FlexCAN: A Flexible Architecture for Highly Dependable Embedded Applications

Juan R. Pimentel
Kettering University – U.S.A.
jpimente@kettering.edu

José Alberto Fonseca
DET / IEETA – Univ. de Aveiro - Portugal
jaf@det.ua.pt

Abstract

A new architecture (FlexCAN) and associated protocol (SafeCAN) are sumarized. The new architecture and constituent protocol are based on the CAN protocol and supports embedded safety-critical applications such as those in vehicles (trucks, cars, boats, snowmobiles, etc). The paper concentrates on the paradigms used by the architecture and the various constituent components.

1. Introduction

As communication networks for embedded and industrial applications become common there is an increasing need to provide dependable services [1],[2],[3],[8]. As used in the FlexCAN architecture, dependability involves the concepts of reliability, availability, safety, and security as detailed extensively by the research community [4]. The field of application of FlexCAN is control networks, i.e., those which are part of a control system that could operate in an open loop or closed loop fashion. Furthermore, FlexCAN deals with dependable issues in an explicit fashion unlike other somewhat dependable networks such as CanOpen, J1393, and other automotive networks based on the native CAN protocol.

Not all applications require a high level of dependable services. One can distinguish three main types of applications with varying degree of dependability requirements: low, medium, and high. For example, automotive applications involving steering and stability control have a high level of dependability requirements.

The following is a list (not exhaustive) of dependability requirements for embedded and industrial applications grouped in the categories of availability and reliability, and safety and security.

A. Availability and Reliability (AR) Requirements

1. Configuration management: a) Starting the network; b) identification of equipment; c) replacing/adding devices; d) downloading/loading communication configuration; e) downloading/uploading application process.

2. System integrity: a) Control architecture; b) monitoring and control of equipment; c) reading, setting, and synchronization of system clock and application time.

3. System characteristics: a) operational start/stop functions; b) reset of function/device; c) input data acquisition (sensors); d) data processing; e) output data (actuators).

4. Performance management: a) data characteristics; b) distribution and synchronization of time on the network; c) prediction of execution times; d) maximum jitters in processing and communication.

5. Working conditions: a) environmental parameters; b) exposure time.

B. Safety and Security (SS) Requirements

1. System safety structure: a) safety of input sensors; b) of the control system; c) of the output actuators; d) of the communication subsystem.

2. System safety integrity: a) safety architecture; b) monitoring of data, functions, or components; c) detection of faults; d) fault tolerance; e) error handling; f) response time limit; g) reading or setting of parameters related to redundancy.

3. System safety characteristics: a) safety start (restart) function; b) safety stop; c) emergency stop function; d) reset of safety function or device; e) human factors.

4. Security management: a) security in the presence of multiple system managers; b) password security for applications and technical personnel.

5. Parameter check: a) system (e.g., mechanical) overload; b) limit values (e.g., high and low pressure).

The CAN protocol was developed specifically for control applications and is being successfully used for applications with low levels of dependability requirements. It is generally agreed that the native CAN protocol does not meet all the needs of applications with medium and high dependability requirements. Some approaches for designing a highly dependable protocol use other paradigms such as the time-triggered medium access technique known as TDMA (time division multiple access) for example TTP/C [5] and FlexRay [6]. Still another approach used is to add TDMA features on top of the CAN protocol such as TTCAN and FTTCAN [10] protocols. FlexCAN, on the other hand, builds on the native CAN protocol by adding a special protocol termed SafeCAN to meet more stringent levels of dependability. Currently, the FlexCAN architecture only supports reliable, available, and safe features. Security features will be added later. Thus, FlexCAN inherits some important dependable attributes of CAN that include:

1. Deterministic, real-time bus access mechanism. Given a number of messages to be sent over a CAN bus together with real-time requirements, a message schedule meeting the deterministic and real-time requirements can be found if the messages satisfy some traffic conditions.

2. Atomic broadcast. Nodes receive the same set of messages in the same order that were sent on the bus. However some authors claim that there is a small probability of inconsistent message transmissions [9].

3. Error handling. The CAN protocol detects the following errors: bit error, stuff error, CRC error, form error, and acknowledgement error.

4. Error signaling. A node that detects an error condition signals the error by transmitting the following flags: active error flag and passive error flag.

5. Fault confinement. A node in error may be in one of the following three states: error-active, error-passive, or bus-off depending upon the state and history of two counters: TEC and REC (transmit/receive error counter).

FlexCAN adds the following dependable attributes [7]:

- Redundancy management. Detects node errors and manages a set of replicated nodes.
- Replicated CAN channel management.
- Membership management on a FTU and application basis.
- Sequence numbers for application messages.
- Timed-triggered application messages.
- Application oriented synchronization (Sensor, controller, actuator, processes synchronized in time).
- Application management. Keeps track of various applications (e.g., control loops).
- Producer-initiated timed synchronization.
- Error, fault, and failure management.
- Enable node fail silent behaviour.
- Enable CAN channel fail operational behaviour.
- Enable application fail safe behaviour.

C. Fieldbus network architectures. Networks for embedded applications belong to a more general class known as fieldbuses. In general fieldbuses either use two (1, 2) or three (1, 2, 7) OSI layers because the functionality of the remaining is not generally needed. Sometimes other layers (e.g., transport) may be required or the corresponding functions are embedded in the application layer. Without loss of generality, in this paper we assume that we are dealing with two or three layer fieldbus architectures, the layers being identified as layer 1, 2, and 7 (see Fig. 1).

The goal of this paper is to provide an overview of the FlexCAN architecture including its constituent components and its paradigms, and in addition to provide a summary of the SafeCAN protocol. Unfortunately, due to space limitations we cannot provide all the details of FlexCAN and SafeCAN.

2. FlexCAN Paradigms

Designing a dependable system architecture meeting all the requirements listed in section ***1 A*** and ***B*** is not trivial. From the viewpoint of offering technical solutions to this problem, these requirements can be grouped into three main groups: reliability and availability, safety, and security. Reliability and availability requirements are met by adopting the well known paradigm of component replication. In this respect, FlexCAN offers node replication and CAN channel replication. In addition, a mechanism is needed to manage the various replicated components. This is accomplished by the SafeCAN protocol. An important question regards the positioning of SafeCAN in the protocol stack. There are two possibilities: inside the OSI layers or outside (i.e., part of the actual application). We decided to position SafeCAN outside the OSI layers in a safety layer as depicted in Fig. 1.

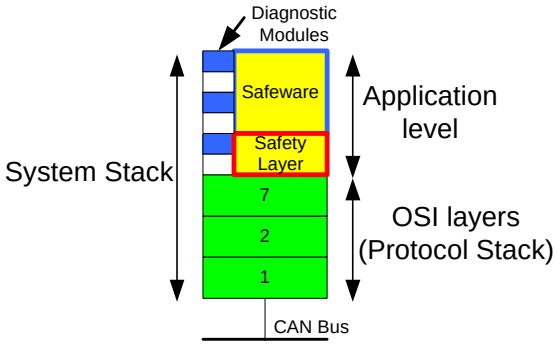

Fig. 1 Safety layer and safeware components in the context of the OSI stack

Trying to develop a protocol to handle safety requirements is not easy because the set of safe states is highly application dependent. Thus, attempting to design a safety protocol may not be fruitful. The idea exploited by FlexCAN is to provide a set of relatively simple mechanisms that enable end users to write safe applications. In addition, these should be carefully defined, designed, tested, verified, validated, and possibly certified. Thus we need to deal with safe software components that we term *safeware* and reside on top of the safety layer (See Fig. 1).

FlexCAN deals with control systems featuring three types of devices: sensors, controllers, and actuators. Accordingly, we distinguish three types of safeware: sensor safeware, controller safeware, and actuator safeware. Testing a dependable system meeting the requirements of section ***1 A*** and ***B*** requires a high degree of diagnostic and management mechanisms and these are provided also at the application level as depicted in Fig. 1.

Messages in a highly dependable system need to be not only deterministic and real-time but also synchronized. In TDMA systems that have a synchronized system wide clock, message synchronization can be done easily using this clock. Since there is no global clock in FlexCAN, message synchronization is done on an application basis. FlexCAN deals with control loops involving sensors, controllers, actuators, and plants so any source of data (e.g. a sensor) can be selected to generate time-triggered messages. All remaining nodes in the same application are expected to synchronize with the messages sent by the data source. Thus time-triggered synchronization messages are source and application oriented.

End users deal with simplified models because the details of the protocol are hidden from them by SafeCAN. Thus the end user interface is simple. More specifically, SafeCAN hides details regarding hardware reconfiguration, replicated component management, fault and error detection, and fault confinement.

3. SafeCAN protocol

As noted in section 2, the primary mechanism used to achieve fault tolerance and enhance reliability, availability, and safety is component replication. Two main types of components are replicated in FlexCAN: communication channels and comm. nodes. Fig. 2 shows one communication node with two replicas (i.e., a total of three nodes) configured in a so-called FTU (fault tolerant unit). FTUs provide a means to contain faults to regions called FCR (fault containment regions). Although not shown in Fig. 2, each node is connected to three CAN channels thus resulting into nine network interfaces. Thus the main problem with replicated components is the complexity that results from including one or more replicas. The complexity of a system with replicated components can become unmanageable if proper design steps are not used.

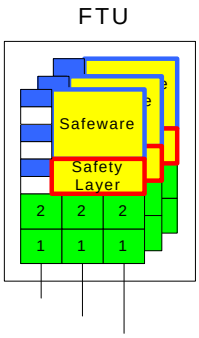

Fig. 2 FTU (fault tolerant unit) components.

To simplify the complexity of the resulting system FlexCAN includes a special protocol termed SafeCAN that is responsible for error detection and node management in an FTU. SafeCAN manages a number of node components and makes them appear as a single node to the remaining of the safety-critical application. In addition, it supports other mechanisms summarized in Table 1 in a manner similar to ProfiSafe.

A. Protocol summary.

Although any safeware node (i.e., sensor, controller, actuator) can be replicated, the following summary assumes that only controllers are replicated. SafeCAN assumes a set of application oriented time-triggered cycles termed application cycles of length Ts. The cycle is initiated by a data producer and termed PM (producer message). Certain consumers (e.g., controllers) receive the message, perform some computations, and in turn may generate additional CAN messages for still other consumers (e.g., actuators). Consumers may also generate feedback messages called CM. The set of replicated components are termed primary (P), secondary (S), tertiary (T), quaternary (Q) and so on. It is assumed that only one controller message goes on each CAN channel and this message (C-P) is sent by the primary controller.

SafeCAN is responsible for selecting just one primary from a set of replicated components identified by a hardware address termed serial number (SN). All remaining components are designated as secondary, tertiary, etc. according to a ranking mechanism. SafeCAN relies heavily on timers. If the primary controller does not send its message C-P after a delay t_S then the secondary controller S assumes that the primary has failed and will switch to primary and output the C-P message as shown in Fig. 3. If the secondary has also failed, the tertiary component will output the C-P message after a delay t_T. This procedure is repeated for additional replicated components. Fig. 3 also depicts the relative timing of the C-P and CM messages. SafeCAN only deals with hardware reconfiguration and not software. The parameters Ts, t_S, t_T, t_Q, and so on must be chosen appropriately for the protocol to work and several strategies are possible for choosing them. One possibility is to minimize the control delays of a number of control loops. These issues are currently under investigation.

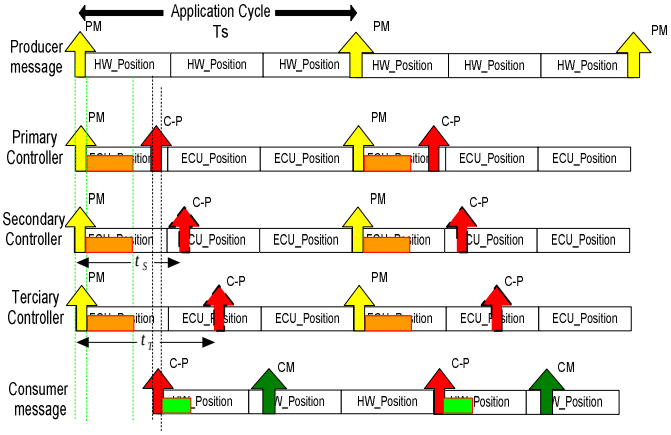

Figure 3. Timed sychronization of SafeCAN messages.

Table 1. Failures and protocol mechanisms to deal with failures.

Mechanism: Failure:	Sequence Number	Timed Acknowl.	Data Protection	Authori-zation
Repetition	X			
Loss	X	X		
Insertion	X	X		X
Incorrect Sequence	X			
Corrupted Data			X	
Delay		X		

The protocol deals with a number of failures including burst errors affecting one or more replicated component (node or bus) and as long as the errors do not affect all of the components at the same time. Extensive testing and verification has been performed on the protocol against MISRA-C rules and the DO-178B standard. In addition, a passive network manager has been designed and implemented. The details will be published elsewhere.

SafeCAN has the following messages: *Active Message*, *Rank Message*, *Serial Message*, and *Identify Message*. Additional details of the protocol can be found in [7].

4. Architectural Components

In this section we describe FlexCAN constituent components. Some networks may support applications that are safety-critical and other that are not. Thus FlexCAN components also include traditional CAN nodes not safety-critical. The main FlexCAN components include:

- Traditional (i.e., not safety-critical) two layer and three layer CAN stacks.
- Two layer and three layer SafeCAN stacks with one CAN channel and no replicas.
- Two layer and three layer SafeCAN stacks with N CAN channels and no replicas.
- Two layer and three layer SafeCAN stacks with one CAN channel and multiple replicas.
- Two layer and three layer SafeCAN stacks with N CAN channels and multiple replicas.
- Safeware (sensor, controller, actuator).
- Two layer and three layer diagnostic modules.
- Two and three layer network management modules.
- N-port gateways.

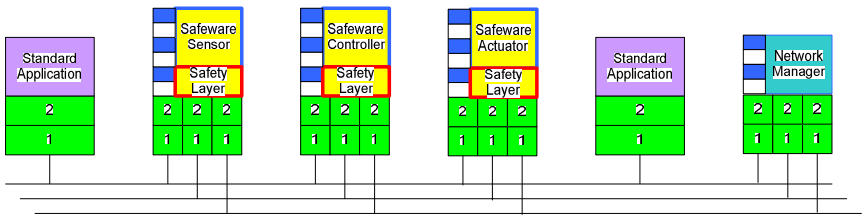

Figure 4. A sample configuration involving non-replicated components.

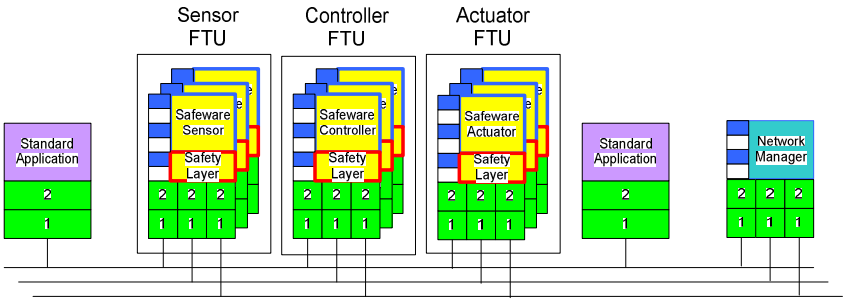

Figure 5. A sample configuration involving replicated components.

These constituent architectural components are illustrated in Figs. 4 and 5 that correspond to the same application. The former includes a six-node network featuring three redundant CAN channels, two conventional nodes on the top CAN channel, three nodes (Sensor, Controller, Actuator) corresponding to a safety-critical control loop, and a network manager to monitor the safety-critical application. In the latter configuration the sensor, controller, and actuator nodes are triplicated into corresponding FTUs. In the network of Fig. 5 there is a total of twelve nodes but, thanks to the SafeCAN protocol, the application only deals with a total of six. The architecture has been fully exercised and tested on a steer-by-wire application at Kettering University.

A. Architectural Features

The FlexCAN architecture is simple, flexible, modular, and scaleable. It shows deterministic and real-time behaviour. It offers a high level of dependable (reliability, availability, and safety) services. The implementation is based on COTS CAN components. Systems are expandable to 16 controllers and any number of buses (current implementation). Nodes may be added or removed without disrupting applications. Messages are not missed unless an entire FTU fails. Finally, replicated nodes are fail-silent.

5. Summary and conclusions

Some features and characteristics of FlexCAN and SafeCAN have been discussed. These make them attractive for immediate implementation and application of safety-critical systems because the technology is based on COTS components. The protocol mechanisms are simple, flexible, and yet generic to a wide variety of applications.

References

[1] Kopetz, H.(2003), Fault Containment and Error Detection in the Time-Triggered Architecture. In *Proc. IEEE Int. Symp. On Autonomous Descentralized Systems, ISADS 2003*, pp. 139-148.

[2] Muller, et al, (2002). Fault Tolerant TTCAN Networks. In *Proc. 8th Int. CAN Conference*, Las Vegas.

[3] Ferreira J., Pedreiras P., Almeida L., Fonseca J. (2002). Achieving Fault Tolerance in FTT-CAN. In *Proc. 2002 Int. Workshop on Factory Comm. Systems (WFCS2002)*, pp. 125-132, Vasteras, Sweden.

[4] Kopetz H. (1997), *Real-Time Systems, Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers.

[5] Kopetz H. and Grunsteidl G. (1994), TTP – A protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer*, pages 14-23.

[6] FlexRay Consortium, http://www.flexray-group.com

[7] Pimentel J. and Kaniarz, J. (2004), A CAN-Based Application Level Error Detection and Fault Containment Protocol, *INCOM'2004*, April 2004, Salvador, Brazil.

[8] PROFIBUS Nutzer Organization: PROFIBUS Profil, PROFISafe – Profile for Safety Technologies, Version 1.11, July 2001.

[9] Rufino J., et al (1998), Fault-Tolerant Broadcast in CAN. Digest of papers, *28th Int. Symp. Fault Tolerant Computer Systems*, pages 150-159.

[10] Almeida L., Pedreiras P., Fonseca J. (2002), The FTT-CAN Protocol: Why and How. *IEEE Transactions on Industrial Electronics*, 49(6), 2002.