SCHEDULING HARD AND SOFT REAL-TIME COMMUNICATION IN THE CONTROLLER AREA NETWORK (CAN)

M.A. LIVANI[*], J. KAISER[*] and W.J. JIA[**]

[*]*University of Ulm, Department of Computer Structures, 89069 Ulm, Germany*
[**]*City University of Hong Kong, Department of Computer Science, Hong Kong*

Abstract. The paper introduces a mechanism to implement distributed scheduling for CAN-bus resource in order to meet the requirements of a dynamic distributed real-time system. The key issues considered here, are multicasting, distinguishing between hard real-time, soft real-time, and non real-time constraints, achieving high resource utilization for CAN-bus, and supporting dynamic hard real-time computing by allowing dynamic reservation of communication resources.

Key Words. Real-Time Communication, CAN, Scheduling.

1. INTRODUCTION

A real-time communication system (RTCS) constitutes the backbone for distributed control applications. RTCS substantially differ in many respects from general purpose communication systems. In general, while the goals of general purpose communication systems center around throughput, RTCS focus on predictability of communication. Predictability means that the system exhibits an anticipated behaviour in the functional and the temporal domain.

In the area of industrial automation and the automotive industry, field busses are used to disseminate time critical messages. Field busses exhibit bounded message latency, reliability of transfer, efficiency of the protocol and, to a certain extent, preventing a node from monopolizing the network. Among field busses, CAN bus (BOSCH (1991)) provides advanced built-in features, which make it suitable for complex real-time applications. Some of these features are priority-based, multiparty bus access control using carrier sense / multiple access with collision avoidance (CSMA/CA), bounded message length, efficient implementation of positive/negative acknowledgement, and automatic fail-silence enforcement with different fault levels.

As a common resource, the CAN bus has to be shared by all computing nodes. Access to the bus has to be scheduled in a way that distributed computations meet their deadlines in spite of competition for the communication line. Since the scheduling of the bus cannot be based on local decisions, a distributed consensus about the bus access has to be achieved.

There exist several alternative approaches to solve this problem based on the assumptions about the behaviour of the system and the environment.

The deadline-monotonic priority assignment (Audsley et al. (1991), Tindell et al. (1994, 1995)) achieves meeting deadlines as guaranteed by an offline feasibility test for a static system with periodic tasks. Although static systems can be scheduled easily by this approach, it does not allow scheduling of dynamic systems, where an offline feasibility test has incomplete knowledge about the future behavior of the system. Moreover, the deadline-monotonic approach does not distinguish hard and soft deadlines.

A scheduling mechanism for the CAN bus, based on a mixture of dynamic and static priorities, has been approached by Zuberi and Shin (1995). However, this approach makes unrealistic assumptions about CAN – e.g. 10 Mbits/s – and exhibits a rather restricted scheduling ability due to a short time horizon (cf. section 4).

Livani et al. (1998) have introduced a mechanism to assign dynamic priorities to CAN messages, in order to achieve an EDF resource access consensus among the participating nodes. Based on the EDF access mechanism, soft real-time communication will be scheduled optimally with EDF approach. To guarantee deadlines of hard real-time communication, calendar-based resource reservation is applied. An application of this scheduling approach in a distributed object-oriented real-time control system has been introduced by Kaiser et al. (1998a).

The presented approach takes advantage of the built-in CSMA/CA access protocol of CAN bus to realize the EDF access regulation. The CSMA/CA protocol is comparable with a priority-based dispatcher. Due to this analogy, it is possible to express scheduling decisions for the CAN-bus resource by static or dynamic priority orders.

The paper is organized as follows: Chapter 2 introduces the application system model. Chapter 3 describes some features of CAN. In chapter 4 the authors briefly describe a dynamic priority assignment mechanism, resulting in EDF access regulation. Based on the EDF access regulation, chapter 5 introduces a real-time scheduling approach for communication on CAN bus. A summary concludes the paper.

2. THE SYSTEM MODEL

The system model in this paper is influenced by the anticipated hardware basis. The authors assume a number of different micro-controllers with different performance attributes ranging from 8-bit to 32-bit architectures. All micro-controllers are equipped with a CAN-Bus interface. The model exploits the inexpensive availability of 8-bit micro-controllers to structure the overall task of the system into small packages, i.e. objects with a well-defined message interface to other objects. Of course, it is possible to locate several tightly related objects in one node.

Higher control instances are available to control groups of objects or eventually the entire system using more powerful microprocessors. Thus, the higher control instances have a well defined instrumentation interface which is composed from objects, each encapsulating a certain functionality.

An object has a unique name and a set of associated operations. The unique name of the object is translated to a short form system name during run-time and maintained by a configuration service.

Object groups

In conventional object-oriented languages a method invocation is synchronous and directed to a single object. It is usually not possible to express a request to a group of objects. However, in a real-time control system it is beneficial to provide groups of objects and to use asynchronous multicasts to invoke methods of these groups. The motivation ranges from simple and fast distribution of information, like sensor data and alarm messages, to replicated objects forming a group to achieve fault-tolerance.

In this system model, an object participating in group communication, does not necessarily have any knowledge about the number and location of other group members. Therefore, the sender of a multicast usually does not know whether it has to send the message locally, or remote. This simplifies the design and implementation of the objects, and minimizes the configuration effort when adding or removing an object.

But consequently, there must be an instance in the system, which knows the configuration. The configuration is given by object groups, where a group contains one or more objects. This configuration information is maintained by distributed multicast agents, one residing on every computing node. Every object wishing to send a multicast message, requests its local multicast agent to deliver the message to other group members. The multicast agent is described in more details in Kaiser et al. (1998a).

Group Communication Protocol

By using CAN bus as the basic communication medium, one can relax many assumptions necessary for arbitrary networks, resulting in a rather low overhead protocol. Because the communication medium, the CAN-Bus, supports consistent view on the low system level, it is beneficial to exploit this feature.

In order to support real-time object groups with consistent information, the group communication protocol must deliver real-time messages to all members of a group both *timely*, and in a *consistent order*. In chapters 4 and 5 the authors describe how to guarantee timely multicast transmission in CAN. Given the guarantee of timely message transmission, Kaiser et al. (1998b) have shown that atomic multicast can be achieved by consistent ordering of messages based on application-defined delivery deadlines.

3. LOW-LEVEL CONSISTENCY IN CAN

Since in the CAN bus every bit is propagated over the whole bus length while it is still transmitted, all correct nodes have a consistent view of every bit. In the case that different nodes send different bit values simultanously, the dominant value – which changes the bus level actively – overrules the recessive value. In all common implementations of CAN, ‘0’ is the dominant bit value, i.e. whenever different bit values are sent on the bus simultanously, the logical AND function of their values is observed by every node. Thus, the nodes of a CAN bus are logically connected by a wired-AND function.

The consistent view of every bit among all nodes, is exploited in the CAN protocol to implement a low-latency error detection and signaling mechanism. During the transmission of every message, all CAN controllers monitor every bit, and detect any violation of the consistency rules (e.g. CRC, bit-stuffing) with a high probability (BOSCH (1991)). After a node detects an error, it sends a sequence of six dominant bits, thus violating the bit-stuffing rule. At this time, every node in the network becomes aware of the error, the message is discarded, and retransmitted by the sender.

Also the basic medium access protocol of CAN, called CSMA/CA, relies on a consistent global view of every bit. According to this mechanism, all

nodes competing for the bus begin to send the unique identifier of their messages, which consists of 11 or 29 bits, according to the „standard“ or „extended“ message format. whenever a node sends a ‘1’ and senses a ‘0’, it stops transmitting, and switches to the receiver mode. At the end of the identifier field, only the node which is sending the message with the lowest identifier value, is still transmitting.

CAN provides a connectionless protocol. This means, that the message identifier indicates the content of a message rather than the source or destination. Each communication controller has a programmable filter through which it can associatively detect relevant messages. By setting some of the filter bits as „don’t care“, it is possible to receive groups of messages.

4. EDF ACCESS REGULATION

As it is shown in chapter 5, an EDF medium access regulation among all network nodes can serve as a basis for a hybrid bus scheduling, where timeliness of hard real-time messages is guaranteed by resource reservation, while optimal scheduling of soft real-time communication is achieved by EDF scheduling strategy.

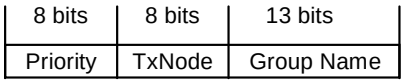

Fig. 1. Partitioning of a CAN-message identifier

In order to realize the EDF access regulation in a CAN network, one must define a mapping of the transmission deadline into the message priority, such that a message with an earlier transmission deadline wins the bus arbitration against a message with a later deadline. However, one cannot use the whole identifier as a deadline-driven dynamic priority because of following reasons: Firstly, the sender node identifier must be included into the identifier field of CAN messages, to ensure that competing messages have always different identifiers. Secondly, one should encode the message group name into the identifier in order to support group addressing and multicast message filtering by the CAN controller hardware. Fig. 1 illustrates how a CAN identifier is partitioned in order to fulfill all functionalities mentioned above.

Due to the requirement to schedule three types of communication, namely hard real-time, soft real-time, and non real-time, the value domain of the 8-bit dynamic priority field is divided into two distinct ranges. The higher priorities are reserved for real-time messages, and the lower priorities are assigned to non real-time messages. This ensures that real-time messages always win the bus against non real-time messages.

In the priority field of a real-time message, the time remaining until its transmission deadline (let’s call it *transmission laxity*) is encoded. The *transmission deadline* is a point of time specified by the sending application object, when a message must be completely transmitted to receiving nodes. As long as a sending node is pending for the bus, its communication subsystem checks and updates the transmission laxity of the ready message periodically. As soon as the transmission laxity of a hard real-time message becomes zero, it is withdrawn, and the sending application object is notified of the deadline failure. Soft real-time messages, however, are sent in spite of missed deadline.

Each value of the transmission laxity is mapped to a portion of future time, a *priority tick* Δt_p. At the end of each priority tick, a pending transmitter increases the priority of its real-time message by decrementing its transmission laxity field. The priority ticks are time intervals of a fixed length, with the first one beginning at the present time. Since there is only a limited number of different priorities, only a limited number of priority ticks are visible. Hence, one cannot distinguish different points of time within and after the last identified priority tick. Since one can make no order decision between any two points of time falling into the last priority tick, the beginning of the last priority tick will be the *time horizon* of the proposed deadline-driven access decision.

Def. The Time Horizon: the time horizon is defined as the point of time, where a time-driven decision mechanism (like EDF scheduler) cannot see beyond.

In this case, if the time until the time horizon is only long enough for the transmission of n messages, then the EDF bus scheduler cannot guarantee correct scheduling of n+1 pending real-time messages with deadlines beyond the time horizon.

5. SCHEDULING HARD AND SOFT REAL-TIME MESSAGES

Different urgency classes

The authors assume four classes of activities in the system, which have been introduced by Stankovic et al. (1989). Critical activities are hard real-time and their deadlines are absolute in time. To guarantee the timely execution of critical activities, all their occurrences have to be predicted and respective resources have to be scheduled in advance in a periodic calendar. Essential activities have deadlines relative to their start times. If the system grants an essential task, it guarantees the

resources for a timely execution by reserving the appropriate time slots in the resource scheduling calendar. However, the system can refuse a guarantee.

Soft real time activities have deadlines which are considered by the system but no guarantees are given to meet such a deadline. Soft real-time activities are executed on a best effort basis. However, as shown later, the EDF scheduling strategy is used for optimal resource utilization. Non real-time activities can only use resources which are not requested by a real-time activity.

The global scheduling in a distributed system requires consensus between all participants about the usage of shared system resources. Particularly, if a joint action will be performed, all local resources must be available and reserved for the respective computation (Gergeleit et al. (1994b)). The global schedule has to be enforced by all participants, based on their local information. In a completely static system, a global calendar is available and each participant has its relevant entries referring to its activities in a global time scale. A local activity may only be started according to this schedule (Kopetz et al. (1985, 1992)). In a more dynamic system where critical, essential, soft real-time and non-real-time tasks coexist, things are more complicated. If a computing resource is free, a less critical task may start computation and request resources. In this case, it must be guaranteed that it does not block an activity with a higher criticality. In this section, the enforcement of the global schedule for the shared bus resource is described.

Timeliness of hard real-time communication

For hard real-time communication, the deadline is guaranteed by reserving a time slot on the bus. The reserved time slots are entered into a calendar, which contains both periodic fixed reservations and dynamic reservations. This calendar is contained in each node of the system, thus the dynamic time slot reservation is performed by a single atomic multicast message only.

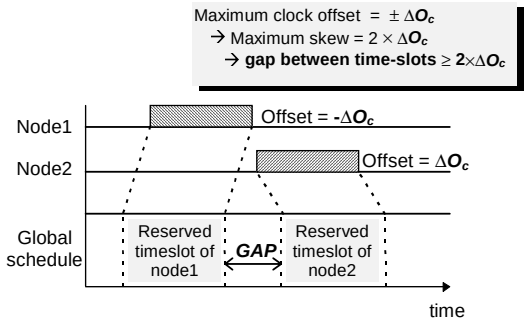

Fig. 2. The gap between reserved time-slots due to clock inaccuracy

The scheduling approach for hard real-time communication requires access to a global time reference with bounded inaccuracy. Once a time slot is reserved, the respective action can be started locally. To guarantee that it does not interfere with another time slot, the time reference of all nodes must be synchronized. The lower the clock accuracy, the larger the minimum gap between two subsequent time slots in the global bus schedule (Fig. 2). In order to provide a global time reference with high accuracy, clock synchronization mechanisms have to be applied, e.g. as described by Kopetz et al. (1987) or Gergeleit et al. (1994a).

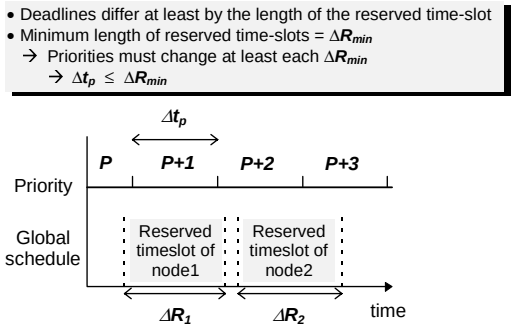

Fig. 3. Maximum length of priority tick

Efficient reservation of time slots for hard real-time communication is supported by defining the end of the reserved time slot of a message as its transmission deadline, thus applying EDF scheduling to hard real-time communication. The sender of a hard real-time message enforces its access right by dynamically increasing the priority of the message according to its laxity relative to the reserved time slot. Due to this scheme, a hard real-time message gains the highest possible priority at the beginning of its reserved time slot. In order to guarantee that two subsequent reserved time-slots are always assigned different priorities in the right order, the priority tick must not be longer than the minimum possible length of any reserved time-slot (Fig. 3). In the current prototype, the minimum time-slot length for a single-fault tolerant message is equal to the transmission time for two empty frames plus error detection and recovery time, which take together at least 175 bit-times.

Note that every message may be delayed by one message, which is started before the ready time of it. ΔT_{max} is defined as the longest possible message transmission time. Then, a hard real-time message k – with a reserved time slot beginning at s_k – must be ready before $s_k - \Delta T_{max}$, in order to tolerate the non-preemptive transmission of any single message.

In order to guarantee the collective timeliness of hard real-time communication, the following requirements must be met:

(R1) for each hard real-time message, an exclusive time-slot is reserved, which ends at the transmission deadline of the message,
(R2) the reserved time-slot of each message is as long as the worst-case transmission time of the message, including all overheads for error detection, recovery, and retransmissions under an anticipated fault hypothesis,
(R3) the gap between any two different reserved time slots for hard real-time messages is greater or equal to the maximum time-skew between any two correct nodes in the system,
(R4) every hard real-time message is ready for transmission at least ΔT_{max} before the beginning of its reserved time slot. This means that the laxity of every hard real-time message at its ready-time is large enough to allow the longest possible message of the system to be transmitted first,
(R5) the priority of a hard real-time message depends on its transmission laxity, (cf. chapter 4).

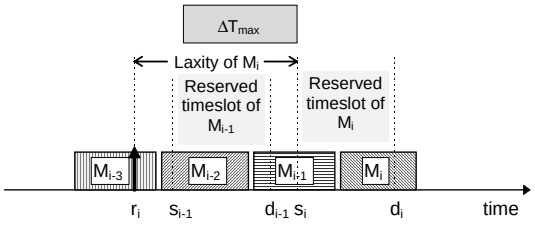

Fig. 4. A hard real-time message missing its deadline

Given these assumptions, every hard real-time message will be transmitted timely under anticipated fault conditions. Assume (Fig. 4) that M_i is the first hard real-time message after the system startup, which cannot start at the beginning of its reserved time slot s_i, hence missing its transmission deadline d_i. Since M_i is ready at $s_i - \Delta T_{max}$, it participates in at least one arbitration process before s_i. If M_i does not start before s_i, then another message M_{i-1} wins the arbitration against M_i, and following conditions are true:

(C1) M_{i-1} is a hard real-time message,
(C2) the reserved time slot of M_{i-1} ends at d_{i-1}, where $d_{i-1} < d_i$, because of R5, and
(C3) M_{i-1} is not completely transmitted until s_i.

From R3 and C2 it is concluded that the transmission deadline of M_{i-1} lies before s_i, i.e. $d_{i-1} < s_i$. Due to C3, M_{i-1} misses its deadline d_{i-1}, because it is completed after s_i, and $d_{i-1} < s_i$. This is a contradiction to the assumption that M_i is the *first* hard real-time message after system startup, which misses its deadline.

Timeliness of soft real-time communication

For the soft real-time communication, the presented mechanism does not guarantee a deadline. This is because it is always possible to dynamically schedule additional hard real-time („essential“) communication. However, this approach guarantees optimal scheduling of soft real-time messages, because the priority of a soft real-time message depends directly on the time remaining until its delivery deadline, thus realizing EDF scheduling, which is known to be optimal (Liu et al. (1973)).

Conflicts of soft and hard deadlines

Since soft real-time communiation does not make any time-slot reservation, the deadlines of soft real-time messages are arbitrarily spread over the time. This can lead to conflicts while scheduling several soft real-time messages with deadlines close together. While deadline conflicts among soft real-time messages are always acceptable, they must never affect the timeliness of a hard real-time message. This is achieved by guaranteeing that a hard real-time message has a higher priority than any soft real-time message within a critical interval before its dealine. This critical interval begins after the latest ready time of a hard real-time message:

worst-case latest ready time of any HRT-message = Deadline - ΔRmax - ΔTmax
where:
ΔR_{max} *= longest possible reserved time-slot of any HRT-message*
ΔT_{max} *= longest possible single transmission time of any message*

From this requirement the priorities of hard and soft real-time messages are derived depending on their transmission laxity:

Hard real-time:
$P_H(laxity) = min(P_{min} + \lfloor laxity / \Delta t_p \rfloor , P_{max})$
where:
$P_H(laxity)$ *is the priority of a hard real-time message with a given 'laxity'*
P_{min} *is the highest priority = lowest binary value for real-time priorities*
P_{max} *is the lowest priority = highest binary value for real-time priorities*
Soft real-time:
$P_S(laxity) = max(P_H(\Delta R_{max} + \Delta T_{max}) , min(P_{min} + \lfloor laxity / \Delta t_p \rfloor , P_{max}))$
where:
$P_S(laxity)$ *is the priority of a soft real-time message with a given 'laxity'*

Non real-time communication

Non real-time messages are assigned fixed priorities, because the importance of a non real-time message does not change by the passage of time. Note that in this approach any non real-time message has always a lower priority than any real-

time message. It means: $P_N > P_{max}$ *where* P_N *is the priority of an arbitrary non real-time message.*

Dealing with faults

In order to guarantee timely hard real-time message transfer in the presence of faults, redundancy must be provided. Space redundancy would require a second CAN bus. Since a fault model with only *crash* and *omission failures* is assumed, time redundancy can be applied instead of space redundancy. This means that several subsequent transmissions plus the failure detection/recovery of CAN bus, which consumes bounded time (Rufino et al. 1995)), have to be scheduled. This application of time redundancy is similar to strategies used in other real-time communication protocols, e.g. TTP (Kopetz et al. (1992)). However, in contrast to statically planned communication, this can use the redundant time slot for low-priority communication, if the first transmission was successful.

6. CONCLUSION AND FUTURE RESEARCH

In order to guarantee timely delivery of hard real-time messages in a CAN network, the authors have introduced a calendar-based scheduling mechanism, which coexists with the EDF scheduling used for soft real-time messages. In contrast to pure EDF scheduling, where resource conflicts may occur due to conflicting deadlines, the presented approach coordinates the deadlines of hard real-time communication by reserving resources in a global calendar. Thus, the timeliness of hard real-time communication is achieved despite overload failures by guaranteeing exclusive access right to the network during the reserved time-slots.

This approach guarantees timeliness in presence of communication errors, by applying time redundancy. However, in case of non-faulty message transfer, the unused redundant resources are utilized for low-priority communication. This is possible because a hard real-time message releases the network resources after its successful transmission. Thus, optimal resource utilization in fault-free situations as well as in presence of faults is achieved.

The low-level communication protocol presented in this paper will serve as a basis for a high-level atomic multicast protocol with bounded termination time. The concept of the high-level atomic multicast protocol is reported by Kaiser et al. (1998b).

7. REFERENCES

Audsley, N.C., A. Burns, M.F. Richardson, A.J. Wellings (1991). Hard Real-Time Scheduling: The Deadline Monotonic Approach. *Proceedings of 8th IEEE Workshop on Real-Time Operating Systems and Software.*

Gergeleit, M. and H. Streich (1994a). Implementing a Distributed High-Resolution Real-Time Clock using the CAN-Bus. *Proceedings of the 1st International CAN-Conference, Mainz.*

Gergeleit, M., J. Kaiser, H. Streich (1994b). DIRECT: Towards a Distributed Object-Oriented Real-Time Control System. *Workshop on Concurrent Object-based Systems.*

Kaiser, J. and M.A. Livani (1998a). Invocation of Real-Time Objects in a CAN Bus-System. *1st Int'l Symposium on Object-Oriented Distributed Real-Time Computing Systems, Kyoto.*

Kaiser, J. and M.A. Livani (1998b) Predictable Atomic Multicast in Controller Area Network (CAN). *Technical Report, University of Ulm.*

Kaiser, J., M.A. Livani, W. Jia (1997). Membership and Order in a CAN-Bus Based Real-Time Communication System. *Proc. of Int'l Workshop Advance Parallel Processing Technology.*

Kopetz, H. and G. Grünsteidl (1992). TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems. *Research Report No. 12/92, Inst. für Techn. Informatik, Techn. University of Vienna.*

Kopetz, H. and W. Ochsenreiter (1987). Clock Synchronization in Distributed Real-Time Systrems. *IEEE Trans. on Computers 36(8).*

Kopetz, H. and W. Merker (1985). The Architecture of MARS. FTCS-15.

Liu, C.L. and J.W. Layland (1973). Scheduling Algorithms for Multiprogramming in a Hard real-Time Environment. *J. ACM 20(1):46-61.*

Livani, M.A. and J. Kaiser (1998). EDF Consensus on CAN Bus Access for Dynamic Real-Time Applications. *Lecture Notes in Computer Science 1388 (Jose Rolim Ed.), pp. 1088-1097, Springer Verlag Berlin, 1998.*

BOSCH (1991). CAN Specification Version 2.0.

Rufino, J. and P. Veríssimo (1995). A Study on the Inaccessibility Characteristics of the Controller Area Network. *2nd Int'l CAN Conference 95.*

Stankovic, J.A. and K. Ramamritham (1989). The Spring Kernel: A New Paradigm for Real-Time Operating Systems. *ACM Operating Systems Review 23(3).*.

Tindell, K. and A. Burns (1994). Guaranteeing Message Latencies on Control Area Network (CAN). *1st International CAN Conference 94.*

Tindell, K., A. Burns, A. Wellings (1995). Calculating Controller Area Network (CAN) Message Response Times. *Control Engineering Practice, 3(8):1163-1169.*

Zuberi, K.M. and K.G. Shin (1995). Non-Preemptive Scheduling of messages on Controller Area Network for Real-Time Control Applications. *Proc. Real-Time Techn. and Appl. Symposium.*