Combining Operational Flexibility and Dependability in FTT-CAN

Joaquim Ferreira, Luís Almeida, *Member, IEEE*, José Alberto Fonseca, *Member, IEEE*, Paulo Pedreiras, *Member, IEEE*, Ernesto Martins, Guillermo Rodríguez-Navas, Joan Rigo, and Julián Proenza

***Abstract*—The traditional approaches to the design of distributed safety-critical systems, due to fault-tolerance reasons, have mostly considered static cyclic table-based traffic scheduling. However, there is a growing demand for operational flexibility and integration, mainly to improve efficiency in the use of system resources, with the network playing a central role to support such properties. This calls for dynamic online traffic scheduling techniques so that dynamic communication requirements are adequately supported. Nevertheless, using dynamic traffic management mechanisms raises additional problems, in terms of fault-tolerance, related with the weaker knowledge of the future system state caused by the higher level of operational flexibility. Such problems have been recently addressed in the scope of using flexible time-triggered CAN (FTT-CAN) in safety-critical applications in order to benefit from the high operational flexibility of this protocol. This paper gathers and reviews the main mechanisms that were developed to provide dependability to the protocol, namely, master replication and fail-silence enforcement.**

***Index Terms*—Distributed computing, fault tolerance, real-time systems, safety.**

I. Introduction

DISTRIBUTED embedded systems (DES) have been widely used in the last few decades in several application fields, ranging from industrial machinery to avionics and automotive systems. In some of these application domains, the dependability requirements are of utmost importance, since failing to provide services in a timely and predictable manner may cause important economic losses or even put human life in risk [1].

The adoption of the best practices in the design of distributed embedded systems does not fully avoid the occurrence of faults, arising from the non-deterministic behavior of the operational environment. Thus, fault-tolerance mechanisms need to be included in the DES to prevent possible faults from causing system failure. To be effective, fault-tolerance mechanisms require *a priori* knowledge of the correct system behavior to be capable of distinguishing it from erroneous behaviors. Traditionally, when designing fault-tolerance mechanisms, the *a priori* knowledge means that all possible operational modes are known at system design time and cannot change at runtime [2]. This leads to systems that are either fully static or allow a small number of operational modes only. On the other hand, higher operational flexibility would allow supporting evolving requirements, adapting to environment or system configuration changes, simplifying maintenance and repair, and improving the efficiency in resources utilization. This efficiency might impact positively on the system cost because with the same resources, one can add more functionality or one can offer the same functionality with fewer resources.

Also, in the recent years, real-time systems have been used in more versatile and, often, dynamic scenarios. According to [3], the application requirements of new and planned autonomous and time-critical systems include a flexible distributed system infrastructure that can adapt robustly to dynamic changes in mission requirements and environmental conditions. This requires the support of the lower communication infrastructure to deliver flexible yet dependable services to the application.

However, flexibility and dependability are often regarded as conflicting concepts [1] because flexibility implies the ability to deal with evolving requirements that, in turn, can lead to unpredictable and possibly unsafe operating scenarios. However, this is true only when assuming unbounded flexibility. If flexibility is adequately bounded and the adaptation to evolving requirements is appropriately controlled, then it might be possible to enforce continuous dependability. This paper addresses such possibility within the scope of the flexible time-triggered CAN (FTT-CAN) protocol and proposes a set of mechanisms that support dependable adaptation.

The rest of this paper is organized as follows. The next section compares the flexibility attributes of some communication protocols for safety-critical embedded applications. Section III briefly presents the FTT-CAN protocol, while Section IV describes the basic design decisions made to provide fault-tolerance in FTT-CAN and describes the fault hypothesis considered. Section V addresses the replication of the FTT-CAN master, and Section VI addresses the enforcement of fail-silence in FTT-CAN nodes. Finally, Section VII presents the concluding remarks.

II. Flexibility of Some Communication Protocols for Safety-Critical Embedded Applications

Concerning communication protocols for dependable embedded applications, there is a wide range of available options. In this section, we consider a few that are representative of the

Manuscript received November 21, 2005; revised January 29, 2006.

J. Ferreira is with the Polytechnic Institute of Castelo Branco, 6000-767 Castelo Branco, Portugal (e-mail: jjf@est.ipcb.pt).

L. Almeida, J. A. Fonseca, P. Pedreiras, and E. F. V. Martins are with the Electronics and Telecommunications Department, University of Aveiro, 3810-193 Aveiro, Portugal (e-mail: lda@det.ua.pt; jaf@det.ua.pt; pedreiras@det.ua.pt; evm@det.ua.pt).

G. Rodríguez-Navas and J. Proenza are with the University of the Balearic Islands, 07122 Palma de Mallorca, Spain (e-mail: guillermo.rodriguez-navas@uib.es; julian.proenza@uib.es).

J. Rigo is with Telefónica de España and also with the University of the Balearic Islands, 07122 Palma de Mallorca, Spain (e-mail: juan.rigovadell@telefonica.es).

Digital Object Identifier 10.1109/TII.2005.875508

automotive and avionics industries, namely, CAN [12] [22], [23], TTCAN [4], TTP/C[5], FlexRay[6], and ARINC-629[7], as well as a research protocol, namely, FTT-CAN[8].

When addressing operational flexibility, event-triggered communication systems such as native CAN are typically well positioned because they react promptly to communication requests that can be issued at any instant in time, letting the bus available in the absence of events to communicate. Conversely, time-triggered systems such as TTP/C and TTCAN are not so flexible because communication takes place at predefined instants only, not taking into account run-time variations in the application communication requirements. This can be improved, as in TTP/C, allowing mode changes between a set of predefined (static) modes. Moreover, any time-triggered protocol allows reserving space at design-time for nodes and message streams that can then be added online, but this is an inefficient technique because extra bandwidth is kept allocated, even when it is not needed for long periods.

The systems that combine both event- and time-triggered paradigms present an intermediate level of operational flexibility due to the event-triggered part, despite the low flexibility of the time-triggered one. This is the case of TTCAN, FlexRay, and ARINC-629. This is even the reason why FlexRay claims to be flexible. Moreover, TTCAN, FlexRay, and TTP/C block online changes to the traffic schedule matrix or table. These can only be performed in configuration mode, which implies halting the system.

The FTT-CAN protocol deserves a particular reference in what concerns operational flexibility since it was built explicitly with the purpose of improving this aspect. Therefore, it not only supports event- and time-triggered traffic with temporal isolation, but it also delivers flexible time-triggered communication services, allowing online changes to the periodic communication requirements with timeliness guarantees.

Concerning dependability, TTP/C and FlexRay are probably the most robust solutions, with an advantage toward TTP/C because of its membership service. The star topology that these protocols support also helps improve error detection and isolation capabilities. On the other hand, the philosophy of ARINC-629 is to leave all dependability issues to the application level, removing that concern from the communication protocol. TTCAN does not seem to meet all requirements for safety-critical distributed systems (e.g., redundant communication, fail-silent nodes). In fact, some authors [9]–[11] claim that native CAN is more dependable than TTCAN. The original proposal for FTT-CAN did not consider dependability aspects and thus presents a level similar to that of TTCAN in this aspect. Table I summarizes some of the properties of the communication protocols as discussed above. Among these protocols, FTT-CAN is in a favored position to combine a high level of operational flexibility with a high level of dependability as appropriate mechanisms are added to it. This is the main motivation for this paper.

TABLE I
SUMMARY OF COMMUNICATION PROTOCOLS' PROPERTIES

Protocol	MAC Protocol	Online Scheduling[3]	Built-in Fault-tol.	TT Flex.
CAN	CSMA-BA	n.a./yes	+++	n.a.
TTCAN	TDMA/CSMA-BA	no/yes	++	No
FAT-CAN	M-MS[2]/CSMA-BA	yes/yes	++	++++
TTP/C	TDMA	no/no	+++++	++
FlexRay	TDMA/FTDMA	no/yes	++++	No
ARINC-629	FTDMA	yes[1]/yes	+	+

[1]Possible, but not common; [2]Master-Multislave; [3]TT traffic/ET traffic

A. FTT-CAN: A Brief Presentation

The FTT-CAN protocol (flexible time-triggered communication on CAN) has been developed with the main purpose of combining a high level of operational flexibility with timeliness guarantees and high efficiency in the bandwidth utilization. It uses the dual-phase elementary cycle concept for isolated time and event-triggered communication. The time-triggered traffic is scheduled online and centrally in a particular node called master, facilitating online admission control of requests, thus being managed in a flexible way, under guaranteed timeliness.

The protocol relies on a relaxed master-slave medium access control in which the same master message triggers the transmission of messages in several slaves simultaneously (master/multi-slave). The eventual collisions between slave messages are handled by the native distributed arbitration of CAN. FTT-CAN slots the bus time in consecutive elementary cycles (ECs) with fixed duration. All nodes are synchronized at the start of each EC by the reception of a particular message known as EC trigger message (TM), which is sent by the master node.

Within each EC, the protocol defines two consecutive windows—asynchronous and synchronous—that correspond to two separate phases. The former one is used to convey event-triggered traffic, herein called asynchronous because the respective transmission requests can be issued at any instant. The latter one is used to convey time-triggered traffic, herein called synchronous because its transmission occurs synchronously with the ECs. The synchronous window of the nth EC has a duration that is set according to the traffic that is scheduled for it. The schedule for each EC is conveyed by the respective EC trigger message (see Fig. 1). Since this window is placed at the end of the EC, its starting instant is variable, and it is also encoded in the respective EC trigger message.

The sequence of asynchronous windows maintains the real-time native properties of CAN, namely, the arbitration mechanism, despite the interruptions caused by the interlacing with the synchronous windows. For a deeper description of the native mechanisms of CAN, refer to [12]. The strict temporal isolation between windows is enforced by preventing the start of transmissions that could not complete within the respective window.

The communication requirements are held in a database located in the master node, the system requirements database (SRDB). This database holds several components, one of which is the synchronous requirements table (SRT) that contains the description of the periodic message streams. Based on the SRT, an online scheduler builds the synchronous schedules for each EC (EC-schedules). These schedules are then inserted in the data area of the respective EC trigger message (see Fig. 1) and

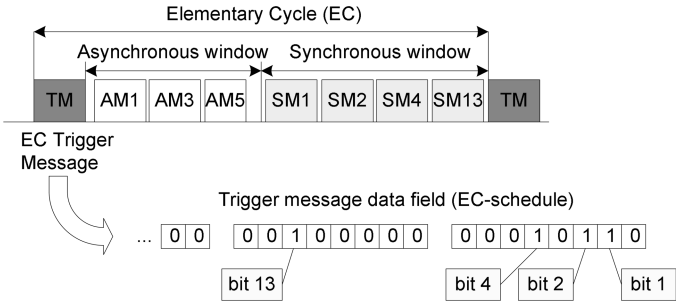


Fig. 1. Master/multislave access control and EC-schedule coding scheme.

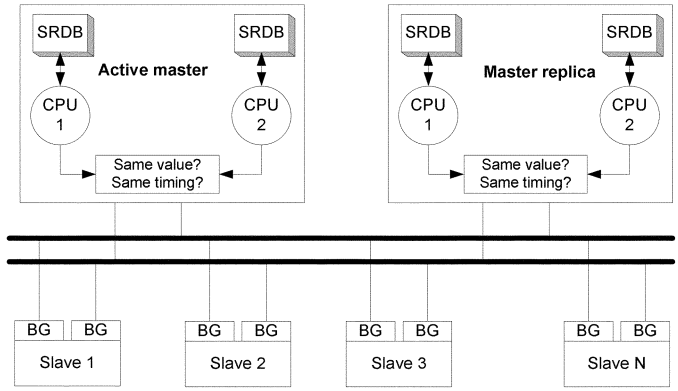


Fig. 2. Fault-tolerant FTT-CAN architecture based on a replicated broadcast bus, master replication, and bus guardians.

broadcast with it. Due to the online nature of the scheduling function, changes performed in the SRT at run-time will be reflected in the bus traffic within a bounded delay, resulting in a flexible behavior.

III. Achieving Fault-Tolerance in FTT-CAN

When addressing fault-tolerance in FTT-CAN, the most obvious issue that must be dealt with is the single point of failure formed by the master holding the SRT and the traffic scheduler. In fact, if the master node fails, no more trigger messages with EC-schedules are issued, and thus, communication is disrupted. This can be handled using replication, with one or more similar nodes acting as backup masters. However, all master replicas must be synchronized so that they can replace the active one without a discontinuity of the traffic schedule. In the case of protocols supporting online adaptation, such as FTT-CAN, that synchronization is more difficult as the communication requirements can evolve in time.

To facilitate the task of designing mechanisms that enforce masters synchronization, we consider that nodes are fail-silent. This, however, must also be enforced by using adequate components as nodes can fail uncontrollably.

The remainder of this paper is dedicated to describing the mechanisms that were developed to support masters synchronization and enforcing nodes fail-silence, which are the two cornerstones of our approach to support dependable adaptation. The former aspect includes a protocol to synchronize (re)starting masters by transferring the current SRT and another protocol to ensure consistent SRT updates upon change requests. The latter aspect considers bus guardians to enforce fail-silence in the time domain, which is suitable for slaves only, and a dual port network interface to support internal replication promoting fail-silence in the time and value domains, which is suitable for both masters and slaves.

The bus itself is another single point of failure, and thus, replication of the transmission path is essential to tolerate physical partition of the bus. Given these considerations, the proposed fault-tolerant FTT-CAN architecture is depicted in Fig. 2.

IV. Fault Hypothesis

The fault hypothesis that will be used throughout this paper is as follows.

- **Node faults**—Both masters and slaves are assumed to exhibit fail-silence failure semantics. This means that they can only fail by not issuing any message to the network. For the masters, this assumption is substantiated by their internal redundancy. For the slaves, this assumption can hold if internal redundancy is also used, or, if bus guardians are used instead, the assumption is restricted to timing faults.
- **Channel transient faults**—Only *transient* faults that change the value of at least one bit are considered. Experimental data concerning CAN bit-error rate [11] showed that CAN bit-error rate in an aggressive environment can be as low as 2.6×10^{-7}. Adapting the results of [13] to FTT-CAN, this value translates to at most four inconsistent message omissions per hour. Nevertheless, this paper does not consider any particular constraint on the bit-error rate nor on the duration of the error bursts, but it is assumed that the resulting network inaccessibility periods will not go beyond the physical controllability limit of the system.
- **Channel permanent faults**—The transmission medium is a single point of failure of FTT-CAN. However, permanent faults of the transmission medium, such as bus partition, are not considered in this paper. To tolerate such faults, medium redundancy must be included. One particular way of achieving such redundancy that is particularly simple and can be easily used to complement our work is Rufino's *Columbus egg* [14].
- **Synchrony assumptions**—It is assumed that nodes, both masters and slaves, are always synchronized. This assumption is based on the fact that the trigger message transmitted by the active master, besides conveying the scheduling information, also acts as a synchronization mark to all network nodes. That is, the master node is also the time master, and it is assumed that in between two consecutive trigger messages (typically 5–10 ms), the clock counters of each node do not diverge more than a negligible amount of time. In particular, bus guardians also have independent clock oscillators and synchronize on the reception of the trigger messages. As the synchronization requirements of FTT-CAN are relatively coarse, the assumption that masters and slaves are synchronized is assumed to hold, even

in the presence of error bursts that last for a relatively high number of consecutive ECs.

The reader should refer to [15] for further details on the fault hypothesis and on some underlying properties of CAN and FTT-CAN.

V. Master Replication

In order to achieve a successful master replication, so that a master backup takes over the bus scheduling in case the currently active master fails to transmit the trigger message, it is mandatory that all replicas maintain synchrony with the current active master. In our case, the replica that assumes the role of primary master must have the same knowledge of the faulty master, i.e., they must be replica determinate. The master nodes are replica determinate if, starting from the same initial conditions (same SRT) and fed by identical inputs (SRT update requests), they produce the same result (EC-schedule) at the same time (every EC).

Replication is a common technique to provide fault-tolerance, and it is used, for example, in TTCAN to cope with failures of the time master. However, master replication in FTT-CAN includes an extra complexity, resulting from the dynamic nature of the SRT. Since the system requirements, replicated at all masters SRTs, are flexible and may evolve over time, there is a potential for inconsistency in the replicated SRT images, compromising the replica determinism requirement. The following two situations are particularly relevant.

1) During an asynchronous startup/restart, a master node may lose the contents of its SRT. This calls for the definition of a protocol to transfer the SRT from the active master to the unsynchronized one.
2) During the processing of an SRT update request, it must be ensured that all replicas process the same request, in the same order, and commit the request synchronously. This calls for the definition of an adequate protocol to enforce consensus, preventing a master to accept a request while others may reject it or different masters committing a request at different instants.

Apart from these situations and during standard operation mode, there are no reasons for the replicas to diverge. To cope with these issues, the master node replication in FTT-CAN is addressed by means of four mechanisms:

- a policing mechanism that allows backup masters to detect loss of synchronization;
- a synchronization protocol to allow out-of-sync backups to re-synchronize in a short interval;
- an agreement protocol to handle SRT updates in order to minimize the potential for loss of synch;
- a master replacement mechanism upon active master failure.

A. Masters Synchronization

Whenever the active master receives a synchronization request, it initiates the download of its current SRT and scheduler state as necessary to resume scheduling synchronously [15]. This data transfer is carried out using high-priority asynchronous messages. The process may take a few ECs, depending on the size of the SRT and on the current network utilization. This is a time-critical process since during its execution modifications to the SRT are not allowed, and the synchronizing master is not yet capable of replacing the primary in case of failure.

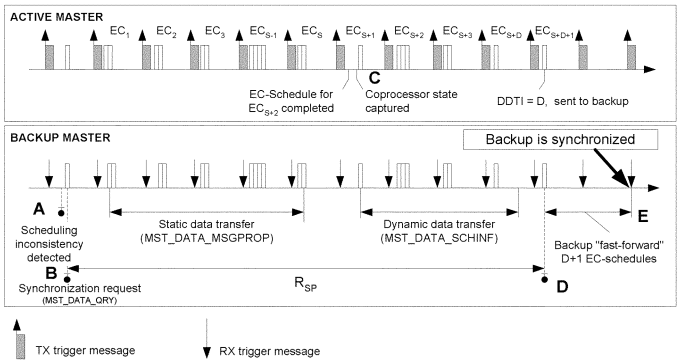


Fig. 3. Master synchronization protocol timeline.

The timeline of the synchronization process is depicted in Fig. 3. It begins with (A) a backup master detecting inconsistency on the received EC-schedule and (B) issuing a synchronization request (MST_DATA_QRY message) to the active master. This causes the latter to download the scheduling data in two rounds. The first round comprises the transmission of the static portion of the SRT (MST_DATA_MSGPROP), namely, IDs, periods, deadlines, and lengths. These data are fragmented in a set of CAN messages and transmitted in the asynchronous windows of several consecutive ECs.

In the second round, the active master transmits the scheduling state data (MST_DATA_SCHINF). This comprises the relative phases of all messages and their respective states. Because this transmission may span more than one EC, the data actually sent to the backup master are a snapshot of the phases and message scheduling states, just before the start of this round (C).

At the end of this round (D), the active master sends a special message containing the dynamic data transfer interval (DDTI), which is the count of the number of elapsed ECs since the scheduler state was captured. This number tells the backup master the scheduling delay (in ECs) relative to the active master. Since the backup master scheduler is now updated with the scheduling data just received from the primary, the backup master scheduler just needs to generate that number of consecutive EC-schedules (fast-forward). After that, the backup master is synchronized (E). Note in Fig. 3 that the backup master synchronizes only one EC after the reception of the DDTI message. This is to guarantee that the backup master has enough time (at least one EC long) to generate the required number of EC-schedules specified in the DDTI message, even when this message arrives late in the EC. The worst-case synchronization time can be upper bounded under certain assumptions, as explained in [15].

B. SRT Update Protocol

There is only one master operation that may modify the local copy of the SRT, i.e., the processing of an update request. Therefore, as long as each request is consistently processed by all the

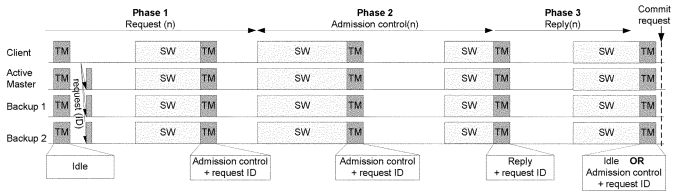


Fig. 4. Phases of the update protocol.

masters, the consistency of their SRT copies is preserved. In particular, if more than one request is received, then these requests must be processed in the same sequence. However, inconsistent channel errors, e.g., inconsistent omissions [13], turn out to be an impairment to this consistency because they may cause different masters to process requests in a different order.

This problem has been addressed previously from different perspectives and for different scenarios. For the specific case of FTT-CAN, a new agreement protocol was proposed in [15]. In order to eliminate inconsistencies among masters, the active master assumes the role of *leader*, whereas the backup masters assume the role of *followers*. Instead of using specific messages, the active master uses the TM to spread its state. In particular, the TM conveys the identifier of the request that is being processed as well as the protocol state. Whenever a backup master is not able to follow the active master (e.g., because it did not receive the request to be processed), it considers itself as being unsynchronized and initiates re-synchronization.

Fig. 4 illustrates how the SRT update protocol works in the absence of channel and node faults. The whole process is divided into three phases: **request, admission control**, and **reply**. In the request phase, the slave broadcasts an update request using a specific asynchronous message. If there is no pending request with higher priority, the active master changes its state to admission control and piggybacks the request ID and the protocol state on the next TM. Notice that the request ID is the FTT-CAN ID of the message the request refers to, taking 6 bits. The protocol state takes other 2 bits. After receiving this TM, all the masters, active and backup, wait until the end of the asynchronous window (following the one of the initial request) and start the admission control test. This delay is used to guarantee that slaves have at least a full asynchronous window to retransmit a request in case of error, reducing the probability of inconsistent omission.

As soon as the admission control test concludes, the active master broadcasts the result. Again, the result of the admission control and the new protocol state are piggybacked on the TM. This phase is called reply phase. At the end of this phase, all nodes (both masters and slaves) know which request has been processed as well as the result of the admission control test. Therefore, each master can update its local replica of the SRT, according to the request. Once the processing of an update request has concluded and the result of the admission control has been broadcast, the active master is ready to process another request.

A potential problem of a leader–follower approach, like the one described, is that an inconsistent transmission of the TM might cause inconsistency among the backup masters. However, the active master knows whether the transmission of its TM has been inconsistent [17] and takes advantage of this feature by keeping the same protocol state in the following TM. In this way, the delayed leader delays all followers. The result is similar if the active master crashes while processing an update request. Then, the backup that takes over relays the same protocol information received in the last TM to enforce its consistency, thus delaying the protocol.

Therefore, although replicated masters could face transient inconsistencies, such inconsistencies would be eventually solved after a consistent transmission of the TM. Thus, channel errors may enlarge the protocol execution but do not jeopardize consistency. A partial formal validation of this protocol using a PROMELA model and the SPIN model checker is presented in [15]. This SRT update protocol and the other mechanisms related with master replication were also experimentally validated in an autonomous mobile robot [18]. The system included a nine-node FTT-CAN network with 9 synchronous messages, 6 synchronous task triggers, 12 asynchronous messages, and an elementary cycle of 5 ms. A description of the experimental setup can be found in [19]. The experimental results confirm the proper operation of the referred fault-tolerance mechanisms.

VI. Enforcing Fail Silence in FTT-CAN

This section presents two different approaches to enforce fail-silent behavior: dynamic bus guardians and internal replication with temporized agreement. The latter mechanism enforces fail-silence both in the time and in the value domains, and it was designed to be used primarily on the master node, given its central role. Notice, however, that it can also be adopted in the slave nodes whenever needed, but it is a more costly technique. The former mechanism, in turn, is meant for slave nodes only, and it cannot be adopted in the masters because of the causal relationship between the master node computed schedule and the dynamic bus guardian operation.

A. Slave Nodes Fail-Silence Enforcement

Bus guardians are usually adopted to enforce fail-silence behavior in nodes of a distributed system because they are simpler than using node replication and some sort of voting mechanism. Apart from the cost issue, simplicity is also important because a simpler component is often more dependable than a complex one. Being that FTT-CAN is a two-phase protocol with event- and time-triggered parts, an FTT-CAN compliant bus guardian should police both protocol phases.

B. Bus Guardian

This bus guardian implements a subset of the CAN protocol and is able to examine the validity of an ongoing message transmission, right after the transmission of the respective ID and DLC message fields. The operation of the bus guardian is based on its capacity to observe the network state (Rx_{bus}) and the node transmission state (Tx_{node}), independently and bit by bit. In this way, the bus guardian needs to decode the node transmission, which accounts for the node contribution to the overall bus traffic in parallel with the decoding of the bus traffic.

Two classes of errors could be detected by this bus guardian: the logical errors related with the FTT-CAN protocol and the errors at CAN bitstream level. Examples of logical protocol errors

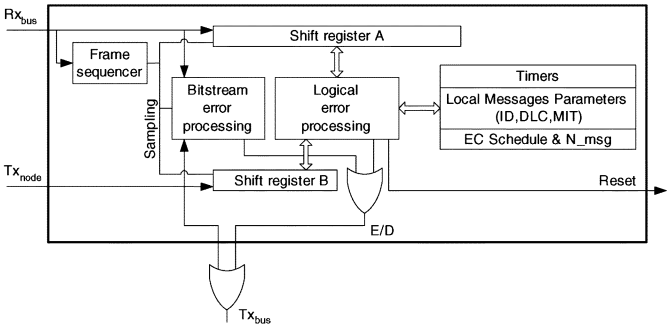


Fig. 5. Main building blocks of the bus guardian.

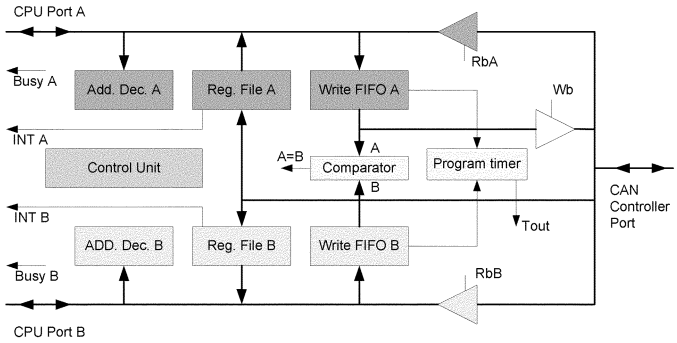


Fig. 6. Interfacing a pair of processors with a CAN controller in a master node to enforce fail-silent behavior.

are unsolicited synchronous message transmission and asynchronous message transmissions not respecting the minimum inter transmission time. Examples of bitstream errors include a node constantly sending active error frames or permanent dominant bits (stuck at dominant failure mode).

Fig. 5 depicts the main functional blocks of the bus guardian. The frame sequencer receives the input signal, resulting from the contributions of all nodes, including the local node when it is transmitting, and uses it to synchronize the bus guardian with the bitstream flowing in the bus. Thus, the frame sequencer extracts the clocking information from the bitstream and feeds it (sampling signal in Fig. 5) to the bitstream error processing unit and to the shift registers A and B. The bitstream error processing unit makes a bitwise comparison of the most recent bits received in the shift registers to detect any possible mismatch. In case the local node transmits, for example, an active error frame that is not present in the bus, possibly meaning that the local CAN controller is faulty, the bitstream error processing unit isolates the node from the bus. Notice that the bitstream error processing unit can evaluate the correctness of every transmission issued by the local node, because it receives the same bitstream that is also received by the local node CAN controller.

Shift register B stores the ID and the DLC of each message transmitted by the local CAN controller (15 bits). Shift register A stores the trigger message ID and DATA fields (75 bits). After receiving each TM and storing its content in the shift register A, the logical error processing unit extracts the node schedule for that EC, derives the number of messages to transmit (N_{msg}) in that EC, and sets the timer to enforce the temporal isolation between protocol phases. During the asynchronous window, the logical error processing unit analyzes the ID and DLC of the messages transmitted by the local node (shift register B) and blocks message transmissions that violate the predefined minimum inter-transmission time or ID or size. To validate the minimum inter-transmission time property of each message, a timer is set with that interval each time an asynchronous message is transmitted. A transmission is allowed when the timer interval has elapsed only.

During the synchronous window and with the knowledge of N_{msg}, the bus guardian drives the bus transmission line (Tx_{bus}) according to the following rules.

1) If $N_{\mathrm{msg}} > 0$ and the bus is idle, then $Tx_{\mathrm{bus}} = Tx_{\mathrm{node}}$ ($E/D = 0$). This allows the node to start a message transmission. N_{msg} is decremented for each successful transmission.
2) If $N_{\mathrm{msg}} = 0$ and the bus is idle, then $Tx_{\mathrm{bus}} =$ recessive$E/D = 1$), i.e., the node cannot start a message transmission.
3) If a transmission from other node is in progress, then $Tx_{\mathrm{bus}} = Tx_{\mathrm{node}}$ but only to transmit acknowledge bits, error frames, or overload frames. At all other times, $Tx_{\mathrm{bus}} =$ recessive$E/D = 1$).

Whenever the node starts a transmission during the synchronous window with $N_{\mathrm{msg}} > 0$, the bus guardian policies the transmission bit by bit until the end of the message ID and the DLC fields. If one of these fields is incorrect, the bus guardian immediately aborts the ongoing transmission, inducing an error in the bus and isolating the node from the network, possibly generating a hard reset to the host node processor and CAN controller. The worst-case fail-silence enforcement latency is close to 52 bit times, which is the transmission time of a minimum-sized CAN frame. In this way, there is no interference in the bus schedule by aborting the transmission of an erroneous message.

C. Master Nodes Fail-Silence Enforcement

For the case of the master node, and possibly some slave nodes, it is necessary to enforce fail-silence both in the time and value domains. This is mandatory in the master, to guarantee the correctness of the EC-schedule that is broadcast to the network. Fail-silence enforcement in FTT-CAN master nodes requires a replicated processing/voting scheme.

The proposed approach [21] is based on a specific network interface that supports internal duplication of the node in a transparent way. Basically, this interface enforces an agreement both in the time and in the value domain between all the messages generated by both internal replicas. In case of disagreement, no message is actually sent to the network. Fig. 6 depicts such a network interface designed for an FTT-CAN system called dual processor–CAN controller interface (DPCI). The CPUs run from independent clocks, and each of them is connected to a dedicated DPCI port (on the left side) through which it sees the CAN controller programming interface as if it was physically connected to it. Synchronization between DPCI and each CPU is enforced by semi-synchronous port interfaces. The reader should refer to [21] for further details concerning the DPCI.

VII. Conclusion

This paper discussed the growing interest for operational flexibility in current distributed embedded systems, including for safety-critical applications. However, current safety-critical communication protocols that consider static communication requirements or, at most, a set of predefined static operational modes do not support such a high level of operational flexibility.

Conversely, other protocols provide high flexibility but lack appropriate fault-tolerance capabilities. This is the case of FTT-CAN, which combines the advantages of time- and event-triggered paradigms while providing flexibility to the time-triggered traffic. This paper puts together the mechanisms that were recently developed to provide FTT-CAN with an adequate level of fault-tolerance to make it suitable for safety-critical applications.

This paper showed the fault hypothesis and presented an FTT-CAN system architecture with replicated masters and fail-silent nodes. Then, this paper focused on the specific problems and mechanisms related with master replication, particularly a protocol to enforce consistency during updates of replicated SRT and another protocol to transfer the SRT to an unsynchronized node upon asynchronous startup or restart.

Moreover, this paper discussed the implementation of fail-silence in FTT-CAN nodes and proposed two solutions, both based on hardware components that are attached to the node network interface. One solution relies on bus guardians that allow enforcing fail-silence in the time domain, and the other solution relies on a special network interface with duplicated host processor ports that provides transparent node internal replication.

This paper shows that it is possible to build fault-tolerant mechanisms for FTT-CAN that preserve its high level of operational flexibility, particularly concerning the time-triggered traffic. With such mechanisms, we argue that FTT-CAN is suitable for safety-critical applications, as indicated by the results obtained so far. Nevertheless, the verification and validation of the complete protocol is not yet complete. Further efforts are still being carried out to establish that suitability, including a thorough safety analysis and fault-injection.

References

[1] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications.*. Norwell, MA: Kluwer, 1997.

[2] RTCA/EUROCAE, Requirements Specification for Avionics Computer Resource (ACR), Radio Technical Commission for Aeronautics/European Organization for Civil Aviation Equipment, 2000, Tech. Rep.

[3] D. Schmidt, R. Schantz, M. Masters, J. Cross, D. Sharp, and L. Di-Palma, Towards Adaptive and Reflective Middleware for Network-Centric Combat Systems, CrossTalk, November 2001. [Online]. Available: http://www.cs.wustl.edu/schmidt/PDF/crosstalk.pdf. Feb. 21, 2005, 2001.

[4] Road vehicles-controller area network (CAN)–part 4: Time triggered communication, ISO, 2001.

[5] Time-Triggered Protocol TTP/C High-Level Specification Document (edition 1.0), TTTech. [Online]. Available: http://www.ttagroup.org, , 2002.

[6] FlexRay Consortium, FlexRay Communications System—Protocol Specification, version 2.0, 2004, FlexRay Consortium Tech. Rep.

[7] *Aeronautical Radio INC, ARINC Specification 629 Multi-Transmitter Data Bus*, , 1990.

[8] L. Almeida, P. Pedreiras, and J. A. Fonseca, "The FTT-CAN Protocol: Why and How," *IEEE Trans. Ind. Electron.*, vol. 49, no. 6, pp. 1189–1201, Dec. 2002.

[9] I. Broster, A. Burns, and G. Rodríguez-Navas, "Comparing real-time communication under electromagnetic interference," in *Proc. IEEE Computer Society 16th Euromicro Conf. Real-Time Systems*, Sicily, Italy, Jun. 2004.

[10] G. Rodríguez-Navas and J. Proenza, "Analyzing atomic broadcast in TTCAN networks," in *Proc. 5th IFAC Int. Conf. Fieldbus Systems Their Applications*, Aveiro, Portugal, 2003, pp. 153–156.

[11] J. Ferreira, A. Oliveira, P. Fonseca, and J. A. Fonseca, "An experiment to assess bit error rate in CAN," in *Proc. 3rd Int. Workshop Real-Time Networks Satellite (held in conjunction with the 16th Euromicro Int. Conf. Real-Time Systems)*, Jun. 2004.

[12] *ISO 11898. Road Vehicles—Interchange of digital information—Controller area network (CAN) for high speed communication, International Standards Organization*, , 1993.

[13] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodigues, "Fault-tolerant broadcast in CAN," in *Digest of Papers, 28th Int. Symp. Fault Tolerant Computer Systems*, 1998, pp. 150–159.

[14] J. Rufino, P. Veríssimo, and G. Arroz, "A Columbus' egg idea for CAN media redundancy," in *Digest of Papers, IEEE 29th Int. Symp. Fault-Tolerant Computing Systems*, Madison, WI, Jun. 1999, pp. 286–293.

[15] G. Rodríguez-Navas, J. Proenza, J. Rigo, J. Ferreira, L. Almeida, and J. Fonseca, "Design and modeling of a protocol to enforce consistency among replicated masters in FTT-CAN," in *Proc 5th Workshop Factory Communication Systems*, 2004, pp. 229–238.

[16] J. Ferreira, P. Pedreiras, L. Almeida, and J. Fonseca, "Achieving fault tolerance in FTT-CAN," in *Proc. 4th Workshop Factory Communication Systems*, 2002.

[17] J. Ferreira, L. Almeida, J. Fonseca, G. Rodriguez-Navas, and J. Proenza, "Enforcing consistency of communication requirements updates in FTT-CAN," in *Proc. Workshop Dependable Embedded Systems (held in conjunction with the 22nd Symp. Reliable Distributed Systems)*, Oct. 2003, pp. 7–12.

[18] R. Marau, V. Silva, J. Ferreira, L. Almeida, and J. A. Fonseca, "Assessment of FTT-CAN master replication mechanisms for safety-critical applications," in *Proc. SAE World Congress*, 2006.

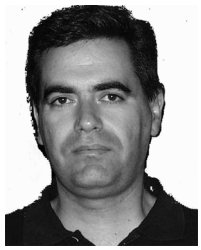
[19] V. Silva, R. Marau, L. Almeida, J. Ferreira, M. Calha, P. Pedreiras, and J. Fonseca, "Implementing a distributed sensing and actuation system: The CAMBADA robots case study," in *Proc. 10th IEEE Int. Conf. Emerging Technologies Factory Automation*, Sep. 2005.

[20] J. Ferreira, L. Almeida, and J. Fonseca, "Bus guardians for CAN: A taxonomy and a comparative study," in *Proc. Latin-American Workshop Dependable Automation System*, 2005, pp. 3–10.

[21] J. Ferreira, L. Almeida, E. Martins, P. Pedreiras, and J. Fonseca, "Components to enforce fail-silent behavior in dynamic master-slave systems," in *Proc. 5th IFAC Int. Symp. Intelligent Components Instruments Control Applications*, Jul. 2003.

[22] T. Nolte, M. Nolin, and H. A. Hansson, "Real-time server-based communication with CAN," *IEEE Trans. Ind. Informat.*, vol. 1, no. 3, pp. 192–201, Aug. 2005.

[23] S. Cavalieri, "Meeting real-time constraints in CAN," *IEEE Trans. Ind. Informat.*, vol. 1, no. 2, pp. 124–135, May 2005.



Joaquim Ferreira received the licenciatura degree in electronics and telecommunications engineering, the M.Sc. degree in electrical engineering, and the Ph.D. degree in informatics from the University of Aveiro, Aveiro, Portugal, in 1991, 1997, and 2005, respectively.

Currently, he is an Adjunct Professor at the Polytechnic Institute of Castelo Branco, Castelo Branco, Portugal. His research interests include real-time communication systems, fault tolerance, and distributed embedded systems.

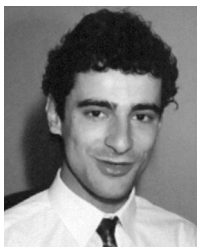
Luís Almeida (S'86–M'89) received the licenciatura degree in electronics and telecommunications engineering and the Ph.D. degree in electrical engineering from the University of Aveiro, Aveiro, Portugal, in 1988 and 1999, respectively.

He is an Assistant Professor in the Electronics and Telecommunications Department and a Senior Researcher at the IEETA research unit of the University of Aveiro. Formerly, he was a Design Engineer in a company producing digital telecommunications equipment. His research interests lie in the fields of real-time networks for distributed industrial/embedded systems and control architectures for mobile robots.

Dr. Almeida is a Senior Member of the IEEE Computer Society.

José Alberto Fonseca (M'00) received the licenciatura degree in electronics and telecommunications engineering and the Ph.D. degree in electrical engineering from the University of Aveiro, Aveiro, Portugal, in 1980 and 1992, respectively.

He has been an Associate Professor in the Electronics and Telecommunications Department of the University of Aveiro since 2000. His current research interests are embedded systems, distributed systems, and industrial communications.

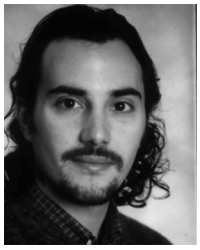


Paulo Pedreiras (M'03) received the licenciatura degree in electronics and telecommunications engineering and the Ph.D. degree in electrical engineering from the University of Aveiro, Aveiro, Portugal, in 1997 and 2003, respectively.

Currently, he is an Invited Assistant Professor in the Department of Electronics and Telecommunications at the University of Aveiro. His research interests include distributed embedded systems, real-time networks, real-time operating systems, and mobile robotics.

Ernesto Martins was born in Porto, Portugal, in 1962. He received the electronics engineer degree and the Ph.D. degree in electrical engineering from the University of Aveiro, Aveiro, Portugal, in 1986 and 1999, respectively.

Currently, he is an Assistant Professor in the Electronics and Telecommunications Department and a Senior Researcher at the IEETA research unit of the University of Aveiro. His technical and research interests include coprocessor architectures and reconfigurable custom computing machines for real-time applications as well as distributed embedded systems with real-time and fault-tolerance constraints.



Guillermo Rodríguez-Navas received the telecommunication engineer degree from the University of Vigo, Vigo, Spain, in 2001. He is currently pursuing the Ph.D. degree in the Department of Mathematics and Informatics, University of the Balearic Islands (UIB), Palma de Mallorca, Spain.

He is currently a Lecturer in the Department of Mathematics and Informatics at UIB, where he is also a member of the System, Robotics and Vision (SRV) research group. His research is focused on dependable and real-time distributed embedded systems. In particular, he has addressed various issues related to the CAN field bus, such as fault tolerance, clock synchronization, and response time analysis.

Joan Rigo received the licenciatura degree in computer science from the Universitat Politécnica de Catalunya, Barcelona, Spain, in 1990, and he is currently pursuing the Ph.D. degree in the Department of Mathematics and Informatics, University of the Balearic Islands (UIB), Palma de Mallorca, Spain.

He joined Telefónica de España, Palma de Mallorca, in 1990 and is currently designing data and voice network customer solutions. He is also a Part-Time Lecturer in the Department of Mathematics and Informatics at UIB. His research interests include specification and design of concurrent and real-time systems.

Julián Proenza received the licenciatura degree in physics from the University of the Balearic Islands (UIB), Palma de Mallorca, Spain, in 1989, and he is currently pursuing the Ph.D. degree in the Department of Mathematics and Informatics at UIB.

He is currently a Lecturer in the Department of Mathematics and Informatics at UIB. His research interests include dependable and real-time systems, fault-tolerant distributed systems, clock synchronization, and field-bus networks, such as CAN.