Proceedings of the 4th IFAC International Symposium on Intelligent Components and Instruments for Control Applications - SICICA 2000, Buenos Aires, Argentina.

REAL-TIME COMMUNICATION ON THE CAN-BUS FOR DISTRIBUTED APPLICATIONS WITH DECENTRALIZED CONTROL

Jörg Kaiser

University of Ulm
Dept. of Computer Structures
kaiser@informatik.uni-ulm.de
http://www.informatik.uni-ulm.de/rs/core/

Abstract: The paper discusses a transport protocol for the CAN-Bus (CAN: Controller Area Network) for distributed control systems. The protocol supports synchronous hard real-time traffic and best effort soft real-time traffic. Furthermore, the protocol guarantees the same order of messages in all nodes and allows synchronized delivery of messages in all nodes even in the presence of overload and transmission errors. The protocol is tailored for a CAN-Bus environment and needs very low overhead by exploiting the specific CAN features.

Keywords: Real-time communication, Fieldbus, Decentralized control, Resource allocation, Consistency.

1. INTRODUCTION

Smart sensors and actuators, powered by microcontrollers and connected via a communication network support in many ways extensibility, reliablity and cost effectiveness of control systems. The built-in computational component enables the implementation of a well defined higher level interface which does not just provide raw (sometiems analogue) transducer data, but a preprocessed, application related set of process variables. The communication network represents on a physical level a standardized interface over which the devices can exchange information. In essence, the device can be used as a building block, a black box which can be configured by process specific parameters and communicates process relevant data. Consequently, the interfaces and the functions of these smart components are not just related to the raw physical values of the controlled device but they may include functions related to overall control, supervision and maintenance issues. Perhaps the most challenging property of these intelligent devices is their ability to spontaneously interact with the overall system. This leads to a modular system architecture in which smart autonomous components cooperate to control a physical process without a central coordination facility.

Many of the today's control systems and the respective field-bus protocols maintain a centralized, cyclic model of control like the (CiA Draft Standards 301,1999), (DeviceNet Specification 2.0),(Smart Distributed Systems, 1996), and (PROFIBUS). A central master periodically polls its peripheral sensory I/O components, makes decisions and distributes commands to the actuators. At a first glance, this model has many advantages concerning predictability. However, when systems are becoming more complex, latency issues arise with the increasing number of components, which have to be accommodated. Secondly, the master constitutes a single point of failure which result in reliability or even safety problems for many advanced applications. Perhaps

The most important for the long term development of complex control systems is the fact that distributed processing power in the devices cannot be exploited properly by the centralized, cyclic model. As an example, consider a smart inclination sensor which allows to determine the slope of the terrain for a robot. It can not only give the actual inclination of the robot but also calculate a gradient along the robot's track. This sensor could generate alarm and warning messages, which are directly interpreted by smart actuators which stop the robot or move it backward. The underlying computational model is based on spontaneous events related to some physical process, which are sensed, processed and propagated by the autonomous components of the control system.

To meet the communication requirements of such systems, it is necessary to depart from the conventional field busses and master/slave-based protocols and move to networks, which support an event driven communication model. The problem here is to maintain the timeliness properties, the reliability, and the simplicity dictated by the cost/performance restrictions in control applications. Protocols using fixed priority mechanisms like rate monotonic message scheduling have been proposed by Tindell and Burns (1994). Although these protocols meet timeliness properties, they poorly support sporadic events and the coexistence of hard and soft real-time requirements. In this paper, we present a communication protocol for the CAN-Bus (CAN=Controller Area Network) (Robert Bosch GmbH, 1991). This protocol firstly schedules the messages according to their deadlines. Secondly, it provides transmission guarantees for hard real-time traffic, and thirdly gurantees that the order of messages delivered to the application is the same in all nodes and all messages are delivered at the same point of time. This is achieved in a completely decentralized way and supports periodic and sporadic messages. By carefully exploiting the properties of the CAN standard and the functionality of the CAN controller, the load for the host microcontroller is comparitively low.

The rest of the paper is organized as follows. First we will introduce some basic CAN properties necessary to introduce our protocol. Then we will describe our network scheduling mechanism based on a least laxity first approach. We will show how this mechanim is used to guarantee transmission of hard real-time traffic. Finally, we describe how a consistent order is achieved and a summary concludes the paper.

2. CAN PROPERTIES

CAN handles arbitration, message validation, error detection and error signaling in a very specific way. The properties of the CAN-Bus can be summarized in two points:

- Efficient use of available network bandwidth by providing a non-destructive priority-based arbitration of the bus. The arbitration mechanism schedules the messages which are competing for the bus according their priorities. The priority of a message is indicated in the message header.
- Immediate error detection, signaling and automatic retransmission of messages. The validation mechanisms of CAN achieves a consistent view about the status of a message at the end of every individual message transfer. Every message will be accepted or rejected by all participants. There are some very rare cases in which inconsistencies may occur. These cases are treated in detail in (Kaiser and Livani, 1999) and (Rufino et al 1998).

For a further detailed description of the CAN-protocol, the reader is refered to (Robert Bosch GmbH, 1991).

3. ACHIEVING TIMELINESS AND ORDER

Cooperative real-time actions need consensus about which action to be performed and when to perform the action. Consider the simple example depicted in fig.1. Two motors spin synchronously driving a workpiece. The main objective is to guarantee synchronism of the motors. For this purpose the motor controllers (μC) communicate sending actual speed and, in case of a serious local problem an alarm message. Additionaly, the speed of the motors can be adjusted externally by defining a new setpoint. Even if the command to adjust the speed to the actual setpoint may not be considered to be a hard real-time

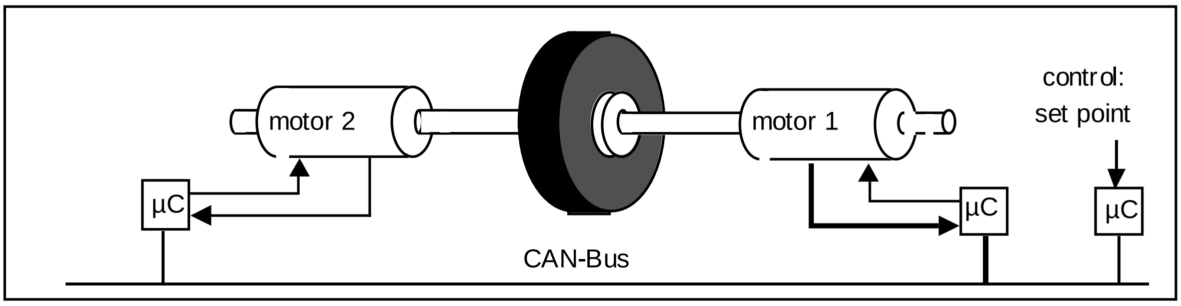

Fig. 1 Example of a control system

message, it is obvious that it has to be delivered to both motor controllers at the same time. This firstly requires that the command actually arrives at both nodes, a problem of reliable message transfer, and secondly, that there is a very small relative jitter for the delivery of messages at the distinct controllers.

If a new set point can be defined by multiple sources, or, more general, multiple messages which influence the motor speed are possible, it also has to be guaranteed, that the order of messages is the same for the two motor controllers. If messages have deadlines, they can be delivered at the deadline thus fulfilling the requirement of low jitter. However, soft real-time messages may miss their deadlines. Even in these situations, the order of messages has to be maintained as sketched in the example. We now first introduce our mechanism to schedule hard real-time messages and then present the ordering scheme.

3.1 Message Scheduling

In a real-time control system, a message has to be delivered at its deadline. We assume the co-existence of critical hard real-time messages and less critical soft real-time messages. The former have to meet their deadlines otherwise the system fails unpredictably. Soft real-time messages may miss their deadlines under transient overload. The priority mechanism provided by CAN is used as a basis to enforce deadline constraints. The basic idea is to dynamically increase the priority of a message the closer it comes to its transmission deadline[1]. This corresponds to a least-laxity-first scheduling scheme and is depicted in Fig. 2.

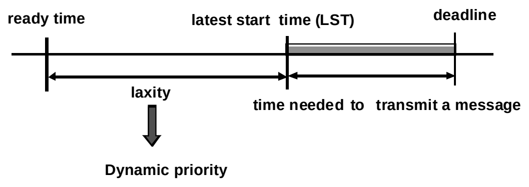

Fig. 2 Scheduling of messages

The entire CAN arbitration field defines a priority according to which messages are scheduled. In our protocol, the 8 most significant bits are used to explicitely define a priority level. The rest of the arbitration field (we use the extended 29-bit-format) is divided into two additional fields. One of them carries a node identifier which is necessary to guarantee uniqueness of the message ID as required by the basic CAN protocol and one field is exploited as a specific tag identifying the content of the message (Kaiser and Mock, 1999). These fields only implicetely participate in the priority scheme discriminating messages with the same high order priority bits.

For hard real-time messages, we have to guarantee that they are received before their deadlines even in case of transmission errors or overload situations. EDF alone cannot provide predictability under these assumptions. Therefore, for hard real-time messages we reserve fixed time slots like in a TDMA (Maruti-3, 1995) or TTP (Kopetz and Grünsteidl, 1992) approach. Conflicting resource requirements between hard real-time messages are resolved off-line, i.e. at run-time two hard real-time messages never will compete for the CAN-bus. If a time slot for a specific hard real-time message arrives, the dynamic priority mechanism assures that the message has the highest priority of all messages currently competing for the bus. To cope with ongoing message transfers and communication failures the reserved time slot must include additonal time margins to handle these situations.

Soft real-time messages and non-real-time messages share the portion of time that is not preallocated. Non real-time messages have a fixed low prioirity that does not conflict with any real-time message. Soft real-time messages are straightforwardly scheduled according to the LLF (Least Laxity First) algorithm. This means that in overload situations deadlines may be missed. A detailed description of the scheduling scheme is provided in (Livani and Kaiser, 1999a). An evaluation of the scheme can be found in (Livani and Kaiser, 1999c).

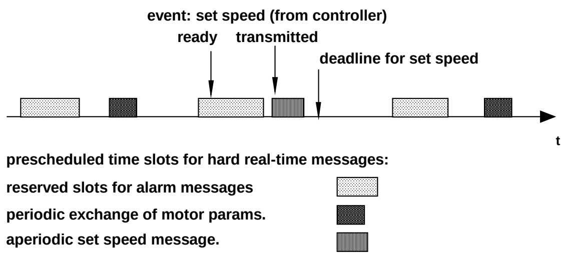
event: set speed (from controller)
ready
transmitted
deadline for set speed
t
prescheduled time slots for hard real-time messages:
reserved slots for alarm messages
periodic exchange of motor params.
aperiodic set speed message.

Fig. 3 Example of a message schedule

An example of a message schedule for the setting depicted in Fig. 1 is sketched in Fig. 3. The prescheduled slots are dedicated to specific hard real-time messages. The dynamic priority scheme prevents that the message which communicates the event "set speed" is transmitted immediately because this would affect the transmission of the prescheduled periodic exchange of motor parametres as speed, torque etc.

[1] We distinguish between a transmission deadline and a delivery deadline. The transmission deadline is defined as the delivery deadline minus the time to transmit the message.

The dynamic priority scheme, obviously, requires a certain overhead during run-time to update the priorities according to the laxity. However, it has to be noted that on each node, only the message currently competing for the bus has to be maintained. The other messages in the send queue remain untouched until they are transferred to the transmit buffer of the CAN controller. We calculated the CPU time needed to update the dynamic priority on a Siemens C167 16-Bit microcontroller (Siemens, 1996). It is about 13 µsec or 5% total overhead in a typical setting (Livani and Kaiser, 1999c).

3.2 Order of messages

The mechanism to achieve order relies on the reliable message transfer and on the knowledge about the message transmission deadlines. The *transmission deadline (TD)* of a message denotes the point of time, at which the message must be successfully transmitted to all non-faulty destination nodes. The transmission deadline is tightly related to the point in time, where the message delivery to all receiving application objects is expected. Therefore, the ordering mechanism reflects an application specific order related to temporal requirements. Let *m* and *m'* be two real-time messages with deadlines dm and dm'. Then the deadline-based ordering algorithm implements the following rule: $dm < dm' \Rightarrow delj(m) \rightarrow delj(m')$. This means that if the deadline of the message *m* is before the deadline of the message *m'*, then *m* must be delivered to destination objects before *m'*. For hard real-time messages this is easy to show. Because hard real-time messages are scheduled off-line and are transmitted in reserved time slots, the above rule follows immediately. The order between hard real-time messages can straightforwardly be formulated by the following delivery rule: Every hard real-time message is delivered at its deadline. Total order of hard real-time messages is guaranteed according to their deadlines.

To establish a total order between soft real-time messages is more complicated. Soft real-time messages can become ready for transmission at any point in time. No a priori reservation or conflict resolution is performed. We may have messages with different and identical transmission deadlines. This results in the same laxity that is mapped on the same priority. Under transient overload, multiple messages that already have missed their deadlines end up with the same laxity. In both cases we cannot use the priority field (8 most significant bits) in the CAN message ID alone to establish the order. However, because CAN IDs all must be different (to enable an unambiguous arbitration decision), some arbitrary order is determined by the implicit priority defined by the node ID and the content tag (cf. 3.1). Let us denote this least significant portion of the CAN-ID as message-IDm. Note, that in case of equal deadlines and in the situation that all laxities are "0", no causal relationship between the messages can exist because none of the messages has yet been sent. The following rule is used to ensure a global decision on the delivery order of messages of m and m': $dm = dm' \Rightarrow (delj(m) \rightarrow delj(m') \Leftrightarrow \text{message-IDm} < \text{message-IDm'})$.

The consistent order is based on the following observation: A CAN controller automatically resends a message in case of a transmission error until all receivers have correctly received it. If a node has received a message *m*, and then another message with lower priority is observed on the CAN bus, or the bus is idle, then the sender of *m* will not retransmit it in the future. Hence, in this case, the receiver can be sure that the higher priority message has been received by all other nodes. To deliver this message, the receiver must also be sure that it is delivered in the same order in all nodes. This order should be based on the transmission deadline. Therefore, we now must first clarify the question whether a message with an earlier deadline still can arrive. Based on our previous considerations we can conclude that a soft real-time message with a higher priority has an earlier deadline than a soft real-time message with a lower priority. If after the transmission deadline of a soft real-time message m another message with lower priority is transmitted on the CAN bus, or the bus is idle, then there is no other soft real-time message m' with an earlier or equal deadline, which is pending for transmission. Therefore, we can state the condition for total order as:

Every received soft real-time message can be delivered in total order as soon as either another message with later deadline, or an idle time on the bus is observed after its transmission deadline.

A formal proof for this theorem can be found in (Livani and Kaiser, 1999b). It should be noted that these events are detected synchronously by all nodes so that the delivery of the respective message happens at the same point in time.

In situations which are not affected by overload, soft real-time messages are transmitted and delivered in strict deadline order. In overload situations, by definition, no temporal guarantees can be given. The deadlines cannot be used for ordering the messages. However, total order between soft real-time messages is still preserved. It should be noted that the consistent order of messages is established without extra communication overhead.

Related work considering order in a CAN-Bus system is presented by Zuberi and Shin (1995) and Rufino *et al.*, (1998). In (Zuberi and Shin, 1996) a CAD-tool is used to determine off-line, which messages in an application may be causally related. The deadlines of related messages are adjusted according to this analysis. Then, they use a dynamic priority scheme (Zuberi and Shin, 1995) which is similar to our scheme to schedule the messages on the CAN-bus according to the fixed deadlines. The approach is only valid in systems in which all causal relationships can be determined off-line. Secondly, the approach treats all messages as hard real-time messages. In (Rufino *et al.*, 1998) a protocol called TOTCAN is proposed which uses a two phase approach to achieve total order of messages. This approach results in a considerable overhead compared to our scheme. Particularly, this approach needs many additional messages for the establishment of a consistent order.

4 CONCLUSION

The paper presents a transport level protocol for the CAN-Bus. The goal was to provide a protocol which maintains the essential predictability features found in centralized approaches like master/slave configurations but works in a event-based control system in a completely dynamic, decentralized way. This is possible because the underlying CAN-Bus was just designed to support distributed control and thus represents a good starting basis for the efficient realization of decetralized mechanisms. The priority-based arbitration mechanism of CAN is exploited to realize a dynamic LLF message scheduling scheme. Combined with off-line reservation, the protocol enforces transmission guaranties for hard real-time messages. Soft real-time messages are usually delivered at their deadlines but may miss their deadlines in overload situations. However the protocol guarantees that they are delivered at the same order and at the same point in time in all nodes. The protocol carefully exploits the features provided by CAN controller hardware. As a result the overhead of the protocol is low. It needs some local effort in every node to maintain dynamic priorities but no additional communication bandwidth is needed to achieve consistent order.

5 REFERENCES

CiA Draft Standards 301: „CANopen Application Layer and Communication Profile“, *Version 4.0, June1999*

DeviceNet Specification 2.0 Vol. 1, *Published by ODVA, 8222 Wiles Road - Suite 287 - Coral Springs, FL 33067 USA.*

Kaiser, J. amd M.A. Livani:"Achieving Fault-Tolerant Broadcasts in CAN", *Proc. of the 3rd European Dependable Computing Conference, (EDCC-3), Prague, Sept. 1999*

Kaiser, J. and M. Mock : "Implementing the Real-Time Publisher/Subscriber Model on the Controller Area Network (CAN)", *2nd Int'l Symposium on Object-Oriented Distributed Real-Time Computing Systems, San Malo, May 1999*

Kopetz, H. and and G. Grünsteidl (1992): „TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems“, *Res. Report 12/92, Inst. f. Techn. Informatik, Tech. Univ. of Vienna, 1992.*

Livani, M.A. and J. Kaiser (1999a): "Scheduling Hard and Soft Real-Time Communication in the Controller Area Network (CAN)", *Control Engineering Practice, Vol 7, No. 12, pp.1515-1523, Dec. 1999*

Livani, M.A. and J. Kaiser (1999b):"A Total Ordering Scheme for Real-Time Multicasts in CAN", *Proc. of the Joint 24th IFAC/IFIP Workshop on Active and Real-Time Database Systems, Schloß Dagstuhl, Germany, May 1999*

Livani, M.A. and J. Kaiser (1999c): „Evaluation of a Hybrid Real-time Bus Scheduling Mechanism for CAN“, *7th Int'l Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'99), San Juan, Puerto Rico, Apr. 1999.*

Maruti 3, Design Overview 1st Edition, System Design and Analysis Group, *Dept. of Comp. Science, Univ. of Maryland, 1995.*

PROFIBUS: Technical Overview *http://www.profibus.com/data/technic/index.html*

ROBERT BOSCH GmbH: „CAN Specification *Version 2.0“, Sep. 1991*

Rufino, J. , P.J. Verisimo, C. Almeida and L. Rodrigues:"Fault-Tolerant Broadcasts in CAN", *Proc. FTCS-28, Munich, Germany, June 1998*

Siemens AG: „C167 User's Manual 03.96“, *Published by Siemens AG, Bereich Halbleiter, Marketing-Kommunikation, 1996.*

Smart Distributed Systems, Application Layer Protocol Version 2, *Honeywell Inc, Micro Switch Specification GS 052 103 Issue 3, USA, 1996*

Tindell, K. and A. Burns : „Guaranteed Message Latencies for Distributed Safety-Critical Hard Real-Time Control Networks“, *Report YCS229, Department of Computer Science, University of York, May 1994*.

Zuberi, K.M., and K.G. Shin:"Non-Preemptive Scheduling of messages on the Controller Area Network for Real-Time Control Applications", *Tech. Report, University of Michigan, 1995*

Zuberi, K.M., and K.G. Shin:" A Causal Message Ordering Scheme for Distributed Embedded Real-Time Systems", *Proc. Symp. on Reliable and Distributed Systems, Oct. 1996*