Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Systems and Software 77 (2005) 27–36

The Journal of Systems and Software

www.elsevier.com/locate/jss

COSMIC: A real-time event-based middleware for the CAN-bus

Jörg Kaiser *, Cristiano Brudna, Carlos Mitidieri

Department of Computer Structures, University of Ulm, Ulm, Germany

Received 8 November 2002; received in revised form 1 September 2003; accepted 12 December 2003
Available online 11 September 2004

Abstract

The paper describes the event model and the architecture of the COSMIC (COoperating SMart devICes) middleware. Based on the assumption of tiny smart sensors and actuators, COSMIC supports a distributed system of cooperating autonomous devices. COSMIC considers quality of service requirements in the event model and provides an application interface which allows to express the respective temporal and reliability attributes on a high, application related abstraction level. According to the need in most real-time systems, COSMIC supports 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 the layered COSMIC architecture to map the different channel classes to the CAN-Bus.
© 2004 Elsevier Inc. All rights reserved.

Keywords: Middleware; Cooperative control; Real-time event channel; Message scheduling; CAN-Bus

1. Introduction

Many control applications are based on distributed devices controlling a physical process. Sensors and actuators connected by a network observe and evaluate aspects of the physical environment and act in response to it. At a first glance, this is not new because distributed sensors and actuators connected by field busses are common in industrial automation since years. However, although sensing and acting is physically distributed in these systems, the logic control of the process still is centralized. The sensors are polled periodically by a central instance, which then calculates new values and set points and eventually disseminates them to dedicated actuators. This is reflected in many respective application level protocols in the field bus area, e.g. CiA (1999), Profibus (1999), Noonen et al. (1994) and SDS (1996).

New applications and recent technological developments motivate a new system and control architecture. From a technological point of view, sensors and actuators comprise special purpose hardware (sometimes also mechanical elements) for signal reception and conditioning together with computational elements and a network interface. The smart nodes therefore constitute autonomous entities which allow to capture sensor data at the real-world interface, transform it to a machine readable form and spontaneously disseminate it as a standardized message to the distributed system. Moini (1997) makes the point that "smart sensors are information sensors, not transducers and signal processing elements". Recently, standardization efforts are expended for such systems by OMG (2001) and by IEEE (Bryzek, 1996). The most interesting and challenging property of

* Corresponding author.
E-mail addresses: kaiser@informatik.uni-ulm.de (J. Kaiser), cristiano.brudna@informatik.uni-ulm.de (C. Brudna), carlos.mitidieri@informatik.uni-ulm.de (C. Mitidieri).

these intelligent devices is their ability to spontaneously interact with the overall system. This enables a modular system architecture to control physical processes reactively without the need of a central co-ordination facility. In such a system, multiple different sensors will co-operate to augment perception of the environment and autonomous actuators will co-ordinate actions to increase speed, power and quality of actuation thus forming decentralized sensor and actuator networks.

To ease the interaction of cooperating objects, the system should provide an adequate level of abstraction. The application designer should not be forced to deal with the low-level message passing of the raw field bus. The communication model should reflect the requirements of a control system composed from autonomous active components. This demands for the spontaneous generation and dissemination of events when they are detected at the sensor interface or triggered by internal state changes. A many-to-many communication mechanism is needed to efficiently disseminate sensor data and control information because many entities may be interested in the data supplied by a sensor and multiple sources may disseminate similar information. Finally, to maintain autonomy of components and reduce the side effects of communication, the data transfer should not be coupled with a transfer of control, as it is the general case in client–server interactions (Pereira et al., 2001).

COSMIC supports an event-based communication model. The term "event-based" in this context does not refer to any specific model of synchrony. Events are typed information carriers. They may be spontaneously generated and immediately disseminated triggered by an occurrence in the environment or the system itself. Alternatively, an event may be triggered periodically by a clock and contain the current state of a variable or a previous change of state. Events in COSMIC are disseminated in a publisher/subscriber style. Publisher/subscriber protocols are well known to support spontaneous, many-to-many communication relations and reflect autonomy of communicating entities (Rajkumar et al., 1995; Eugster et al., 2001b; Kaiser and Mock, 1999). COSMIC extends the publisher/subscriber scheme by integrating a real-time event model which reflects that events disseminated in the system may represent real-world events requiring real-time dissemination in the control system. An event is characterized by a context of occurrence which at least is defined by a location and a point in time. Additionally, because the event may represent a time/value entity for which the value changes over time, quality properties which are related to the temporal validity have to be assigned. The context and quality attributes (sometimes called "functional" and "non-functional") are complementary to the event data. To control the quality of dissemination we introduce the concept of event channels. An event channel is an abstraction of the communication mechanisms. Events can be published to an event channel which then ensures the required dissemination quality.

The paper starts with introducing the notions of events and event channels in Section 2. Section 3 describes the layered COSMIC architecture. Related work is reviewed briefly in Section 4 and conclusions and future work are summarized in Section 5.

2. Events and event channels

An event may occur in the physical environment or in the computing system. Hence, both, an observation of a real-time entity and a state transition of a variable are uniformly characterized, represented and disseminated as an event. Smart sensors substantially support this view on the system because they encapsulate all low-level computations. On the respective abstraction level they are represented as active objects, taking the roles as publishers and subscribers producing or consuming events, respectively. An architecture which describes this model of "generic events" has first been presented by Veríssimo and Casimiro (2003) in the context of CORTEX (Veríssimo et al., 2002). The event layer represents the only means of information dissemination and hides the network as well as the transformation process of the I/O subsystems. Because all events appear at the event layer, it is possible to treat events consistently whether they are generated in the environment or in the system. The event and event channel model introduced by COSMIC constitute a specific way to realize such an architecture.

Different from simple messages, an event is a typed information carrier. In COSMIC it includes the context in which an event has been generated and quality attributes. An event instance is specified by a tuple comprising a subject, attributes and contents:

$$\begin{aligned} Event := \langle & subject, context\_attribute\_list, \\ & quality\_attribute\_list, data \rangle \end{aligned}$$

A subject defines a type of event and thus is related to the event contents. The pro and cons of subject-based addressing are discussed in detail elsewhere (Oki et al., 1993; Eugster et al., 2001a; Pereira et al., 2001) and are not reviewed again in this paper. In our system, a subject is represented by a unique identifier. An event is further characterized by a set of attributes. The attributes related to context comprise e.g. a location, the time of occurrence or a certain network zone. The non-functional attributes include quality aspects as a validity interval (expiration time), a deadline and a tolerated omission degree. Finally, the data comprise event's contents specified by the application.

Events are propagated from publishers to subscribers through event channels. We introduce event channels as abstractions of network resources and therefore assign QoS related properties attributes of these resources. An event channel is an instance of an event channel type characterized by a subject, quality attributes and handlers which are invoked to notify an object or handle exceptional situations:

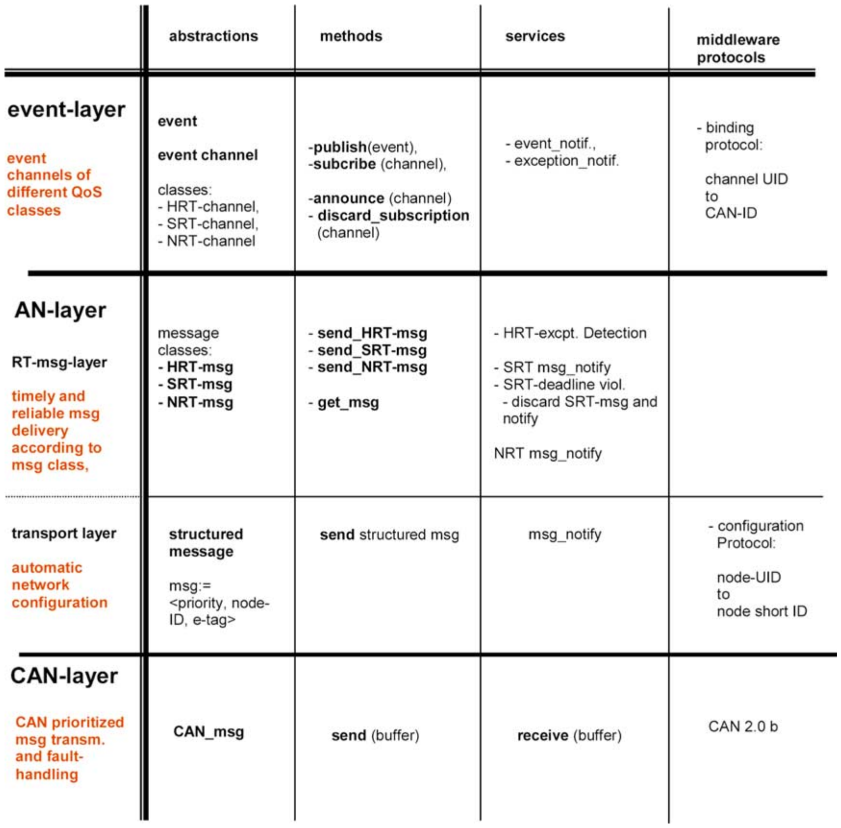
$$EventChannel := \langle subject, attribute\_list, handlers \rangle$$

An event channel exclusively disseminates events that are compatible with its respective subject. As attributes, an event channel may include e.g. a latency, dissemination constraints and reliability parameters. A data structure representing an event channel is dynamically created in the local middleware whenever a publisher first announces a publication or a subscriber subscribes to a channel. According to the need in most real-time systems, COSMIC supports event channels with different timeliness and reliability properties. 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 delivery guarantees 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.

3. The COSMIC architecture

Implementing the event model requires to map the abstractions of that model (publisher, subscriber, event type, event instance) to the elements provided by the technical infrastructure of the system such as objects, messages and addresses. More precisely, we can identify publishers and subscribers with application objects, and event instances with messages that are sent over the CAN-Bus (Bosch GmbH, 1991). The respective functionality to perform these mapping in COSMIC is encapsulated in the Event Channel Handler which resides in every node. The ECH provides the interface for the application and maintains all the data structures to enable the event-based interaction.

Fig. 1 presents the abstraction layers which are handled by the ECH for the CAN-Bus, a standard field bus developed for the automotive industry (Bosch GmbH, 1991). It depicts the abstractions which are available at the layers, i.e. the objects which are provided by the layer, the methods which can be applied to these objects, the services which are provided, and the specific protocols. On the lowest level, we assume a



	abstractions	methods	services	middleware protocols
event-layer event channels of different QoS classes	**event** **event channel** classes: - HRT-channel, - SRT-channel, - NRT-channel	-**publish**(event), -**subcribe** (channel), -**announce** (channel) - **discard_subscription** (channel)	- event_notif., - exception_notif.	- binding protocol: channel UID to CAN-ID
AN-layer RT-msg-layer timely and reliable msg delivery according to msg class,	message classes: - **HRT-msg** - **SRT-msg** - **NRT-msg**	- **send_HRT-msg** - **send_SRT-msg** - **send_NRT-msg** - **get_msg**	- HRT-excpt. Detection - SRT msg_notify - SRT-deadline viol. - discard SRT-msg and notify NRT msg_notify	
transport layer automatic network configuration	**structured message** msg:= <priority, node-ID, e-tag>	**send** structured msg	msg_notify	- configuration Protocol: node-UID to node short ID
CAN-layer CAN prioritized msg transm. and fault-handling	**CAN_msg**	**send** (buffer)	**receive** (buffer)	CAN 2.0 b

Fig. 1. Abstractions layers of COSMIC.

standard CAN-Bus. CAN is a broadcast bus which provides message identifiers to characterize the contents of a message rather than a source or destination address. For a detailed discussion of the CAN features the reader is referred to the CAN standard specification CAN 2.0 (Bosch GmbH, 1991), Rufino et al. (1998), Fredriksson (2002) or Livani and Kaiser (1999). The CAN message identifier is exploited for a prioritized bus arbitration effectively providing a global distributed priority-based message dispatching for all messages which are ready to be sent. The CAN standard 2.0 B defines a 29-bit identifier. It is often argued that the 29-bit identifier is a substantial overhead because a CAN message body may contain at most 64 bits of payload and therefore the CAN standard 2.0 A is preferable which only needs 11-bit IDs. However, the small size of the payload is also an argument for the extended format because useful information can be transferred to the ID and therefore saves rare space in the message body (Fredriksson, 2002). As one of the consequences of a short identifier it is impossible in most application level protocols for CAN to assign individual priorities to messages. Instead, only a fixed and very limited priority assignment of message classes can be achieved (see e.g. CiA, 1999; Noonen et al., 1994; SDS, 1996). We will exploit the long identifier on the abstract network layer for an explicit priority control and to identify nodes and events. The abstract network (AN-) layer is subdivided in two levels. The lower level defines the structure of the message identifier.

The 29-bit identifier is subdivided in (1) an 8-bit priority field, (2) a unique 7-bit node identifier field (node-ID) and (3) in a 14-bit event tag (e-tag). The priority field allows 256 priority levels which are exploited in the RT-msg-layer to realize the different real-time message classes. The node-ID-field is used to solve the problem of unique message identifiers because equal CAN message identifiers would result in an arbitration conflict which cannot be resolved. To support the dynamic deployment of smart components, a configuration protocol dynamically assigns this node-ID when a node is attached to the CAN network or during the start-up configuration phase. Every node has a 64-bit long identifier assigned at production time and stored in non-volatile memory. The configuration protocol converts this long node UID in a temporary 7-bit node-ID. The details of the configuration protocol which is functionally similar to configuration protocol in Can Application Layer protocol CiA (1993) can be found in Kaiser and Mock (1999).

The RT-msg-layer is responsible for the real-time properties of message dissemination. It is built up on the structured message identifier and provides messages with different real-time properties. In this layer, the 8-bit priority field of the structured CAN message identifier is evaluated to schedule the message on the medium. COSMIC provides three different real-time classes, hard real-time, soft real-time and non-real-time.

3.1. Scheduling hard real-time messages

Hard real-time messages have delivery guarantees and use reserved time slots in a TDMA (time division multiple access) scheme. Reservations are organized in rounds. This is similar to the time-triggered protocols like TTP (Kopetz and Grünsteidl, 1992; Kopetz et al., 2001; Führer et al., 2000). A round specifies the cycle in which the schedule of the communication medium is repeated. The intention of the reservation-based scheme is to avoid collisions by statically planning the transmission schedule. 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. Hence any conflict between hard real-time messages is avoided. Reservation based system need a global time base to determine when a respective message slot has arrived. Global time in COSMIC is based on synchronized clocks according to the algorithm proposed in Gergeleit and Streich (1994). At synchronization intervals of 1 s, the maximum measured offset between two clocks was below ± 2.5 μs after the clocks have stabilized (14 s). Therefore a gap of at least $\Delta G_{\min} > 5$ μs has to be inserted between reserved slots to prevent any conflict between reserved HRT slots because of the time-skew of the synchronized clocks. Different from other reservation-based schemes, COSMIC exploits the priority mechanism of CAN to enforce that the respective hard real-time message is sent in the reserved time slot. Whenever the reserved time slot is reached, the hard real-time message obtains the maximum possible priority in the system and therefore will be transmitted whenever the bus becomes free. Fig. 2 sketches the temporal properties of a time slot.

Because a message cannot be preempted, there may be a non-hard real-time message transfer still going on when the reserved slot is reached. Therefore, we have to consider a maximum waiting time which, in the worst case, is the longest possible CAN message and add it to the time slot. This prevents that a hard real-time message does not reach the deadline because of a lower priority message. In Führer et al. (2000) a non-hard real-time message is prevented from being sent if it would affect a reserved time slot. This scheme, however, would imply that (1) all nodes have synchronized clocks and (2) that before sending any non-hard real-time message, the global schedule of reserved slots has to be checked. In our scheme there may be simple smart devices not participating in the clock synchronization. They are allowed to send a message at any time, the only restriction is that they never may use the priority reserved for hard real-time messages.

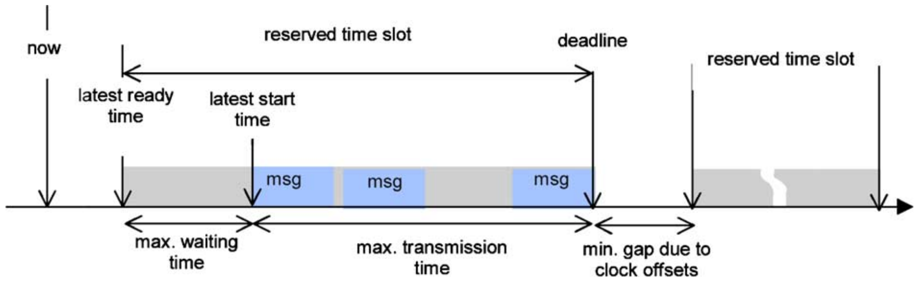


Fig. 2. Structure of a time slot.

To cope with transient transmission faults, a hard real-time message may be sent multiple times depending on the required degree of reliability. This means that the respective slot has to be extended accordingly. Our protocol relies on the fault-handling mechanisms of the standard CAN which has an impact on the fault classes which we can handle. Standard CAN provides mechanism for transient transmission errors and some permanent node failures based on time redundancy. An automatic retransmission of messages copes with transient faults. An additional mechanism is provided to handle permanent faults in the physical CAN interface. An analysis of the fault-handling times of the CAN-Bus can be found in Rufino and Veríssimo (1995) and in Livani and Kaiser (1999). For a message with B bytes of data, the maximum length of the message including header and bit-stuffing is: [1] $Message_{\text{length}} = 75 + \lfloor B \times 9.6 \rfloor$. Under the assumption of f single transmission failures, the required minimum time-slot length is: $Slot_{\text{length}} = 2 \times t_{\text{message}} + (t_{\text{message}} + 18) \times f + 3$, expressed in bit-times. Assuming a single message failure of an 8 bytes message at 1 Mbit/s (message transmission time: 151 μs, fault detection and retransmission 18 μs) and a gap between the slots of 50 μs, approximately 1900 slots/s can be allocated. If it is necessary to tolerate a permanent controller failure, this number drops down to an approximate number of 350 slots/s.

Compared to a maximum throughput of about 6500 maximum length messages, the number of possible slots is low. However, it should be noted that these numbers only refer to the number of guaranteed slots not to the number of messages which actually can be sent. This distinguishes our approach from the purely time-triggered one as proposed by Kopetz et al. (2001) and Führer et al. (2000). Firstly, we exploit not only time but also the priority mechanism to enforce hard real-time guarantees. Time is used to separate hard real-time messages which all have the same maximum priority in the system. In most situations, [2] the CAN-Bus allows to determine, without any additional overhead, whether all operational nodes have received a message successfully. In this case, the reserved redundant message slots are not needed and the sending node will stop to further transmit the message. In case there are pending SRT- or NRT-messages, the priority mechanism of CAN automatically will schedule the message with the highest priority for transmission. Thus time redundancy only costs bandwidth if faults really occur, which may be relatively rare compared to the overall traffic. Therefore, very conservative fault assumptions are possible because the penalty is low in the average. Another crucial problem is jitter. TT-CAN (Führer et al., 2000) enforce the constraints on jitter on the network level by mechanisms to guarantee the start times of messages. In TT-CAN there are particularly assigned empty slots to prevent that any ongoing non-real-time message transfer interferes with a reserved hard real-time slot. In COSMIC, jitter is prevented by defining the delivery deadline of a message. The event channel handler which has access to the global clock ensures that the message is delivered at that point in time. Thus, jitter is handled on the middleware layer rather than on the network layer. It also should be noted that the correct reception of a HRT-message within a slot cannot be predicted because faults may occur or not. In TTP and TT-CAN, messages are just resend up to a certain number of times which corresponds to a specified omission degree. This fills up the reserved slot and avoids jitter but for the price of bandwidth.

3.2. Scheduling soft real-time- and non-real-time-messages

The priority-based arbitration mechanism is also exploited to schedule soft real-time (SRT) and non-real-time (NRT) messages. SRT messages are scheduled according to an EDF scheme while NRT messages use fixed low-level priorities. The 8-bit explicit priority field in the CAN message identifier enables to represent 256 priority levels. Hard real-time (HRT) messages reserve the highest priority. The relation between the priorities of hard real-time, soft real-time and non-real-time messages can be expressed by the relation: $P_{\text{HRT}} < P_{\text{SRT}} < P_{\text{NRT}}$ (a lower numerical value represents a higher priority). The assignment enforces that a message of a lower

[1] The factor 9.6 is because of the bit stuffing mechanism.

[2] There are situations of inconsistent replicas and even inconsistent omissions (according to Rufino et al., 1998, inconsistent omissions occur with a probability in the order of 10^{-9}). Kaiser and Livani (1999) describe a transparent mechanism to handle these situations.

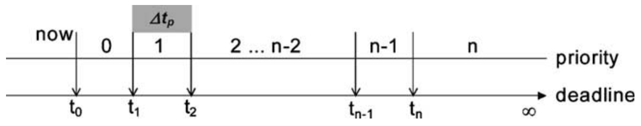


Fig. 3. Mapping of deadlines to priorities.

real-time class never will interfere with one of a higher class during bus arbitration. SRT messages are scheduled according to an EDF scheme which means that the assigned priorities reflect the deadline order of message transmissions. We assume that we have the highest priority (0) for HRT messages and for NRT messages a small number of fixed low priorities are used. The remaining priority levels are available for scheduling SRT messages. They 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 (see Fig. 3). 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 a priority slot. The priority of a SRT message will reach the highest possible value at its transmission deadline. 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 arbitrarily 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 $\Delta H = (P_{\max} - P_{\min}) \times \Delta t_p$. 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. in the approach of Zuberi and Shin (1997) which use the CAN 2.0A with 11-bit identifiers. Considering around 250 priority slots available in our scheme and a priority slot length of approximately one CAN-message, we can accommodate 250 message transfers within the time horizon.

3.3. The real-time event layer

The event layer constitutes the interface to the application. It provides the abstractions of events and event channels as described in Section 2. The event layer makes use of the message classes in the AN-layer to provide event channels of different real-time classes. Additionally, the event layer comprises the functionality to route and filter events based on their subjects. A subject is represented by a 64-bit unique ID specified by the application. There are many different ways for subject-based routing. A subject can be related to a multi-cast group (Meier and Cahill, 2002) and the subscribers can listen to whatever is sent to this address. In other systems, every message is sent to the same broadcast address and the subject is contained in the message body (Oki et al., 1993). In the COSMIC implementation for the CAN-Bus, we exploit the specific broadcast and message filtering properties of CAN. A dynamic binding protocol is performed when a publisher announces a publication or a subscriber subscribes to a channel. It relates the long 64-bit subject UID of an event to the 14-bit e-tag in the CAN message ID. The filtering functions of the CAN controller hardware then can be used to filter subscribed events from the message stream. This takes a considerable burden from the embedded controller which otherwise would have to examine every message on the CAN-Bus to recognize such an event (Kaiser and Mock, 1999; Kaiser and Brudna, 2002).

According to the message classes provided by the AN-layer, the event channels are classified as hard real-time (HRTEC), soft real-time (SRTEC) and non-real-time (NRTEC) event channels. The transfer of events through a hard real-time event channel (HRTEC) is certain, i.e. all the necessary resources are reserved to transmit event messages timely under specified fault assumptions. The API for a HRTC is presented in Fig. 4. HRTECs need to set up the infrastructure before communication in compliance with the required guarantees. This is initiated by an application through calling the "announce" method:

channel.announce(subject, attribute_list,
  exception_handler)

```
class hrtec {
  private:
    subject subject_uid;
  public:
  // constructor and destructor of the class
    hrtec(void);
    ~hrtec(void);
  // methods used for publication:
    int announce(subject, attribute_list, exception_handler);
    int publish(event);
  // method used for subscription:
    int subscribe(subject, attribute_list, event_queue,
                  notification_handler, exception_handler);
  }
```

Fig. 4. Declaration of a HRTEC class in C++.

Because reservations are statically allocated for hard real-time event channels, event channel data structures for HRTEC are also statically created by the Event Channel Handler. Therefore, the announcement of a HRTEC does not result in the creation of a new event channel data structure. The Event Channel Handler searches in the event channel list for a HRTEC with the respective subject.

The announcement is accepted if the attribute list declared by the publisher (deadline, data length in bytes, omission degree) matches the static attribute list of the channel data structure.

Soft real-time events become ready at any time and are scheduled according to their transmission deadlines by an earliest deadline first (EDF) algorithm. Although the declaration of a SRTEC looks similar to the HRTEC, the differences are crucial and primarily are substantiated in the different attributes defined for SRTECs. Events published to a SRTEC specify a transmission deadline and a temporal validity parameter. The transmission deadline is defined as the latest point in time when a message has to be transmitted. This deadline is exploited for scheduling the event on the communication medium. Due to other messages competing for the communication medium, a message may be delayed. Additionally, delays may occur in the receiving node because soft real-time events are stored in a local queue and may not be immediately delivered due to ongoing computations and other pending events. This is the reason why we consider a validity expiration time additionally. The validity expiration is used to determine at the subscriber side whether the information is still valid. Thus, the expiration time constitutes an end-to-end temporal constraint. Two exceptional situations may occur: a missed transmission deadline and an expired validity. When missing the transmission deadline a message will be discarded from the sending queue and a local exception is raised. This prevents outdated messages to still compete for the bus but allows the application to react if necessary. The transmission deadline is defined in the interval: $t_{\mathrm{valid}} - t_{\mathrm{ready}}$, where t_{valid} is the point in time until the event is temporally valid and t_{ready} when it is ready to be sent. The validity of an event is checked when the respective application is notified and an exception is raised on expiration.

NRTECs are used for events that do not have timeliness requirements. 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. A NRTEC has a fixed priority. The priority is specified by the application during the announcement of the channel. NRTEC are particularly used to configure and maintain the smart networked devices of the system. This may require sending 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 "fragmentation" mechanism for NRTEC to chain individual CAN messages to a larger application specific message is provided by the middleware.

Fig. 5 shows first results. We tested in a scenario where hard, soft, and non-real-time channels are created. Two PCs running the COSMIC middleware under RT-Linux are connected to the CAN Bus (250 Kbit/s) and an Infinion C167 micro-controller is generating load. The measured clock skew is below 5 µs if no (normal) Linux task are running, thus we are able to obtain high precision timestamps for the measurements. There is one publisher and one subscriber using the channel. Every 250 ms an event is pushed to a channel.

The upper part of Fig. 5 shows the behavior of a non-real-time channel. The micro-controller is an independent load generator and disseminates CAN messages with a fixed priority of "2", i.e. the third highest priority. The load is varied continuously between 700 and 1700 messages/s. A message transfer (8 bytes) takes around 600 µs. Events are disseminated with a fixed low priority and always are delayed when the load generator sends a message. As expected, the jitter is low for the initial load because there are only rare conflicts. At some point, the probability to find a point in time when the CAN-Bus is idle and no message from the load generator is pending becomes low and delays rise up to 39 ms. The situation is much better for soft real-time events. Soft real-time messages start at some low priority but reach a max. priority of "1" shortly before the latest start time thus always winning the arbitration process against the load messages. The delay seen in the middle part of Fig. 5 comes from the fact that there may be ongoing message transfers which cannot be preempted. Finally, hard real-time channels exhibit a very stable behavior. Because ongoing message transfers are anticipated by the reservation scheme and messages are delivered at their specified deadline, the jitter only varies from 2 µs to 18 µs due to load variations.

4. Comparison with related work

Our work on an event-based interaction system for embedded systems is inspired by many research efforts in very different areas. Event-based systems in general have been introduced to meet the requirements of applications in which entities spontaneously generate information and disseminate it, e.g., as in Bacon et al. (2000), Meier and Cahill (2002) and OMG (2000). Intended for large systems and requiring complex infrastructures, these event systems do not consider quality aspects like timeliness and dependability issues. Harrison et al. (1997) introduced a real-time event system

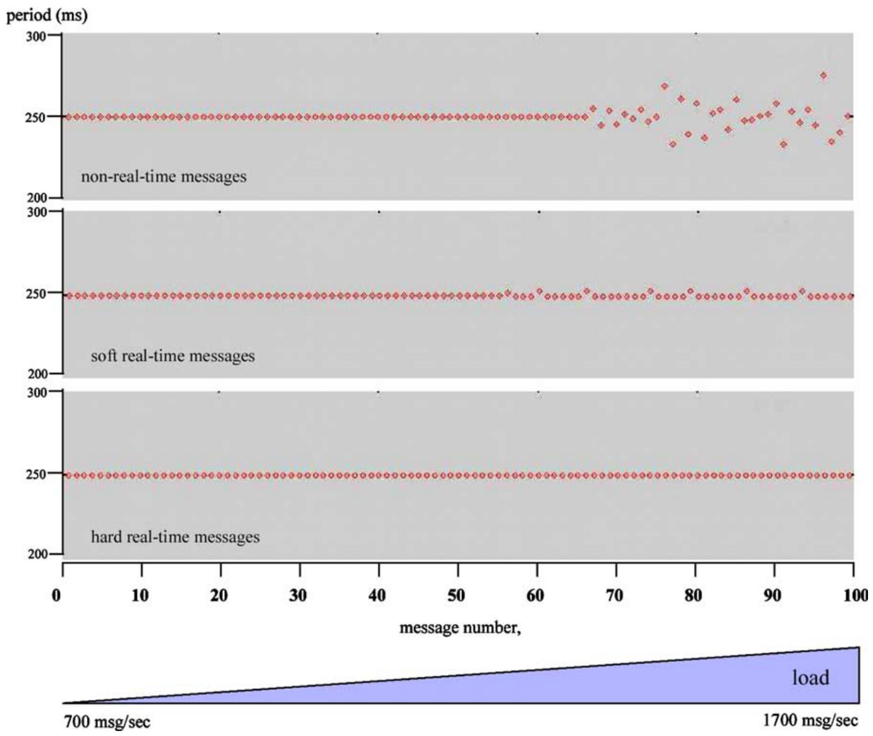


Fig. 5. Event dissemination through different channels.

for CORBA. The events are routed via a central event server which provides scheduling functions to support the real-time requirements. Here, a central component is exploited to guarantee temporal requirements. Such a component is not available in an infrastructure envisaged in our system architecture. Secondly, to exploit the underlying CORBA system, event dissemination and notification is based on the remote invocation mechanism of CORBA which again introduces the problems of control dependencies. Finally, the proposed system needs a quite complex middleware unsuitable for directly integrate smart devices. There are efforts by Kim et al. (2000) and Lankes et al. (2003) to implement CORBA for the CAN-Bus. However, in these approaches no, or only very limited, timeliness or dependability issues can be supported. A new scheme to integrate smart devices in a CORBA environment is proposed in Kopetz et al. (2001) and has lead to a proposal of a standard (OMG, 2001, Smart Transducer Interface). Smart transducers are organized in clusters which are connected to a CORBA system by a gateway. Hence, this standard also suggests a WAN-of-CANs structure. The proposal is based on the concepts of an interface file system (IFS) and a time-triggered communication protocol (TTP/A) which are intimately intertwined. The IFS defines a global address space within each cluster in which the individual devices can be identified and accessed uniformly. The TTP/A provides a low overhead efficient communication protocol in which the sources and destinations of data are specified in terms of the IFS. In contrast to our event channel model, all communication inside a cluster relies on a single synchronous communication channel. Although the temporal behavior of a single cluster is rigorously defined, no model to specify temporal properties for cluster-to-CORBA or cluster-to-cluster interactions is provided. Another effort to specify interactions between smart devices is the IEEE 1451 Standard for a smart transducer interface (Bryzek, 1996). Interfaces are specified on various levels, however, a general higher-level interaction model is not provided. Particularly, to our knowledge, the network interface (NCAP: Network Capable Processor) and the inter-operability model have not yet been specified. For the CAN-Bus various application level protocols have been standardized such like CAL (CiA, 1993, CAN Application Layer), SDS (1996), and DeviceNet (Noonen et al., 1994). They are based on fixed priority schemes which are not able to reflect temporal requirements. To overcome this problem, the deadline-monotonic priority assignment of Tindell and Burns (1994) uses an off-line feasibility test to ensure 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 Davis (1994) or the schemes proposed by Eriksson et al. (1996) and Zuberi and Shin (1997) support the use of both hard and soft real-time communication on the CAN-Bus. However, they have the disadvantage of providing a very restricted time horizon.

5. Conclusion

We presented the COSMIC middleware which relies on an event channel model that supports functional and non-functional attributes. 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. Assuming the coexistence of different event channel classes and the need to consider sporadic events, substantial advantages concerning a better utilization of bandwidth can be achieved compared to other time-triggered schemes. Firstly, when a reserved slot is not used, the priority mechanism of CAN automatically assigns this slot to some other (lower priority) message. Secondly, 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 has been implemented. It is available for Linux, RT-Linux and the small memory footprint and efficient CAN-Bus implementation enables its use on 16- and even 8-bit micro controllers (Infinion C167 and Motorola 68HC08 micro-controllers). However, the preliminary version does not fully support the real-time properties described in this paper. A recent implementation including all real-time channel classes is available for a CAN-Bus environment under RT-Linux. It is currently evaluated before it is ported to the embedded micro-controllers. The results presented in this paper have been measured for this version of COSMIC.

Future work will concentrate on two main directions. The current version of COSMIC already supports event channels across multiple CAN-Bus and TCP/IP networks. This enables applications of cooperating robots which are composed from smart sensors and actuators locally connected by a CAN-Bus and cooperating over a wireless TCP/IP connection. The event channel concept of the COSMIC middleware hides this heterogeneity and allows for a dynamic interaction between the robots. However, it is not yet possible to specify quality attributes for these channels. Because the degree of predictability is different in a wired CAN-Bus compared to the wireless network, awareness of the underlying network structure and the respective quality of service in the various networks has to be included. A second focus will be on filtering mechanisms to exploit the context attributes in the event description.

Acknowledgement

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).

References

Bacon, J., Moody, K., Bates, J., Hayton, R., Ma, C., Mcneil, A., Seidel, O., Spiteri, M., 2000. Generic support for distributed applications. IEEE Computer 33, 68–76.

Bosch GmbH, 1991. CAN Specification version 2.0.

Bryzek, J., 1996. Introduction to IEEE-P1451: the emerging hardware independent communication standard for smart transducers. In: Eurosensors X. Leuven, Belgium.

CiA, 1993. CAN application layer (CAL) for industrial applications. Published by CAN in Automation.

CiA, 1999. Draft Standards 301: CANopen Application Layer and Communication Profile. Published by CAN in Automation, version 4.0.

Davis, R., 1994. Dual priority scheduling: a means of providing flexibility in hard real-time systems. Tech. Rep. YCS230, University of York, UK.

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

Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A.-M., 2001a. The many faces of publish/subscribe. Tech. Rep. DSC ID:200104, EPFL, Lausanne, Switzerland. Available from <www.citeseer.nj.nec.com/442483.html>.

Eugster, P., Guerraoui, R., Damm, C., 2001b. On objects and events. In: Proceedings for OOPSLA 2001, October.

Führer, T., Muller, B., Dieterle, W., Hartwich, R, Hugel, R., Walther, M., 2000. Time triggered communication on CAN. Available from <http://www.can-cia.org/can/ttcan/fuehrer.pdf>.

Fredriksson, L.B., 2002. CAN for critical embedded automotive networks. IEEE Micro 22 (4), 28–36.

Gergeleit, M., Streich, H., 1994. Implementing a distributed high-resolution real-time clock using the CAN-Bus. In: 1st International CAN-Conference. Mainz, Germany, September.

Harrison, T.H., O'Ryan, C., Levine, D.L., Schmidt, D.C., 1997. The design and performance of a real-time CORBA event service. In: OOPSLA '97, ACM, Atlanta, GA, October.

Kaiser, J., Brudna, C., 2002. A publisher/subscriber architecture supporting interoperability of the CAN-Bus and the internet. In: 2002 IEEE International Workshop on Factory Communication Systems (WFCS2002), Vasteras, Sweden, August. Available from <www.citeseer.nj.nec.com/kaiser99implementing.html>.

Kaiser, J., Livani, M.A., 1999. Achieving fault-tolerant ordered broadcasts in CAN. In: Third European Dependable Computing Conference (EDCC-3).

Kaiser, J., Mock, M., 1999. Implementing the real-time publisher/subscriber on the controller area network (CAN). In: 2nd International Symposium on Object-Oriented Real-time distributed Computing, Saint-Malo, France, May.

Kim, K., Jeon, G., Hong, S., Kim, T., Kim, S., 2000. Integrating subscription-based and connection-oriented communications into the embedded CORBA for the CAN-Bus. In: IEEE Real-time Technology and Application Symposium, May.

Kopetz, H., Griinsteidl, G., 1992. TTP—A time-triggered protocol for fault-tolerant real-time systems. Tech. Rep. 12/92, Inst. fur Techn. Informatik, Techn. University of Vienna.

Kopetz, H., Holzmann, M., Elmenreich, W., 2001. A universal smart transducer interface: TTP/A. International Journal of Computer Systems, Science and Engineering 16 (2).

Lankes, S., Jabs, A., Bemmerl, T., 2003. Integration of a CAN-based connection-oriented communication model into real-time CORBA.

In: Workshop on Parallel and Distributed Real-time Systems, April.

Livani, M., Kaiser, J., 1999. Evaluation of a hybrid real-time bus scheduling mechanism for CAN. In: 7th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'99), San Juan, Porto Rico, April.

Meier, R., Cahill, V., 2002. Steam: event-based middleware for wireless ad hoc networks. In: International Workshop on Distributed Event-Based Systems.

Moini, A., 1997. Vision chips or seeing silicon. Available from <http://www.eleceng.adelaide.edu.au/Groups/GAAS/Bug-eye/visionchips>.

Noonen, D., Siegel, S., Skeen, D., 1994. DeviceNet application protocol. In: 1st International CAN Conference, Mainz, Germany.

Oki, B., Pfluegl, M., Siegel, A., Skeen, D., 1993. The information bus—an architecture for extensible distributed systems. In: ACM Symposium on Operating System Principles, pp. 58–68.

OMG, 2000. Notification service, version 1.0. Object Management Group. Available from <http://www.omg.org>.

OMG, 2001. Omg: Smart transducer interface, initial submission—orbos/2001-06-03.

Pereira, C.E., Kaiser, J., Mitidieri, C., Villela, C., Becker, L.B., 2001. On evaluating interaction and communication schemes for automation applications based on real-time distributed objects. In: 4th Int. Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'Ol), Magdeburg, Germany.

Profibus, 1999. Technical overview. Available from <http://www.profibus.com/data/technic>.

Rajkumar, R., Gagliard, M., lui Sha, 1995. The real-time publish/subscribe inter-process communication model for distributed real-time systems: design and implementation. In: IEEE Real-Time Technology and Applications Symposium. IEEE Real-Time Technology and Applications Symposium, June.

Rufino, J., Veríssimo, P., 1995. A study of the inaccessibility characteristics of the controller area network. In: Proceedings of the 2nd Int. CAN Conference, London, UK.

Rufino, J., Veríssimo, P., Almeida, C., Rodrigues, L., 1998. Fault-tolerant broadcasts in CAN. In: Proceedings of FTCS'98, Munich, Germany.

SDS, 1996. Smart distributed systems, application layer protocol version 2, micro switch specification gs 052 103 issue 3, honeywell inc.

Tindell, K., Burns, A., 1994. Guaranteeing message latencies on controller area network (CAN). In: First Int. CAN Conference, Mainz, Germany.

Veríssimo, P., Casimiro, A., 2003. Event-driven support of real-time sentient objects. In: Proc. of the 8th IEEE International Workshop on Object-oriented Real-time Dependable Systems, Guadalajara, Mexico, January.

Veríssimo, P., Cahill, V., Casimiro, A., Cheverst, K., Friday, A., Kaiser, J., 2002. Cortex: towards supporting autonomous and cooperating sentient entities. In: European Wireless Conference, Florence, Italy.

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