The design of the CANbids architecture

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

Luís Almeida
DEC-FEUP. Universidade do Porto. Portugal
lda@fe.up.pt

Abstract

Despite the significant advantages of the Controller Area Network (CAN) there is an extended belief that CAN is not suitable for critical applications, mainly because of several dependability limitations. During the CANbids project each one of these limitations has been addressed and a complete architecture for CAN-based fault-tolerant systems has been devised. This architecture allows building highly-reliable systems. This paper describes the design of such an architecture and the prototyping of its fundamental parts.

1. Introduction

The Controller Area Network (CAN) [5] protocol is a fieldbus communication protocol that was first devised for in-vehicle control applications and that has been widely adopted in many other areas within the distributed embedded control systems field. CAN is nowadays a mature technology whose tremendous success has been mainly caused by its error control features, low latency, network wide bus access priority and real-time response. In addition, CAN's widespread use has caused the price of its components to drop to some levels where other protocols cannot compete.

Despite these significant advantages, there is an extended belief that CAN is not suitable for critical applications, mainly because of the following dependability limitations [12]: (1) Limited data consistency; (2) Limited error containment; (3) Limited support for fault tolerance and (4) Lack of clock synchronization. Nevertheless, several researchers state that CAN will be able to support safety-critical applications if these limitations are overcome with the proper enhancements [12]. This possibility is very appealing for many application domains, since CAN components are much cheaper than those of the natural competitors of CAN in highly dependable systems, e.g., FlexRay or TTA, and because the use of CAN permits to take advantage of the know-how and expertise that engineering teams have gained in using this technology during the last decades.

For the last three years, the CANbids (*CAN-Based Infrastructure for Dependable Systems*) project has purported to design, implement and validate a CAN-based infrastructure for supporting the execution of highly-dependable distributed control applications. CANbids has used as building blocks (i.e. subsystems) various mechanisms and enhancements intended to overcome each and every one of the aforementioned dependability limitations of CAN.

Specifically, in order to eliminate the potential data inconsistencies, a device called CANsistant [18] that is capable of detecting certain inconsistencies has been designed. Other possible inconsistencies are eliminated with a new mechanism called Aggregated Error Flag Transmitter (AEFT) [20]. In order to increase the error containment capacity, a star topology called CANcentrate [3] has been proposed that has an active hub that disconnects from the rest of the network the nodes and links that fail in various manners according to a wide fault model. In order to provide the resulting system with suitable support for fault tolerance CANbids includes two functionalities. The first one allows replicating the star topology in such a way that the single point of failure that would imply having a single hub is eliminated. This is achieved with the replicated star topology called ReCANcentrate [3]. The second one supports the active replication of nodes. This means that several nodes execute the same functionality in such a way that after each partial computation they exchange the results obtained so far and each one votes on the results received from all nodes. The mechanisms included to support the replication of nodes are an adaptation of our work presented in [17]. The last limitation of CAN has been overcome by designing a clock synchronization subsystem that is able to tolerate its own faults [19].

It is important to mention that the replicas that nodes execute in order to achieve node fault tolerance can be ei-

 doi:10.1109/ETFA.2012.6489646

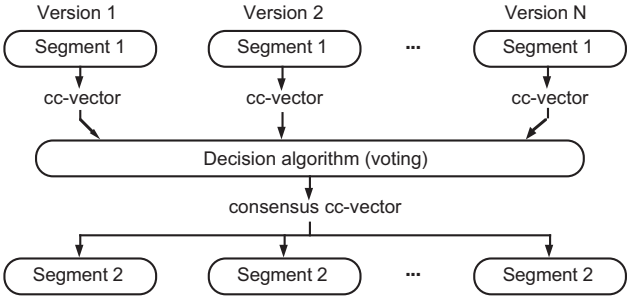

Figure 1. NVP execution

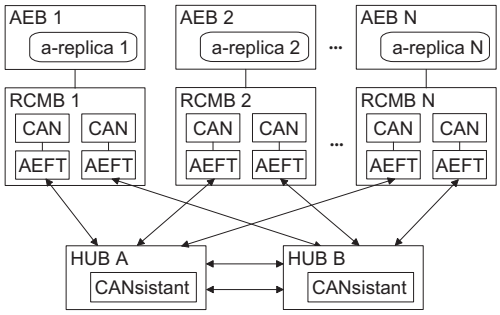

Figure 2. The CANbids circuitry

ther identical pieces of software or diverse versions of the same program. The first approach allows tolerating hardware faults whereas the second one software (design) faults as well. For diverse versions a suitable classical paradigm exists that is called *N-Version Programming* (NVP) paradigm [1]. In NVP, N diverse versions of the same program are developed by independent teams. Each version is partitioned into a set of *segments*. Corresponding segments in different versions are intended to perform the same function. In execution, each time a version finishes a segment, it issues a vector of results of this segment, called *cc-vector* (see Figure 1). Then a *decision algorithm* is executed on to obtain a *consensus cc-vector* which is sent back to all versions to be used in the continued computation. This mechanism, called *cc-point*, provides both synchronization among versions and masking of faults in a minority of versions. This voting mechanism can also be used when we have identical replicas instead of diverse versions. That is why we have adopted NVP for CANbids and we are going to refer always to the NVP mechanisms and terminology (segment, cc-point, cc-vector, etc.) no matter whether we are considering identical or diverse replicas of the application program. We will use the term *a-replica* to refer to a generic replica of the application program, that can present or not design diversity.

Figure 2 gives a first idea of the general appearance of CANbids. As can be seen, several nodes are connected by means of replicated stars and each one of the nodes executes an a-replica (either an identical copy or a diverse version of the same application). Each time one of the a-replicas finishes one of the segments, the corresponding cc-vector is immediately broadcast through the replicated star. Once the cc-vectors from each a-replica have been broadcast, each node executes the decision algorithm on them and returns the consensus cc-vector to the local a-replica. This voting is performed in CANbids in such a way that the determinism of the replicated nodes is enforced, i.e. all non-faulty nodes obtain the same values for their consensus cc-vectors. In this architecture the transmission of cc-vectors among nodes is triggered each time one of the a-replicas finishes a segment, which corresponds to an event-triggered communication scheme. This is the basis of the Event Synchronous System (ESYS) approach, proposed in [8] as the most suitable way of executing applications which follow the NVP paradigm, given the diverse execution times which are to be expected from diverse software versions. As indicated above, the approach is equally suitable when identical replicas of software are executed. Moreover, the ESYS approach perfectly fits in with the asynchronous character of many applications based on CAN.

In Section 2, a brief review of previous work is provided. Section 3 discusses the fault model we have considered. Section 4 describes the basic organization of CANbids. Section 5 describes the different subsystems that constitute the basic parts of our architecture. Section 6 summarizes the integration of the different subsystems into CANbids. Section 7 presents the prototype and the video demonstration created to prove the concept. Finally, Section 8 concludes the paper with a list of advantages of our approach.

2. Related work

Due to space constraints we are not going to elaborate on the different alternatives to each one of the specific mechanisms for dependability improvement that are integrated in the CANbids architecture. However it is important to say that each one of these mechanisms outperforms its previously published alternatives. Specifically, CANcentrate includes the CAN hub that presents by far the most complete error-containment capabilities among those described in the literature. On the other hand, no solution that could be compared to ReCANcentrate for star replication in CAN has been presented so far. Similarly, for the problem of node replication in the context of CAN there are no published alternatives, to the authors' best knowledge. The solution developed within the CANbids project for data inconsistencies covers much more error scenarios than other proposals. Finally, our clock synchronization subsystem is the one providing more independence from the application and more complete fault-tolerance capabilities. For specific details on the advantages that each one of CANbids subsystems exhibits in front its competitors in its specific tasks, the reader is referred to the

corresponding publications [17, 3, 18, 20, 19]

Looking at CANbids as a whole, it is possible to compare it to the two alternative complete infrastructures also proposed to improve the dependability of CAN-based systems: FlexCAN [12] and CANELy [21, 22]. A basic aspect to point out about FlexCAN it that it uses a TDMA scheme to organize the message exchanges. This does not correspond to the native event-triggered nature of CAN and thus it reduces the generality of the solutions that FlexCAN incorporates for CAN's dependability limitations. Moreover, the FlexCAN design does not explicitly address the data consistency issue and it does not present a replicated star topology, which implies loosing the superior error-containment capacities that are associated to this kind of topologies.

Regarding CANELy [21, 22], it is important to say than it should be seen not as the architecture of a complete distributed embedded system, but as a set of mechanisms based on CAN COTS components that system designers may consider when designing a fault-tolerant CAN system. In particular CANELy does not include instructions on how to achieve tolerance to node faults. Moreover, CANELy keeps the bus topology, with its inherent impairments related to error-containment. Furthermore, it considers only a subset of the inconsistency scenarios addressed by CANbids. Last, some of the services it proposes, such as its protocol for fault-tolerant clock synchronization, create a substantial overhead in terms of message exchanges that can make them unsuitable for specific applications.

3. Fault Model

The fault model considered in CANbids is actually a superset of the fault models taken into account for the design of all its subsystems [17, 3, 18, 20, 19]. For the sake of clarity it is worth to mention that both temporary and permanent faults are considered for nodes, connections, links and hubs. These faults can have an internal (e.g. short-circuits) or external (e.g. electromagnetic interference) origin and can be physical or human-made, although intentional faults are not explicitly considered. Also software faults can be addressed for the a-replicas if they present design diversity. Due to space limitations, the reader is asked to find the details of the considered faults in the cited papers.

4. Architecture overview

Providing an infrastructure that does not exhibit any of the referred dependability limitations of CAN requires the definition of an architecture that integrates the mechanisms developed to overcome such limitations. This is simplified by the orthogonal orientation followed in their design.

The general architectural approach is similar to that used in the design of our node replication scheme [17]. Here we focus on two particular architectural features, namely the definition of *error containment boundaries* and the organization of fault-tolerance operations.

4.1. Definition of error containment boundaries

In the proposed architecture, the error containment boundaries are enforced by restricting the failure semantics of the nodes and by preparing the nodes to deal with the errors that may still occur. Restricting the failure semantics of a node simplifies the operations that other nodes have to perform to deal with its potential errors.

We consider different levels of failure semantics for different functions the nodes have to perform. The main function of a node is the application task it is executing. Since cyclic votings on the results of this task are going to be carried out, it is all right if it exhibits *incorrect computation failure semantics* [2], i.e. be able to fail in the worst case by providing incorrect results either in the value or in the time domains. However we decided to prevent it from presenting more arbitrary failure modes, such as impersonating other nodes, sending different values to different nodes or babbling in the channel. The reason for this decision is that in order to deal with them the voting protocol would have to be much more complicated.

For all other node functions that send messages through the network, e.g., error detection functions that send heartbeats, we consider a *crash failure semantics* [13] since it makes no sense to perform a majority voting on the messages they transmit.

To enforce the above mentioned failure semantics, we divided each node hardware in two parts (see Figure 2), the *Application Execution Board* (AEB) that executes the application, and the *Redundancy and Communication Management Board* (RCMB) that handles communications and manages redundancy (e.g. performing the votings), being responsible for enforcing the failure semantics of nodes through *self-checking* and *policing* of the attached AEB [17].

The self-checking mechanism disconnects the respective RCMB upon error detection, thus providing a crash failure semantics. The policing mechanism verifies the messages sent by the attached AEB, thus preventing the broadcast of messages that may constitute either *babbling idiot failures* [6], two-faced behaviour or impersonations of other AEBs [13].

Asuming NVP is used as mechanism of node replication, each RCMB only allows a single cc-vector per segment to be sent by its attached AEB, thereby preventing babbling idiot behaviour. Likewise, the RCMB sends all the cc-vector messages from its attached AEB in a broadcast mode in order to eliminate any chance of an AEB sending different messages to different nodes. Finally, the RCMB actually indicates in the cc-vector messages which is the identity of the corresponding AEB-version, thereby eliminating the possibility of

a faulty AEB-version impersonating other nodes. This policing is what gives to the AEBs and the versions they execute an incorrect computation failure semantics. If node replication is achieved by means of identical versions of software, the same can be done with the messages conveying the intermediate results periodically sent for voting.

Moreover, beyond the RCMBs and AEBs local mechanisms, enforcing the desired failure semantics still requires certain properties of the underlying communication protocol. Specifically, we need a protocol providing *Reliable Broadcast* [4] as well as the ability to prevent babbling idiot behaviour generated by the channel itself, e.g., continuous retransmission caused by a permanent fault.

Concerning the specific case of the CAN protocol [5], its mechanisms to globalise errors supposedly provide *Reliable Broadcast*. More precisely, what CAN is claimed to provide is *data consistency*, which corresponds to Reliable Broadcast plus *Total Order* (TO) [4], i.e., if two correct nodes N_1 and N_2 receive two messages m_1 and m_2, then N_1 receives m_1 before m_2 if and only if N_2 receives m_1 before m_2. According to [4], this means that CAN is supposed to provide *Total Order Broadcast* (AB).

Despite not strictly necessary, TO is a very useful property that helps simplifying the consistent management of node replication (Section 5.1.1). As plentifully discussed over the years, CAN fails to provide AB under certain circumstances, but its enforcement can be achieved using specific techniques. Section 5.3 presents the technique used in CANbids.

CAN is also usually assumed to be able to prevent a babbling idiot behaviour generated at the channel level, given its native error counting and node isolation mechanisms. However, these are only effective when the primary errors affect only some nodes, and thus the errors are effectively eliminated by disconnecting the nodes in question from the network. For cases, in which there are direct electrical connections between multiple nodes without proper error containment, such as in buses, those mechanisms are not enough [3]. As a consequence, a fault in the components involved in such electrical connections may generate errors that propagate to all other nodes and prevent further communication, e.g., a transceiver that continuously sends a dominant value.

Star topologies with a central hub that disconnects the faulty ports are the usual solution for this problem. However, this solution introduces a single point of failure in the system, the hub, typically eliminated by redundancy. Our approach replicates the whole star, thereby tolerating link faults as well (Section 5.2). Moreover, the hub replicas are interconnected in a way that, in the absence of faults, mirrors every single bit in all interfaces. This significantly simplifies the redundancy management, that is performed by each node in the form of a specifically designed driver.

Beyond the mechanisms presented before for restricting the nodes failure semantics and contain certain errors, other errors may still cross the nodes interfaces, e.g., a wrong result in the value domain produced by an AEB and sent to the other nodes, or an omission created by the hub upon a node disconnection. Thus, in order to complete the definition of the error containment boundaries, we also need to provide the nodes with mechanisms to properly deal with these errors. This is the responsibility of the RCMB. For each received message, the RCMB has to determine if it originates from the RCMB or from the AEB of the transmitting node. In the former case, it is assumed correct since RCMBs exhibit crash failure semantics. The latter case is more complicated since both AEBs and the a-replicas executed on them exhibit incorrect computation failure semantics, i.e., upon failure they may generate incorrect outputs, both in the value and in the time domains. In CANbids we handle these failures with voting, taking into account that the messages may be incorrect also in the time domain. Finally, RCMBs must consider the potential omission of any message.

4.2 Organization of the fault-tolerance operations

In this section we focus on the fault tolerance operations performed at the upper layer of the CANbids architecture, which deals with the execution of the application software. Other lower-level fault tolerance operations will be discussed later, in the sections that describe how the different subsystems have been designed. The main operation performed in this upper layer to achieve fault tolerance is the voting on the cc-vectors issued by the a-replicas (identical or diverse versions) at the end of each segment. This voting is executed at each RCMB and provides *error compensation* [7] (i.e. fault masking).

To improve the global dependability, the RCMBs perform the following three additional fault-tolerance operations: *error detection* of individual nodes; *fault passivation* [7]; and *recovery* of components that have suffered transient faults.

Each RCMB performs error detection of individual nodes by comparing the consensus cc-vector calculated by the voting with the cc-vectors sent by each node. Additional error detection is provided by checking the reception of specific messages. As the omission of a cc-vector may be diagnosed as caused by either a faulty a-replica or a faulty RCMB, RCMBs must send an “I am alive” message at the beginning of the cc-vector exchange. If this message is received, the fault is assigned to the a-replica, otherwise it is assigned to the RCMB.

Fault passivation is performed by disconnecting the components that are affected by faults. To prevent a quick attrition of redundancy, disconnection should not be permanent for components which are affected by transient faults. Therefore, each RCMB maintains an error counter for itself and its a-replica which are increased each time a new error

is detected, and decreased when no error is detected. Only when an error counter reaches a pre-established threshold, the RCMB considers the corresponding component permanently faulty and disconnects it from the rest of the system. At any instant, the values of all these counters are considered to represent the *status* of the system.

Although not strictly necessary, in CANbids each RCMB keeps (consistently with the other RCMBs) the value of all counters (those of all nodes) for two reasons: for being able to send the values of the necessary error counters to an RCMB that is in the process of reintegration after having suffered a transient fault (Section 5.1.1); and for all RCMBs to have a consistent view of the available redundancy in the system, which is a very convenient feature in a distributed architecture.

Concerning recovery after transient faults, for a-replicas we rely on NVP that already has appropriate mechanisms, such as the cc-points. NVP even provides a mechanism that can recover a-replicas exhibiting more severe errors, e.g. execution of the wrong segment. This is called *recovery point* [1] and it is basically a cc-point in which the complete state of the computation is exchanged and voted. This allows the recovery of versions whose internal state (i.e. memory) has been corrupted.

In contrast, faults in RCMBs always manifest themselves as omissions caused by their self-disconnection from the network upon error detection by the self-checking mechanisms (Section 5.1). In this case, a recovery mechanism prevents transient RCMB faults to be converted to permanent omissions (crashes), which would lead to the quick attrition of redundancy. This recovery mechanism, which we call *reintegration*, also resynchronizes the affected RCMB with the non-faulty ones, leading it to a consistent state [15].

5. Subsystem design

The next step in the description of the CANbids architecture is to explain how the different subsystems that have been mentioned are designed to be able to make their contribution to the operation of the whole system. Due to space limitations, only the basic ideas of these designs will be presented. In most cases, suitable references are available for more detailed descriptions.

5.1. Node design

The CANbids architecture is based on the assumption that nodes exhibit the restricted failure semantics indicated above. Therefore, the effectiveness of the entire design, and thus the level of reliability which is finally reached, dramatically depend on the coverage of our failure semantics assumptions, and thus on the RCMB design. Since in order to disconnect itself from the channel (crash) the RCMB needs first to detect an error, this *assumption coverage* [14] strongly depends, in our architecture, on the error detection coverage of the RCMBs. The higher the latter is, the higher the former will be.

In order to reach a high level of reliability we have chosen duplication with comparison as a technique for error detection in the RCMBs. This technique is considered very effective, and we have applied it extensively in a previous design of the RCMBs [17] that is not fully compatible with the rest of the CANbids subsystems (e.g. with ReCANcentrate). Since the application of this technique is a rather straightforward although time-consuming engineering task, we do not consider its use as a requirement in order to prove the concept of CANbids. Therefore, the current prototype of CANbids, which we will introduce in Section 7, uses regular non-duplicated circuits for the implementation of the RCMBs.

5.1.1 RCMB software

The (duplicated) processor of the RCMB must execute software for a number of functions. First the driver *ReCANdrv* that manages the star replication; second the *core-TOTCAN* protocol (based on TOTCAN by Rufino et al. [23]) that reestablishes the TO property with the assistance of the CANsistant device placed at the hubs; and third some routines, called *RCM routines*, to consistently manage the node redundancy. These three layers of software can be seen in Figure 3). We discuss next the third of these layers and the other two will be described in later sections.

Two issues related to the consistent management of the redundancy constitute the main focus of the RCM routines: *replica determinism enforcement* [13] of all replicated operations and *consistent reintegration* of RCMBs after transient faults. Replica determinism enforcement ensures, for instance, that all non-faulty replicas of the voting procedure executed by the different RCMBs produce the same consensus cc-vector. Reintegration allows an RCMB that has been disconnected in order to prevent error propagation caused by transient faults to again be integrated in the system. The purpose of reintegration is to make sure that the redundancy of the system does not permanently attrite (degrade) when RCMBs are being disconnected due to transient faults. It should be noted that mechanisms for reintegration of a-replicas and AEBs that have suffered a transient fault are provided by NVP itself. Indeed given that in the cc-points described above all versions receive the resulting consensus cc-vector, not only fault masking is achieved, but also versions which have issued a wrong cc-vector —e.g. because of a transient fault in the corresponding computer— have an opportunity to recover using the consensus cc-vector values to resume computation. In contrast, the reintegration of RCMBs affected by transient faults is a new problem that is solved by our architecture. Essentially, the RCMB that has disconnected itself after detecting an internal error with

its duplicated and compared structure asks the other RCMBs for the necessary reintegration information, and the others respond making sure that the recovered node does not miss any relevant information even during its reintegration. Since all RCMBs consistently keep the status information, any of them can send it to the RCMB in reintegration.

For the detailed description of our approaches to consistency and reintegration it is necessary to refer to [17, 15]. However the most relevant aspect of them is that consistency among replicas is achieved by ensuring that all of them consider the same information both for calculating the consensus cc-vector and for calculating the update of the status. Since all the relevant information is obtained from the CAN channel, what is necessary is to choose the same set of messages for those calculations. The specific mechanism for consistently delimiting the set of messages uses the TO property and that is why it is necessary to reestablish that property after inconsistencies (see Section 5.3).

5.2. Replicated Star: ReCANcentrate

CANbids uses the replicated star topology called ReCANcentrate [3] in order to increase error containment and to provide tolerance to communication media faults. The general appearance of this topology can be seen in Figure 2, which shows two hubs and each one of the nodes connected to both hubs through dedicated links. Each one of the ReCANcentrate hubs is able to individually observe the contribution of each node to the value in the channel and detect which node (or link) is faulty. The hub keeps an error counter for each one of its ports and disconnects the corresponding port when its counter reaches a threshold. In this sense, the hub performs the same operations (error detection and fault passivation) for the physical layer of the network as the RCMBs performed for a-replicas and other RCMBs.

On the other hand, the replication of hubs and links provides tolerance to hub and media faults. In order to manage this replication, each node executes a driver called *ReCANdrv* (mentioned in Section 5.1.1) that has to hide the star replication for the upper layers of the RCMB software, which will see a single logical channel. In order to simplify the design of this driver, the two hubs are connected to each other to be able to couple the values of both stars into a single value, as it would happen in a regular CAN bus. Thereby, in the absence of faults, the driver expects to receive exactly the same values through both links. Each hub takes care of this coupling by disconnecting its links with the other hub in case it decides that the latter is faulty.

5.3. Consistent communication enforcement

As indicated in Section 4.1, the RCM routines need that the communication protocol provides AB (including TO). However, there are several circumstances in which CAN fails to provide AB. Specifically, when a CAN controller gets into the *error passive* state, it can no longer signal its errors in a way that forces all the other nodes to see them. Second, there are some scenarios in which channel errors in the last bits of the frame can create some inconsistent message exchanges [23, 16]. And third, some new scenarios have been recently identified in which channel errors in other bits of the frame can create the same inconsistencies [20]. It has to be noted that the actual probability of the error scenarios leading to inconsistencies in CAN is still an open issue. Nevertheless, CANbids includes mechanisms to solve all these problems. If some of them are deemed not so relevant, the corresponding solutions can be removed.

Specifically, all those mechanisms are included in CANbids as an holistic integrated solution we call TOBE-CAN (*Total Order Broadcast Enforcement in CAN*). First, the ReCANdrv in each RCMB would simply discard one CAN controller (and use the other) when the controller generates an error warning interrupt, which indicates that the controller's error counters reached a threshold prior to its change to the error passive state.

Second, all possible last-bits scenarios are detected by a specifically designed hardware module, called CANsistant [18]. CANsistant could be connected to the system as a regular node but it is placed in one of the hubs in order to have a privileged view of the network. Due to its fundamental role, CANsistant must be replicated and thus one replica can be placed in each hub. In order to easily manage this replication, each CANsistant must be internally duplicated and compared to have crash failure semantics. However, detecting all possible last-bits scenarios is not enough and in order to reestablish TO, we have designed a higher-layer protocol called *core-TOTCAN* (C-TOTCAN), which is executed by each RCMB on top of the ReCANdrv. C-TOTCAN is based on the TOTCAN protocol proposed by Rufino et al. [23], from which it basically keeps the strategy each recipient node uses for ordering the frames. In TOTCAN is is assumed that in the event of an inconsistency, the sender will retransmit the affected frame. Then, only when a frame is exchanged without possible inconsistency, an ACCEPT message is eagerly broadcast by all nodes, so that each one of them eventually receives it and, hence, knows that the last received replica of the initial frame is the one to be delivered. In TOBE-CAN, CANsistant is responsible for retransmitting the potentially inconsistent frames, since the sender is not able to detect all last-bit scenarios. Moreover, the ACCEPT frame is not eagerly broadcast by all nodes, but retransmitted by CANsistant, only when this frame suffers from an inconsistency, what is more efficient. The details of TOBE-CAN and its integration within CANbids will be the subject of a future paper.

Finally, in order to resolve the new inconsistency scenarios described in [20] a circuit called *Aggregated Error Flag*

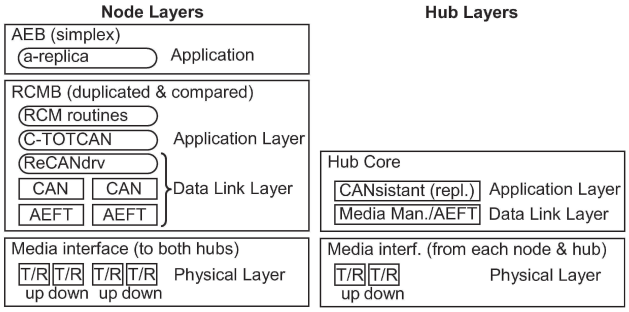

Figure 3. Layers in the CANbids architecture

Transmitter (AEFT) has been added to each CAN node and CANsistant replica. The role of the AEFT is to adequately prolong the duration of the error flags, so as to ensure that all nodes and CANsistant replicas can globalize the errors they detect locally, thereby enforcing that the affected frame is consistently rejected by all [20]."

5.4. Clock synchronization

All CANbids' fault tolerance mechanisms have been designed to assume almost no synchronization among nodes. Therefore, a clock synchronization service is not a requirement of CANbids. If the application requires it, then OCS-CAN[19] could be added to the system. The current design of OCS-CAN is devised for the CAN's bus topology but it can be used for a replicated star topology as well.

6. Systemwide integration

In complex fault-tolerant systems, it is fundamental that the various mechanisms are integrated to each other in such a way that they do not interfere. CANbids has been devised with this principle in mind, trying to design each mechanism as orthogonal as possible from the rest of the system. More specifically, each mechanism either acts in a different physical location solving its specific problems or it is integrated in a multi-layer structure and provides services to upper layers at the same time that it receives services from the lower ones. Figure 3 shows the layer structure of CANbids, somehow summarizing the descriptions that have been provided in previous sections, and allowing to appreciate how the different subsystems interact with each other. Note that rounded rectangles represent software modules, whereas regular ones represent hardware modules. In the cases in which the module carries out communication tasks, the corresponding OSI layer is indicated on the right.

7. CANbids implementation

For each of the different mechanisms discussed above, an individual prototype has been already described [17, 3, 18, 20, 19]. In this section we present the prototype we have built to integrate the two most important parts of CANbids: the node replication and ReCANcentrate.

The prototype is comprised of three nodes and two hubs. Each hub consists of an XSA-3S1000 prototyping board from XESS [24], which contains a Xilinx Spartan 3 XC3S1000 FPGA. Each board was connected by means of a flat IDE cable to a custom circuit board that provides an external oscillator for the FPGA and which in addition functions as the hub's I/O module. This circuit board provides a total of six RJ-45 connectors, three of which allow the connection of up to three nodes, two of which are used as interlinks, and one that can optionally be used as an interface to a personal computer (PC). The hubs' logic was implemented using VHDL.

For each of the nodes we used a dsPICDEM prototyping board from Microchip [11], which contains a dsPIC30F6014A microcontroller [10], four LEDs, and a serial port. All nodes were connected by means of the serial port to a single PC. This allowed us to easily initialize the nodes through the serial port and to retrieve logging information that could then be displayed on the PC's screen. Each dsPICDEM board was also connected by means of a flat cable to a dedicated custom circuit board that allows the corresponding node to be connected to each of the hubs using UTP Cat5 Ethernet cable. The integrated voting mechanism and ReCANcentrate driver, as well as the user application executing on the nodes, were written in the C programming language and compiled with the MPLAB C30 compiler from Microchip [9]. The code was compiled using the maximum optimization level and function inlining.

The voting mechanism was tested by initializing the nodes such that they generate disagreeing cc-vectors. The port isolation mechanisms were tested by physically disrupting the connections between nodes and hubs. We also tested both mechanisms simultaneously by having a node generate disagreeing cc-vectors while at the same time disrupting some of the connections.

A video demonstration of this prototype can be watched at http://www.youtube.com/watch?v=7fd9eW4Tr50. Please note that in the video the operation of the system has been deliberately and significantly slowed down to allow observing the proper operation step by step.

8. Conclusions and future work

The complete architecture of CANbids has been presented. In this infrastructure all the widely accepted dependability problems of CAN are addressed without changing the event-triggered character of the protocol. Moreover, the performance of the used solutions surpasses those of their alternatives. As a result it is possible to execute applications on CANbids in such a way that permanent faults in

the communication media, the nodes and even the application software can be tolerated. Furthermore, the resulting structure presents a series of additional advantages. First the voting process does not represent a single point of failure since it is consistently replicated in each node. Second, the mechanisms for fault tolerance are concentrated in the added RCMBs and hubs, being thereby orthogonal to the application level. This means that it is possible to change the application and even the AEBs, and CANbids will still be able to provide the specified fault tolerance services. Third, hardware *Commercial Off The Shelf* (COTS) components are used as main processors and as building blocks for the design of the RCMBs. Fourth, the architecture is scalable in the sense that it allows increasing the number of replicated nodes (and a-replicas) in order to tolerate an arbitrary number of faulty nodes. And fifth, the restriction of the failure semantics of the nodes makes unnecessary to have a number of nodes that is bigger than $2t + 1$ (where t is the number of nodes whose failure is to be tolerated), unlike what happens when nodes can exhibit byzantine failures [15].

Future work will include the quantification of the reliability increase achievable with CANbids.

9. Acknowledgement

This work was supported by the Spanish Science and Innovation Ministry with grant DPI2008-02195, by the Spanish Economy and Competitiveness Ministry with grant DPI2011-22992, and by FEDER funding.

References

[1] A. Avižienis. The N-Version aproach to fault-tolerant software. *IEEE Transactions on Software Engineering*, 11(12):1491–1501, December 1985.

[2] M. Barborak, M. Malek, and A. Dahbura. The consensus problem in fault-tolerant computing. *ACM Computing Surveys*, 25(2):171–220, June 1993.

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

[4] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. J. Mullender, editor, *Distributed Systems*, ACM-Press, chapter 5, pages 97–145. Addison-Wesley, second edition, 1993.

[5] ISO. *International Standard 11898 – Road Vehicles – Interchange of Digital Information – Controller Area Network (CAN) for High-Speed Communication*. 1993.

[6] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Real-Time Systems. Engineering and Computer Science. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[7] J.-C. Laprie, editor. *Dependability: Basic Concepts and Terminology*. Springer-Verlag Wien New York, 1992.

[8] S. V. Makam. *Design Study of a Fault-Tolerant Computer System to Execute N-Version Software*. PhD thesis, Computer Science Department. University of California, Los Angeles (UCLA), December 1982.

[9] Microchip Technology Inc. MPLAB C30 C Compiler User's Guide, 2005.

[10] Microchip Technology Inc. dsPIC30F6011A/ 6012A/ 6013A/ 6014A Data Sheet, 2006.

[11] Microchip Technology Inc. dsPICDEM 80-Pin Starter Development Board User's Guide, 2006.

[12] J. Pimentel, J. Proenza, L. Almeida, G. Rodríguez-Navas, M. Barranco, and J. Ferreira. Dependable automotive CAN networks. In N. Navet and F. Simonot-Lion, editors, *Automotive Embedded Systems Handbook*. CRC Press.

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

[14] D. Powell. Failure mode assumptions and assumption coverage. In *Digest of Papers of the IEEE 22th Int. Symp. Fault-Tolerant Computing FTCS-22*, pages 386–395, Boston, Massachusetts-USA, July 1992.

[15] J. Proenza. *RCMBnet: A Distributed Hardware and Firmware Support for Software Fault Tolerance*. PhD thesis, DMI. University of the Balearic Islands, 2007.

[16] J. Proenza and J. Miro-Julia. MajorCAN: A modification to the Controller Area Network protocol to achieve Atomic Broadcast. In *IEEE Int. Workshop on Group Communications and Computations. IWGCC. Taipei, Taiwan*, 2000.

[17] J. Proenza, J. Miro-Julia, and H. Hansson. Managing redundancy in CAN-based networks supporting N-Version Programming. *Computer Standards and Interfaces. Elsevier*, 31(1):120–127, January 2009.

[18] 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 *Proceedings of the 14th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2009), Palma de Mallorca, Spain*, September 2009.

[19] G. Rodríguez-Navas, S. Roca, and J. Proenza. Orthogonal, fault-tolerant and high-precision clock synchronization for the controller area network. *IEEE Transactions on Industrial Informatics*, 4(2):92–101, May 2008.

[20] G. Rodríguez-Navas, C. Winter, and J. Proenza. Injection of aggregated error flags as a means to guarantee consistent error detection in CAN. In *Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2011), Toulouse, France*, September 2011.

[21] J. Rufino, P. Veríssimo, and G. Arroz. Node failure detection and membership in CANELy. In *IEEE Int. Conf. on Dependable Systems and Networks. DSN 2003.*, June 2003.

[22] J. Rufino, P. Veríssimo, G. Arroz, and C. Almeida. Control of inaccessibility in CANELy. In *Proceedings of the 6th IEEE International Workshop on Factory Communication Systems (WFCS 2006). Torino, Italy*, June 2006.

[23] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcast in CAN. In *Proceedings of the IEEE 28th Int. Symp. Fault-Tolerant Computing. FTCS-28. Munich (Germany)*, June 1998.

[24] XESS Corporation. XSA-3S1000 Board V1.1 User Manual, 2007.