A Real-Time Event Channel Model for the CAN-Bus*

Joerg Kaiser, Cristiano Brudna, Carlos Mitidieri
University of Ulm
Dept. of Computer Structures
Ulm, Germany
{kaiser, cristiano.brudna, carlos.mitidieri}@informatik.uni-ulm.de

Abstract

The paper describes a real-time event channel model in a publisher/subscriber communication scheme. The model specifically considers temporal and reliability attributes and suggests an API that integrates the real-time aspects in the event channel model. According to the need in most real-time systems, we support event channels with different timeliness and reliability classes. Hard real-time event channels are considered to meet all temporal requirements under the specified fault assumptions. The resource requirements for this type of channel are statically assigned by an appropriate reservation scheme. Soft real-time event channels are scheduled by their deadlines, but they are not guaranteed under transient overload conditions. Non real-time event channels are used for events without any specified timeliness requirements in a best-effort manner. The paper finally presents how the different channel classes are mapped to the mechanisms necessary to implement the model on the CAN-Bus.

1 Introduction

The publisher/subscriber (P/S) model has been recognized as an appropriate high-level communication scheme to connect autonomous components in large distributed control systems [21], [20]. Particularly, P/S supports autonomy of components by an asynchronous notification mechanism omitting any implicit control transfer coupled with the exchange of information. This is in contrast to other high-level interaction models as remote procedure calls or remote invocations which create global dependencies and side effects which require complex mechanisms to handle temporal and functional fault situations. Secondly, P/S relies on a content-based communication scheme. This means that the content of a message is used to route a message rather than an address. A subscriber, which is interested in particular information, e.g. the temperature data of some sensor, subscribes to the particular information rather than to a specific sensor. This has the advantage that the subscriber does not have to know any specific sensor name or address and thus, a content-based addressing mechanism substantially encourages dynamic adaptability and extensibility requirements. Moreover, it may be the basis of transparent fault-tolerance mechanisms in which multiple sources provide the same information. In our P/S protocol we adopted a slightly less general variant of content-based addressing which maps an arbitrary content to a subject field. Subject-based addressing supports our event channel concept and is more suited in a real-time environment because of predictability reasons. Additionally, subject-based addressing can be optimized to meet the requirements of the restricted computational resources found in a control system composed from smart sensors and actuators [12]. A more detailed discussion of the P/S model is beyond the scope of this paper and can be found in [6], [18]. In this paper we will describe our P/S model which is based on event channels and particularly intended for real-time control applications. We will briefly review the specific problems when implementing the P/S model in a system composed from smart sensors and actuators which usually have only limited computational performance. The focus of our paper is the description of different real-time event channel classes and their programming interface. Finally, we will show, how to map this model to lower level mechanisms of the CAN-Bus [2] to enforce the required real-time guarantees.

2 Events and event channels

Event and the associated event channels are central conceptual constructs in our system. An event is related to an occurrence in the real world, e.g. observed by a sensor, or

*This work has been supported by the European Union's Information Society Technology Program under contract IST-2000.26031 (CORTEX: Cooperating real-time sentient objects: architecture and Experimental evaluation).

an in the control system itself, e.g. generated by some control program. Subscribers, which have stated their interest in an event, are asynchronously notified when this event occurs. An ***event*** is an instance of an event type, which is characterized by a subject, attributes and content:

event := <subject, attribute_list, content >

According to the subject-based addressing in the P/S protocol, the subject is a tag related to the content of an event. In our system, a subject is represented by a unique identifier. The attributes are related to the context, in which an event is generated like location, time, mode of operation, etc. and to quality aspects like a validity interval (expiration time) and a deadline[1]. They represent non-functional properties of the event. The content of an event carries the data and is represented as a structured set of functional parameters. The fields of the content are accessible by specific methods.

An ***event channel*** is an architectural component of the middleware which disseminates all events of a certain subject. Publishers and subscribers interact with the event channel to publish events or receive notifications. An event channel is an instance of an event channel type. It comprises the subject of events which can be disseminated by the channel and attributes:

event_channel := <subject, attribute_list >

In contrast to the attributes of an event which describes the properties of a single individual occurrence of an event, the attributes of the event channel abstract the properties of the underlying communication network and dissemination scheme. Therefore attributes include e.g. latency, dissemination constraints and reliability parameters. An event channel is dynamically created whenever a publisher makes an announcement for publication or a subscriber subscribes for an event notification. For every event type there is at most one event channel. An event channel may handle multiple publishers and multiple subscribers, thus representing a many-to-many communication channel. When a publisher announces a publication, the respective data structures of an event channel are created by the middleware. When a subscriber subscribes to an event channel, it may specify attributes of the event and the event channel. As described later, these attributes are used for type checking and filtering.

2.1 Routing, filtering and binding

Implementing the publisher/subscriber model requires to map the abstractions of that model like events and event channels to the elements provided by the technical infrastructure of the system such messages and addresses. In the publisher/subscriber scheme, events are routed by their content from a publisher to an interested subscriber. Therefore the issue of how to map this to a network includes the problem of 1. getting a message to the right destination (routing) and 2. that the destination should only receive those messages which it is interested in (filtering).

We adopted a subject-based routing mechanism [20][6] and introduced dynamic binding as an additional optimization. Dynamic binding trades the degree of flexibility of the subject based mechanism against improved routing performance and filtering overhead and makes this approach feasible for small embedded controllers. Dynamic binding relates a subject of an event to an low level address of the underlying network. Then, the network hardware automatically takes on the job of routing the message to the respective destinations. The local communication controller filters all messages that don't match the subject out of the message stream. Hence, the subject filtering does not put any burden to the embedded computational component of a smart sensor or actuator. The details of this approach, particularly for a CAN-Bus have first been presented in [13] and later extended in [12].

2.2 Real-time event channels

According to the need in most real-time systems, event channels with different timeliness and reliability properties should be supported. Therefore, we distinguish three event channel classes: hard real-time event channels (HRTEC), soft real-time event channels (SRTEC) and non real-time event channels (NRTEC). A HRTEC offers rigorous guarantees for discrete control based on sporadic events as well as for continuous control requiring periodic events like sensor readings and control feedback. For sporadic events a maximum latency will be guaranteed while for periodic events the goal is to achieve a low period- and latency-jitter. The guarantees are maintained under an anticipated number of network failures. Events published to a SRTEC are scheduled according to the earliest deadline first (EDF) algorithm. As outlined below, deadlines may be missed in situation of transient overload or due to the arbitrary arrival times of messages. Finally, a NRTEC disseminates events that have no timeliness requirements.

The transport of events through a hard real-time event channel (HRTEC) is synchronous and reliable. The properties of a HRTEC are defined by: 1.) a known upper bound for the transport latency, i.e. the interval between the point in time when an event message becomes ready and its delivery; 2.) a known upper bound for the latency jitter, i.e. the variance of the transport latency; 3.) a known upper bound of the period jitter for periodic events, i.e. the variance on the period; 4.) a fault assumption under which the properties 1.) - 3.) are valid. In order to offer such properties, a

[1]Note that for a soft real-time event, the deadline may be missed. In this case the expiration time defines the interval after which the event may be dropped entirely.

HRTEC transparently handles redundant transmissions of events and guarantees that the respective publisher has a privileged access to the communication network. Access is based on the reservation of network resources according to a TDMA mechanism (TDMA: Time Division Multiple Access) similar to the time-triggered protocol [14]. It means that events published to the HRTEC have to be ready at the start of the assigned TDMA time-slot. As explained below, in contrast to most TDMA schemes, we exploit the specific priority mechanisms of the CAN-Bus to enforce the temporal guarantees in a more flexible way.

A SRTEC has timeliness requirements which are expressed by deadlines and validity intervals (expiration time). Different from HRTECs, SRTECs do not use reservations. Soft real-time event messages become ready at any time and are scheduled according to their transmission deadlines by an earliest deadline first (EDF) algorithm. The ***transmission deadline*** is defined as the latest point in time when a message has to be transmitted. As described later, the priority mechanism of CAN-Bus is exploited for this purpose. However, because a message can not be interrupted during its transmission and messages may become ready at arbitrary points in time, EDF will not always take the right scheduling decisions (only a clairvoyant scheduler would be able to do so) and situations of temporal conflicts and transient overload may occur. In theses situations, messages will still be transmitted at a later time in a best effort manner. An SRT event message eventually will be discarded if its transmission time is delayed beyond its temporal validity specified by the ***expiration time***. The expiration time is an application specific parameter, which may be defined according to some value function [11].

NRTECs are used for events that do not have timeliness requirements. They are primarily intended for configuration and maintenance purposes. While HRTEC and SRTEC disseminate events of restricted length to meet the responsiveness requirements of real-time systems, NRTEC may transfer bulk data in a sequence of message fragments.

Subsequently, we will describe the application-programming interface of the different event channel types in more detail.

2.2.1 Hard real-time event channels

The transfer of events through a HRTEC is certain, i.e. all the necessary resources are reserved to transmit event messages timely under specified fault assumptions. The API for a HRTEC is presented in fig. 1. HRTECs need to set up the infrastructure, before communication will be possible with the required guarantees. This is initiated by an application through calling the method: *channel.announce(subject, attribute_list, exception_handler);*

The *announce()* method enables the local middleware components to set up the data structures representing the respective event channel and performing the binding of the event channel subject to a network address. Three arguments are specified for the method: The *subject*, represented by the unique identifier of the event channel, the *attribute_list*, and an exception handler. The *attribute_list* describes the specific attributes of the channel, e.g. whether the publication is periodic or aperiodic, reliability requirements and data rates. This information is used to allocate and reserve the respective resources. For a hard real-time channel it is not common to provide exception handling because it is based on fault masking and worst-case assumptions about temporal properties. However, it should be noted that in a distributed system, local exception handling may contribute to an early detection of a fault and thus may increase the safety of the system. The lower levels of the communication system may detect a failure, which cannot be handled by the fault masking mechanism, and propagate this information through the middleware to the respective subscribers of a channel.

```
class hrtec {

private:
subject subject_uid;

public:
// constructor and destructor of the class
hrtec(void);
~hrtec(void);
// methods used for publishing
int announce(subject, attribute_list, exception_handler);
int publish(event);
// methods used for subscribing
int subscribe(subject, attribute_list, event_queue, not_handler, exception_handler);
int cancelSubscription(void);
}
```

Figure 1. Declaration of a HRTEC class in c++

When the HRTEC is established, the application can publish events to the channel using the method: *channel.publish(event);*

The *subscribe()* method establishes the necessary channel data structures and creates the binding of the subject to a network address. It corresponds to the *announce()* method for publishers. The *attribute_list* specifies a list of attributes used for allocating the respective resources and for filtering. For instance, we generally assume that publishers and subscribers are connected by a channel which spans multiple networks, e.g. a field bus, a wireless network and a wired wide area network[2]. In such a scenario, a subscriber may be interested in receiving events only from publishers in the same network, i.e. those connected to the same field bus. In such a case, the respective attribute can be set

[2]An example is described in [12].

accordingly and any event, which has been generated outside the field bus, will be filtered out and will not trigger a local event notification. It should be noted, however, that the HRT-channels are statically assigned to time-slots and have predefined temporal and reliability attributes. Therefore, the filtering is usually less important for this channel class because only a particular publisher is allowed to publish in a certain time-slot (see section 3). The subscriber to such a channel is thus always aware which entity is expected to transmit. The known time of transmission itself therefore will be exploited as a filter for a HRT-channel.

Because events can be aperiodic, the event notification service of the middleware provides an asynchronous notification mechanism for applications. When an event has passed the filters, the middleware stores the event in some predefined memory area and calls the application's notification handler *not_handler*. The notification handler comprises application code that is executed when an event is received. Thus, the notification handler retrieves the event from memory using the *getEvent()* primitive and then proceeds performing the respective operations. As for the publisher of a HRTEC, an exception handler is also specified for the subscriber. Because a HRTEC is based on reservations, the time when a message is expected is known and thus, the event channel handler on the subscriber side can detect a missing message.

Finally, the *cancelSubscription()* method removes a subscription. Note that a cancel subscription is a strictly local operation and releases the resources in the local event handler. Only subscribers can dynamically cancel subscription to a HRTEC.

2.2.2 Soft real-time event channels

SRTECs do not use reservations. In SRTECs transmission deadlines are used to dynamically schedule the event traffic. Fig. 2 depicts the declaration of the SRTEC class. Although the structure looks similar to the HRTEC, the differences are substantial and primarily are substantiated in the different attributes defined for SRTECs. Events published to a SRTEC specify a transmission deadline and an expiration parameter in the attribute list of the event. As already discussed, events are scheduled by the EDF algorithm which may lead to missed deadlines because of the non-preemptive nature of the message transmission and because of overload situations. This situation requires notifying the application for awareness reasons. Two exceptional situations may occur: a missed deadline and an expired validity. In both cases, the local exception handler is called. This local notification allows the application to react and adapt to such situations. When the validity interval is expired, the event is completely removed from the local send queue.

Section 3 provides further details about the realisation on the specific CAN network infrastructure.

```
class srtec {

private:
subject subject_uid;

public:
// constructor and destructor of the class
srtec(void);
~srtec(void);
// methods used for publishing
int announce(subject, attribute_list, exception_handler);
int cancelPublication();
int publish(event);
// methods used for subscribing
int subscribe(subject, attribute_list, event_queue, not_handler, exception_handler);
int cancelSubscription(void);
}
```

Figure 2. Declaration of a SRTEC class in c++

2.2.3 Non real-time event channels

Non real-time event channels are used for events that do not have timeliness requirements. A NRTEC has a fixed priority. The priority is specified by the application during the announcement of the channel. However, as further discussed in the next chapter, only priorities within a predefined range are accepted by the middleware.

NRT-channels are particularly used to configure and maintain the smart networked devices of the system. This may require to send a considerable amount of data over the network, like memory images, electronic data sheets, or test patterns. Because message frames on the CAN-Bus are limited to a payload of 8 data bytes, a mechanism to chain individual CAN messages to a larger application specific message is needed. Such a "fragmentation" mechanism for NRT channels which publishes long event messages in multiple fragments is provided by the middleware. Fragmentation is an inherent attribute of a NRT-channel and therefore, on the publisher side, fragmentation is defined during the announcement of the event channel as an entry in the *attribute_list*.

3 Event channels on a CAN-Bus network

We now present a mapping of the described abstractions to a field bus network. We will first describe the reservation scheme for the HRTECs which is similar to a scheme used in time-triggered protocols like TTP [14], TTP/A [15], and TTCAN [7]. However, a substantial advantage over a TDMA scheme is that due to CAN-Bus properties, bandwidth which was reserved but is not needed by a HRTEC can be used by less critical traffic. Because of the conservative worst-case assumptions of a HRTEC, this can be a large share of the overall bandwidth. For SRTECs the priority-based message dispatching of the CAN-Bus is exploited to realize an EDF-based scheduling of messages. Constraints on the priority level of SRTECs prevent any interference with reserved HRTECs. Finally, the low fixed priorities of NRTECs enable the use of any free bandwidth for non-critical bulk data transfer.

3.1 The reservation scheme

Hard real-time communication is organized in rounds. A round is divided into time slots that are assigned to HRTECs. A round specifies the cycle in which the schedule of the communication medium is repeated. The data structure which stores the schedule of a round is called a calendar and corresponds to the Round Descriptor List (RODL) in the TTP protocol [14]. The intention of the reservation-based scheme is to avoid collisions by statically planning the transmission schedule. Multiple slots may be reserved for an individual node within a round. The correctness of the reservations regarding timing conflicts and temporal overlap are checked by an admission test. We assume that this is done before any new reservation is confirmed and that the reservations are made off-line.

The event channel approach of the P/S protocol leaves open which publisher provides the respective information to a subscriber. However, when defining a HRT event channel, the slot reservation has to be done according to a specific node, which is allowed to exactly send a message within this time interval. Hence, if multiple publishers provide input to the same channel, multiple slots have to be reserved. As it becomes clear later, the specific handling of reserved slots on the CAN-Bus allows a flexible treatment and reuse of reserved slots. Thus, if a node has already successfully published a certain event in a hard real-time channel, subsequent publications of the same subject can possibly be suppressed and the slot can be used by less critical traffic.

3.2 Structure of the time-slots

Hard real-time messages use time-slot reservations. This mechanism is based on a global time for which we currently adopted a standard solution [9][3]. Because we must prevent any temporal overlap between adjacent hard real-time slots, a minimal gap ΔG_{min} has to be allocated between the slots. This gap depends on the quality and frequency of clock synchronization and is conservatively assumed at 40μs [17]. The length of the time-slot is defined by the worst-case transmission time of a HRT message. The worst-case transmission time considers the length of a message and includes assumptions about the failure modes during message transmission. We use time redundancy and forward error recovery to cope with network omission faults and temporary node faults. An analysis of the worst-case transmission times under fault assumptions is published in [16]. To deliver a message at its predefined deadline, the transmission has to be launched at the Latest Start Time (LST) (see fig. 3). We exploit the CAN priority mechanism to enforce that a HRT message will definitely be sent at this point in time. The highest priority is exclusively reserved for HRT messages. When the LST is reached this priority is assigned to the message. The priority mechanism of the CAN-Bus guarantees that this message will win the bus arbitration. Thus, our scheme exploits the global time to avoid any conflicts between HRT messages. The priority mechanism of the CAN-Bus enforces the reservations and omits any interference with less critical messages. To guarantee that SRT- and NRT-messages do not interfere with HRT-messages, we additionally have to consider that an ongoing message transfer cannot be preempted. It may happen that messages (SRT or NRT), which can be sent at any time, will be started just before the HRT-message becomes ready, i.e. at the LST. In this case, it would steel reserved time from the HRT-message and jeopardize HRT guarantees. Therefore, the slot for HRT has to be extended and HRT-messages must be ready for transmission at the latest ready time (see fig. 3) which is: *LST* - ΔT_{wait}. The time ΔT_{wait} corresponds to the transmission time of the longest CAN-message (154 μsec at 1Mbit/sec). The actual successful transmission of the HRT-message can then occur at any time inside the time-slot which introduces the problem of jitter. To avoid the jitter for the application, HRT messages are always delivered by the middleware at the predefined transmission deadline.

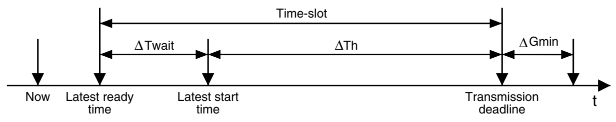

Figure 3. Structure of a time-slot

Thus, our reservation mechanism differs from other time-triggered approaches in many substantial points. Firstly, we exploit time and the priority mechanism to enforce HRT guarantees. Time is used to separate HRT-messages which all have the same maximum priority in the system. This guarantees that they are sent when they become ready. However, the CAN-Bus allows to determine, without any additional overhead, whether all operational nodes including the sending node have received a message successfully. Observing that we included time redundancy to tolerate transmission faults, this redundancy is not needed any more if all nodes have correctly received the message. Therefore, the sending node will stop to further transmit the message in this case. Now, if there are pending SRT- or NRT-messages, the priority mechanism of CAN will automatically schedule the message with the highest priority for transmission. Thus, time redundancy only costs bandwidth if faults really occur, which, compared to the overall traffic, may be relatively rare. Therefore, very conservative fault assumptions are possible because the penalty is low in the average. This is not possible in schemes which only use global time to enforce reservations.

Secondly, most TDMA schemes as TTP [14] and TTCAN [7] enforce the constraints on jitter on the network level by sophisticated and expensive mechanisms to guarantee start times of messages. In the time-triggered CAN-Bus protocol (TTCAN [7]) there are extra slots to prevent that an ongoing message transfer interferes with a reserved slot, thereby wasting additional bandwidth. In our protocol, jitter is prevented by defining a precise delivery deadline of a message and the event channel handler which has access to the global clock assures that the message is delivered at that point. Thus, jitter is handled on the middleware layer rather than on the network layer. It also should be noted that a HRT-message will be correctly received at some time within the slot, which cannot be predicted because faults may occur or not. In TTP and TTCAN, messages are just resend up to a certain number of times which corresponds to a specified omission degree. This of course fills up the reserved slot and avoids jitter but for the price of valuable bandwidth. To reuse the remaining slot time means that a message transfer is successfully completed within the slot and the delivery has to be handled independently of the network at a higher system layer.

3.3 Scheduling soft real-time- and non-real-time-messages

One of the most important properties of the CAN-Bus is its priority-based distributed arbitration mechanism. This mechanism is exploited to schedule SRT messages according to the EDF scheme. NRT messages use a low fixed priority. We use an 8-bit explicit priority field in the CAN message identifier which results in 256 priority levels. As described in 3.2, HRT-messages reserve the highest priority 0. Scheduling SRT-messages requires a range of priorities which reflect the deadlines in time. This is outlined in 3.4. NRT messages are assigned to fixed low priorities. The relation between the priorities of HRT, SRT and NRT messages can be expressed by the relation[3]: $0 \leq P_{HRT} < P_{SRT} < P_{NRT}$. The assignment avoids that NRT and SRT messages ever gain access to the bus against any pending HRT message. The middleware rigorously has to enforce the above relation.

[3]On CAN-Bus the highest priority corresponds to the lowest binary value.

3.4 Soft real-time messages

SRT messages are scheduled according to an EDF scheme which means that the assigned priorities reflect the deadline order of message transmissions. Although there is some overhead related to the use of a dynamic scheduling scheme as EDF, the choice is motivated by the following consideration that there is a substantial share of aperiodic and sporadic traffic in the system which can not adequately be mapped to static priorities. We assume that we have 250 priority levels for SRT messages (this can be flexibly determined by application needs. In this example we define a single priority level to HRT-messages and 5 levels to NRT-messages). These 250 priority levels have to be mapped on a time scale to express the temporal distance of a deadline. The closer the deadline, the higher is the priority. Mapping deadlines to priorities will cause two problems. The first problem is that static priorities cannot express the properties of a deadline, i.e. a point in time. A priority corresponding to a deadline can only reflect this deadline in a static set of messages. When time proceeds and new messages become ready, a fixed priority mechanism cannot implement the deadline order any more. It is necessary to increase the priorities of a message when time approaches the deadline, i.e. with decreasing laxity. Therefore, the priority of a message is dynamically increased with a granularity of Δt_p which defines the length of a priority slot. A SRT message will reach its highest possible priority at its transmission deadline. The dynamic increase of the message priority causes an overhead which is evaluated in [16]. Secondly, there is a trade-off between the length of a priority slot and the quality of the derived schedule. When mapping a transmission deadline for a message to a priority slot, two deadlines which are close may be mapped to the same slot and thus, to the same priority. The order between the deadlines is then arbitrary determined by the other fields of the CAN identifier. This would motivate a very small slot length, which decreases the probability of equal priorities. However, this raises the problem of a tight time horizon. The time horizon is given by $\Delta H = (P_{max} - P_{min}) * \Delta t_p$ (Δt_p defines the length of a priority slot) [17]. Any deadlines, which are beyond this value are mapped to the same priority and thus may be scheduled incorrectly. The described trade-off is particularly a problem when the number of priority levels is low (e.g. as in [23]). Considering the 250 priority slots provided in our scheme and a priority slot length of approximately one CAN-message, we can accommodate 250 message transfers within the time horizon. Because the number of nodes connected to a CAN-Bus is usually in the range of 32 to 64, the time horizon seems to be sufficiently large. A decision about the trade-off has also to consider specific application requirements.

3.5 Structuring the CAN message identifier

The transport mechanism for CAN-Bus uses the basic communication mechanisms described in [13] to support different event channel classes on the CAN-Bus. An event channel class (HRTEC, SRTEC, NRTEC) is mapped to the respective message class on the CAN-Bus. The CAN 2.0B standard with 29-bit identifiers is used to express the priority of the message as well as to identify the channel. The identifier is structured into multiple fields to represent the priority and the subject of an event channel. It is often argued that a long CAN-ID is a waste of bandwidth. These arguments do not recognize that the CAN-ID is not an ID in a conventional sense but a message tag which comprises useful information which otherwise has to be moved to the payload of the frame. As correctly observed in [8] the CAN-IDs can be a message of its own.

On the bus level the CAN message identifier is used to identify the event channel (subject) and to express the priority of the event message. The priority of the message is expressed in an 8-bit priority field providing 256 priority levels. A 7-bit field *TxNode* is used to ensure the uniqueness of the CAN-ID. This is a requirement of the CAN-Bus protocol [2] because after the arbitration phase, a consistent decision has to be derived which node is allowed to use the bus. The *TxNode* is dynamically assigned during the configuration phase [12]. The remaining 14 bits of the identifier are exploited as the *etag* field representing the subject of the event channel. Whenever a new channel is created in a node (by an announce or a subscribe), the binding protocol assigns an *etag* to the respective unique subject identifier of the channel. A detailed description of both, the configuration and the binding protocols, can be found in [13].

4 Comparison with related higher level CAN protocols

On the CAN-Bus [2], schemes using message priority and time have been explored to achieve predictability of communication. Standard protocols based on CAN-Bus such like CanOpen [3], SDS [10], and DeviceNet [19] are based on fixed priority schemes which are not able to reflect temporal requirements. To overcome this problem, the deadline-monotonic priority assignment [22] uses an off-line feasibility test to meet the deadlines of periodic and sporadic messages. However, it supports only static systems and does not distinguish hard and soft deadlines. More flexible protocols like the dual priority scheme [4] or the schemes proposed in [5], [23] support the use of both hard and soft real-time communication on the CAN-Bus. However, they have the disadvantage of providing a time horizon that is too short.

There are a couple of recent protocols that uses time-slot reservation on CAN-Bus, like TTCAN [7] and FTT-CAN [1]. As already mentioned, TTCAN does not fully exploit the CAN-Bus facilities and therefore, uses bandwidth less efficiently. Secondly, non real-time messages can only be sent in the respective contention slots. This is less flexible compared to our approach. FTT-CAN is more intended for industrial automation applications and exploits the master slave mechanism to dynamically define the schedule, in principle for every round. However, both protocols are based on a master-slave mechanism which we wanted to avoid in our system because the master constitutes a single point of failure. The TTP/A protocol [15] aims to the same application area as TTCAN and uses similar mechanisms. Here, the master always initiates the communication with the slaves sending their own messages in a predefined manner.

5 Concluding remarks and future work

In the paper we presented an event channel model that supports functional and non-functional attributes. Three types of channels are provided for applications: HRTEC, SRTEC, and NRTEC. The concept of event channels represents a high level programming interface for real-time communication. It provides the abstractions to ease the programming and enables the programmer to reason about non-functional attributes like temporal and reliability characteristics without being involved in low level network details. Communication via HRT channels is certain, i.e. it guarantees timely delivery under the specified fault assumption. SRT channels constitute a versatile and flexible way to express deadlines as timing constraints but provide awareness if the constraint is violated. It also allows specifying an expiration attribute, which is used to discard an event when the validity has gone to zero. Again, the application will be notified to enable corrective application related actions. Finally, NRT channels are established to transport non real-time traffic. Here, we allow arbitrary long messages to communicate configuration and maintenance data like ROM-images or electronic data sheets, describing a device.

The paper also shows how these channel abstractions are represented on a CAN-Bus, which is a popular field-bus in the area of industrial automation and particularly in the automotive area. On the CAN-Bus, the channel classes are directly mapped to respective message classes. The protocol for the CAN-Bus provides three types of messages: hard real-time, soft real-time, and non real-time. Hard real-time messages use a reservation scheme to avoid any conflicting requirements between them. The slot reservation mechanism also allows using time redundancy to tolerate omissions failures. The priority mechanism of CAN is exploited to enforce the bus access in the asserted time-slot. For less stringent timeliness and reliability requirements, SRT messages are provided. In this type of communication deadlines are considered and flexible mechanisms to handle deadline violations are introduced. Finally, messages that do not have timeliness requirements will exploit the basic priority and fault handling mechanism of the CAN-Bus.

An additional advantage of our protocol compared to other time-triggered schemes is a better utilization of bandwidth. Firstly, when a reserved slot is not used, the priority mechanism of CAN will automatically assign this slot to some other (lower priority) message. This may occur when a sporadic HRT message has a reservation but is not sent. Secondly, the CAN-Bus provides a unique technique to determine when all nodes correctly receive a message. This property is exploited to reduce the overhead of time redundant message transfer. If all receivers have received the message correctly, the sender can detect this and skip the redundant message transmission.

A first prototype of the P/S protocol is available. Presently, it provides the API for a single event channel class without non-functional attributes. The middleware has a low memory footprint, which enables the use on various 16- and even 8-Bit micro controllers. A Linux and an RT-Linux version have also been realized. Future work includes the integration of the different event channel classes presented in this paper.

References

[1] L. Almeida, J. Fonseca, and P. Fonseca. Flexible time-triggered communication on a controller area network. In *Proc. of the Work in Progress Session of the 19th IEEE Real-Time Systems Symposium*, 1998.

[2] Bosch. CAN specification version 2.0. Published by Robert Bosch GmbH, September 1991.

[3] CiA. Canopen communication profile for industrial systems. cia draft standard 301 version 3.0. Published by CAN in Automation.

[4] R. Davis. Dual priority scheduling: A means of providing flexibility in hard real-time systems. Technical Report YCS230, University of York, UK, May 1994.

[5] C. Eriksson, H. Thane, and M. Gustafsson. A communication protocol for hard and soft real-time systems. In *Proceedings of the EURWRTS'96*, 1996.

[6] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. Technical Report DSC ID:200104, EPFL, Lausanne, Switzerland, 2001.

[7] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther. Time triggered communication on CAN (time triggered can-TTCAN). In *7th international CAN Conference*, 2000.

[8] L.-B. Fredriksson. CAN for critical embedded automotive networks. *IEEE Micro*, 22(4), July/August 2002.

[9] M. Gergeleit and H. Streich. Implementing a distributed high-resolution real-time clock using the can-bus. In *1st International CAN-Conference*, Mainz, Germany, September 1994.

[10] Honeywell. Smart distributed systems, application layer protocol version 2, micro switch specification gs 052 103 issue 3. Published by Honeywell Inc., 1996.

[11] E. Jensen and J. Nothcuttn. Alpha: A non-proprietary os for large, complex distributed real-time systems. In *Proc. of the IEEE Workshop on Experimental Distributed Systems*, pages 35–41, Alabama, USA, 1990.

[12] J. Kaiser and C. Brudna. A publisher/subscriber architecture supporting interoperability of the CAN-Bus and the internet. In *2002 IEEE International Workshop on Factory Communication Systems (WFCS2002)*, Västeras, Sweden, August 2002.

[13] J. Kaiser and M. Mock. Implementing the real-time publisher/subscriber model on the controller area network (CAN). In *2nd International Symposium on Object Oriented Distributed Real-Time Computing Systems*, San Malo, France, May 1999.

[14] H. Kopetz and G. Grünsteidl. TTP - a time-triggered protocol for fault-tolerant real-time systems. Technical Report 12/92, Inst. für Techn. Informatik, Techn. University of Vienna, 1992.

[15] H. Kopetz, M. Holzmann, and W. Elmenreich. A universal smart transducer interface: TTP/A. *International Journal of Computer System, Science & Engineering*, 16(2), March 2001.

[16] M. Livani and J. Kaiser. Evaluation of a hybrid real-time bus scheduling mechanism for CAN. In *7th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS´99)*, San Juan, Puerto Rico, April 1999.

[17] M. Livani, J. Kaiser, and W. Jia. Scheduling hard and soft real-time communication in the controller area network. *Control Engineering Practice*, 7(12):1515–1523, December 1999.

[18] R. Meier and V. Cahill. Taxonomy of distributed event-based programming systems. Technical Report TCD-CS-2002, Dept. of Computer Science, Trinity College Dublin, Ireland, 2002.

[19] D. Noonen, S. Siegel, and P. Molaney. DeviceNet application protocol. In *1st. International CAN Conference*, 1994.

[20] B. Oki, M. Pfluegl, A. Seigel, and D. Skeen. The information bus®- an architecture for extensible distributed systems. In *14th ACM Symposium on Operating System Principles*, Asheville, NC, December 1993.

[21] R. Rajkumar, M. Gagliardi, and L. Sha. The real-time publisher/subscriber inter-process communication model for distributed real-time systems: Design and implementation. In *IEEE Real-time Technology and Applications Symposium*, June 1995.

[22] K. Tindell and A. Burns. Guaranteeing message latencies on controller area network (CAN). In *1st International CAN Conference*, 1994.

[23] K. Zuberi and K. Shin. Scheduling messages on controller area network for real-time CIM applications. *IEEE Transactions On Robotics And Automation*, 13(2):310–314, April 1997.