ENFORCING DEPENDABILITY AND TIMELINESS IN CANELy
Application to Spaceborne Data Communication Systems[*]

José Rufino, Paulo Verissimo
Universidade de Lisboa, Faculdade de Ciências, LaSIGE, Campo Grande, 1749-016 Lisboa, Portugal

Ricardo Pinto, Carlos Almeida, Guilherme Arroz
Universidade Técnica de Lisboa, Instituto Superior Técnico, Avenida Rovisco Pais, 1049-001 Lisboa, Portugal

Keywords: Dependability and real-time, Controller area network, Spacecraft data communication.

Abstract: The Controller Area Network (CAN) has played along the last decade a crucial role in the design and implementation of distributed embedded systems. However, the native CAN protocol exhibits a set of availability, reliability and timeliness limitations. Given the large practical base of off-the-shelf microcontrollers integrating standard CAN interfaces and the emergence of CAN protocol open cores, a fundamental question is whether (and how) those components can be used for highly dependable applications of CAN?
This paper identifies a fundamental set of shortcomings of the native CAN protocol and discusses how existing CAN controllers can be combined with additional hardware/software components to secure the provisioning of strict dependability and timeliness guarantees. Furthermore, the paper discusses the main issues in the design and implementation of CANELy, a CAN-based infrastructure able of extremely reliable hard real-time communication, and shows how CANELy components can be integrated in the onboard data communication and processing infrastructure currently being designed for future space vehicle avionics.

1 INTRODUCTION

The Controller Area Network (CAN) has played along the last decade a crucial role in the design and implementation of distributed embedded systems in areas as diverse as industrial automation, automotive, train transportation, medical, oil drilling, aeronautics and space. Standard CAN-based profiles have been defined for a diversified set of specific devices and application domains. Recently, the domains of aeronautics (AEEC, 2010) and space (ECSS, 2005) have been approached.

In the course of our current research aiming at building a time- and space-partitioned architecture for the next generation of space vehicle avionics we are tackling the difficult problem of integrating input/output (I/O) functions, such as sensors, actuators and networks while maintaining overall system responsiveness (Rufino et al., 2010). The architectural principle of time- and space-partitioning (TSP) enables the safe integration of applications with different degrees of criticality in a single computing platform. Applications are segregated into logical containers, the *partitions*, for the benefit of fault containment and to ease verification, validation and certification. Each partition uses a predefined dedicated memory addressing space; access to a given I/O device is granted to the *system partition* hosting the corresponding agent for I/O operations. Partitions and therefore I/O operations are scheduled under a predetermined, cyclically repeated, sequence of time windows.

Given the reasonable body of research in CAN dependable communications and the on-going standardisation activities in the space domain, we are approaching the use of the CAN data bus with the purpose of integrating responsive remote terminal units (RTU) and simple sensors/actuators in a TSP system aboard a spacecraft (Figure 1).

CAN is traditionally viewed as a robust data bus. However, the native CAN protocol exhibits a

[*]This work was partially developed within the scope of the European Space Agency Innovation Triangle Initiative program, through ESTEC Contract 21217/07/NL/CB, Project AIR-II (ARINC 653 in Space RTOS – Industrial Initiative, http://air.di.fc.ul.pt). This work was partially supported by Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology), through the Multiannual Funding and CMU-Portugal Programs.

Rufino J., Verissimo P., Pinto R., Almeida C. and Arroz G..
ENFORCING DEPENDABILITY AND TIMELINESS IN CANELy - Application to Spaceborne Data Communication Systems.
DOI: 10.5220/0003376004560463
In *Proceedings of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems* (PECCS-2011), pages 456-463
ISBN: 978-989-8425-48-5

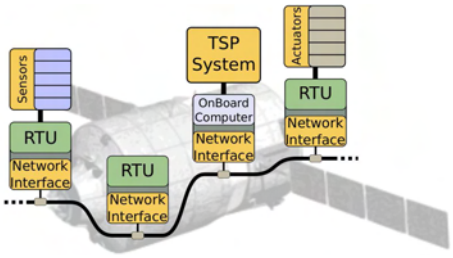

Figure 1: Utilization of CAN in a spacecraft TSP system.

set of severe limitations with regard to provisioning of strict availability, reliability and timeliness guarantees, which are a must for spaceborne applications. Given the large practical base of off-the-shelf standard CAN interfaces, a fundamental question is whether (and how) these components can be used for highly dependable applications of the CAN data bus?

In fact, what is missing in the standard CAN data bus to attain high levels of dependability is a set of fault tolerance and timeliness-related services. These can be provided off-the-shelf (i.e. without modifications to the CAN standard or to existing CAN controllers), through the use of properly encapsulated additional hardware/software components. The materialization of this concept is called **CAN Enhanced Layer** (CANELy), which is made from several hardware and software building blocks (Rufino, 2002).

This paper discusses the main issues in the design and implementation of CANELy and how such functionality can be effectively integrated with a TSP architecture. The paper is organized as follows: Section 2 provides a short description of CAN and analyses its dependability; Section 3 discusses the system model; Section 4 analyses how to improve the availability of the network infrastructure; Section 5 discusses how to secure CAN timely behaviour in the presence of faults; Section 6 addresses the integration of a semantically rich CANELy service interface in a TSP architecture and the separation of implementation issues between hardware and software components. Finally, Section 7 concludes the paper.

2 CAN DATA BUS

CAN is a multi-master data bus that uses a twisted pair cable as transmission medium (CAN, 1993; CiA, 1994). The network maximum length depends on the data rate. Typical values are: 40m @ 1 Mbps; 1000m @ 50 kbps. Data bus signalling takes one out of two values: *recessive* (r), also the state of an idle bus; *dominant* (d), which always overwrites a recessive value. This behaviour, together with the use of unique frame identifiers, is exploited for bus arbitration. A *carrier sense multi-access with deterministic collision resolution* policy is used. When several nodes compete for bus access, the node transmitting the frame with the lowest identifier always goes through and gets the bus. Frames that have lost arbitration or have been destroyed by errors are automatically scheduled for retransmission. A *data frame* is a piece of encapsulated information, which may contain a *message*, a user-level piece of information. A *remote frame* has no field for message encapsulation.

In the signalling of abnormal network operation incidents, the CAN protocol uses: *error frames*, for (global) error signalling; *overload frames*, to react to violations of the standard interframe spacing, which has a nominal duration of three bit-times and is known in CAN terminology as *intermission* (CAN, 1993).

Although the standard CAN physical layer allows a few cabling faults (one wire open/short failures) to be tolerated (CAN, 1993; CAN, 1997), no standardized mechanism exists to provide resilience against network partitioning if both wires of the network cable get simultaneously interrupted. A solution to the problem of (physical) network partitioning has to be built as an extension to the standard specification (NOB, 1998; Rufino et al., 1999).

Furthermore, the occurrence of certain incidents in CAN operation (such as: bit errors; transmitter/receiver glitches) produces a subtle form of (virtual) network partitioning, called *inaccessibility*. Though the standard CAN protocol has means of recovering from these situations, the recovery process takes time, leading to increase the network access delay as seen by one or more nodes. This may induce a violation of the expected network timeliness properties and therefore provisions to tolerate such kind of faults are required (Verissimo et al., 1997; Rufino et al., 2006).

3 SYSTEM MODEL

The definition of a systemic model for CAN proved extremely useful, showing the weaknesses of CAN with regard to dependability and providing the grounds to handle those problems effectively. The fault assumptions for the system and a relevant set of CAN protocol properties are drawn from previous works on CAN (Rufino et al., 1998; Rufino et al., 1999; Rufino, 2002).

3.1 Fault Model

The CAN infrastructure is composed of $\mathcal{N}$ nodes interconnected by a Channel. The Channel is the physi-

cal path, i.e. the cable medium and transceivers, used by Medium Access Control (MAC) entities to communicate.

A component is **weak-fail-silent** if it behaves correctly or crashes if it exhibits more than a given number of omission failures – the component's *omission degree* – in a time interval of reference, T_{rd}. In CAN, an omission is an error that destroys a data or remote frame. The following failure semantics are defined for **CAN network components**:

- individual components are **weak-fail-silent** with *omission degree* f_o;
- failure bursts never affect more than f_o transmissions in a time interval of reference;
- omission failures may be inconsistent (i.e., not observed by all recipients);
- there is no permanent failure of the Channel (e.g. the simultaneous partitioning of all redundant media).

3.2 CAN Protocol Properties

For the sake of completeness, a discussion of a relevant set of CAN properties is summarized next. The foundation of CAN operation is described by the physical layer properties formalized in Figure 2.

Property PCAN1 formalizes the *quasi-stationary* propagation of signals in the CAN Channel (Stuart, 1999; Rufino et al., 1999). A *Bit* is the physical layer information unit and has a constant nominal duration. A single *Bit* is broadcast in the Channel at a time, as described by PCAN3. In absence of faults, a *Bit p* at *s* assumes one and only one logical value $v_B^s(p)$. The symbol $\prod$ is used in PCAN2 to specify a logical AND function combining the signals from multiple simultaneous transmitters into a single *Bit* value.

A key set of CAN MAC sub-layer properties is also enumerated in Figure 2. Property MCAN1 derives from CAN built-in error handling mechanisms, implying that frame errors are transformed into omissions. The residual probability of undetected frame errors is negligible (Charzinski, 1994). Property MCAN2 maps the system model failure semantics onto CAN operational assumptions, being $k \geq f_o$.

The behaviour of CAN in the time domain is described by property MCAN4. In absence of faults, T_{td} includes the normal queuing, access and frame transmission delays. It depends on message latency classes and offered load bounds (Davis et al., 2007; Zuberi and Shin, 1997; Livani et al., 1998). In general, T_{td} also needs to include the extra delays resulting from the additional queuing effects caused by the *periods of inaccessibility* (Pinho et al., 2000; Punnekkat et al., 2000). The maximum frame transmission delay includes a corrective term, T_{ina}, which accounts for the worst case duration of inaccessibility events (MCAN3). The inaccessibility characteristics of CAN are obtained by analysis of the CAN protocol (Verissimo et al., 1997; Rufino, 2002).

Physical-level properties

PCAN1 - *Bit* **Simultaneity**: for any *Bit p* of any transmitter *s* starting at $t_B^s(p)$, if $t_B^r(p)$ is the start of *Bit p* as seen by receiver *r*, for any *r*, then in absence of faults, $t_B^s(p) = t_B^r(p)$.

PCAN2 - Wired-AND Multiple Access: for all transmitters *s* in $\mathcal{N}$, the value of any *Bit p* seen by the channel *c* is, in absence of faults, $v_B^c(p) = \prod_{s \in \mathcal{N}} v_B^s(p)$.

PCAN3 - *Bit* **Broadcast**: in absence of faults, for any *Bit p* on the channel *c*, and for any receiver *r*, $v_B^r(p) = v_B^c(p)$.

MAC-level properties

MCAN1 - Error Detection: correct nodes detect any corruption done by the network in a locally received frame.

MCAN2 - Bounded Omission Degree: in a known time interval T_{rd}, omission failures may occur in at most *k* transmissions.

MCAN3 - Bounded Inaccessibility: in a known time interval T_{rd}, the network may be inaccessible at most *i* times, with a total duration of at most T_{ina}.

MCAN4 - Bounded Transmission Delay: any frame queued for transmission is transmitted on the network within a bounded delay of $T_{td} + T_{ina}$.

Figure 2: Relevant CAN protocol properties.

4 NETWORK AVAILABILITY

The first problem to be addressed concerns the availability of the CAN infrastructure. A commercial solution (NOB, 1998) uses a self-healing ring/bus but it does not solve the problem efficiently: ring reconfiguration may last as long as 100 *ms*, an extremely high inaccessibility figure (Rufino, 2002).

In CANELy, resilience to network physical partitioning is achieved through replication of the physical path (bus medium and transceivers) used by MAC entities to communicate. Replication of channel media assumes that: each cable replica is routed differently, being reasonable to consider failures in different media as independent; any bit issued from a MAC sublayer is simultaneously transmitted on all the redundant media interfaces.

Basic Media Redundancy Mechanisms

An innovative strategy to handle replicated media is based on a Columbus' egg idea and extends the wired-AND nature of CAN (property PCAN2, in Figure 2)

to the media interface level (Rufino et al., 1999): the signals from the different $\mathcal{M}$ redundant media receivers, $M_{Rx}(m)$, are combined in a conventional AND function, before interfacing the standard MAC sub-layer, Ch_{Rx}.

The specification of such strategy in VHDL[2] is drawn in Figure 3. This simple solution, feasible given property PCAN1, ensures resilience to medium physical partitions and stuck-at-recessive failures (Rufino et al., 1999).

```
-- MediaRX: Vector aggregating the several media.
-- ChRx   : Channel incoming (Rx) bit stream, is
---- logical '1' if all media are '1', else '0'.
---- In CAN, logical '1' <=> recessive (r)
----         logical '0' <=> dominant  (d)

ChRx <= '1' when MediaRX = (MediaRX'range => '1')
        else '0';
```

Figure 3: The AND-based media redundancy management strategy in VHDL.

Resilience to bus stuck-at-dominant failures is achieved exploiting the identity value of the logical AND function. The contribution of each medium interface for that function can be selectively disabled, through the assertion of the $M_{dis}(m)$ signal, as specified in Figure 4.

```
-- MediaRX : Vector aggregating the several media.
-- M_Rx(m) : Medium m Rx signal (transceiver).
-- M_dis(m): Medium m disable signal.

procMediumRXOR: process is
begin
-- Parallelizing selective actions on each medium
  for m in 1 to NumberMedia loop
    MediaRX(m) <= M_Rx(m) or M_dis(m);
  end loop; -- m
end process procMediumRXOR;
```

Figure 4: Media selection functions in VHDL.

The $M_{dis}(m)$ signal is asserted in conformity with the specified in Figure 5, upon the detection of a stuck-at-dominant failure, signalled through the assertion of $M_{stkd}(m)$, or after a Medium has exceeded its omission degree bound. The $M_{dis}(m)$ signal is locked in the assert state until the negation of the $M_{lock}(m)$ signal, by CANELy *media quarantine* entities or upon a request issued from high-level management entities. The function specified in Figure 5 contributes to ensure a safe operation, preventing a faulty Medium of being unseasonably enabled.

[2]Very High-Speed Integrated Circuits (VHSIC) Hardware Description Language.

```
-- M_dis(m) : Medium m disable signal.
-- M_stkd(m): Medium m is stuck-at-dominant.
-- M_Od(m)  : Medium m omission degree.
-- k_m      : Medium m omission degree bound.
-- M_lock(m): Medium m lock.

MediaDisable : for m in 1 to NumberMedia generate
------ Parallelizing disable actions on each medium
  -- Generate the disable signal for Medium m
  M_dis(m) <=
          M_stkd(m) or (M_Od(m)>k_m) or M_lock(m);
end generate MediaDisable;
```

Figure 5: Generation of the media disable functions in VHDL.

The AND-based Media Redundancy Management is a central component in the CANELy architecture of Figure 6. The remaining modules identified in the diagram of Figure 6 provide additional monitoring and fault treatment functions.

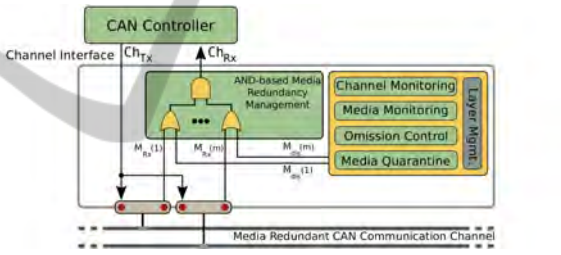

Figure 6: CANELy Media Redundancy Mechanisms.

Media and Channel Monitoring Functions

The set of Media and Channel monitoring functions identified in Figures 7 and 8 are needed to complement the bare functionality provided by the AND-based Media Redundancy Management strategy.

A combination of Media and Channel monitoring signals is used to provide the following functionality:

- disable operation of Medium m, if a stuck-at-dominant failure is detected, as reported through $M_{stkd}(m)$;
- perform receiver-based frame monitoring, comparing Channel and Medium incoming frame data on a bit-by-bit basis. This mechanism is fundamental to detect Medium omissions;
- detect and account for omissions at each Medium interface and evaluate the corresponding *Medium Omission degree*, $M_{Od}(m)$;
- disable operation of Medium m, if it exceeds the allowed omission degree bound, k_m, i.e. if $M_{Od}(m) > k_m$.

Basic Media Monitoring	
$M_{stkd}(m)$	**Stuck-at dominant medium**
	asserted if dominant for more than a given threshold;
	negated upon detection of a recessive bit.
$M_{Od}(m)$	**Medium omission degree**
	incremented upon detection of an omission failure;
	unchanged if common-mode or unkown-source errors;
	reset upon correct frame transfer on the Medium.
$M_{lock}(m)$	**Lock Medium m disable status**
	asserted upon disable of Medium m, by $M_{dis}(m)$ (Figure 5);
	negated by *media quarantine* or by high-level entities.
Extended Media Monitoring	
$M_{idle}(m)$	**Medium idle**
	asserted if recessive for more than a given threshold;
	negated upon assertion of Ch_{EOT} (see Figure 8).
$M_{ds}(m)$	**Medium dominant signaling**
	asserted upon detection of a dominant bit;
	negated upon assertion of Ch_{EOT} (see Figure 8).

Figure 7: Media monitoring functions in CANELy.

Basic Channel Monitoring	
Ch_{SOF}	**Start Of Frame**
	asserted at beginning of frame transmission;
	one bit-time duration.
Ch_{Fok}	**Frame Correct**
	data or remote frame received without errors;
	negated upon assertion of Ch_{EOT}.
Ch_{Err}	**Frame Error**
	asserted upon violation of CAN bit-stuffing rule;
	negated upon assertion of Ch_{EOT}.
Ch_{EOT}	**End Of Transmission**
	asserted after detection of minimum bus idle period;
	negated upon assertion of Ch_{SOF}.
Extended Channel Monitoring	
Ch_{stk-Tx}	**Stuck-at dominant Channel**
	asserted if dominant for more than a given threshold;
	negated upon management request.

Figure 8: Channel monitoring functions in CANELy.

These mechanisms provide effective resilience against *all* the cabling failures discussed in Section 2. They are not hard to implement in VHDL as FPGA[3]-based components.

The VHDL/FPGA machinery of a functionally effective CANELy unit should also integrate specific means for the preservation of dependability coverage, as follows:

- detection of medium partition and medium stuck-at-recessive failures and their signalling to high-level management entities;
- early detection of stuck-at-dominant Channel failures, allowing a prompt shutdown of the incorrect node;
- operation of a CAN-oriented *media quarantine* scheme, allowing an optimal $k=1$ Channel omission degree bound, if at least one channel media replica is unaffected by errors (permanent or transient).

[3]Field Programmable Gate Array.

Management Interface

The layer management entities identified in Figure 6 are elements of the CANELy machinery implemented as FPGA-based components. They provide an interface between the hardware infrastructure and the high-level network management protocol entities. Both invocation and notification primitives are included, as specified in Figure 9, given the parameters: (i) *baud*, the bus bit signalling rate; (ii) k_m, the media omission degree bound; (iii) *m*, the failed Medium; (iv) *mid*, the message identifier.

Invocation Primitives (canely-msu.req)	
Description	
Initialize ($baud$, k_m)	
Notification Primitives (canely-msu.nty)	
Description	*Issuing Condition*
Omission degree exceeded (m)	$M_{Od}(m) > k_m$
Stuck-at-dominant Medium (m)	$M_{stkd}(m)$
Stuck-at-recessive Medium (m, mid)	$M_{idle}(m) \wedge \neg M_{ds}(m)$
Medium partition (m, mid)	$M_{idle}(m) \wedge M_{ds}(m)$
Stuck-at-dominant Channel	Ch_{stk-Tx}

Figure 9: CANELy redundancy management primitives.

The parameters signalled upon stuck-at-recessive and Medium partition failures permits a high-level diagnose application to establish a node connectivity matrix, useful to pinpoint the location of the failure in the network cabling.

Although such mechanisms may be useless in unmanned spacecraft, they may be important for manned space flights or in permanent planetary bases where the crew may perform some repair actions.

5 INACCESSIBILITY CONTROL

Normal CAN operation can be hindered by *periods of inaccessibility*, which derive from incidents in network operation (e.g. bit errors) that temporarily prevent communication. Service is not provided to some or all of the nodes and this may have the effect of increasing the corresponding queueing and network access delays. Analysis of message transmission latencies performed under the assumption the network always operates normally (Davis et al., 2007; Zuberi and Shin, 1997; Livani et al., 1998) are relevant and, undoubtedly, useful for optimal system configuration. However, bounds are established that may be violated

upon the (even if rare) occurrence of inaccessibility events.

To avoid timing failures due to network inaccessibility incidents it is required to control inaccessibility (Rufino, 2002; Rufino et al., 2006).

5.1 CAN Inaccessibility Boundedness

The first step to control inaccessibility implies the study of network accessibility constraints, ensuring that the number of inaccessibility periods and their duration have a bound. The analysis in (Verissimo et al., 1997; Rufino, 2002) provides a comprehensive set of easy-to-use formulas to evaluate the worst-case bounds of the periods of inaccessibility.

The results of such analysis are summarized in Figure 10. It is worth noticing: the single bit errors (on the leftmost part of Figure 10) are not reduced because they affect only the transmission of one frame; the worst-case inaccessibility bound for bus multi-burst errors is reduced, due to the effects of CANELy *media quarantine* mechanisms; the extremely low worst-case figure of bus reconfiguration delay (209 μs @ 1Mbps), compared with other failure scenarios and, in particular, with the 100 *ms* of existing commercial systems (NOB, 1998).

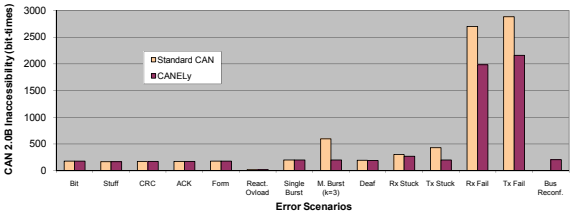

Figure 10: Normalized durations of inaccessibility periods.

On the other hand, the actions taken in (Rufino, 2002) to enforce the weak-fail-silent assumption for the network components: are based on CAN own error confinement mechanisms (Rufino et al., 1998); induced only a moderate, though interesting, reduction of inaccessibility durations for receiver and transmitter failure scenarios, as shown in Figure 10.

The avoidance of "babbling idiot" failures has further been studied: the inaccessibility constraint derived in (Broster and Burns, 2003) for CAN settings has a normalized duration of 41 bit-times, much lower than the values inscribed in Figure 10. Babbling idiot failures are not detectable by the native CAN error handling and fault confinement mechanisms. Protection has to be provided by special-purpose machinery (bus guardian) (Broster and Burns, 2003).

5.2 Message Schedulability Analysis

Next, it is required to show that inaccessibility bounds are suitably low for service requirements. This requires a comprehensive analysis of message schedulability guarantees given known traffic patterns and offered load bounds. Both error free and worst-case error analysis are relevant. The former, is intended to provide the parameters required for optimal system configuration (Davis et al., 2007; Zuberi and Shin, 1997; Livani et al., 1998). The latter, given a worst-case pattern of inaccessibility incidents, provides hard real-time guarantees of message schedulability and defines worst-case message delivery delays (Pinho et al., 2000; Punnekkat et al., 2000).

Extended versions of existing message schedulability analysis tools and methodologies (Pinho et al., 2000) should be able to provide relevant parameters for system configuration, including a bound for the time the effects of inaccessibility last in the system.

5.3 CAN Inaccessibility Control

To enable low-level control of inaccessibility, the Channel monitoring functions of Figure 8 should be extended with the additional functionality summarized in Figure 11. The Ch_{Ina} signal, is used for:

- the evaluation of the real durations of inaccessibility incidents, t_{ina}, and of the extra message queuing and network access delays, t_{p_ina};
- the evaluation of inaccessibility upper bounds with respect to the total number of incidents, i, and their total duration, T_{ina}, in a period of reference (property MCAN3);
- the evaluation of the worst-case duration of the entire period where the effects of inaccessibility last in the system, which we have defined as inaccessibility epoch, T_{p_ina}.

Extended Channel Monitoring	
Ch_{Bidle}	**Bus Idleness**
	asserted if bus is idle for more than the nominal *intermission*; negated upon detection of a dominant bit.
Ch_{Ina}	**Channel Inaccessibility status**
	asserted upon assertion of Ch_{Err} (Figure 8); negated upon assertion of Ch_{Bidle}.

Figure 11: Extension of Channel monitoring functions for the control of inaccessibility.

The layer management interface of Figure 9 is extended with the primitives described in Figure 12, which are used to manage the effects of inaccessibility in protocol execution, at all the relevant levels of the system (Rufino, 2002; Rufino et al., 2006).

Invocation Primitives (canely-icu.req)	
Description	*Bounds*
Get Channel status (Ch_{Ina})	
Get Channel inaccessibility events (Ch_i)	i
Get Channel inaccessibility times (t_{ina}, t_{p_ina})	T_{ina}, T_{p_ina}
Notification Primitives (canely-icu.nty)	
Description	
Channel status change (Ch_{Ina})	

Figure 12: CANELy inaccessibility control primitives.

At application-level, a corrective term accounting for the worst-case duration of an inaccessibility epoch is simply added to optimal timeout values. At low-level protocols, advanced inaccessibility control mechanisms allow: to account for the real duration of an inaccessibility epoch; to selectively add a corrective term to (optimal) timeout values, only when inaccessibility affects protocol timeliness.

6 TSP SYSTEM INTEGRATION

In TSP systems the exchange of data between partitions (either local or remote) is restricted to authorized interpartition communication channels. Two paradigms are used: *sampling ports*, holding only one fixed-size message; *queueing ports*, holding room for a given number of variable-size *atomic* and *totally ordered* messages. The size of each message has a predetermined maximum size. Such approach is in conformity with the ARINC-653 specification (AEEC, 2006).

The CANELy architecture and its companion real-time protocol suite, supporting group communication (Rufino et al., 1998), clock synchronization (Rodrigues et al., 1998), node failure detection and site membership (Rufino et al., 2003), constitutes an excellent candidate for supporting interpartition communication in distributed TSP systems.

The CANELy machinery supporting the execution of the real-time communication protocol suite, dubbed CANELy Dependability Engine, can be extended to support interpartition communication in distributed TSP systems, as depicted in Figure 13. Such extension is achieved through provision and management of the buffers necessary for implementing the sampling and queueing communication ports. The architecture proposed in Figure 13 decouples network operations from message data processing by the corresponding partition in the distributed TSP system.

6.1 Proof-of-Concept Prototype

In the prototype of the CANELy architecture currently being implemented (Figure 14), the CANELy processing infrastructure required for the execution of CANELy low-level protocols is materialized using the state-of-the-art Dallas/Maxim DS80C390 High-Speed Microprocessor (Dallas, 2005).

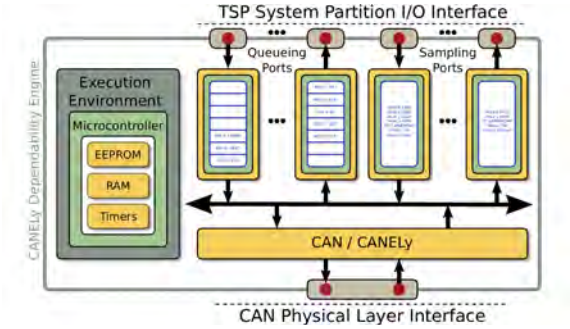

Figure 13: CANELy Dependability Engine extension with support for distributed TSP systems.

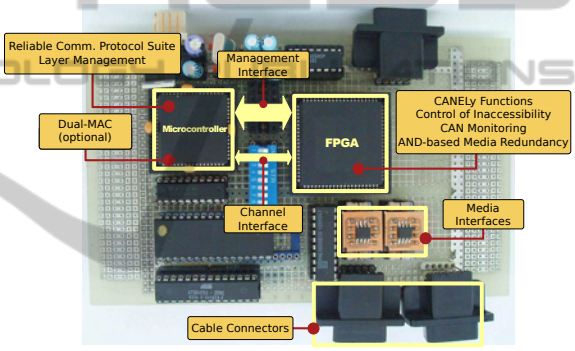

Figure 14: CANELy prototype board.

The support for the low-level special-purpose functions is implemented by a single, medium capacity, programmable logic device (Field Programmable Gate Array - FPGA) (Xilinx, 2009).

7 CONCLUDING REMARKS

Given the increasing demand for embedded distributed fault-tolerant systems based on low-cost network technologies, it has been: investigated the shortcomings of CAN, with regard to dependability and timeliness; defined a systemic model of CAN that not only did it show those weaknesses, but it provided the grounds to handle those problems effectively.

This paper discussed the implementation of the main components in the CANELy architecture, the CAN Enhanced Layer (Rufino, 2002), a combination of the CAN standard layer with some simple machinery resources and low-level protocols achieving

highly dependable real-time communications. The CANELy mechanisms enhance the dependability and timeliness of CAN-based systems and allow the assessment of real system parameters (w.r.t. timing, omission), thus making possible to monitor the coverage of both dependability and timeliness models.

In the context of spaceborne applications, the CANELy architecture can be used to support inter-partition communication in distributed TSP systems.

Finally, this paper identified the set of functions to be implemented as FPGA-based components and the functionality that has to be integrated at CANELy (software) protocols.

REFERENCES

AEEC (2006). Avionics application software standard interface. ARINC Specification 653, Airlines Electronic Engineering Committee (AEEC).

AEEC (2010). General standardization of CAN (Controller Area Network) for airborne use. ARINC Spec. 825-1, Airlines Electronic Engineering Committee (AEEC).

Broster, I. and Burns, A. (2003). An analysable bus-guardian for event-triggered communication. In *Proc. of 24th Real-time Systems Symposium*, pages 410–419, Cancun, Mexico. IEEE.

CAN (1993). *International Standard 11898 - Road vehicles - Interchange of digital information - Controller Area Network for high-speed communication.* ISO.

CAN (1997). *TJA1053 - Fault-tolerant CAN transceiver*. Philips Semiconductors.

Charzinski, J. (1994). Performance of the error detection mechanisms in CAN. In *Proc. of the 1st Int. CAN Conference*, pages 1.20–1.29, Mainz, Germany. CiA.

CiA (1994). *CAN Physical Layer for Industrial Applications - CiA Draft Standard 102 Version 2.0*. CiA - CAN in Automation.

Dallas (2005). *DS80C390 Dual-CAN High-Speed Microprocessor*. Maxim/Dallas Semiconductors.

Davis, R. I., Burns, A., Bril, R. J., and Lukkien, J. J. (2007). Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35:239–272.

ECSS (2005). *ECSS Draft Standard ECSS-E-ST-50-15C. Recommendations for CAN Bus in Spacecraft Onboard Applications*. European Cooperation for Space Standardization (ECSS).

Livani, M., Kaiser, J., and Jia, W. (1998). Scheduling hard and soft real-time communication in the controller area network (CAN). In *Proc. of the 23rd IFAC/IFIP Workshop on Real-Time Programming*, Shantou - China. IFAC/IFIP.

NOB (1998). RED-CAN a fully redundant CAN-system. NOB Elektronik AB Product Note - Sweden.

Pinho, L., Vasques, F., and Tovar, E. (2000). Integrating inaccessibility in response time analysis of CAN networks. In *Proc. of the 3rd Int. Workshop on Factory Communication Systems*, Porto, Portugal. IEEE.

Punnekkat, S., Hansson, H., and Norstrom, C. (2000). Response time analysis under errors for CAN. In *Proc. of the Real-Time Technology and Applications Symposium*, pages 258–265, Washington, USA. IEEE.

Rodrigues, L., Guimarães, M., and Rufino, J. (1998). Fault-tolerant clock syncronization in CAN. In *Proc. of 19th Real-Time Systems Symposium*, pages 420–429, Madrid, Spain. IEEE.

Rufino, J. (2002). *Computational System for Real-Time Distributed Control*. PhD thesis, Technical University of Lisbon - Instituto Superior Técnico, Lisboa, Portugal.

Rufino, J., Craveiro, J., and Verissimo, P. (2010). Building a time- and space-partitioned architecture for the next generation of space vehicle avionics. In *Proc. of the 8th IFIP Int. Workshop on Software Technologies for Embedded and Ubiquitous Systems*, pages 179–190. IFIP, Springer.

Rufino, J., Verissimo, P., and Arroz, G. (1999). A Columbus' egg idea for CAN media redundancy. In *Digest of Papers, The 29th Int. Symposium on Fault-Tolerant Computing Systems*, pages 286–293, Madison, Wisconsin - USA. IEEE.

Rufino, J., Verissimo, P., and Arroz, G. (2003). Node failure detection and membership in CANELy. In *Proc. of the 2003 International Conference on Dependable Systems and Networks*, pages 331–340, San Francisco, California, USA. IEEE.

Rufino, J., Verissimo, P., Arroz, G., and Almeida, C. (2006). Control of inaccessibility in CANELy. In *Proc. of the 6th. Int. Workshop on Factory Communication Systems*, pages 35–44, Torino, Italy. IEEE.

Rufino, J., Verissimo, P., Arroz, G., Almeida, C., and Rodrigues, L. (1998). Fault-tolerant broadcasts in CAN. In *Digest of Papers, The 28th Int. Symposium on Fault-Tolerant Computing Systems*, pages 150–159, Munich, Germany. IEEE.

Stuart, R. (1999). CAN bit timing requirements. Application Note AN1798, Motorola, Inc.

Verissimo, P., Rufino, J., and Ming, L. (1997). How hard is hard real-time communication on field-buses? In *Digest of Papers, The 27th Int. Symp. on Fault-Tolerant Computing Systems*, Washington - USA. IEEE.

Xilinx (2009). Spartan-3E FPGA family data sheet.

Zuberi, K. and Shin, K. (1997). Scheduling messages on Controller Area Network for real-time CIM applications. *IEEE Transactions on Robotics and Automation*, 13(2):310–314.