently, there is a need for architectures that are designed for validation [Johnson and Butler, 1992]. A key element for controlling validation and certification cost is architectural support for modular certification. Modular certification [Rushby, 2001b] is a certification strategy that promises a massive reduction in certification cost through modularization and reuse of certification arguments. The presented integrated architecture offers modular certification by separating the certification of architectural services from applications and by supporting independent safety arguments for different DASs.

- **Separating certification of architectural services from certification of applications.** The clear interfaces between the platform and applications are a prerequisite for the separation of the certification of architectural services from the certification of applications. The certified architectural services of the integrated architecture establish a baseline safety argument for the certification of the overall system [Nicholson et al., 2000].
- **Separating certification of different DASs.** The integrated architecture allows the independent certification of different DASs, instead of considering the system as an indivisible whole in the certification process. The safety argument for each DAS is provided to

the integrator by the suppliers along with the compiled application code of the jobs in the corresponding DAS. In order to construct the safety argument for the overall system, the system integrator combines the safety arguments of the independently developed DASs and acquire additional evidence, such as results of a formal verification of the architectural services. The deconstruction of the overall system into encapsulated DASs with different criticality levels reduces the overall certification efforts and allows to focus on the most critical parts. Furthermore, the separate certification of DASs is beneficial, if functionality is reused in different systems. In this case, the safety argument for the functionality needs to be constructed only once.

A prerequisite for modular certification are the error containment and encapsulation services of the integrated architecture. These services permit incremental certification [Nicholson et al., 2000], i.e. the maintaining of a safety argument despite the modification or addition of DASs. Incremental certification requires the ability to integrate new DASs without the need to re-certify the whole system.

7. CONCLUSION

Already partly deployed in avionics, the importance of integrated architectures is continually rising also in other domains such as the automotive industry. The transition from a federated to an integrated architecture makes it possible to realize substantial technical and economic benefits in large dependable embedded systems such as a reduction of the number of nodes with less hardware and cabling cost. In this paper we have presented the blueprint of an integrated architecture that is built on the core services of the well-established Time-Triggered Architecture (TTA) and has been implemented during the European Integrated Project DECOS within the 6th Framework Programme.

The key design choice of the DECOS architecture is a fine structuring of application subsystems into small DASs, each provided with encapsulated communication and computational resources. We partition node computers into a safety-critical subsystem and a non safety-critical subsystem by a basic connector unit that enforces a clear separation of these two subsystems and leads to a noteworthy reduction of the certification effort of a mixed criticality node. The communication resources for the different DASs are provided through virtual networks, which are realized as overlay networks on top of a time-triggered physical network. Each virtual network forms the communication infrastructure of a DAS and runs a communication protocol tailored to the needs of the respective DAS. A virtual network exhibits specified temporal properties, which are independent from the communication activities in other

virtual networks. The partitioning of the overall system into DASs with encapsulated virtual networks helps in managing complexity, because a DAS can be understood independently from other DASs. Furthermore, we provide a solution to the controlled export and import of information between distributed application subsystems. We give the designer the ability to coordinate application services and exploit redundancy in a system to either improve reliability or reduce resource duplication. We introduce virtual gateways for the coupling of virtual networks by the selective redirection of messages. Virtual gateways not only resolve property mismatches between distributed application subsystems, but also preserve encapsulation.
A prototype implementation has shown the feasibility of the presented architecture. Given the advances in the field of semiconductor technology it is expected to implement a node of this integrated architecture as a System-on-a-Chip (SoC).

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

Atkinson, R. (2002). COTS tools reduce the cost of embedded software certification. *COTS Journal*, pp. 27–31.

Beal, D. et al. (2000). RTAI: Real-Time Application Interface. *Linux Journal*. April, 2000.

Birch, S. (2003). Pre-safe headlines S-Class revisions. *Automotive Engineering International*, pp. 15–18.

Butler, R.W., Caldwell, J.L., and Vito, B.L. Di (1991). Design strategy for a formally verified reliable computing platform. In *Proceedings of the 6th Annual Conference on Systems Integrity, Software Safety and Process Security*, pp. 125–133.

Coulouris, G., Dollimore, J., and Kindberg, T. (1994). *Distributed Systems: Concepts and Design.* International Computer Science Series. Addison-Wesley, Reading, MA, USA, second edition.

Cristian, F. (1991). Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78.

DECOS (2005). Virtual Communication Links and Gateways – Implementation of Design Tools and Middleware Services. Dependable Embedded Components and Systems (DECOS) Project Deliverable D2.2.3.

Deicke, A. (2002). The electrical/electronic diagnostic concept of the new 7 series. In *Convergence International Congress & Exposition On Transportation Electronics*, Detroit, MI, USA. SAE.

DeLine, R. (1999). *Resolving Packaging Mismatch.* PhD thesis, Carnegie Mellon University, Computer Science Department, Pittsburgh.

Driscoll K., and Hoyme, K. (1993). SafeBus for avionics. *IEEE Aerospace and Electronics Systems Magazine*.

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

Hansen (2002). The Hansen Report on Automotive Electronics, Portsmouth NH USA, www.hansenreport.com.

Hayhurst, K., Dorsey, C., Knight, J., Leveson, N., and McCormick, G. (1999). Streamlining software aspects of certification: Report on the SSAC survey. Technical report, NASA Technical Memorandum 1999-209519.

Hexel, R. (1999). *Validation of Fault Tolerance Mechanisms in a Time Triggered Communication Protocol using Fault Injection*. PhD Thesis. Technische Universität Wien, Institut für Technische Informatik.

Huber B., Peti P., Obermaisser R., and El Salloum C. (2005). Using RTAI/LXRT for Partitioning in a Prototype Implementation of the DECOS Architecture. In *Proceedings of the Third International Workshop on Intelligent Solutions in Embedded Systems (WISES)*. May, 2005.

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

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

Kopetz, H. (1999). *Elementary versus composite interfaces in distributed real-time systems.* In Proceedings of ISADS'99, Tokyo, Japan.

Kopetz, H. (1999b). Specification of the TTP/C Protocol. TTTech. Schönbrunnerstraße 7, A-1040 Vienna.

Kopetz, H. (2003). Fault containment and error detection in the time-triggered architecture. In *Proceedings of the Sixth International Symposium on Autonomous Decentralized Systems*.

Kopetz, H. and Bauer, G. (2003). The time-triggered architecture. *IEEE Special Issue on Modeling and Design of Embedded Software.*

Kopetz H., and Nossal R. (1997). Temporal firewalls in large distributed real-time systems. *Proceedings of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems.*

Kopetz H., Obermaisser R., Peti P., and Suri N. (2004). From a Federated to an Integrated Architecture for Dependable Embedded Systems. Technical Report 22/2004. Technische Universität Wien, Institut für Technische Informatik.

Kopetz, H. and Ochsenreiter,W. (1987). Clock synchronization in distributed real time systems. *IEEE Transactions on Computers*.

Kopetz, H. and Suri, N. (2003). Compositional design of RT systems: A conceptual basis for specification of linking interfaces. In *Proceedings of the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 51–60.

Lala, J.H. and Harper, R.E. (1994). Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, 82:25–40.

Langley, NASA (2003). Formal methods site. shemesh.larc.nasa.gov.

Leohold, J. and Schmidt, C. (2004). Communication requirements of future driver assistance systems in automobiles. *In Proceedings of the International Workshop on Factory Communication Systems*, pp. 167 – 174.

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

MOST Cooperation (2002). *MOST Specification Version 2.2*. MOST Cooperation, Karlsruhe, Germany. www.mostnet.de.

Nicholson, M., Conmy, P., Bate, I., and McDermid, J. (2000). Generating and maintaining a safety argument for integrated modular systems. In *Proceedings of 5th Australian Workshop on Safety Critical Systems and Software*, pp. 31–41.

Obermaisser, R.. and Peti, P. (2005). Specification and Execution of Gateways in Integrated Architectures. *In Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'05)*, Page(s): 689 – 698.

Obermaisser R., Peti P., Kopetz H., (2005) Virtual Networks in an Integrated Time-Triggered Architecture. *Proceedings of the 10th IEEE International Workshop on Object-oriented Real-time Dependable Systems.*

Obermaisser R., Peti P., Kopetz H., (2005b) Virtual Gateways in the DECOS Integrated Architecture. *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems.*

OMG (2001). Model Driven Architecture. Technical Report document number ORMSC/2001-07-01, Object Management Group. Available at http://www.omg.org.

Peti P., Obermaisser R., Tagliabo F., Marino A., Cerchio S. (2005). An Integrated Architecture for Future Car Generations. *Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing.*

Robert Bosch Gmbh (1991). *CAN Specification*, Version 2.0. Robert Bosch Gmbh, Stuttgart, Germany.

Poledna, S., Maier, R., Bauer, G., and Stöger, G. (2002). Time-triggered architecture: A consistent computing platform. *IEEE Micro*, 22(4):36–45.

RTCA (1992). DO-178B: Software Considerations in Airborne Systems and Equipment Certification. Radio Technical Commission for Aeronautics, Inc. (RTCA),Washington, DC.

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

Rushby, J. (2001). Bus architectures for safety-critical embedded systems. In Henzinger, Tom and Kirsch, Christoph, editors, *Proceedings of the First Workshop on Embedded Software (EMSOFT 2001)*, volume 2211 of Lecture Notes in Computer Science, pp. 306–323, Lake Tahoe, CA. Springer-Verlag.

Rushby, J. (2001b). Modular certification. Technical report, Computer Science Laboratory SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025, USA.

Rushby, J. (2002). An overview of formal verification for the time-triggered architecture. In Damm, Werner and Olderog, Ernst-Rudiger, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 2469 of Lecture Notes in Computer Science, pp. 83–105, Oldenburg, Germany. Springer-Verlag.

Simon, H.A. (1996). *The Sciences of the Artificial*. MIT Press.

Swanson, D.L. (1998). Evolving avionics systems from federated to distributed architectures. In *Proceedings of the 17th Digital Avionics Systems Conference (DASC)*, volume 1, pp. D26/1–D26/8. AIAA/IEEE/SAE.

Swingler, J. and McBride, J.W. (1998). The synergistic relationship of stresses in the automotive connector. In *Proceedings of the 19th International Conference on Electric Contact Phenomena*, pp. 141–145.

Toyota (2004). A Guide to Hybrid Synergy Drive. Toyota Motor Corporation. Japan.

Walter, C. J., Hugue, M. M., and Suri, Neeraj (1995). *Advances in Ultra-Dependable Distributed Systems.* IEEE Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720.