A first qualitative evaluation of star replication schemes for FTT-CAN

David Gessner, Manuel Barranco, Julián Proenza
Dpt. Matemàtiques i Informàtica
Universitat de les Illes Balears, Spain
david.gessner@uib.es

Michael Short
Electronics & Control Group
Teesside University, Middlesbrough, U.K.
m.short@tees.ac.uk

Abstract

Highly dependable distributed embedded systems (DES) have traditionally been developed using static approaches, i.e., assuming a mostly constant environment. However, the little flexibility of such approaches does not allow continuous operation under dynamic environments. The Flexible Time-Triggered (FTT) communication paradigm is a promising approach to introduce the required flexibility. However, for continuous operation reliability is also crucial. Replicated star topologies are particularly well-suited to provide an increased reliability. Nevertheless, for FTT-CAN, the implementation of FTT for CAN, no replicated star topology that takes advantage of FTT-CAN's features to increase reliability and error containment exists. This paper discusses important design questions that need to be solved to create such a novel solution.

1. Introduction

Traditionally, highly-dependable distributed embedded systems (DES) have been developed assuming that the communication requirements remain static during a system's operation. This has led to static approaches that prevent a system to function correctly when continuous operation under dynamic environments is required. The alternative are approaches that introduce flexibility. A particularly promising one is the Flexible Time-Triggered (FTT) communication paradigm [8], which has been applied to both the Ethernet and the Controller Area Network (CAN) protocols. However, flexibility alone is not enough for continuous operation under dynamic environments. It is also necessary to ensure that the system's reliability is high enough.

The application of the FTT paradigm to CAN is known as FTT-CAN [1]. It allows to dynamically change, at runtime, the communication requirements of a CAN-based DES while at the same time guaranteeing that all accepted changes will satisfy real-time constraints. Moreover, it supports both event- and time-triggered communication.

FTT-CAN has originally been designed on top of the standard CAN bus. A bus topology, however, has several reliability limitations [2]. Moreover, FTT-CAN introduces a master node that needs to be replicated in order to avoid a new single point of failure. As a solution, a replicated bus approach for FTT-CAN has been proposed [14], as well as mechanisms to replicate the FTT master [6].

These solutions, however, still have important dependability limitations. First, the master replication does not consider certain types of CAN fault scenarios [10, 12] that may prevent a node from detecting that a frame it has transmitted has been rejected by some nodes. Second, replicated bus topologies are vulnerable to common-mode failures such as spatial proximity failures due to the bus replicas coming near together at every node [2]. In contrast, a replicated star is less vulnerable to common-mode failures and can provide better error containment.

Recently, an approach to integrate ReCANcentrate [2], a replicated star topology for CAN, with FTT-CAN has been proposed [3]. ReCANcentrate is comprised of two hubs that can contain media errors at the ports to which the nodes are connected. The mentioned integration consists in considering FTT-CAN and ReCANcentrate as two orthogonal components and composing the former on top of the latter. Although this approach takes advantage of some of the dependability-related benefits of star topologies, it prevents the hubs from exploiting certain FTT-CAN features to further improve reliability and error containment, e.g., to treat errors in the time and value domains.

This paper explores several questions that need to be addressed to design a star replication scheme for FTT-CAN that takes advantage of FTT-CAN's features.

Section 2 describes the basics of FTT-CAN and Section 3 the fault model, assumptions, and goals under which different star replication schemes will be considered; Section 4 discusses the main problems that need to be solved; Section 5 concludes the paper and indicates future work.

2. FTT-CAN

FTT-CAN implements a centralized master/multi-slave access to the network. The master divides the communication time into rounds of fixed duration E called *Elementary Cycles* (ECs). Each EC is comprised of an *asynchronous window* followed by a *synchronous window*. The FTT master initiates each EC with the transmission of a *Trigger Message* (TM), to which the slave nodes synchronize. This message dictates the schedule for the next synchronous window, i.e., it tells the slave nodes which messages they should transmit during that window. The schedule is cal-

 doi:10.1109/ETFA.2012.6489702

culated by the master from the contents of a *System Requirements Database* (SRDB), which specifies the communication requirements for different message streams. Example requirements are deadlines, periods, and minimum inter-arrival times. Slave nodes may request changes to the SRDB, but these requests are subject to an online admission control performed by the master. The admission control basically ensures that the SRDB is only updated with the requested change if the system will still be schedulable afterwards. Regarding the asynchronous window, its access is event triggered.

3. Fault model, assumptions, and goals

Our fault model establishes the following. (i) The FTT masters have crash-failure semantics, which can be achieved through an internal duplication and comparison mechanism. (ii) A *hub channel*—a hub, together with all the cabling media and transceivers at the connected nodes and the hub—cannot generate spurious messages, which is obvious for the transceivers and cabling, and can be enforced for the hubs if necessary. (iii) The hub channels can suffer permanent or transient media faults. (iv) The slave nodes can generate faults in the time and value domains. Specifically, they can transmit messages at the wrong time (or not at all) or with the wrong contents. Regarding the latter, it is important to deal with messages with spurious CAN IDs because they can cause blocking and interference to legitimate messages that need to be delivered in a timely manner. However, wrong contents in the data field of messages is considered application-specific and is out of the scope of this paper, although it could be addressed, e.g., through the well-understood mechanisms of node replication and voting.

Regarding assumptions, we assume that each hub channel provides a *CAN broadcast domain* (simply broadcast domain from here onwards), which is defined as a set of one or more CAN segments (cabling plus CAN transceivers) interconnected in such a way that they satisfy two properties: (1) a new bit is not transmitted on any of the segments until the previous bit has stabilized in all of them, i.e., CAN's in-bit response is preserved, and (2) the value of the stabilized bit is the logical AND of all the latest bits that have been transmitted over each of the segments, i.e., the behavior is equivalent to CAN's wired-AND. CANcentrate [2] is an example of a hub that provides a broadcast domain.

We also assume that each hub, like the ones in CANcentrate [2], can detect media errors at its ports and isolate the ports if the media errors persist longer or appear more frequently than a given threshold.

Another assumption is that each hub channel provides Total-Order Broadcast (TOB), also known as atomic broadcast [4]. Although CAN does not always guarantee TOB [13], some authors used experimental data to claim that the chances of TOB violation in CAN are negligible [5]. Even if the claim is wrong, solutions have been proposed, e.g., [13, 11], and the issue appears solvable for FTT-CAN.

Finally, as indicated in the introduction, our goal is to allow highly-dependable DES to function correctly and continuously under dynamic environments. This means that we want to maximize both flexibility and reliability. Thus, the star replication schemes to be considered for FTT-CAN must maintain both FTT-CAN's flexibility and the containment of media errors provided by the hubs. In addition, the FTT-CAN masters will have to be replicated. Lastly, for consistency, trigger messages with different schedules must not reach the same node during the same EC.

4. Design questions

Although the solutions to use will be FTT-CAN specific, the problems to solve that we will discuss can be stated in general terms. For instance, any replication scheme needs to determine how to handle duplicates and how to deal with the partitioning of the communication participants (slave nodes and masters) into subsets that cannot communicate with each other. Furthermore, for any master/slave protocol with replicated masters, we need to decide how to enforce replica determinism [9] for the master replicas.

Handling duplicates. Duplicates can be categorized as *application caused duplicates* (a-duplicates), i.e., duplicates whose transmission has been explicitly requested by a correctly functioning application on a node; *redundancy caused duplicates* (r-duplicates), i.e., duplicates result of the channel replication; *faulty duplicates* (f-duplicates), i.e., duplicates caused by a spurious transmission by a faulty node; and retransmissions due to channel faults.

One approach to handling duplicates is to simply accept them. This is basically the approach currently followed in CAN, where it is well known that duplicates can occur. For this reason, CAN applications usually do not transmit messages that carry information that is relative to previous messages. Note that in CAN duplicates only appear occasionally, mainly due to retransmissions; whereas in replicated channels that redundantly send each message through all replicas of the channel (spatial redundancy), even under error-free conditions, duplicates occur for each and every message, namely in the form of r-duplicates.

Using the approach that has already been used in ReCANcentrate [2], it is possible to avoid delivering to the application these additional duplicates. For this, all the hub channels need to be interconnected such that they form a single broadcast domain. This allows to clearly identify r-duplicates because they are the only ones that could be received within the same bit time through the redundant links of a node. A node could then simply ensure that of all the r-duplicates only one is delivered to the application.

With an additional assumption, it is even possible to ensure that only the strictly necessary duplicates, namely a-duplicates, are delivered. The assumption is that it is not required that two indistinguishable messages, i.e., with the same ID and data, are delivered to a node during the same EC. For synchronous messages this assumption holds because FTT-CAN requires them to have a period that is an integer multiple of E [1]. For asynchronous messages there is no equivalent restriction. Nevertheless, these messages

need to have a minimum inter-arrival time for their temporal properties to be verified and we believe that most applications do not require this time to be shorter than a typical EC length. In any case, even if a shorter minimum inter-arrival time is required, messages can easily be made distinguishable, e.g., by adding short sequence numbers to the data field. With the assumption, a node simply needs to keep track of the messages it received since the start of the current EC and, when it receives a duplicate in the same EC, it can discard it because it must be an r- or f-duplicate, or a retransmission of a message it already has. This approach also has the advantage that it does not require the hub channels to form a single broadcast domain.

Avoiding partitioning. If all participants (nodes and masters) are correctly connected to at least one and the same broadcast domain, they can communicate and no partitioning exists. Thus, a way to reduce the chances of partitions when links are affected by faults is to interconnect the hubs to provide the coupling of all the broadcast domains into a single one like in ReCANcentrate.

Another solution is to keep separate broadcast domains and use a store and forward mechanism through hub interconnections to transport messages from one broadcast domain to another when link faults occurred that could lead to a partitioning. An important distinction with respect to forming a single broadcast domain is that errors in one hub channel would not necessarily affect others. If the same messages are transmitted over different broadcast domains, transient faults could thus also be tolerated through spatial instead of through temporal redundancy (retransmissions) alone. However, once there is no longer a shared broadcast domain, e.g., due to faults, it becomes necessary to transport each and every message to ensure communication between all participants, which doubles the required bandwidth. In that case, the initial reduction in bandwidth achieved by tolerating some transient faults through spatial redundancy, instead of by temporal redundancy, becomes negligible. On the other hand, for some applications this could be an acceptable degraded mode of operation.

Enforcing replica determinism of masters. Enforcing replica determinism of the masters depends on several factors. We will discuss the following important ones: (i) the definition of replica determinism for the masters; (ii) the number of active masters; (iii) the channel available for the masters to communicate with each other; (iv) the location of the masters; and (v) whether masters need to schedule retransmissions.

The first factor requires us to define when we consider masters to be replica deterministic. As mentioned in Section 3, different schedules must not reach the nodes during the same EC. This must be true independently of the number of masters. Taking this as the definition of replica determinism of masters, it will be achieved when they have obtained the same schedule for each EC.

The second factor is the number of *active* masters, i.e., masters that are not a backup, but are currently responsible for transmitting a TM in a broadcast domain.

If there is more than one, each responsible for a different set of broadcast domains, avoiding different schedules reaching the same node requires them all to convey the same schedule in their TMs during each EC. This requires an agreement protocol to be executed just before sending the TM in each broadcast domain.

The protocol could be an SRDB synchronization protocol that ensures that all masters have an identical copy of the SRDB just before each of them calculates the schedule. Since the masters have crash-failure semantics, it would be guaranteed that they only send a TM if they calculated the same correct schedule.

Another option is having the agreement protocol be master/slave itself, i.e., one of the active masters is the primary one and it uses TOB to inform the other (secondary) masters of the schedule to transmit in the next TM.

Alternatively, assuming the masters started with equal SRDBs, the agreement protocol could simply consist in ensuring TOB of slave update requests in the set of all broadcast domains (as opposed to TOB in each individual broadcast domain). This is so because, considering the crash-failure semantics of the masters, if two or more masters have different SRDBs, this can only be due to them having received a different subset of the SRDB update requests issued by slaves. This solution, by itself, only seems promising if no new masters can be added to a running system, no reintegration of faulty masters is required, and (as discussed shortly) masters do not reschedule messages corrupted by errors. Otherwise, an SRDB synchronization protocol would be necessary anyway.

If there is only one active master and it is connected to and responsible for all broadcast domains, it must simply transmit the same TM in all of them. This avoids the problem of different schedules reaching the same node. However, there is still the issue of the active master crashing and a backup master taking over as seamlessly as possible, which in fact also exists when there is more than one active master. Thus, an agreement protocol is necessary in any case, even if it is just between active and backup masters, to seamlessly tolerate the failure of an active master. Moreover, it is necessary for the masters to communicate to indicate whether they are alive or not.

In any case, communication between masters is necessary, for which we can distinguish two broad solutions. Use the hub channels themselves or have a separate channel, which we may call *master channel*. How masters communicate impacts the design of the replica determinism enforcement and to make a choice the location of the masters must also be considered.

The basic choices are masters within the hubs or within nodes. If they are within nodes, having a dedicated master channel would be costly since an additional full communication infrastructure just for the masters would be required. It may therefore be more sensible to use the hub channels. However, this would take away bandwidth for the communication of slave nodes. If they are in the hubs, the already necessary infrastructure that connects hubs to solve the par-

titioning problem may be used.

Regarding the location of the masters, there are a few additional worthwhile comments. Since the SRDB is a specification of correct message streams, it would allow a hub to use that information to implement more advanced error treatment mechanisms, e.g., a hub could check whether a node connected to a port is transmitting more frequently than the specification allows. If the masters are in the hubs, the hubs could have direct access to a copy of the SRDB. If they are in the nodes, the access would be more complicated. However, the big disadvantage of masters in the hubs are common mode failures between masters and hub channels. On the other hand, since the masters would be embedded in already necessary components (the hubs), less hardware would be required (which reduces costs and may improve overall reliability). Regarding scalability, adding additional masters is not as trivial in hubs as when they are simply nodes.

Finally, the replica determinism of the FTT masters also depends on whether messages affected by transient errors are autonomously retransmitted by the sender or whether it is the master that reschedules a message corrupted by errors [7]. In the latter case, the schedule calculated by the masters not only depends on the contents of the SRDB, but also on whether messages have been corrupted by errors or not. This means that the masters must all agree on whether a message needs to be rescheduled or not. To achieve this, one possibility is to use, as before, an agreement protocol to decide if a rescheduling should be done.

5. Conclusions and future work

This paper discusses trade-offs and potential solutions for important design questions in a star replication scheme for FTT-CAN, namely, how to handle duplicates, how to avoid the partitioning of the communication participants, and how to enforce replica determinism for the masters.

However, without doing a quantitative dependability evaluation of different replication schemes it is not clear which scheme is best for reliability. This will be the next problem to solve. Nevertheless, from a purely qualitative point of view, the most promising one has the following features. The hubs contain the FTT masters, which gives them direct access to a copy of the SRDB and allows them to use that copy to detect and isolate nodes that behave incorrectly in the time and value domains; the protocol used to ensure replica determinism of the masters is an SRDB synchronization protocol, which allows the reintegration of masters that have become faulty, and the online addition of new masters. Finally, separate broadcast domains are kept initially, allowing spatial redundancy to tolerate some transient faults. Only after a partitioning is detected, the hubs would reconfigure themselves to form a single broadcast domain, which would allow all nodes that have at least one correct link to a correct hub to communicate with each other, but without the negative impact on bandwidth that a store and forward mechanism would have.

As further future work we can point out the design and implementation of a concrete star replication solution for FTT-CAN.

Acknowledgement

This work was supported by the Spanish Economy and Competitivity Ministry with grant DPI2011-22992 and by FEDER funding.

References

[1] L. Almeida, P. Pedreiras, and J. A. Fonseca. The FTT-CAN protocol: Why and how. *Industrial Electronics, IEEE Trans. on*, 49(6):1189–1201, 2002.

[2] M. Barranco, J. Proenza, and L. Almeida. Boosting the Robustness of Controller Area Networks: CANcentrate and ReCANcentrate. *Computer*, 42(5):66–73, May 2009.

[3] M. Barranco, G. Rodriguez-Navas, D. Gessner, and J. Proenza. Towards the integration of flexible-time-triggered communication and replicated star topologies in CAN. In *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conf. on*, pages 1–4. IEEE, 2011.

[4] X. Défago, A. Schiper, and P. Urbán. Total Order Broadcast and Multicast Algorithms: Taxonomy And Survey. *ACM Computing Surveys*, 36:2004, 2003.

[5] J. Ferreira. *Fault-Tolerance in Flexible Real-Time Communication Systems*. Phd, Universidade de Aveiro, 2005.

[6] J. Ferreira, L. Almeida, J. A. Fonseca, P. Pedreiras, E. Martins, G. Rodriguez-Navas, J. Rigo, and J. Proenza. Combining operational flexibility and dependability in FTT-CAN. *Industrial Informatics, IEEE Trans. on*, 2(2):95–102, 2006.

[7] L. Marques, V. Vasconcelos, P. Pedreiras, and L. Almeida. Tolerating Transient Communication Faults with Online Traffic Scheduling. In *IEEE Int. Conf. on Industrial Technology*, Athens, 2012.

[8] P. Pedreiras and L. Almeida. The flexible time-triggered (FTT) paradigm: an approach to QoS management in distributed real-time systems. In *Proc. Int. Parallel and Distributed Processing Symposium*, page 9. IEEE Comput. Soc, 2001.

[9] S. Poledna. *Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism*. Kluwer Academic Publishers, 1996.

[10] J. Proenza and J. Miro-Julia. MajorCAN: A Modification to the Controller Area Network Protocol to Achieve Atomic Broadcast. *IEEE Int. Workshop on Group Communication and Computations, Taipei, Taiwan*, 2000.

[11] J. Proenza and E. Sigg. A first design for CANsistant: A mechanism to prevent inconsistent omissions in CAN in the presence of multiple errors. In *Conf. on Emerging Technologies & Factory Automation*, pages 1–4. IEEE, Sept. 2009.

[12] G. Rodriguez-Navas, C. Winter, and J. Proenza. Injection of aggregated error flags as a means to guarantee consistent error detection in CAN. In *Int. Conf. on Emerging Technologies and Factory Automation*, pages 1–4, Toulouse, Sept. 2011. IEEE.

[13] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-Tolerant Broadcasts in CAN. In *FTCS '98: Proc.s of the The Twenty-Eighth Annual Int. Symposium on Fault-Tolerant Computing*, page 150, Washington, DC, USA, 1998. IEEE Computer Society.

[14] V. F. Silva, J. A. Fonseca, and J. Ferreira. Adapting the FTT-CAN Master for multiple-bus operation. In *5th IEEE Int. Conf. on Industrial Informatics*, pages 305–310. IEEE, July 2007.