FlexCAN: A Flexible Architecture for Highly Dependable Embedded Applications

Juan R. Pimentel
Kettering University
Flint, Michigan USA
and
Jose A. Fonseca
University of Aveiro
Aveiro, Portugal

Preprint Version: *Submitted to the 3rd Int. Workshop on Real-Time Networks to be held in Catania, Italy, July 2004)*

Abstract
A new architecture (FlexCAN) and associated protocol (SafeCAN) is 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 more common there is an increasing need to provide more dependable services services (Kopetz 2003, Muller et al 2002, and Ferreira et al 2002). As used in the FlexCAN architecture, dependability involves the concepts of *reliability*, *availability*, *safety*, and *security* as detailed extensively by the research community (Kopetz, 1997). The field of application of FlexCAN involves control networks rather than generic communication networks such as those based on TCP/IP. By a control network we mean that the network is 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 and one can distinguish three main types of applications with varying degree of dependability requirements: low, medium, and high. One example of an automotive application with a low level of dependability requirement is the so called body network involving the control of windows, doors, seats, etc. On the other hand, automotive applications involving steering and stability control systems 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 on the network
c) Replacing/adding equipment
d) Downloading/uploading communication configuration
e) Downloading/uploading application process

2. System integrity

a) Control architecture

b) Monitoring and control of equipment
c) Reading, seting, 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) safety of the control system
c) safety of the output actuators
d) Safety of the communication network

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 (and restart) function
b) safety stop function
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 of 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). Examples of TDMA highly dependable protocols include TTP/C (Kopetz and Grunsteidl, 1994) and FlexRay (FlexRay Consortium). Still another approach that has been used is to add TDMA features on top of the CAN protocol such as the TTCAN and FTTCAN 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, dependable, and safe features. Security features will be added in subsequent work on the architecture. Thus, FlexCAN inherits some important dependable attributes of CAN that include:

1. *Deterministic, rea-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 as long as 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 there is not a total agreement on this issue as there are some authors that claim that there is a small probability of inconsistent message transmissions (Rufino et at, 1998)
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 (transmit error counter) and REC (receive error counter).

In addition, FlexCAN adds the following dependable attributes (Pimentel and Kaniarz, 2004):

- Redundancy management. Detects node errors and manage a set of replicated nodes.
- Replicated CAN channel management
- Membership management on an FTU and application basis
- Sequence numbers
- Timed messages
- Application oriented synchronization. (Sensor, controller, actuator, process 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 behavior
- Enable CAN channel fail operational behavior
- Enable application fail safe behavior

Examples of networks with high levels of dependability: ProfiSafe (Profibus Users Organization, 2001), TTP/C, and FlexRay. ProfiSafe is an industrial network with a high degree of safety.

C. Fieldbus network architectures. Networks for embedded applications belong to a more general class of networks known as fieldbus networks. It is well known that fieldbus networks either have two (layers 1 and 2) or three (layers 1, 2, and 7) OSI layers because the functionality of the remaining layers are not generally needed. In some cases, additional layers (e.g., transport) may be required or the correspondent functions must be embedded in the application layer. Without any loss of generality, in this paper we assume that we are dealing with two or three layer fieldbus architectures where the layers are identified as layer 1, 2, and 7 (see Fig. 1).

2. FlexCAN Paradigms

Designing a dependable system architecture that meets all of 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.

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 would enable end users to write safe applications. In addition, end safe applications 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.

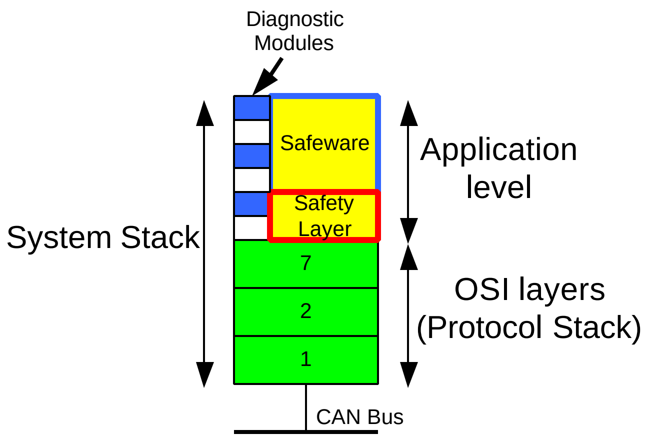

Fig. 1. Safety layer and safeware components in the context of the OSI stack.

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 the system wide 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 is hidden from end users 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

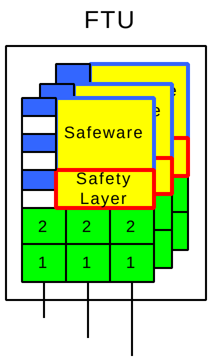

Fig. 2. FTU (fault tolerant unit) components.

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 communication 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). FTU's 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.

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 make 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.

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

Mechanism: **Failure:**	**Sequence Number**	**Timed Acknowledgement**	**Data Protection**	**Authorization**
Repetition	**X**			
Loss	**X**	**X**		
Insertion	**X**	**X**		**X**
Incorrect Sequence	**X**			
Corrupted Data			**X**	
Delay		**X**		

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 T_s. 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.

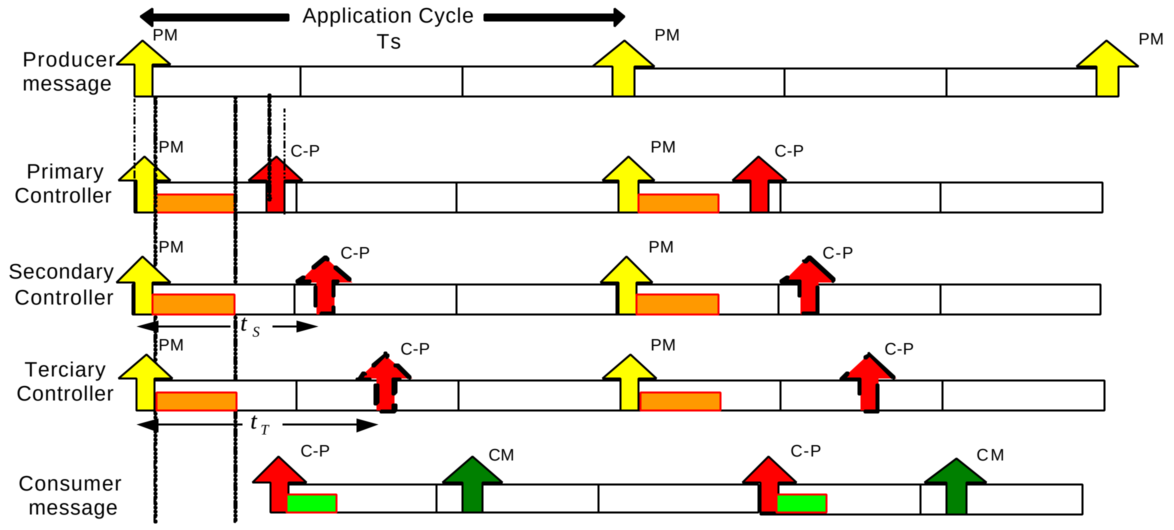

Fig. 3. Timed sychronization of SafeCAN messages.

The SafeCAN protocol format is shown in Fig. 4. Each message also contains a sequence number that is sent as the first data byte.

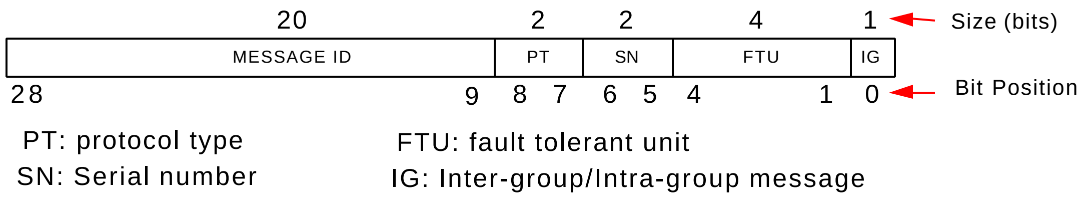

Fig. 4. SafeCAN protocol fields in a CAN2.0B format.

SafeCAN has the following messages:

Active Message. This message declares that the sender is the primary node.
Rank Message. This message requests that all nodes in a certain FTU (other than the primary node) start their rank procedure. If a node is already executing its ranking procedure this message is ignored.
Serial Message. The *Serial* message is used during the ranking procedure and is sent immediately after a Rank message is received. As the name implies, the Serial message contains the serial number of the sending node.
Identify Message. The identify message is used whenever there is a perceived inconsistency on what nodes are primary, secondary, tertiary, etc. For example, a primary node receiving a message with the same ID as that just sent (i.e., two C-P messages in the same cycle) in the current application cycle will conclude that there is another primary node and will issue the identify message to resolve the inconsistency.

Additional details of the protocol can be found in (Pimentel and Kaniarz, 2004).

B. Network Management. A network manager has been designed and implemented that is helpful for determining an accurate state of the distributed system and for some debugging. The network manager is mostly passive (i.e., it does not introduce load into the network) and only when required (e.g., for determining the status of a backup controller) it uses the *Identify Message*.

4. Architectural Components

In this section we describe the constituent components of FlexCAN. Some networks may support applications that are safety-critical and other that are not. Thus FlexCAN components also include traditional CAN nodes that are 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 layer and three layer network management modules
- N-port gateways

These constituent architectural components are depicted in Figs. 5 through 10 and described in the following.

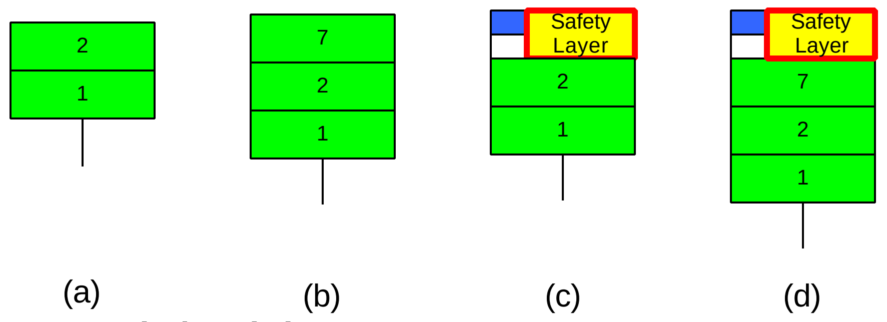

Fig. 5. Single channel FlexCAN components.

As noted, FlexCAN supports conventional two layer and three layers fieldbus communication architectures that are not safety-critical as depicted in Fig. 5 (a) and (b). To provide compatibility with applications that are not safety-critical, FlexCAN positions a safety layer outside the communication stack as depicted in Fig. 5 (c) and (d). The component on the left of the *Safety Layer* of Fig. 5 (c) include diagnostic mechanisms to help with error, fault, and failure management.

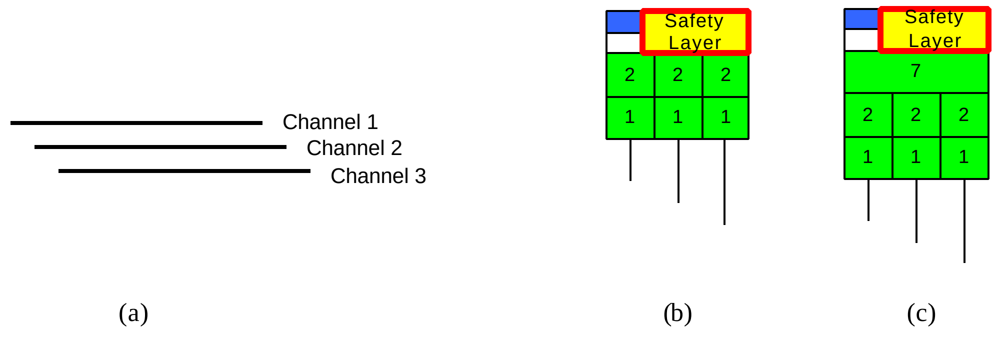

Fig. 6. Multiple channel FlexCAN components.

To enhance reliability and safety, FlexCAN supports a number of CAN channels as shown in Fig. 6 (a). This figure shows three channels but the architecture and protocol support any number of channels as available in the target microcontroller. For example the Motorola HCS12 microcontrollers supports five CAN channels. The corresponding two layer and three layer safety-critical stacks supporting three CAN channels are shown in Fig. 6 (b) and (c).

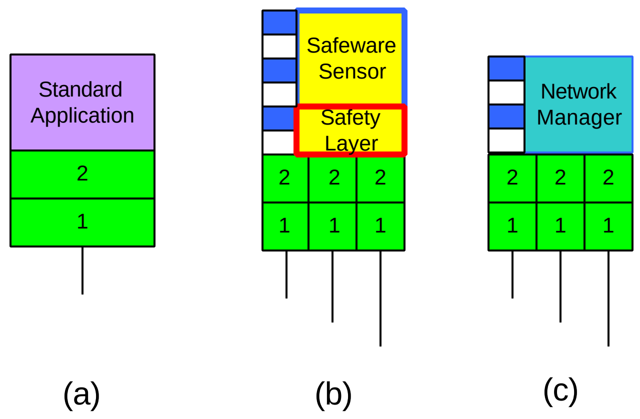

Fig. 7. Standard application, safeware, and network management components.

The safety layer simply contains the SafeCAN protocol and does not include any application code. All of the application software responsible for providing safety functions is contained in a *Safeware* (safe software) module as shown in Fig. 7 (b). Network management is an important application for any industrial network and Fig. 7 (c) shows a network management node suitable for a safety-critical application. Fig. 7 (a) shows a conventional application that may co-exist in a safety-critical network.

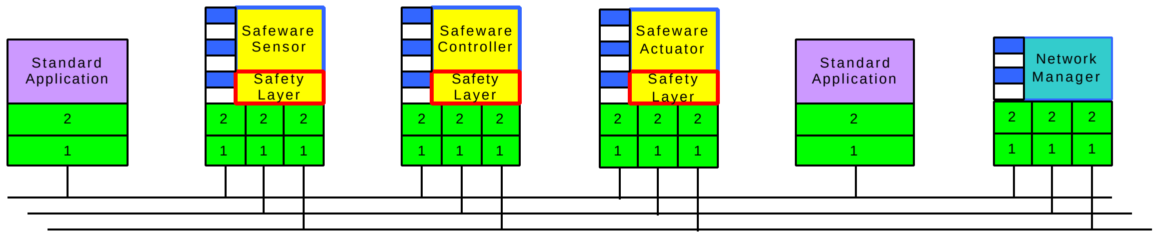

Fig. 8. A sample configuration involving non-replicated components.

The various constituent components described thus far can be combined to form a six-node network as depicted in Fig. 8. This network features three redundant CAN channels, two conventional nodes on CAN channel 1, three nodes (Sensor, Controller, Actuator) corresponding to a safety-critical control loop, and a network manager to monitor the safety-critical application.

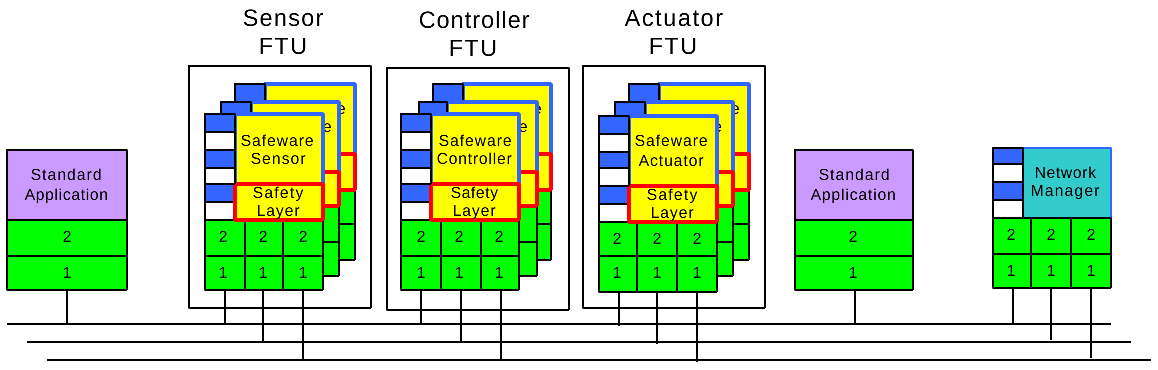

Fig. 9. A sample configuration involving replicated components.

The equivalent configuration of the six node network considered earlier is shown in Fig. 9 where the sensor, controller, and actuator nodes are triplicated into corresponding FTU's. In the network of Fig. 9 there is a total of twelve nodes but thanks to the SafeCAN protocol, the application only deals with a total of six nodes.

A final architectural element is that of a gateway useful to interconnect two or more networks as shown in Fig. 10 where two gateways are used. The gateways could reside on different microcontrollers or in just a single microcontrollers depending upon the availability of CAN channels on the microcontrollers and the gateway performance desired.

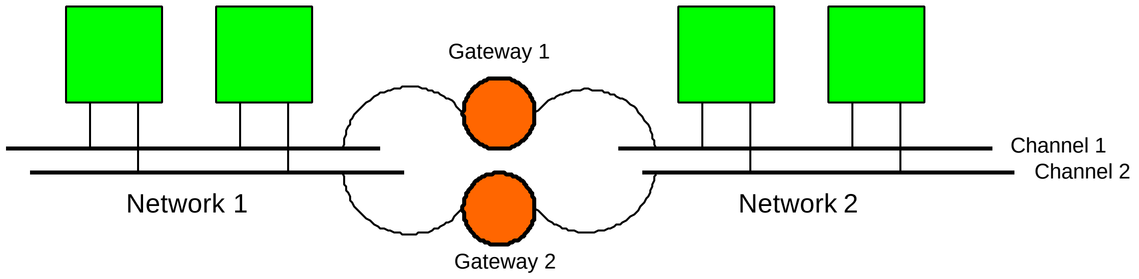

Fig. 10. Gateway component.

A. Architectural Features

1. Simple, flexible, modular, and scaleable
2. Deterministic and real-time behavior
3. High level of dependable (reliability, availability, and safety) services
4. Implementation based on COTS CAN components
5. Expandable to 16 controllers and any number of buses (current implementation)
6. Nodes may be added or removed without disrupting applications
7. Messages are not missed unless an entire FTU fails
8. Replicated nodes are fail-silent.

5. Summary and Conclusions

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

6. References

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.

Muller, B., Fuhrer, T., Hartwick, F., Hugel, R., and Weiler, H. (2002). Fault Tolerant TTCAN Networks. In *Proc. 8th Int. CAN Conference*, Las Vegas.

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

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

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

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

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

PROFIBUS Nutzer Organization: PROFIBUS Profil, PROFISafe – Profile for Safety Technologie, Version 1.11, July 2001.

Rufino, J, Verissimo, P., Arroz, G, Almeida, C., and Rodriguez, L., Fault Tolerant Broadcasts in CAN. In Digest of Papers, The 28th IEEE Int. Symp. On Fault Tolerant Computing Systems, Munich, Germany.