Towards the Integration of Flexible-Time-Triggered Communication and Replicated Star Topologies in CAN

Manuel Barranco, Guillermo Rodriguez-Navas, David Gessner, Julián Proenza
Dpt. Matemàtiques i Informàtica, Universitat de les Illes Balears, Spain
{manuel.barranco, guillermo.rodriguez-navas, david.gessner, julian.proenza}@uib.es

Abstract

There is a growing interest in making the CAN fieldbus more suitable for dependable applications. In the past years, several dependability limitations of CAN have already been addressed and a significant number of solutions are available. Nevertheless, the integration of these solutions into a single communication infrastructure is still an open issue. In this paper we discuss the integration of two specific solutions: FTT-CAN and ReCANcentrate. FTT-CAN is a higher-layer protocol that guarantees flexible real-time scheduling of CAN messages; whereas ReCANcentrate is a duplicated star topology for CAN that includes several enhanced mechanisms for media fault tolerance. We show how they are integrated into a single architecture that preserves the properties of each solution.

1. Introduction

Highly-dependable distributed embedded systems have traditionally been developed using static approaches, i.e. assuming that the environment and operating conditions remain mostly constant. However, the little flexibility of these approaches makes them very inefficient when the system is used in dynamic environments. For this reason, several researchers are proposing ways to add more flexibility to these systems, especially in the communication services provided by the network. Nevertheless, the introduction of new flexible communication services requires a careful evaluation since they may conflict with other services of the system, particularly with the fault-tolerance mechanisms.

The Flexible Time-Triggered CAN protocol (FTT-CAN) [4] is a higher-layer protocol intended to introduce more flexibility in the scheduling of real-time messages in CAN. In order to be suitable for dependable systems, FTT-CAN includes some mechanisms for fault tolerance, such as on-line rescheduling, master replication, and bus guardians [4]. Besides, the application of FTT-CAN with replicated CAN buses has also been investigated [5]. However, replicated buses and bus guardians have important dependability limitations [3] that replicated star topologies do not have. Thus, in our current project CANbids we are taking the next step to make FTT-CAN suitable for highly-dependable systems. In this paper we assess the integration of FTT-CAN with ReCANcentrate, a replicated star topology for CAN that provides advanced media fault-tolerance capabilities [3].

The paper is organized as follows. Section 2 describes the architecture and basic features of FTT-CAN and ReCANcentrate; Section 3 shows the approach followed to integrate the communication services and the fault-tolerance mechanisms of these two technologies; Section 4 describes new functionalities that, for the integration, should be added to an existing ReCANcentrate software driver executing on the nodes; Section 5 covers a few implementation aspects of the proposed integration approach. Finally, Section 6 concludes the paper and indicates future work.

2. FTT-CAN and ReCANcentrate basics

FTT-CAN implements a centralized master/multislave access to the network. The master divides the communication into temporal rounds called *Elementary Cycles* (ECs), by periodically triggering a special message, called *trigger message* (TM), to which the slave nodes synchronize [4]. Each EC combines both event- and time-triggered traffic based on the schedule information spread by the master within the TM. Moreover, the master is able to conduct on-line scheduling updates, thus providing further communication flexibility.

With respect to its fault-tolerance mechanisms, FTT-CAN's on-line re-scheduling capabilities, master replication, and bus guardians allow tolerating faults that manifest as an *unscheduled transmission* (UTX), a *scheduled message omission* (SMO), or an *inconsistent reception of a management message* (IMR). A UTX happens when a node tries to transmit a message that was not scheduled by the TM or when the TM itself is untimely transmitted. An SMO occurs when a node does not transmit/receive a scheduled message or the TM itself. Finally, an IMR happens when a message that conveys information related to the management of the protocol, e.g. scheduling update information, is not consistently received by all nodes.

Regarding ReCANcentrate, it is a replicated star with two hubs (Figure 1) interconnected by means of at least two interlinks [3]. Each hub sends to the other the signal obtained after coupling the non-faulty contributions of its own uplinks. Then, each hub couples its own contribution with the other's, providing a single broadcast domain. This is done in a fraction of the bit time [3], so that both hubs behave like one, thus being transparent to the nodes while providing error-containment. Specifically, each hub is able to detect and isolate, at the corresponding port, faults that compel any node or link/interlink to generate stuck-at-recessive, stuck-at-dominant, or bit-flipping streams [3].

A ReCANcentrate node is constituted by commercial off-the-shelf (COTS) components: a microcotroller, two CAN controllers, and four transceivers (Figure 1). Each CAN controller is connected to only one hub using one transceiver for the uplink and another for the downlink. The single broadcast domain allows each node to easily

 doi:10.1109/ETFA.2011.6059170

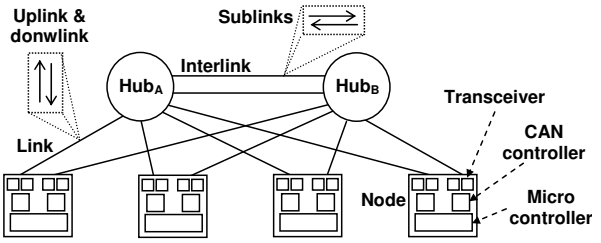

Figure 1. ReCANcentrate architecture

manage the replicated traffic [2]. Basically, one of the controllers acts as the transmission (tx) controller and the other one is the non-transmission (non-tx) controller. The node receives simultaneously (bit by bit) from both controllers, but transmits through the tx controller only. Furthermore, the node uses a tx timer and the native fault-treatment mechanisms of CAN (based on the TEC/REC [1]) to diagnose when it cannot communicate through a given star or a controller crashed. When this happens, the node discards the affected controller for communicating and uses the other one only, both for transmissions and receptions.

3. Integration approach

This section analyzes the possible interactions between the communication services and fault tolerance mechanisms of FTT-CAN and ReCANcentrate. Then, this analysis is used to discus the approach we are following to integrate all these mechanisms, while ensuring that they will still work properly after the integration.

3.1. Functional integration

FTT-CAN is implemented as an extra layer between the application software and the CAN controller. Therefore, according to the OSI model it can be seen as a part of the Application layer, since it provides common communication services that are used by the applications executed by the nodes. For these applications FTT-CAN is just a software driver that provides the required FTT communication primitives, such as transmission requests and on-line change scheduling requests, among others. This driver is called FTTdrv and it is responsible for all the interaction with the CAN controller. This means that in an FTT-CAN node, the application software does not access the CAN controller directly, but only through the FTTdrv driver.

ReCANcentrate has been designed with the idea of hiding its own mechanisms from the application. In this sense, each node is provided with a media management software driver called reCANdrv, which works at the Data Link Layer level to manage the replicated traffic (and treat media and controller faults) transparently to the application. ReCANdrv provides a simple interface equivalent to what we call a *virtual CAN controller*, so that the application sees reCANdrv as an API with the basic configuration and communication primitives of CAN.

As mentioned, to provide their communication services, both FTTdrv and reCANdrv need to access the CAN controller/s. This raises potential conflicts between both drivers when accessing the same device. Nevertheless, there is a significant difference between both drivers that eliminates this potential problem: whereas FTTdrv works as a user of the CAN services, reCANdrv works as a provider of them. This means that FTTdrv can be put on top of reCANdrv and use the services this driver supplies. The only requirement for this architecture is that the interface provided by reCANdrv (the so-called virtual CAN controller) must actually provide all the services that FTTdrv requires. The advantage of maintaining each driver at its original OSI layer is that it allows to reuse the FTT-CAN drivers already implemented at the University of Aveiro, e.g. [6], and the ReCANcentrate driver we developed [2].

When comparing the CAN communication services assumed by FTTdrv and the ones provided by reCANdrv, we noticed three aspects of reCANdrv that should be improved. First, reCANdrv must provide a primitive to abort messages, given that in FTT-CAN it is mandatory to abort the transmission of a message once the end of the corresponding temporal window is finished. Second, reCANdrv should include interruptions to notify of transmissions, receptions and aborts. This feature is mandatory to reduce the latencies. Third, reCANdrv could provide several transmission buffers to reduce the priority inversion problem. Due to the significant amount of design and implementation details related to the changes to be perform on reCANdrv to add these CAN services, we dedicate Sections 4 and 5 to discuss them.

Finally, note that to integrate the services of both drivers does not require any particular change on the hardware architecture of the system. On the one hand, FTT-CAN makes no assumption about the underlaying hardware. On the other hand, none of the hardware pieces of ReCANcentrate depends on any particular aspect of the applications (or protocols) executed by the nodes. The latter is particularly clear in the case of the hub, which is exclusively concerned with information at the bit and frame-field levels.

3.2. Fault-tolerance integration

In order to integrate the fault-tolerance mechanisms of ReCANcentrate and FTT-CAN, first it is necessary to identify the levels at which these mechanisms operate. As concerns ReCANcentrate, its mechanisms are implemented at the physical level of the network as well as at the Data Link Layer. For instance, the hub passivates faults acting at the physical layer, i.e. by disconnecting hub ports, whereas reCANdrv acts at the Data Link Layer level, e.g. it discards a controller for communicating if faulty.

Regarding FTT-CAN, its fault-tolerance mechanisms also work at different levels. On the one hand, *scheduled message omission* (SMO) and *inconsistent management message reception* (IMR) faults are treated by FTTdrv at the level of the FTT-protocol itself (application level). For example, when an SMO of a message that is not the TM occurs, the FTTdrv of the master re-schedules that message. Another example is the distributed algorithm executed by FTTdrv at each node to force consistent updates of the scheduling information.

On the other hand, *unscheduled transmission* (UTX) faults are treated by the *FTT bus guardians*. Each node has it own guardian, which is a hardware device that disables the contribution of the CAN controller to the bus during the appropriate phases of the EC, thereby preventing the node from transmitting unscheduled messages [4]. A bus guardian thus operates at two different levels. It uses the scheduling information contained in the TM, which belongs to the application level, whereas it passivates the output of the CAN controller by hardware, which corresponds to the physical network level.

The fault-tolerance mechanisms of ReCANcentrate can be seamlessly integrated with those of FTT-CAN that specifically address SMO and IMR faults. This is because the mechanisms of ReCANcentrate operate at a different level than these ones of FTT-CAN, so that they do not interact which each other.

Similarly, the FTT-CAN mechanisms that deal with UTX faults, i.e. those carried out by the bus guardian, can also be easily integrated. Concerning fault diagnosis, the guardian acts at the application level, so that it does not interfere with the ReCANcentrate mechanisms. As regards fault passivation, note that although the guardian and the hub act at the physical level, they perform their actions at a different location, i.e. the hub at its ports and the guardian at the output of the CAN controller. Therefore, bus guardians and ReCANcentrate are fully compatible with each other. A possibility of integration could be to attach one guardian to each one of the two CAN controllers of each node. This is easy to deploy, since the guardian only needs to have access to the transmision and reception signal of the CAN controller, which are available in a ReCANcentrate node. However, a most efficient solution would be to enhance the hub fault-diagnosis mechanisms in order to make it able to address UTX faults.

4. New functionalities for reCANdrv

4.1. Abort transmission primitive

The current API of reCANdrv already provides primitives to request a frame transmission, to read from the reCANdrv's reception buffers, and to configure several CAN communication parameters, such as the bit-rate. However, it does not include a primitive to abort a frame, which is required by FTTdrv to cancel transmissions that cannot be completed within a given EC phase.

In principle, such a primitive has to instruct the tx controller to halt the frame located at the hardware tx buffer corresponding to the to-be-aborted frame. However, no CAN controller is able to abort a frame that is being transmitted; instead, the abort will only take place after the ongoing transmission ends. This is the reason why CAN controllers typically include an interrupt or status bit to indicate the completion of an abort.

Thus, it is not enough to enhance reCANdrv with an abort transmission primitive, but we will also need to provide reCANdrv with the capability of forwarding notifications of completed aborts from the tx controller to its API.

4.2. Support for interrupts

The second new functionality required for reCANdrv has to do with the capacity of notifying of different events or situations. The current version of reCANdrv simply provides status information that can be polled through its API. Thus, it is necessary to allow reCANdrv to deliver notifications through interrupts as well. First, interrupts will eliminate the polling overhead otherwise introduced by FTTdrv to become aware of different events. Second, interrupts open the possibility of implementing a task-based version of FTTdrv or of any other application relying on ReCANdrv. In particular, we will provide reCANdrv with the capacity of generating interrupts for at least notifying about successful transmissions, receptions, and frame aborts, which are the CAN communication events that FTTdrv must be aware of.

To provide this support, first we will add a reCANdrv primitive that allows to specify the *Interrupt Service Routine* (ISR) to be called when a given event of interest occurs. Optionally, this primitive could allow to indicate the priority of the ISR itself. This will enable FTTdrv to set up, during a configuration phase, the ISRs of the FTT layer that should handle each event of interest.

As a second step, we have to provide reCANdrv with the capacity to generate interrupts. The current implementation of this driver already includes a mechanism to trigger the execution of some of its own internal procedures as if they were ISRs. We can now extend this mechanism to also launch the execution of external ISRs, such as those specified by the FTTdrv. Specifically, the mechanism consists in triggering, by software, interrupts of the hardware/software platform that are otherwise unused.

In particular, the current version of reCANdrv [2] uses some unused hardware interrupt sources of the dsPIC^{TM}microcontroller it relies on. To launch a given ISR, reCANdrv simply sets the appropriate bit of the interrupt register corresponding to the ISR's hardware interrupt. This strategy can be adapted to other platforms as well.

4.3. Multiple transmission buffers

So far reCANdrv includes just one transmission (tx) buffer, implemented in software, whose content is transferred to one of the hardware tx buffers of the tx controller for its transmission. However, some FTTdrv versions, e.g. [6], use multiple tx buffers present in some CAN controllers to reduce the priority inversion problem during the EC. Thus, although optional, it would be desirable to extend reCANdrv to include as many software tx buffers as hardware tx buffers are present in the kind of CAN controller used for communicating.

Apart from allocating space for these buffers within reCANdrv, it is necessary to adopt a strategy that allows managing these new buffers easily. For that purpose we propose to map each one to both CAN controllers. Specifically, we will establish a one-to-one correspondence between each software tx buffer and one hardware tx buffer at each one of the CAN controllers.

This mapping has two advantages. First, when the transmission of a frame is requested from reCANdrv, we can avoid the overhead of deciding in which hardware tx buffer of the tx controller to allocate the frame. Second, after instructing the frame transmission through the tx controller, reCANdrv can preventively also write a copy of the frame in the corresponding hardware tx buffer of the non-tx controller. This reduces the time required to instruct the transmission through the non-tx controller when the tx controller fails and the non-tx controller takes over.

5. Other implementation aspects

The overhead introduced by reCANdrv is an issue deserving attention. As explained in [2], for reCANdrv to properly operate, it must handle CAN-related events, e.g. transmission time outs, fast enough. Moreover, the amount of time reCANdrv needs to manage those events is not deterministic, as it depends on the order with which they happen. This would increase the jitter with which FTTdrv will transmit/receive frames, which is particularly harmful to some FTT frames, such as the TM, as it represents the temporal mark all nodes synchronize to.

Section 4 included some comments on how to reduce the overhead of the new reCANdrv functionalities (and thus of the jitter, indirectly). However, further implementation aspects are also important for achieving this objecive. The first one regards the way in which FTTdrv and reCANdrv exchange with each other both, the frames to be transmitted and the frames that are received. In order to minimize the volume of data exchanged between both layers, we will preallocate a memory pool shared by both drivers, so that they can exchange information by means of appropriate memory pointers. The second implementation issue aims at reducing the function-call overhead that results from invoking the primitives of reCANdrv from FTTdrv, e.g. the primitives for requesting a frame transmission or a frame abort. It consist in compiling the reCANdrv primitives as inline functions when they are called from reCANdrv.

6. Conclusions and future work

In the context of highly-dependable control systems, significant efforts are being made to provide field-bus communication protocols that combine advanced real-time features with strong fault-tolerance capabilities. Moreover, the flexibility of real-time and fault-tolerance mechanisms is also gaining in importance in adaptive systems.

The FTT paradigm is one of the most promising solutions to provide flexible real-time communication. In the context of CAN, some solutions have been proposed to enhance the fault-tolerance capabilities of the so-called FTT-CAN protocol. However, while all these solutions rely on the use of bus-guardians and replicated buses, other competing field-bus technologies, such as Ethernet, are evolving towards the use of stars given the stronger potential dependability benefits of this topology. Thus, in this paper we investigate the integration of FTT-CAN with ReCANcentrate, a CAN-compliant replicated star topology we developed to provide media fault-tolerance.

We show that a seamless integration of both technologies is possible, while using already available hardware/software pieces of existing prototypes. For that we analyzed the interactions between the set of communication services and fault-tolerance mechanisms of both technologies and, then, discovered that these sets are orthogonal to each other, so that they can still work properly after the integration. In particular, we showed that it is possible to keep the separation in layers of the communication services provided by the existing software drivers implemented for FTT-CAN and ReCANcentrate, so that the only requirement prior to the integration of both drivers would be to add certain communication services to the API of the ReCANcentrate driver. Moreover, we also explained how these functional additions can be carried out, thereby further clarifying the feasibility of the integration.

In future work we will also investigate other possible approaches to integrate FTT-CAN and ReCANcentrate that may yield real-time and fault-tolerance advantages not present in either of them. In particular, the hubs can be provided with mechanisms based on the scheduling information of the FTT layer in order to increase the fault-treatment capabilities of the hub itself. In this way the hubs could not only improve the efficiency with which certain faults are treated in FTT-CAN, but also address faults that are beyond the current capabilities of both technologies, e.g. masquerading faults.

Acknowledgements

This work was supported by the Spanish Science and Innovation Ministry with grant DPI2008-02195, FEDER funding, and the Portuguese Fundação para Ciência e a Tecnologia with grant SFRH/BPD/70317/2010.

References

[1] ISO11898-1. Controller Area Network (CAN) - part 1: Data link layer and physical signalling., 2003.

[2] M. Barranco, D. Gessner, J. Proenza, and L. Almeida. First prototype and experimental assessment of media management in ReCANcentrate. In *15th IEEE Conf. on Emerging Technologies and Factory Automation*, Sep. 2010.

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

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

[5] V. Silva, J. Fonseca, and J. Ferreira. Adapting the FTT-CAN master for multiple-bus operation. In *Proc. of the 5th IEEE International Conf. on Industrial Informatics*, pages 305 – 310. Industrial Electronics Society, June 2007.

[6] 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 *Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation*, 2005.