Realization of Virtual Networks in the DECOS Integrated Architecture

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

Abstract

Due to the better utilization of computational and communication resources and the improved coordination of application subsystems, designers of large distributed embedded systems (e.g., in the automotive domain) are eager to replace existing federated architectures with integrated ones. This paper focuses on the communication infrastructure of the DECOS integrated system architecture, which realizes for each application subsystem a so-called virtual network as an overlay network on top of a time-triggered communication protocol. Since all virtual networks share a single physical network, virtual networks promise massive cost savings through the reduction of physical networks and reliability improvements with respect to wiring and connectors. Furthermore, virtual networks support application subsystems that range from ultra-dependable control applications (e.g., an X-by-wire system) to non safety-critical applications such as comfort systems. For this reason, two classes (event-triggered and time-triggered) of virtual networks are realized. 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. In order to ensure that the virtual network abstractions hold also in the case of software faults, each application subsystem possesses a dedicated virtual network with statically assigned resources at the underlying time-triggered communication service.

1 Introduction

Integrated architectures share network and component resources among different application subsystems in order to evolve beyond a "1 Function – 1 Electronic Control Unit (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 [3]. 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 [12].

Within the Dependable Embedded Components and Systems (DECOS) EU Framework Programme 6, we have developed the conceptual foundation for the communication infrastructure of the integrated DECOS architecture [7]. We provide each application subsystem 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 paradigm (event-triggered or time-triggered control) and is tailored to the requirements of the respective application subsystem via its temporal properties (e.g., latencies, bandwidth).

This paper describes a realization of virtual networks on top of a physical network executing the protocol TTP [13]. We use the Time Division Multiple Access (TDMA) scheme of the physical network as a starting point and subdivide each TDMA slot into smaller subslots that are dedicated to the different virtual networks used in the system. Within each DECOS node of the integrated distributed system, a TTP communication controller offers a state message interface for the time-triggered exchange of messages. The application computers, which are part of the multiprocessor DECOS nodes, use the real-time operating system Linux/Real-Time Application Interface (RTAI) and employ middleware services as kernel modules for realizing the event-triggered and time-triggered virtual networks. The middleware services provide to the application software a set of interface data structures, which consist of message queues for event-triggered virtual networks and state variables with update-in-place semantics for time-triggered virtual networks. The state variables of the time-triggered virtual networks always provide the most recent version of a real-time entity, which is ideal for the construction of highly regular applications, such as control loops. The message

[0]This work has been supported in part by the European IST projects ARTIST2 (No. IST-004527) and DECOS (No. IST-511764).

queues support exactly-once processing, which is required for handling event information [7], despite message transmissions at a priori unknown points in time.

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 Distributed Application Subsystems (DASs). The construction of virtual networks on top of a time-triggered core network is the focus of Section 3. Section 4 discusses the implementation of virtual networks within an integrated node computer. Node computers consists of three distinct hardware elements that realize a hierarchic subdivision of network resources. The paper finishes with a discussion of the benefits of virtual networks in Section 5.

2 DECOS Integrated Architecture

The DECOS architecture [8] 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. The DECOS architecture aims at offering to system designers generic architectural services, which provide a validated stable baseline for the development of applications.

2.1 System Structuring

For the provision of application services at the controlled object interface, the services of a real-time computer system are 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 is executed within a *partition*. 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.

Jobs exploit a *virtual network* [7] in order to exchange messages with other jobs and work towards a common goal. A *virtual network* is the encapsulated communication system of a DAS. All communication activities of a virtual network are private to the DAS, i.e. transmissions and receptions of messages can only occur by jobs of the DAS unless a message is explicitly exported or imported by a gateway. Furthermore, a virtual network exhibits predefined temporal properties that are independent from other virtual networks.

A *port* is the access point between a job and the virtual network of the DAS the job belongs to. Depending on the data direction, one can distinguish input ports and output ports. In addition, we classify ports into state ports and event ports depending on the information semantics of sent or received messages.

A *state port* aims at messages with *state semantics*. Information with state semantics contains the absolute value of a real-time entity (e.g., temperature in the environment is 41 degrees Celsius). Since applications are often only interested in the most recent value of a real-time entity, a state port contains a memory element that is overwritten with newer state values whenever a message arrives at the port (i.e. update in place).

An *event port* supports *event semantics*, i.e. information about the change in value of a real-time entity associated with a particular event. Messages containing event information transport relative values (e.g., increase of the temperature in the environment by 2 degrees). In order to reconstruct the current state of a real-time entity from messages with event semantics, messages are queued at an event port in order to process every message exactly-once. The loss of a single message with event information could affect state synchronization between a sender and a receiver.

2.2 Architectural Services

The DECOS architecture distinguishes a minimal set of *core services* and an open-ended number of *high-level services*. The core services include predictable time-triggered message transport, clock synchronization, and fault isolation. 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 gateway services, virtual network services, and encapsulation services. On top of the time-triggered physical network, different kinds of virtual networks are established and each type of virtual network can exhibit multiple instantiations.

3 Virtual Networks

A virtual network is an overlay network that is established on top of a physical network [7]. In the DECOS integrated system architecture, we provide virtual networks on top of the time-triggered 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., broad-cast 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

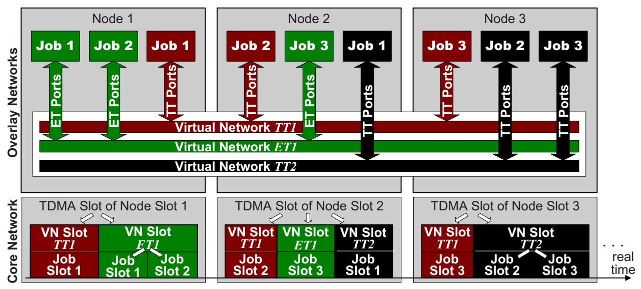

Figure 1. Hierarchic Subdivision of Communication Resources

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.

For the realization of virtual networks at the physical level, we employ the time-triggered core communication service (e.g., TTP [13] or FlexRay [4]) and perform a hierarchic temporal subdivision of the communication resources (see Figure 1). The media access control strategy of the time-triggered core communication service is TDMA. TDMA statically divides the channel capacity into a number of slots and controls access to the network solely by the progression of time. Each 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.

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.

4 Implementation

In the following we describe the implementation of virtual networks. We will explain the employed 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 communication resources by means of temporal and spatial partitioning. On the basis of Figure 2, we discuss the realization of virtual networks in an integrated DECOS node computer.

4.1 Platform

A DECOS node computer is a multiprocessor node as depicted in Figure 2. 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 partitioning between the safety-critical and the non safety-critical subsystem of a DECOS node computer. The BCU splits the bandwidth between the two application computers implementing the safety-critical and non safety-critical node subsystems.

This design facilitates modular certification [10], 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 TTP communication controller [14].

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 [2] 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. According to the DECOS fault hypothesis (see also [7]), a job is treated as FCR for software faults. Therefore, partitions have to ensure that a faulty job cannot interfere with any other job. This includes the computational as well as the communication resources.

4.2 Realization of Basic Connector Unit

Figure 2 gives an overview of the BCU software in the TTP MPC855 single board computer. Two Linux/RTAI drivers (*TTP driver* and *Ethernet driver*)

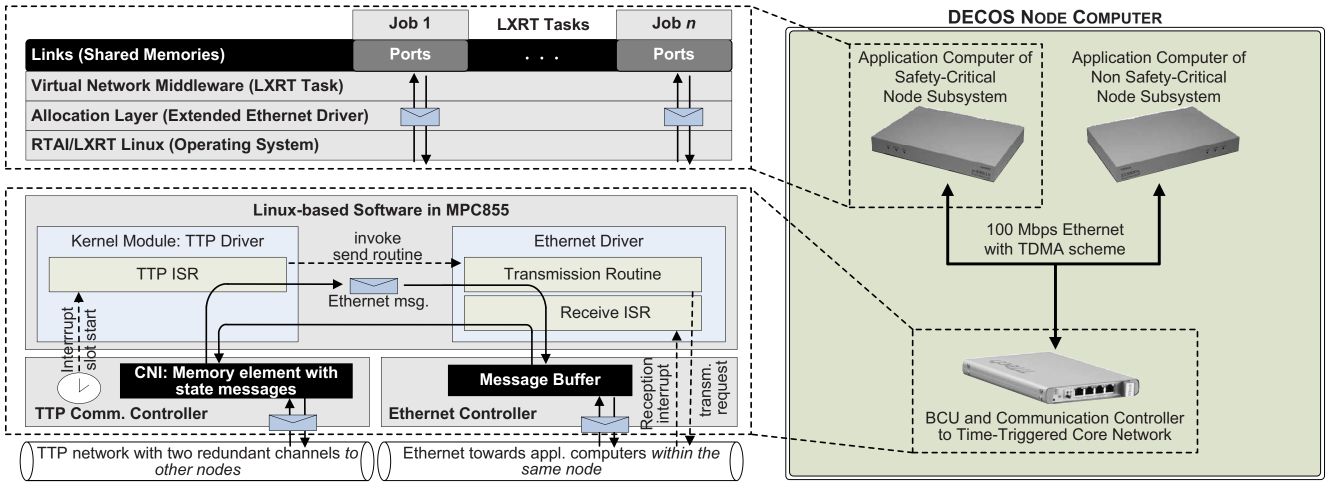
Job 1
LXRT Tasks
Job *n*
Links (Shared Memories)
Ports
. . .
Ports
Virtual Network Middleware (LXRT Task)
Allocation Layer (Extended Ethernet Driver)
RTAI/LXRT Linux (Operating System)
Linux-based Software in MPC855
Kernel Module: TTP Driver
invoke send routine
Ethernet Driver
TTP ISR
Transmission Routine
Ethernet msg.
Receive ISR
Interrupt slot start
CNI: Memory element with state messages
Message Buffer
Reception interrupt
transm. request
TTP Comm. Controller
Ethernet Controller
TTP network with two redundant channels *to other nodes*
Ethernet towards appl. computers *within the same node*
DECOS NODE COMPUTER
Application Computer of Safety-Critical Node Subsystem
Application Computer of Non Safety-Critical Node Subsystem
100 Mbps Ethernet with TDMA scheme
BCU and Communication Controller to Time-Triggered Core Network

Figure 2. Implementation of an Integrated DECOS Node Computer

perform the subdivision of the communication resources for the safety-critical and non safety-critical node subsystems.

TTP Driver The TTP driver is a Linux/RTAI kernel module that is responsible for interacting with the TTP communication controller. The TTP driver contains an Interrupt Service Routine (ISR), which is invoked periodically by a control signal from the TTP communication controller at the beginning of each communication slot on the TTP network. The global points in time of these invocations are a priori defined through the Message Descriptor List (MEDL) of the TTP network.

The TTP ISR is responsible for reading incoming messages from the TTP network and forwarding these messages from the TTP network to the application computers. Upon each invocation, the ISR determines the current MEDL position, i.e. the current TDMA round within the cluster cycle and the actual communication slot within the TDMA round. With the a priori knowledge available via the MEDL, the TTP driver, then, determines which node has sent during the previous slot. The state message that has been broadcast by this node during the previous slot is contained in the CNI, which is realized as a Dual Ported RAM (DPRAM) in the TTP MPC855 single board computer. The TTP driver copies the state message from the CNI and constructs an Ethernet message that is passed to the Ethernet driver for the dissemination on the node-internal Ethernet network.

Ethernet Driver The Ethernet driver is the interface towards the Fast-Ethernet core of the MPC855T. It is a modified version of the Standard 8xx-Ethernet driver, which has been optimized for time-triggered transmissions and receptions in order to minimize the latency jitter. The Ethernet messages to be sent to the application computers are passed to the Ethernet driver by the TTP driver. The Ethernet message with the data from a certain slot of the TTP network is sent on the node-internal network during the subsequent slot, regardless of whether a node has sent or received at the TTP network according to the MEDL. In case the node itself has sent during the slot on the TTP network, the Ethernet message realizes a loopback service. On the one hand, this mechanism enables the application computer to verify that the correct information has reached the BCU. Secondly, the loopback interface is the need for synchronizing application computers to the global time base. The reception of an Ethernet message sent by the BCU is a synchronizing event for the application computers (i.e. master/slave clock synchronization).

Ethernet header	status area	control area	data channel 0	data channel 1	checksum
14 bytes	56 bytes	24 bytes	130 bytes	130 bytes	4 bytes

Figure 3. Structure of Ethernet Messages Exchanged via the Node-Internal Network

The Ethernet messages that are sent from the BCU to the application computers have a structure as depicted in Figure 3. The status and control area of the Ethernet message are snapshots of significant status and control variables in the TTP controller's CNI [14]. Following the status and control areas, the Ethernet message contains two state messages from the two redundant TTP channels, which have been copied from the CNI. The destination Ethernet MAC address in the header is set to a broadcast address, meaning that the message is sent from the BCU to both of the two application computers.

In addition, the Ethernet driver contains an ISR that is invoked when Ethernet messages from an application computer arrive at a priori specified global points in time at the BCU. The purpose of these Ethernet messages is to transfer outgoing messages, which have been sent by jobs, from the two application computers in a node to the BCU, so the BCU can forward these messages to the TTP network.

Upon the invocation of the ISR, the Ethernet driver reads the source MAC address in the message header in order to determine which of the two application computers has sent the message. If a message from the respective node subsystem is actually due in this slot, then the Ethernet driver copies the data contained in the Ethernet message into the CNI of the TTP communication controller.

The arrival of Ethernet messages is implicitly synchronized with the control signals that are generated by the TTP controller at the beginning of communication slots. Hence, there is no concurrency between the TTP driver and the Ethernet driver. This solution not only minimizes latency jitter, because the ISRs are never delayed, but also guarantees that the information read from the CNI is consistently forwarded (i.e. no partially updated messages) to the TTP network and the node-internal network, respectively.

4.3 Realization of Application Computers

Based on the state message interface provided by the BCU, each of the application computers in Figure 2 realizes virtual networks via two software layers. The *allocation layer* is an extended Ethernet driver, which is based on RTnet [5]. It is responsible for interacting with the BCU. The *virtual network layer* is an LXRT task that constructs event-triggered and time-triggered overlay networks using the state message interface of the allocation layer. The jobs, which are also realized as LXRT tasks, exploit the virtual network layer by accessing shared memories containing the ports of the event-triggered and time-triggered virtual networks.

Allocation Layer The allocation layer provides to the higher layers of the application computer a memory region that is updated at a priori specified global points in time. This memory region maps into the application computer the node's CNI towards the time-triggered communication service. The layout of this memory region, which is an image of the node's CNI, is depicted in Figure 4. The CNI is structured into smaller state variables. For each subsystem, node, and channel of the cluster, the CNI contains a dedicated state variable. At a particular application computer, the state variable associated with the application computer (which corresponds to a node subsystem) at the CNI is written by the virtual network service, while all other state variables are read by the virtual network service. The time-triggered communication system behaves inversely, reading the state variable associated with the application computer before broadcasting its contents via the core network. All other state variables are updated with the contents of messages received from other nodes. The global point in time of a

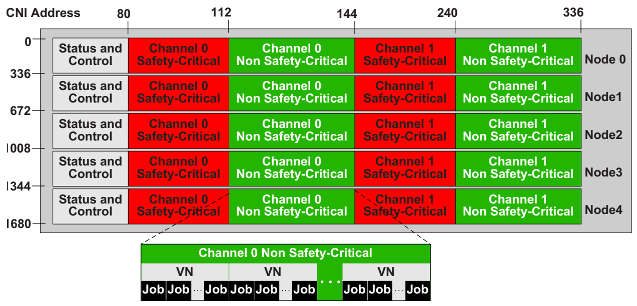
CNI Address
80
112
144
240
336
0
336
672
1008
1344
1680
Status and Control
Channel 0 Safety-Critical
Channel 0 Non Safety-Critical
Channel 1 Safety-Critical
Channel 1 Non Safety-Critical
Node 0
Node1
Node2
Node3
Node4
Channel 0 Non Safety-Critical
VN
VN
VN
Job Job ... Job Job Job ... Job Job Job ... Job

Figure 4. Layout of CNI

message reception not only denotes the identity of the sending node and subsystem, but also determines the state variable in the CNI that is to be overwritten.

Within the RTnet Ethernet driver, the allocation layer provides an ISR that is periodically invoked by Ethernet messages received from the BCU. Upon the reception of an Ethernet message, the allocation layer determines the current position within the TDMA round, which is denoted by a variable within the status area of the Ethernet message (cf. Figure 3). With the knowledge about the position within the TDMA round, the allocation layer can determine the sending node and the corresponding location within the CNI memory region. If the Ethernet message contains a state message from another DECOS node, then the allocation layer copies – except for the Ethernet header – the complete message into the CNI memory region. In case the Ethernet message contains a state message from the same DECOS node, the allocation layer copies only those parts into the CNI memory region, which originate from the other node subsystem. Thus, the parts in the CNI memory region that are updated by the virtual network service are never overwritten by the allocation layer. Nevertheless, the allocation layer provides to the higher layers the state messages from the other node subsystem, which is necessary for the implementation of gateways between safety-critical and non safety-critical DASs.

If the current position within the TDMA round indicates that the sending back of a message to the BCU is due, the allocation layer constructs an Ethernet message with the data from a CNI memory region updated by the virtual network layer. This Ethernet message is sent via the RTnet Ethernet driver with the MAC address of the BCU as its destination.

Virtual Network Layer The main purposes of the virtual network layer are the subdivision of communication resources from subsystem granularity to virtual network and job granularity, the conversion of control paradigms and the switching of messages.

The interface of the virtual network layer to the jobs are state and event ports. A state port contains a state variable that is accessed by the virtual network layer at a priori specified global points in time. The state variable is either updated by the virtual network layer

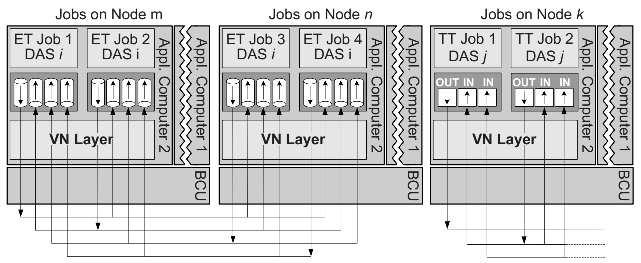

Figure 5. Partitioning at Virtual Network Level

(data copied from CNI memory region to the input state port) or read by the virtual network layer (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 layer in case of an input event port. For an output event port, the virtual network layer retrieves messages from the queue and writes them into the CNI memory region.

The state and event ports are realized with the shared memories in Figure 2 in between the LXRT tasks of the jobs and the LXRT task of the virtual network layer. Each job possesses a dedicated shared memory area that is protected by the Memory Management Unit (MMU) from other jobs, thus preventing unintended interference between jobs via ports. Within the shared memory provided to a job, there is a dedicated input port for each other job of the DAS that sends messages to the job (see Figure 5). Providing a dedicated input port for each sender at all receivers is a key element for encapsulating senders within a DAS:

- **Spatial partitioning by handling masquerading failures.** Masquerading is defined as the sending or receiving of messages using the identity of another sender without authority [1]. 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.

  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 message 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.

- **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.

The conversion of control paradigms is performed for event-triggered virtual networks by mapping the state message interface of the allocation layer to an event message interface. Since the temporal firewall interface provided by the allocation layer is a purely time-triggered communication service, the virtual network layer implements an event-triggered packet service. A job places event messages into the message queues at an output port resulting in a sequential stream of event messages that forms the input to the virtual network layer. The insertion of inactivity messages preserves the assumption of a coherent message stream during time intervals, in which the job does not produce messages. The virtual network layer performs a fragmentation of these outgoing messages into packets and places these packets in the job slot in the Communication Network Interface (CNI) memory region. Virtual network configuration data structures define the addresses and sizes of the job slots in the CNI memory region. The size of event message relative to the size of the job slot in the CNI memory region also determines the transmission latency of the event-triggered virtual network. In case the job slot is smaller than the event message, the transmission of the message will take several TDMA rounds. If the job slot is larger than the event message, data from multiple event messages can be transmitted within a single TDMA round. Adversely, the virtual network layer is also responsible for fusing packets, which are received via the job slots, into event messages and offering these messages to the jobs via the message queues in event input ports.

In order to copy messages between ports and the CNI memory region, the virtual network layer is acti-

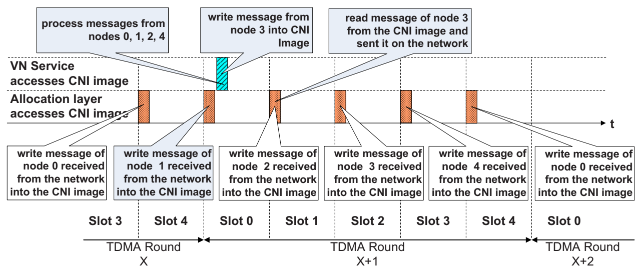
process messages from nodes 0, 1, 2, 4
write message from node 3 into CNI Image
read message of node 3 from the CNI image and sent it on the network
VN Service accesses CNI image
Allocation layer accesses CNI image
t
write message of node 0 received from the network into the CNI image
write message of node 1 received from the network into the CNI image
write message of node 2 received from the network into the CNI image
write message of node 3 received from the network into the CNI image
write message of node 4 received from the network into the CNI image
write message of node 0 received from the network into the CNI image
Slot 3
Slot 4
Slot 0
Slot 1
Slot 2
Slot 3
Slot 4
Slot 0
TDMA Round X
TDMA Round X+1
TDMA Round X+2

Figure 6. Activities in Node Computer 3

vated at the beginning of each TDMA round after the execution of the allocation layer. The virtual network layer reads the CNI areas belonging to other nodes and the second subsystem at the same node. In addition, the virtual network layer writes the CNI area belonging to itself, i.e. the node subsystem in which the virtual network layer is being executed. The virtual network layer accesses the CNI in time-triggered manners and is thereby implicitly synchronized with the allocation layer. Figure 6 illustrates this scheme for node computer 3 in an ensemble of five node computers. This design prevents the transmission and reception of partially updated messages.

Jobs The jobs implement the application services and access the virtual network via the shared memory containing the ports. In order to simplify the implementation of jobs, a *virtual network library* is available that can be linked to a job. This library aims at simplifying startup and provides function calls for accessing the shared memories containing the ports. The library provides the following function calls:

- *t_link *fetch_link(int networkId, int linkId).* During startup, a job can invoke this operation for gaining access to the ports towards a specific virtual network. The link with is identified through two numerical identifiers: the network identification and a link identification that is valid within the network. The link consists of all ports towards the virtual network that are accesses by the job. The fetch-link operation initializes the port configuration data structure and maps the shared memory area containing the port buffer into the address space of the job.

- *int sndEtMessage(t_port *port, t_message *msg).* This operation is used for requesting the transmission of an event message. This function can be invoked at a priori unknown point in time within the execution of a job. The output port is identified by the first parameter and must be part of a link that has been previously acquired with the fetch-link operation. The message, which is passed via the parameter msg is inserted into the message queue associated with the port. The send operation detects queue operations and informs the job through the return value (*ET_OK* or *ET_ERROR*).

- *int rcvEtMessage(t_port *port, t_message *msg).* By invoking this operation, a job can pull a previously received message from an input port. This function is invoked at a priori unknown point in time within the execution of a job. The input port is identified by the first parameter and must be part of a link that has been previously acquired with the fetch-link operation. The second parameter points to a memory area to which the event message will be copied to. The return value (*ET_QUEUE_EMPTY* or *ET_OK*) informs the job, whether a message has been available and copied to the specified memory area.

For time-triggered ports, no send and receive operations are required, because the port buffer already contains a pointer to the periodically updated state variable. The job can simply write (in case of an output port) or read (in case of an input port) the state variable, which is autonomously communicated by a time-triggered virtual network. In order to ensure consistency of communicated state variables, the tasks implementing the jobs are implicitly synchronized with the task implementing the virtual network middleware.

5 Discussion

Virtual networks in the DECOS architecture provide each of the DASs that are executed on the shared distributed computer system with a communication infrastructure that meets the respective requirements w.r.t. to the temporal properties (e.g., control paradigm, bandwidth, latencies) of message exchanges. Each virtual network is encapsulated to prevent unintended interference between DASs at the level of the communication resources. Encapsulation is a key element for cost-effective development of mixed criticality systems with modular certification of different DASs. In addition, encapsulated virtual networks facilitate system integration, because the temporal properties of the messages sent by already integrated jobs are not affected when adding further node computers and jobs.

5.1 Encapsulation of Messages

For capturing the term of encapsulation, we take on a *sender-centric view*. This means that we look at the non interference in the message transmissions between sender jobs, while abstracting over interference between message transmissions from the same sender job. According to the DECOS fault hypothesis, each sender job forms a Fault Containment Region (FCR) w.r.t. to software faults. For analyzing interference, we focus on both the temporal and value domain. When looking at mechanisms for the prevention of intereference, this bivalent distinction maps to the concepts of temporal and spatial partitioning [9].

We regard the following *temporal properties* as potentially subject to interference:

- *Transmission latency.* The end-to-end delay of a message transmission from a sender to a receiver job includes latencies induced by the communication system, namely the access delay and the transmission duration of a message. Encapsulation w.r.t. to the transmission latencies means that the access delays and transmission durations (worst-case, best-case, actual, variability) of messages transmitted by a particular sender to its receivers are not affected by messages transmitted by other senders.

- *Bandwidth.* The bandwidth available to a specific sender is the number of bytes transported within the messages sent by this sender per second. Temporal encapsulation requires the bandwidth (maximum, minimum, average, variability thereof) that is available to a sender job to be independent from the behavior of other jobs.

We regard the following *value domain properties* as potentially subject to interference:

- *Encapsulated name spaces.* In many protocols the message name uniquely identifies the sender of the message. For example, an Ethernet message contains the MAC address of the sender. Similarily, a CAN message contains a unique identifier, which is usually exclusively used by a single sender to avoid collisisions. The arbitration mechanism of the CAN protocol would be apt to fail when more than one sender simulteanously sends a message with the same identifier.

  In order to prevent the misinterpretation of messages at receiver jobs, a sender job must not to use the identity of another sender job (also called masquerading [1]). A communication protocol that satisfies this requirement provides encapsulated name spaces.

- *Data integrity.* A communication protocol ensures data integrity, if for each sender job the contents of the sent messages cannot be affected by other sender jobs.

5.2 System Integration

A seamless system integration phase is closely related to the concept of *composability*. Composability is defined as the stability of component properties across integration. Composability is necessary for correctness-by-construction of component-based systems [11]. Temporal composability is an instantiation of the general notion of composability. A communication system is *temporally composable*, if *temporal correctness* is not refuted by the system integration [6].

A necessary condition for temporal composability is that if n nodes are already integrated, the integration of node $n + 1$ will not disturb the correct operation of the n already integrated nodes. This condition guarantees that the integration activity is linear and not circular. It has stringent implications for the management of the network resources.

In contrast to communication protocols that dynamically share bandwidth at the Media Access Control (MAC) layer, a virtual network supports invariant temporal properties at the communication system during an incremental integration process. The network resources are statically assigned to each node computer. At the cost of reduced flexibility, virtual networks provide to each newly integrated node computer a communication slot that has already been reserved at design time via a static communication schedule.

References

[1] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design.* Addison-Wesley, 2nd edition, 1994.

[2] D. Beal et al. RTAI: Real-Time Application Interface. *Linux Journal*, April 2000.

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

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

[5] J. Kiszka, B. Wagner, Y. Zhang, and J. Broenink. RT-net – a flexible hard real-time networking framework. In *Proc. of 10th ETFA Conference*, 2005.

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

[7] R. Obermaisser, P. Peti, and H. Kopetz. Virtual networks in an integrated time-triggered architecture. In *Proc. of 10th IEEE WORDS*, 2005.

[8] P. Peti, R. Obermaisser, F. Tagliabo, A. Marino, and S. Cerchio. An integrated architecture for future car generations. In *Proc. of the 8th ISORC*, 2005.

[9] J. Rushby. Partitioning for avionics architectures: Requirements, mechanisms, and assurance. NASA contractor report, NASA Langley Research Center, 1999.

[10] J. Rushby. Modular certification. Technical report, SRI International, Sept. 2001.

[11] J. Sifakis. A framework for component-based construction. In *Proc. of 3rd SEFM Conference*, 2005.

[12] J. Swingler and J. McBride. The degradation of road tested automotive connectors. In *Proc. of the 45th IEEE Holm Conference on Electrical Contacts*, 1999.

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

[14] TTTech Computertechnik AG. *TTP/C Controller C2 Controller-Host Interface Description Document, Protocol Version 2.1*, Nov. 2002.