CAN Emulation in a Time-Triggered Environment

Roman Obermaisser, Member, IEEE
Institut für Technische Informatik, TU Vienna, Austria, email: romano@vmars.tuwien.ac.at

Abstract— **The Controller Area Network (CAN) protocol is a widely used event-triggered communication protocol, which offers high average performance, flexibility, and extensibility. However, time-triggered protocols are becoming more and more accepted as the communication infrastructure for safety-critical applications, since they support composability, dependability, and a deterministic behavior of all message transmissions. The desire to reuse CAN based legacy applications as part of time-triggered systems motivates the provision of CAN communication services within a time-triggered environment. This paper elaborates on an approach of layering CAN on time-triggered communication services. A node that participates in this CAN emulation reserves a part of its sending slot for implementing a packet service, thereby establishing a communication channel, a way of transferring a sequential data stream of CAN messages. Furthermore, the emulation offers an improved CAN communication service by addressing deficiencies of the basic CAN protocol. The CAN emulation exploits the fault-tolerance mechanisms of the underlying time-triggered system for extending CAN with support for dependable systems.**

I. Introduction

The Controller Area Network (CAN) was originally developed for in-car use. Industrial control systems and embedded networks became additional application fields [1]. By beginning of 1998 more than 140 million CAN controllers were installed. More than 50 millions of CAN controllers were sold in 1999[1]. CAN represents an event-triggered communication protocol, i. e. the temporal control signals are derived primarily from non-time events. Among its advantages are flexibility and the ability to achieve a high average performance through the statistical multiplexing of bandwidth between components participating in the communication. However, CAN lacks essential properties for systems that have substantial timeliness and dependability requirements. The basic CAN protocol [2] lacks a consistent atomic multicast mechanism and support for fault tolerance by network redundancy. Furthermore, the mechanisms for achieving a faulty node's self-deactivation may cause substantial periods of inaccessibility (2.5 ms at 1 Mbps [3]).

Solutions for addressing fault tolerance by active redundancy [4] and supporting a consistent atomic multicast mechanism have been developed [5, 6, 7], but these approaches either extend the basic CAN protocol or they introduce a solution at the application level, thereby increasing application complexity.

Some mission-critical applications (e.g. x-by-wire systems) require a deterministic behavior of message transmissions. A certain transmission latency of all safety-related messages must be guaranteed even at peak-load. Time-triggered communication protocols can provide this deterministic behavior. In addition to hard real-time performance they support temporal composability and dependability. As a consequence time-triggered protocols are becoming more and more accepted as the communication infrastructure for safety-critical applications [8, 9]. In the automotive industry a time-triggered architecture will provide the ability to handle the communication needs of by-wire cars [10].

Since existing CAN solutions constitute high financial investments, their functionality should be retained without the need of a complete redevelopment. The migration of these CAN-based applications to a time-triggered environment is therefore a major concern. The existing event-triggered CAN applications become legacy systems, i. e. information systems that resist evolution [11].

This paper presents a solution for implementing CAN communication services on top of a time-triggered communication service. This allows the coexistence of event-triggered CAN communication with predictable time-triggered communication services. Furthermore, the CAN application that is integrated into the time-triggered environment can benefit from the properties of an underlying reliable time-triggered communication system. Application complexity is not increased, because the handling of fault tolerance aspects is performed at the protocol level.

The main achievements are the transparent solution of fault-tolerance and predictability aspects not handled by the basic CAN protocol and the possibility to integrate a legacy CAN application into a time-triggered environment. The emulated CAN communication service not only provides the same interface as a commercial CAN controller, but also offers a faithful emulation with a maximum level of authenticity.

Section II explains the two main paradigms for communication systems, namely event-triggered and time-triggered communication systems. Three fundamental approaches for their integration are compared in Section III. Section IV covers a solution for the integration of the event-triggered protocol CAN into a time-triggered environment. The paper is concluded in Section V.

II. Event-Triggered and Time-Triggered Communication Systems

The Controller Area Network (CAN) belongs to the class of event-triggered (ET) protocols. In this section the event-triggered and time-triggered (TT) paradigms are discussed. The properties of a TT communication system and its benefits for real-time systems are elaborated. Furthermore, CAN-specific limitations and peculiarities are described.

A. Concepts

A distributed real-time application can be decomposed into a controlling computer system and a con-

[1]CiA CAN Sales Figures, http://www.can-cia.de/nicc.htm

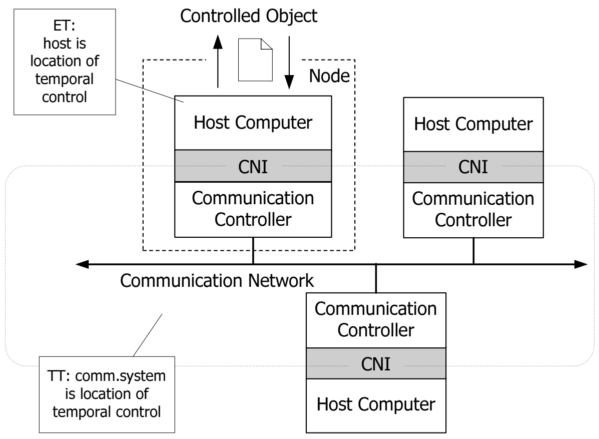

Fig. 1. Computational Cluster

trolled object. The controlling computer system must react to stimuli of the controlled object within periods of time dictated by the controlled object. These periods of time depend on the dynamics of the controlled object. If the controlling computer system is implemented as a distributed computer system, it consists of nodes, which are interconnected by a real-time communication system (see Figure 1). A node consists of a host computer that executes the application and a communication controller. The interface between the host and the communication controller is called the communication network interface (CNI). The set of communication controllers and the common communication network form the communication system. The purpose of the communication system is to transport messages from the CNI of a sender to the CNIs of the receivers. Two fundamentally different paradigms can be used for building the communication system. In an ET system communication activities are initiated whenever a significant change of state occurs. Such a system exploits external control, i. e. the decision when a message is to be transmitted is within the sphere of control of the application software in the host. ET systems allow a flexible allocation of resources which is attractive for variable resource demands. However, multiple nodes may contend for bus access as a reaction to event occurrences. In safety-critical applications it is necessary to guarantee a predictable communication with low latency and low jitter to all participants [9].

In a TT system, activities are initiated at predetermined points in time. Such a system employs autonomous control. The communication controller decides autonomously when a message is transmitted. The CNI forms a temporal firewall, which isolates the temporal behavior of the host and the rest of the system. Local changes of a host cannot invalidate the correct temporal behavior of the communication system. Therefore a TT communication system provides composability with respect to the temporal control. A TT system also helps achieving replica determinism [12], which is essential for establishing fault tolerance through active redundancy. The predetermined points in time of the periodic message transmissions allow error detection and establishing of membership information. Since the system load is independent of the number of events occurring in the controlled object, the latency jitter is minimal. The message transmission latency during peak load scenarios is identical to the latency during normal load.

B. *The Controller Area Network*

The Controller Area Network (CAN) is an ET communication protocol, which uses a broadcast bus with “carrier sense, multiple access with collision avoidance” (CSMA/CA) for medium access control [2]. Bus access conflicts are resolved by observing the message identifier bits on the bus-line. While transmitting a communication message identifier, each node monitors the serial bus-line. If the transmitted bit is recessive and a dominant bit is monitored, the node gives up from transmitting and starts to receive incoming data. The node sending the object with the highest identifier will succeed and acquire bus access. After a loss in the arbitration process or the reception of an error frame, the sender automatically performs a retransmission of the corresponding communication object.

The CAN communication protocol has the following properties:

- CAN communication systems possess a large variability in the transmission latency. A message’s transmission latency depends on the network load. This latency jitter causes an error in the temporal domain and introduces an additional measurement error if there is no global notion of time.
- The arbitration logic of CAN limits throughput, because the propagation delay of the channel must be smaller than the length of a bit-cell.
- Handling of station failures is performed with error counters by recording receive and transmit errors. Thresholds are defined for entering the error passive mode and the bus-off state. Under the assumption that failed nodes reach the bus off state, the worst-case inaccessibility time at 1 Mbps is bounded by 2.5 ms [13].
- The temporal properties of a CAN system are changed during the integration of the system, because host applications can explicitly trigger message transmissions. The temporal coordination of the communication activities is a global issue and depends on the host software in all nodes.
- CAN error recovery mechanisms are unable to ensure a consistent state, if an error is detected in the last but one bit of a frame. Possible consequences are an inconsistent message duplication or an inconsistent message ordering. Establishing consistency requires modifications to the host software [5, 6] or a dedicated hardware component [7].
- Communication errors are handled with immediate message retries and cause increased latencies.

The following properties are not established by the basic CAN protocol:

- CAN does not prevent babbling idiot failures [14]. A node can continuously send high priority messages and thereby prevent communication of other nodes.
- No membership service is provided at the protocol level.

• The CAN protocol does not include a clock synchronization service. If a global notion of time is required, it must be implemented at the host level.

III. Approaches for Integration of ET and TT Communication Services

The desire to leverage investments in existing CAN-based legacy applications motivates the coexistence of CAN-based and TT communication services. This section describes three basic approaches for the integration of an ET communication protocol like CAN with a TT communication service: alternating windows, layering ET on TT services, and layering TT on ET services.

A. Alternating Windows

As depicted in Figure 2, the time axes is divided into windows of two different types. One Window is used for TT, the other one for ET communication. The arrows in Figure 2 represent the points in time when communication activities are started. Communication activities within the ET window are within the sphere of control of the host application and can occur at arbitrary points in time. In the TT windows communication activities start at a priori defined points in time, which are determined by the progression of time, i. e. independent of the application behavior.

In a TDMA based communication scheme the round length determines the update rates of real-time entities. The ET windows limit the achievable update time, because the window dedicated to ET communication must be at least as long as the duration of the longest message. Since many media access protocols designed for ET communication limit the bandwidth (e. g. for CAN bits must stabilize on the medium), this duration may become the limiting factor for the round duration and the update frequencies of real-time data.

The "alternating window" design possesses the following requirements:
• **Collision handling**: Mechanisms for handling or avoiding collisions are required. For example, CAN-based windows need a CSMA/CA media access protocol that supports arbitration.
• **Two media access protocols**: This solution requires two media access protocols. A TDMA based protocol is applied to the TT windows. Within the second window type a dynamic media access strategy is necessary.
• **Fault containment between ET and TT communication**: The ET communication activities must be restricted to the corresponding time windows. No invalidation of the correct temporal behavior of the TT communication activities may be allowed to occur.
• **Forbidden regions**: The available bandwidth of the ET windows cannot be fully used. At the end of the ET window – for a duration equal to the maximum duration of an event message – no message transmission may be started in order to prevent a collision with TT communication.

An example for a communication system using alternating windows is FlexRay [15, 16]. The TT part uses a TDMA media access strategy. In the dynamic part, bus access is performed with static frame priorities according to the Byteflight specification [17].

B. ET Service Implemented on top of a TT Protocol

The approach of implementing the ET service on top of a TT protocol is depicted in Figure 3. The arrows represent the points in time of the periodic message transmissions, which are solely controlled by the progression of time. The TT protocol provides dedicated time windows for all sender nodes. Within such a time window exactly one node may transmit a message. A node that participates in ET communication reserves a part of its sending slot for this purpose, thereby allocating a certain amount of its bandwidth to ET communication. A packet service is implemented with this part of a node's slot. The packet service establishes a communication channel, a way of transferring a sequential data stream of event messages.

A software layer provides the interface of a particular ET communication controller. This software layer accepts messages from the application and receives messages from other nodes. It selects the next message to be placed in the ET communication channel. The decision concerning the selection of a message out of the set of messages the layer has been passed by the application is performed according to a scheduling strategy. This scheduling strategy corresponds to a particular media access protocol. This media access protocol is executed by a software layer layered on top of a different media access protocol - TDMA.

This approach is characterized by the fact that a time slot is written by a single a priori known node. This fact and the handling of media access control by a software layer built on top of a packet service result in the following properties:
• **Single media access protocol**: A single media access protocol is applied for both ET and TT communication. Hence, one communication controller suffices.
• **Arbitrary bandwidth for ET communication**: An arbitrary fraction of the available bandwidth can be reserved for the ET communication. The corresponding time within a communication round may even be shorter than the length of an event message.
• **No forbidden regions**: There is no time window, in which the start of a new message is forbidden to avoid overlapping of static and dynamic parts.

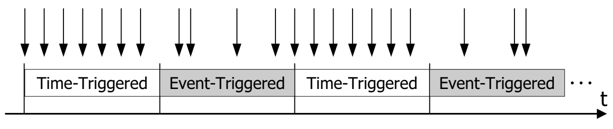

Fig. 2. Alternating Windows

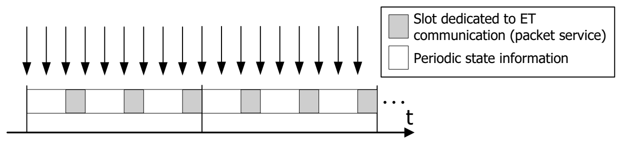
Slot dedicated to ET communication (packet service)
Periodic state information
t

Fig. 3. ET Layered on top of TT

• **Handling babbling idiot failures**: A node participating in the ET communication is called a babbling idiot, if it sends messages at wrong points in time. Such a node can prevent the ability of other nodes to exchange event messages by flooding the network with high-priority messages. By layering the ET service on top of a TT protocol, every node may only send in its particular time slot. The a priori knowledge about potential sending times allows to prevent babbling idiot failures with a special device called bus guardian [18], which prevents nodes from sending during wrong slots.
• **Universal packet service**: The ET media access protocol is layered on top of the TT media access protocol. The interface between these protocols is formed by a packet service. This packet service solution provides a high degree of flexibility. Arbitrary ET communication services can be mapped on this packet service.
• **Knowledge about sender's identity**: If a receiver detects corrupted or missing messages, it possesses knowledge about the identity of the corresponding sender node. This information results from the a priori knowledge about the points in time when a node transmits a message.
• **Gateway functionality**: A conventional CAN network can be interconnected with the TT environment by using a node of the TT network as a gateway. The gateway decouples the TT and CAN communication. Restrictions of the CAN network do not effect the TT network (e. g. bandwidth limitations) and the gateway can perform filtering to minimize the number of messages that are forwarded.
• **No sharing of bandwidth among nodes**: The layering of ET services on top of TT communication services employs no sharing of bandwidth among nodes. The absence of statistical multiplexing of bandwidth among nodes may result in lower bandwidth utilization.
• **Coexistence of multiple ET communication services**: This design can handle several ET communication services simultaneously by sharing the common packet service. Sharing is performed according to a scheduling policy, either statically or dynamically. For example, a system might provide both TCP/IP and CAN services and give priority to CAN messages. The CAN communication service is favored, since it is crucial for the integration of legacy subsystems. The remaining bandwidth of the packet service is available to the TCP/IP service.

C. TT Service Implemented on top of an ET Protocol

This approach uses an ET media access protocol, but the communication activities are controlled by the progression of time. The underlying ET media access protocol eases the task of providing ET communication windows. Since the ET media access protocol is also used for the TT communication, limitations of the ET protocol arising from the need to handle concurrent accesses remain for the TT communication. Furthermore, ET mechanisms of the underlying protocol are not required during the deterministic windows (e. g. arbitration, collision detection).

Fault containment is necessary, otherwise a single faulty node can prevent the correct communication of the other nodes by using the underlying ET interface to monopolize the common communication network. If this approach is not combined with the alternating window approach, the underlying ET media access protocol can be replaced by a TDMA scheme. TTCAN is an example for TT services which are layered on top of an ET protocol [19].

IV. System Model (Functional Overview of Emulated CAN)

This section describes the provision of CAN services on top of a TT communication protocol. CAN messages are exchanged with a packet service and software layers reestablish the temporal properties of a conventional CAN network and provide a particular CAN controller's interface.

A. Basic Architecture

The host computer is shared between the application based upon the TT communication and the CAN-based software. The corresponding node, which is depicted in Figure 4, can be seen as a generalization of a traditional node. It runs two execution environments in parallel, namely an environment providing a TT communication interface and a CAN-based environment. The emulated CAN controller is responsible for relaying ET CAN traffic into a TT communication. The CAN-based application passes CAN messages to the emulated CAN controller and expects it to broadcast the messages to other nodes via the common network. The sequence of CAN messages a node wants to transmit results in a sequential datastream. This datastream is fragmented into packets which are placed in so called *CAN slots*, i. e. periodically recurring periods of time reserved for CAN communication. This fraction of a node's slot is dedicated for establishing the ET communication channel for CAN messages. The rest of the slot can be used for transmitting periodic real-time data (see Figure 5). The

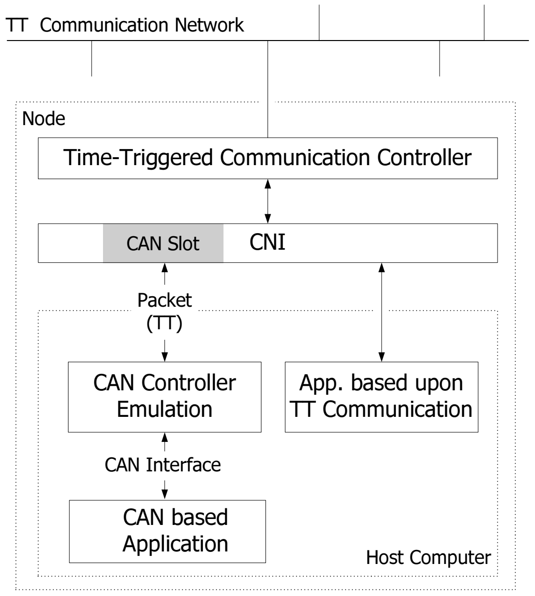

Fig. 4. Node with CAN Message Support

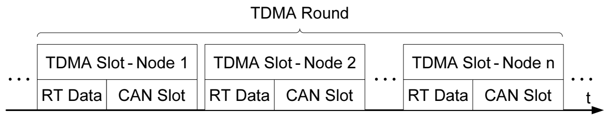

Fig. 5. TDMA Round with CAN Slots

proportion of the slot used for CAN communication is adjusted according to specific application requirements. Depending on the size of the CAN slots, a CAN message transmission can take several TDMA rounds.

B. Software Layers

The CAN controller emulation consists of distinct software layers (see Figure 6). The ET communication channel layer and the protocol engine are responsible for relaying CAN messages into the TT communication service. The application interface provides the application with the programming interface of a specific CAN controller.

1. **CAN based Application**: The application performs CAN communication by accessing the register set of a particular CAN controller, which is provided by the application interface layer. Logically, the application acts according to the information push model for sending activities, i. e. it deposits its output information into the application interface. Message receptions can be mapped into a notification mechanism (information push model) or the application can poll the emulated CAN controller via the application interface (information pull model).

2. **Application Interface**: The application interface layer constitutes the front end for the CAN based application. It emulates the programming interface of a specific CAN controller by providing the corresponding register set. Control of CAN communication services is mapped to this register set.

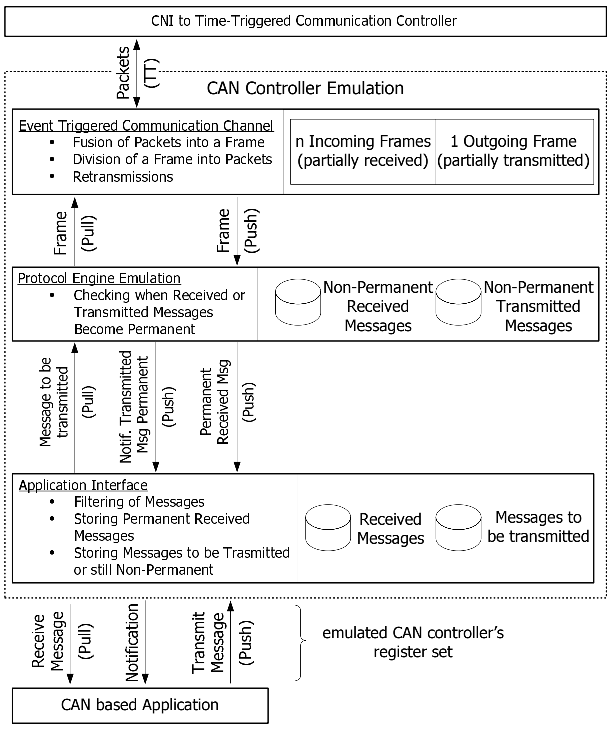
CNI to Time-Triggered Communication Controller
Packets
CAN Controller Emulation
Event Triggered Communication Channel
Fusion of Packets into a Frame
Division of a Frame into Packets
Retransmissions
n Incoming Frames (partially received)
1 Outgoing Frame (partially transmitted)
Frame (Pull)
Frame (Push)
Protocol Engine Emulation
Checking when Received or Transmitted Messages Become Permanent
Non-Permanent Received Messages
Non-Permanent Transmitted Messages
Message to be transmitted (Pull)
Notif. Transmitted Msg Permanent (Push)
Permanent Received Msg (Push)
Application Interface
Filtering of Messages
Storing Permanent Received Messages
Storing Messages to be Trasmitted or still Non-Permanent
Received Messages
Messages to be transmitted
Receive Message (Pull)
Notification
Transmit Message (Push)
emulated CAN controller's register set
CAN based Application

Fig. 6. CAN Controller Emulation: Software Layers

This layer stores permanent messages that have been received and are waiting to be explicitly fetched by the application. Filtering allows to define subsets of the message identifier space for discarding irrelevant messages and to avert a buffer overrun. Furthermore, a set of outgoing messages is maintained. A message is placed in this set by the application and it is removed from the set when it is either canceled by the application or it has become permanent at the receiver.

3. **Protocol Engine Emulation**: This layer establishes the communication semantics of a conventional CAN network. The behavior of the original CAN communication is emulated with respect to the ordering of incoming messages, the point in time of message reception, the acknowledgement for successful message transmissions and the processing of message cancellations. To accomplish this task the protocol emulation layer stores received CAN messages which have not become permanent yet. The permanence of messages is detected by analyzing the transmission request timestamps and by determining the current bus state. Permanent messages can be forwarded to the application interface layer.
Messages that have to be transmitted are also stored. They are handed over to the ET communication layer, if it indicates readiness for a new CAN frame. Messages that have been transmitted remain at the protocol engine emulation layer until they have become permanent at the receiver. During this period of time the messages can still be canceled. At the point in time when a message becomes permanent at a receiver the application interface layer is notified and the message is removed from the set of non-permanent outgoing messages.
Furthermore, this layer exploits the membership information to detect communication failures and to issue message retransmissions.

4. **Event Triggered Communication Channel**: This layer is responsible for dividing CAN frames into packets of fixed size, which can be placed in CAN slots. When an entire frame has been communicated, a new frame is fetched from the protocol engine. At the ET channel layer of the receiver incoming packets are stored individually for each sending node until a complete CAN frame can be restored. When a CAN frame has been assembled, it is forwarded to the protocol engine.
The assembly of frames requires knowledge about future frame starting points. Therefore, the first byte of every frame specifies the frame length. It allows nodes to determine the start of the next frame in the sequential data stream. However, this processing can only be performed by synchronized nodes, i. e. by nodes that have detected the frame starting points of prior CAN frames. If a communication error has invalidated one or more received packets, the global time available in a TT system is exploited to regain synchronization. The ET channel layer periodically inserts a control byte denoting the remainder of the currently communicated frame. The periodically recurring TDMA round during which this control byte is transmitted is

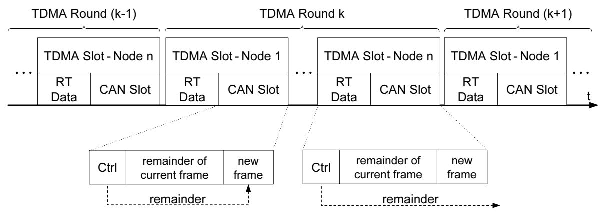

Fig. 7. TDMA Round with Control Bytes for Node-Integration

called the "reintegration round". An example for such a reintegration round is depicted in Figure 7.

C. Task Model

The model for the establishment of CAN communication services on top of a TT communication protocol provides two execution environments. A fundamental requirement for the corresponding task model is the prevention of temporal fault propagation from CAN based applications to the non-CAN environment. This fault containment can be established by realizing both the TT communication based application and the emulated CAN controller as TT tasks and by mapping the CAN based application on a background task. Furthermore, CAN based applications must not block the TT applications through the locking of resources (e. g. explicit synchronization). This prioritization of TT tasks ensures that CAN-based applications cannot delay time-triggered tasks and timing failures do not propagate from a legacy CAN application into the TT environment.

V. Conclusions and Future Research

This paper has described a solution for providing an authentic CAN communication service on top of a time-triggered communication service. The CAN application that is integrated into the TT environment benefits from the properties of the underlying reliable TT communication system. The emulation overcomes deficiencies of CAN like bandwidth limitations, inaccessibility times, and the lack of a consistent atomic multicast mechanism. This occurs at the protocol level without increasing application complexity.

Due to the support for coexistence of event-triggered CAN communication with time-triggered (TT) communication services, CAN legacy applications can be integrated into a TT environment without the need to redevelop existing CAN-based applications. Fault containment mechanisms have been described to avoid the propagation of temporal faults from the CAN legacy application into the TT application.

A realization of the model for the provision of CAN communication services on top of a TT communication protocol is described in [20]. This implementation emulates the behavior of an i82527 CAN controller in a TTP/C communication system. Our future research will focus on the integration of emulated CAN with a physical CAN network. We plan to develop a gateway between the TT environment and the CAN network. Furthermore, we will extend our work to other communication protocols like TCP/IP.

Acknowledgements

This work has been supported in part by the IST Project "DSoS" and the IST Project "Next TTA".

References

[1] W. Lawrenz. Worldwide status of CAN - present and future. In *ICC '95, London 2nd international CAN Conference Proceedings*, pages 0–12, 1995.

[2] C. Version and P. Philips. CAN specification version 2.0, parts A and B., 1992.

[3] P. Veríssimo, J. Rufino, and L. Ming. How hard is hard real-time communication on field-buses? In *Symposium on Fault-Tolerant Computing*, pages 112–121, 1997.

[4] J. Rufino. Dual-media redundancy mechanisms for CAN. Technical Report CSTC RT-97-01, Centro de Sistemas Telemáticos e Computacionais do Instituto Superior Técnico, Lisboa, Portugal, January 1997.

[5] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in CAN. In Digest of Papers, The 28th International Symposium on Fault-Tolerant Computing Systems, pages 150–159, Munich, Germany, IEEE, June 1998.

[6] J. Kaiser and M. A. Livani. Achieving fault-tolerant ordered broadcasts in CAN. In *European Dependable Computing Conference*, pages 351–363, 1999.

[7] M.A. Livani. Share: A transparent approach to fault-tolerant broadcast in CAN. In *Proc.6th International CAN Conference (ICC6), Torino, Italy*, 1999.

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

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

[10] E. Bretz. By-wire cars turn the corner. IEEE Spectrum, p. 68, Apr. 2001, 1997.

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

[12] S. Poledna. Fault-tolerant real-time systems: The problem of replica determinism. Kluwer Acad. Publishers, 1996.

[13] P. Veríssimo, J. Rufino, and L. Rodrigues. Enforcing real-time behaviour of LAN-based protocols. In Proceedings of the 10th IFAC Workshop on Distributed Computer Control Systems, Semmering, Austria, IFAC, September 1991.

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

[15] R. Mores, G. Hay, R. Belschner, J. Berwanger, C. Ebner, S. Fluhrer, E. Fuchs, B. Hedenetz, W. Kuffner, A. Krüger, P. Lohrmann D. Millinger, M. Peller, J. Ruh, A. Schedl, and M. Sprachmann. FlexRay – the communication system for advanced automotive control systems. *SAE 2001 World Congress, Detroit, MI, USA*, Doc.No 2001-01-0676, March 2001.

[16] R. Belschner, J. Berwanger, C. Ebner, S. Fluhrer, T. Führer, F. Hartwich, B. Hedenetz, R. Hugel, A. Knapp, J. Krammer, P. Lohrmann, B. Müller, M. Peller, and A. Schedl. FlexRay – requirements specification. *BMW AG., DaimlerChrysler AG., and Robert Bosch GmbH*, 2001.

[17] M. Peller. Byteflight – a new high performance data bus system for safety-related applications. *IIR Seminar on Latest Innovations in Smart Car Sensing and Safety Systems, London*, June 2000.

[18] C. Temple. Avoiding the babbling-idiot failure in a time-triggered communication system. In *Symposium on Fault-Tolerant Computing*, pages 218–227, 1998.

[19] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther. Time-triggered CAN - TTCAN: Time-triggered communication on CAN. Proc. 6th International CAN Conference (ICC6), Torino, Italy, 2000.

[20] R. Obermaisser. CAN emulation in a time-triggered environment. Technical Report 22, Technische Universität Wien, Institut für Technische Informatik, April 2002.