A Publisher/Subscriber Architecture Supporting Interoperability of the CAN-Bus and the Internet

Jörg Kaiser
University of Ulm
Dept. of Computer Structures
Ulm, Germany
kaiser@informatik.uni-ulm.de

Cristiano Brudna
University of Ulm
Dept. of Computer Structures
Ulm, Germany
Cristiano.brudna@informatik.uni-ulm.de

Abstract

The paper describes a publisher/subscriber protocol and the respective system support tailored for systems which comprise a large number of smart sensors and actuators. These components directly interact to form a reactive control layer. To support large systems structured in clusters of tightly interacting components, we provide interoperability between the fieldbus (CAN) level and a TCP/IP network to allow cooperation between these "islands of control" over a larger area and wireless connections. This requires middleware support which enables the seamless integration of the different networks and provide a uniform communication interface. Finally, the system support requires lightweight mechanisms achieving a memory footprint and performance requirements suitable to run on a wide range of controllers.

1. Introduction

Future control systems may include a large number of cooperating smart devices which may comprise sensor components, computational devices and a network interface. The built-in computational component enables the implementation of a well-defined high level interface that does not just provide raw transducer data, but a pre-processed, application-related set of process variables**.** Consequently, the interfaces and the functions of these smart components may include functions related to overall control, supervision, and maintenance issues. Perhaps the most interesting and challenging property of these intelligent devices is their ability to spontaneously interact with the overall system. This enables a modular system architecture in which smart autonomous components co-operate to control physical processes without the need of a central co-ordination facility.

Our system model assumes smart components which may range from powerful workstations to low-end micro controllers. They cooperate over a hierarchy of networks. In general, the communication infrastructure structure can be viewed as a WAN-of-CANs[1] [VCC02], in which a wide area network (WAN) connects "islands of tight control". The control clusters are interconnected by a controller area network in a general sense. There are a number of goals which we want to reach for the control system architecture:

1. Components of the network are autonomous. Autonomy means that each component is in its own sphere of control and no control signal crosses the boundary of a component. Hence, components only interact on the basis of shared information as e.g. proposed in the data field architecture in Autonomous Decentralized Systems [MOR93].
2. The architecture should support many-to-many communication patterns. A typical situation is that the information gained from a sensor can be used and analyzed in more than one place, e.g., the output of a distance on a mobile robot is interesting for reactive motor control implemented on a small micro-controller as well as for long term navigation strategies implemented on a more powerful device.
3. Communication is spontaneous rather than initiated by a client request. The smart components have to react to external events or are periodically triggered by a local clock. This is well captured by a generative communication model.

A direct consequence of the spontaneous communication is the producer/consumer style of interaction. While popular high level communication models are often based on the request/reply style of a client/server relation, this seems to be less appropriate in a control environment [PBV01]. We developed a publisher/subscriber protocol which supports a producer/consumer model and reflects the requirements defined above. The benefits of a publisher/subscriber protocol for control applications have been pointed out in [OPS93],[PAR97] and [RAJ95]. They centre around the ease of programming, extensibility and scalability arguments. To support interoperability between mobile entities across multiple networks, it is most important that in such a protocol it is content-based routing of messages, i.e. the subscriber has to specify what kind of information it wants to receive rather than

[1] CAN is used in a general sense here. Later in the paper we refer to the CAN-Bus , a specific fieldbus for automotive and industrial applications.

specifying a particular producer which may not be known at design time. However, content-based routing has a substantial drawback concerning the overhead. There is in general no simple way for a controller to filter the message stream for the information it has subscribed for. Therefore, a binding mechanism between the contents of a message and the respective addressing mechanism of the network has to be introduced, i.e. the content of a message has to be related to a network address. This allows the filtering to be performed by the respective communication controller hardware. Because different network addresses have to be bound to the same message content (when delivered by multiple producers) and different networks may use different addressing mechanisms, this binding has to be performed dynamically. A description of the basic problems of implementing the publisher/subscriber protocol on CAN (CAN 2.0 [BOS91]) has been published in [KaM99]. In this paper we focus on:

1. The problems to accommodate a single high level communication paradigm in a decentralized system composed from nodes with widely differing computational capabilities.
2. The problem of integrating networks with different properties concerning network topology and the addressing mechanisms.
3. Assessing the basic performance and memory requirements for the protocol.

We will present the components of the middleware architecture which provide interoperability between CAN-BUS and TCP/IP networks. The protocol runs under Linux and RT-Linux for more powerful computers and on a proprietary kernel for a 16-Bit micro controller.

In the next chapter, we compare related work . Then we will introduce the basic architecture for the our proposed publisher/subscriber protocol which we called UPS (**U**lm **P**ublisher/**S**ubscriber protocol) in chapter 3. Chapter 4 describes the implementation of the communication infrastructure. In chapter 5 we present measurements to assess the basic performance and memory characteristics of the implementation. Finally, chapter 6 concludes the paper.

2. Related Work

There are many elaborated concepts and mechanisms to provide predictable MAC-layer protocols for the CAN-Bus [TiB94], [ZuS95], [AFF99], [LKJ99], [FMD02], [TTC02]. However, less work is available for the higher level communication and for the interoperability between different networks. Concerning interoperability, there is work on integrating wireless communication media and field busses [KVA02], [ATV02]. This work however, maintains the basic fieldbus protocols (CAN [KVA02] and ProfiBus [ATV02]) and adapts the temporal characteristics to a wireless link. Therefore, this is more intended as a replacement for wired connections than for a integration with wider area networks and their standard protocols. The company IXXAT [IXX02] provides a product which links a CANopen network to TCP/IP. But also in this approach the CANopen master handles all communications for its slaves. Thus, there is no seamless integration of smart sensor or actor components in a general wide area communication scheme.

In the area of publisher/subscriber protocols there is also a substantial amount of research ranging from experimental systems [HLS97], [Maf97], to commercial products [OPS93], [PAR97], [TAL02], and standards [NSS00]. Most of them are devoted to dynamic large scale distributed systems. They are not suited nor intended for real-time applications in which tiny smart components directly interact. Particularly, they do not meet the requirements coming from the restrictions in processor performance and network bandwidth which are principal design constraints for our protocol. The publisher/subscriber protocols which specifically have been developed for real time applications are the real-time publisher/subscriber protocol [RAJ95], the commercial NDDS [PAR97] and a protocol for the CAN-Bus [KIM00]. None of the protocols supports a direct interoperability between the CAN-Bus and the Internet world.

Rajkumar et al. [RAJ95] present a subject-based Publisher/ Subscriber protocol which handles all inter-process communications in the distributed system. The target systems for the real-time P/S protocol are more powerful computing nodes with a standard operating system and a IP network. A migration path or an interaction model with respect to smaller systems is not addressed.

A CORBA for CAN-based systems is presented in [KIM00]. It supports both subscription-based group communication as well as point-to-point communication with the CAN 2.0A standard. Here an application object subscribes and publishes to channels, which are related to a certain subject and identified by a unique binary number. This implementation has the disadvantage of having a special version of the inter-ORB (GIOP) protocol developed for CAN that is not compatible with other ORB implementations, thus compromising the interoperability of the system.

The Network Data Delivery Service (NDDS) [PAR97] is a commercial implementation of the (real-time) publisher/subscriber model. Again, the protocol is intended to run on PC-workstations over an IP-network rather than on small controllers.

3. The UPS Architecture

The UPS architecture is based on the concept of event channels [KaM99]. An event channel is related to the content of an event represented by the subject of this event [OPS93]. Applications can publish to a channel related to the respective subject and they may subscribe to such a channel. Channels are identified by a unique identifier (UID), which in our case is a 64-bit binary number. This UID is used by the application to communicate. A UID is independent of any addressing mechanism and identifies a specific event channel independent of any network specific issues. Although this is a desirable feature because it allows an unambiguous identification of an event channel also in a wide area network, it puts some challenges to a realization under the anticipated system constraints. It is hardly feasible on a micro-controller to read every message which occurs on the network and analyse its content. Therefore, we developed a scheme specifically for systems composed from low performance controllers. The basic idea is to map an event channel, characterized by its UID to the low level addressing mechanisms of the network. This enables event routing and filtering on the level of the communication controllers and frees the CPU from the time consuming task to examine every message and decide whether a local application needs this message or not. The CAN-Bus already supports this concept because the principle of operation is close to subject-based addressing, i.e. CAN message identifiers are related to the content of a message rather than to a sender or receiver address. In a TCP/IP network a more complex approach, based on subscription lists has to be adopted. This is described in chapter 4.2.

Our architecture is based on decentralized middleware for the publisher/subscriber architecture. On every node exists a local Event Channel Handler (ECH), which performs filtering for messages and is responsible for maintaining and handling event channels on the node (see fig.1). The ECH manages the local communication controller, filters the message stream on the bus and notifies applications on the arrival of the respective messages.

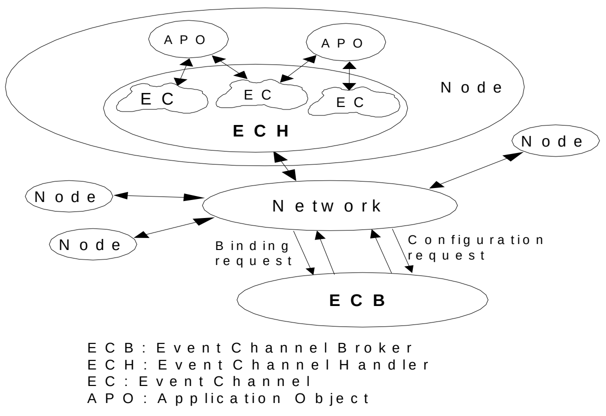

Figure 1 - UPS Architecture Overview

The ECH also initiates the binding protocol. Whenever an application object publishes or subscribes, it uses the event channel UID. The ECH checks whether this event channel already exists in the node and if not, creates the respective data structure. Subsequently, the binding to the network related addressing mechanism has to be performed. This binding has to be consistent, i.e. all identical UIDs which may be used in different nodes should be bound to the same low level address in a specific network. For this purpose a service called Event Channel Broker (ECB) is available. The ECH issues a binding request and the ECB returns a network address. The details of this operation are network specific and are described in the following chapters. It should be noted that the low level network address has only to be valid in the local network, e.g. the CAN. Therefore, the ECB residing in other networks do not need to maintain consistent binding information. The ECB also provides a dynamic configuration service for CAN to enable a dynamic inclusion of new devices in the network. Because low level message IDs must be unique in CAN, the ECH dynamically assigns unique node IDs. The respective configuration protocol is not presented here. The principle of operation is described in [KaM99], a more detailed technical description can be found in [Cas02].

4. Implementation for CAN and TCP/IP

The current implementation (see fig. 2) is basically composed by a CAN part, a TCP/IP part, and by gateways that connect the networks. Note that our architecture allows to have many CAN sub-networks interconnected by a TCP/IP network resulting in a WAN-of-CANs network structure.

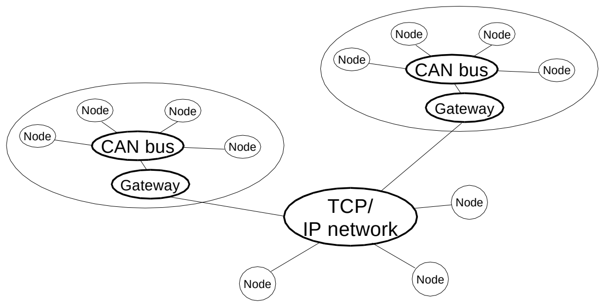

Figure 2 - Implementation Overview

From the point of view of an application the communication infrastructure provides the same services both for application objects running on TCP/IP nodes and CAN nodes. There is an ECH (Event Channel Handler) in each node that provides all the support for local objects and an ECB (Event Channel Broker) running on a TCP/IP node that provides binding information for the TCP/IP nodes. There is also an ECB for the CAN network which in our implementation is provided by the gateway. Thus, the gateway is a special component that can be seen as a "two-faced" ECH, one side for the CAN network and one for the TCP/IP network. It has to be noted that an ECB is required in every network which is separated by a gateway to cope with different, independent addressing schemes. However, there is no communication needed between the ECBs to keep binding information consistent. Because a gateway always acts like an ECH for every network, publications and subscriptions are automatically routed.

4.1 UPS on CAN

The communication over the CAN bus is supported by the local ECH and the CAN-ECB, which in our approach is performed by the gateway. A 29-bit CAN message identifier (CAN 2.0 B specification) is divided in three fields for event messages. One of the fields, which is a 14-bit field called *etag*, identifies the event channel. A second field (*TxNode* with 7 bits), which contains a node identification, is added to ensure the uniqueness of the CAN identifier. Additionally, there is the 8 bit priority field with that is reserved for message scheduling. UPS allows the dynamic binding of CAN message IDs to event UIDs and the dynamic configuration of the CAN nodes. On start up or when integrated in a CAN system, each node connected to CAN needs a unique node identifier to assure the uniqueness of CAN messages. A specific configuration protocol assigns this short node ID (the *TxNode*). The details of the protocol can be found in [Wal01]. Once the CAN node has its *TxNode*-ID, it can start subscribing and publishing to event channels. To subscribe to an event channel an application object uses the *subscribe* primitive (see section 5.3), where the event channel identifier (a 64-bit unique identifier) must be provided.

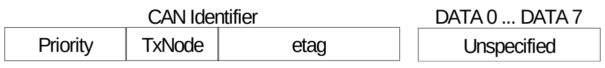

Figure 3 - Event Message

In a similar way, an application object can use *unsubscribe* to remove its subscription. Finally, to publish an event to a certain channel, an application uses the *publish* primitive, where the event channel identifier and the optional data (at most 8 bytes) related to the event are provided.

The *publish*, *subscribe*, and *unsubscribe* requests are handled by the local ECH. If a request is made upon a channel that has not been locally used before, the ECH first requests an *etag* to the ECB. After this, the ECH is able to recognize (by the *etag* field of the CAN identifier) and filter out relevant messages broadcasted on the CAN bus. Filtering may exploit the filter mask mechanism which is available on most CAN-controllers. However, due to largely different properties of the CAN controllers, the problem of configuring the arbitration and mask registers has to be performed in an application and hardware specific way. If an application needs to subscribe to a large number of channels, we therefore do not use the specific controller features for hardware filtering. Because these applications usually run on more powerful controllers, it is feasible to examine every message and perform the filtering in the ECH as indicated by our measurements in chapter 6. Smart sensors or actuators however may have rather restricted CPU performance and may need to selectively receive only the specific CAN messages. On the other hand, they only require to subscribe to a few channels. In this case, subscription and all bindings have to be performed eagerly during the configuration phase as part of an initialization procedure. For notification, the ECH maintains a local subscriber list. Whenever a new event is published the ECH checks whether the related channel is in the list and takes the respective actions.

When a message is published by an application object, the ECH checks its subscriber list for other local objects interested in the respective event channel and sends a copy of the event to them. Furthermore, all events published by a local application object are broadcasted on CAN bus, so that other CAN nodes and the TCP/IP part will be able to receive this event messages (as it is explained in the section 5.1.3).

4.2 UPS on TCP/IP

As in the CAN specific part, the communication in the TCP/IP part of the network is supported by the local ECH and also by a TCP/IP-ECB. Since the TCP/IP is a point-to-point protocol, nodes are identified by their IP numbers and event channels by their UID's. An application uses the same interface, namely *subscribe, unsubscribe, and publish* to respectively request a subscription, to cancel a subscription, and to publish to an event channel. As before, the local ECH handles all these operations. Whenever an application object subscribes to a channel the local ECH adds the application in the list of interested applications for the specified channel. The ECH sends a message to the TCP/IP-ECB requesting the subscription to a channel. Then, the TCP/IP-ECB adds the IP number of the subscribing node in a global list that relates publishers and subscribers with event channels. There is a list for every event channel which contains all subscribers. Thus, when an application publishes an event, the local ECH uses its local channel list to send the event message to every node on the list.

4.3 The Gateway

The gateway enables applications to receive event messages published to the respective event channel anywhere in the WAN-of-CANs network. This capability must be provided transparently by the communication architecture. The gateway has to support two directions of event flow: TCP/IP-to-CAN-

Bus and CAN-Bus-to-TCP/IP. The gateway behaves exactly like an ECH to the CAN-Bus and the TCP/IP network. In the direction TCP/IP-to-CAN-Bus, the gateway maintains a list of all event channels serviced by publishers in the TCP/IP network for which there are subscriptions from objects residing in the CAN system. The gateway hence appears like an ordinary ECH in all registries of the respective event channels. For event flows from CAN-Bus to the TCP/IP network, the gateway maintains an ECH registry for each event channel serviced by a publisher on the CAN-Bus. A benefit from this scheme is that event messages published on the TCP/IP part only will be broadcasted to the CAN bus when a CAN node has subscribed to the respective event channel. Similarly, event messages broadcasted on CAN bus will be disseminated by the gateway only for subscribers on the TCP/IP network. Hence the gateway also acts as a filter preventing the CAN clusters from being flooded by the messages on the TCP/IP network.

4.4 Versions of the UPS middleware

Our current implementation of the communication middleware is available for Linux, RTLinux 3.1 [RTL02], and for the 16-bit Siemens C167 microcontroller [SIE96]. On the C167 two versions are provided: one running on a simple executive developed on our department and a standalone version on the bare silicon. The Linux version supports both TCP/IP and CAN networks, however, no real-time behaviour is enforced. For the CAN-Bus, where we intend to provide scheduling capabilities and increased communication predictability we developed the middleware for RTLinux and the C167. The Linux and RTLinux implementations are currently running in desktop PCs, laptops and Compaq iPAQ. Wireless 802.11 connections are used for mobile operation. Our CAN implementations currently uses SJA1000 CAN controllers[2].

4.5 The Application Programmable Interface (API)

An API composed by a set of functions and structures is provided for applications. Since there are implementations for three different systems the API has two slightly different forms: one for Linux and another for RTLinux/C167. This is because there are no sockets available in RTLinux and on the C167 implementations So, since all tasks (or threads) share the same address space in this operating systems the objects and the local ECH communicate each other trough direct addressing provided by an special interface. Because these details are not relevant for this paper, we only present the Linux API. The interested reader is referred to [Cas02]. The Linux Interface consists of a set of 4 functions:

[2] The hardware consists of "CAN-dongles" (www.phytec.de) which are connected to the parallel port. Linux drivers for this hardware was also developed and are available from the authors.

1. *int publish(msg_obj *message*
2. *int subscribe(u_int64_t channel)*
3. *int unsubscribe(u_int64_t channel)*
4. *msg_obj *get_msg(msg_obj *message)).*

The function 1. used to publish and event. The event channel and the message content is described in a *msg_obj* structure which is described below: The functions subscribe and unsubscribe are similar. The *msg_obj* structure is composed from a 64-bit channel identifier, the data length, and the message data part(max. 8 bytes due to CAN restrictions).

```
typedef struct {
        u_int64_t channel;
        int len;
        unsigned char data[8];
        } msg_obj;
```

4.6 Test Environment

One of the motivations of the prototype implementation was to use it in a mobile robot scenario and obtain quantitative results with respect to memory footprint and performance parameters. No optimization of the implementation has been done yet. Our current test environment is composed by a set of hardware platforms running different versions of the publisher/subscriber implementation. There are PCs with Linux connected by 100Mbit Ethernet and Compaq iPAQ's with wireless Ethernet PC-Cards that can use an access point to communicate with the local network. There is also a gateway connected both to a TCP/IP network and a CAN network where a set of C167 boards can be attached. A view of this environment is shown on the Fig.4.

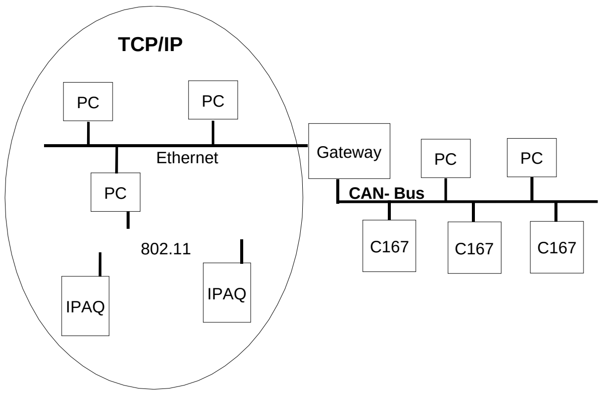

Fig. 4 -Environment Overview

6. Measurements

To better estimate the suitability of UPS, we needed to perform measurements. Particularly interesting were the time the protocol requires on the different platforms supporting UPS. The performance of the current implementations for Linux, RTLinux, and the C167 have been assessed by measurements of local and end-to-end latencies and round-trip delays. All measurements for Linux and RTLinux have been done in Pentium III computers (800 MHz) and measurements for C167 on a micro-controller board with a 20Mhz clock.

We first measured the local latencies for CAN bus, defined by the delay between the message arrival at the node and the delivery of the event by all the local subscribers. This delay includes the processing time used by the CAN driver and the ECH as well as the time necessary for scheduling and all the necessary context switches. Measurements use the system clock for timestamps in the RTLinux version and a hardware timer in the case of the C167. The measurements on CAN were performed with maximum transmission rate of 1Mbit/s and messages with a payload of 1 data byte. Results were obtained taking 100 samples (see table 3) and calculating the average, the best case, and the worst case. From the results on table 3, we can see that an additional subscriber adds around additional 5µs in RTLinux, and ~ 800µs on the C167. The observed jitter is around 15µs for RTLinux, 47µs for C167 stand alone, and 35µs on the C167 with executive. The larger times in the stand-alone version comes from the busy waiting loop rather than a notification by the OS in the other C167 version. Additionally, in the C167 version, the CAN driver needs around 90µs and because we use the interrupt of a successful transmission on the CAN-Bus, another 70µs for the transmission time have to be considered for the results. Thus, to obtain the time which is spent in the ECH these latencies have to be subtracted. Most of the ECH time is used for the sequential table look-up for e-tag/UID matching.

Since the transmission of a CAN message takes an amount of time in the order of 100us, the RTLinux is able to handle every message published on the bus. However, in the case of the C167 on both versions, stand alone and with executive, some constraints are visible. In the stand alone version only around 1400 messages per second can be handled and 540 messages per second in the version with executive. In a worst case scenario, where event messages are broadcasted to CAN bus with maximum rate, the C167 will not be able to handle and/or timely deliver every event to local subscribers.

# of subscribers	*RTLinux min/average/max [µs]*	*C167 stand alone min/average/max [µs]*	*C167 with executive min/average/max [µs]*
1	15.8 / 17.5 / 30.4	674 / 697 / 721	1815 / 1849 / 1851
2	20.5 / 22.6 / 36.1	-	2604 / 2629 / 2640
3	25.4 / 27.9 / 42.1	-	3396 / 3419 / 3424
4	30.9 / 34.3 / 48.9	-	4183 / 4212 / 4223

Table 3 - Local latency (Notification)

The local latency for publishing, which is the delay between a publish request and the actual transmission to the CAN-Bus is depicted in table 4. Again, the latencies were calculated using the system clock as reference in the case of the RTlinux and a specific timer in the C167. The results show a small jitter on RTLinux (7us), C167 stand alone (79us), and C167 with executive (19us). The relative large worst case time on the C167 stand alone version seems to be an artefact of the implementation. As can be concluded from the low variation between the best case and the average case, the worst case only rarely occurs. At the moment, we do not have an explanation and further investigate the reasons for that. As in the previous measurement, the driver latency and the transmission time on the CAN-Bus are comprised in the measured times for the C167.

RTLinux min/average/max [µs]	*C167 stand alone min/average/max [µs]*	*C167 with executive min/average/max [µs]*
150 / 153 / 157	375 / 376 / 454	822 / 823 / 841

Table 4 - Local "Publish" latency for CAN

Because the lack of a sufficiently precise global time, we were not able to measure end-to-end latencies between CAN and TCP/IP directly. Thus, to assess the delays between CAN bus and TCP/IP we used round-trip measurements. Fig. 5 sketches the measurement set-up. For comparison reasons a local round-trip and a round trip involving two different TCP/IP nodes and the network delay are presented.

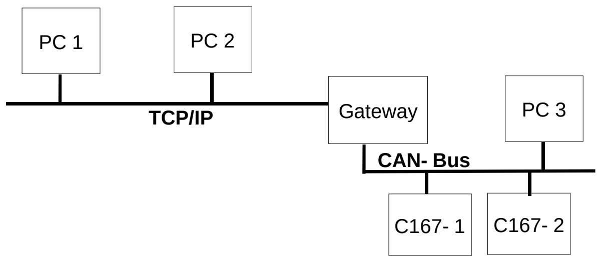

Figure 5 - Measurement Environment

As can be seen from the table, the best case and the average case are pretty close together. The worst case for a CAN to TCP/IP round-trip sometimes exhibits substantial deviations. From this, we can conclude that the worst case is due to some rare events, which, however, have to be considered when aiming at a predictable system. Because there was no additional load on the bus during the measurements, the large variations result from the low predictability of the communication through TCP/IP under the (normal) Linux operating system. This is particularly obvious in the case where the gateway is involved. The average time is about four times as long as the case where no gateway has to be crossed which is reasonable because the gateway just behaves as an additional node which is involved in both direction of the roundtrip.

Publisher on host	*Crossing the Gateway*	*Subscriber on host*	*Round trip min/average/max [µs]*
PC1	no	PC1	361 / 421 / 493
PC1	no	PC2	981 / 1020 / 1083
PC1	yes	C167-1	4060 / 4099 / 4323
C167-1	yes	PC1	4204 / 4350 / 6068
C167-1	no	PC3 (RTLinux)	3757 / 3769 / 3772
PC3 (RTLinux)	No	C167-1	3781 / 3808 / 3821
PC1	yes	PC3 (RTLinux)	1103 / 1248 / 2910

Table 5 - Round trip latencies

6.1 Code Size

The table 6 shows the code size of each of the implementations. Although not being a restriction in PC's, code size is an important issue for micro-controller based applications. As can be seen in table 6, the implementations for the C167, both with and without executive have very small code size. Considering a C167 board with a 256KB ROM, for instance, the full publisher/subscriber implementation would occupy only 2% of the available memory.

	ECB for TCP/IP	*ECH for TCP/IP*	*ECH for CAN*	*Gateway*
Linux	47.1KB	52.8KB	44.9KB	177.1KB
RTLinux	-	-	308.3KB (1)	-
C167 stand alone	-	-	5.7KB (2)	-
C167 with executive	-	-	7.1KB (3)	-

Table 6 - Code Size

(1) This includes 161.2KB of the main part and 147.1KB of the ring buffer.
(2) A ROMmable file with CAN driver is around 10.7KB.
(3) A complete system for the C167 results in a ROMmable file with approx. 13KB, including CAN driver and the small OS.

7 Conclusion

The paper describes an architecture and system support for a publisher subscriber protocol which allows to seamlessly integrate smart sensors and actuators in a large network. Clusters of such components, internally using a CAN-Bus are interconnected via a TCP/IP network resulting in a WAN-of-CAN structure. Applications can communicate via a uniform interface allover the network. As a test scenario, we use the protocol in our mobile robots which communicate over TCP/IP in a wireless IEEE 802.11 network. Smart devices can just subscribe to any sensor, like speed sensors, distance sensors, acceleration sensors or to smart optical devices [KaS01]. We use handheld Linux palmtops to monitor and remotely control these robots. In the future, we intend to exploit the protocol features for distributed sensor fusion, including multiple robots. One of the requirements for the protocol was the suitability for connecting embedded controllers. The measurements show the memory footprint and performance characteristics of the protocol. The memory footprint is sufficiently small to use the middleware also in small micro-controllers. We currently investigate the feasibility of the protocol for 8-Bit embedded controllers. Latencies and delays of messages will be a problem, when they exceed the minimum message transfer time. This means, that the node is no longer able to receive messages at the maximum transfer rate. This problem occurs for the micro-controller version of the middleware. However, this problem is not unique for our protocol and has to be solved by appropriate planning and scheduling mechanisms.
At present, there are no such mechanisms like timeliness guarantees included in UPS. For the CAN networks, we intend to integrate the scheduling and fault-tolerance mechanisms which we developed previously [LKJ99], [KaL99]. The way in which we use the CAN-Bus by UPS, particularly the structure of the CAN-IDs, allows an orthogonal treatment of naming and scheduling issues. For the communication over a wide area network, we plan to replace the TCP/IP protocol by a timely reliable broadcast for a tight cooperation of clusters and awareness mechanisms [CMV01] for wide area communication.

References

[AFF99] L. Almeida, P. Fonseca, J.A. Fonseca, Z. Mammeri - "Scheduling and Synchronization in CAN-based Distributed Systems" ICC'99, Int. CAN Conference, Turin, Italy, November 1999.

[ATV02] Mário Alves, Eduardo Tovar, Francisco Vasques, Klaus Roether, Gerhard Hammer: "Real-Time Communications over Hybrid Wired/WirelessPROFIBUS-based Networks", http://www.hurray.isep.ipp.pt/asp/list_docs.asp

[BOS91] Robert Bosch GmbH, CAN Specification Version 2.0, September 1991.

[CIA99] Can in Automation (CiA). Canopen - Application Layer and Communication Profile. CiA Draft Standard 301, V4.0, 1999

[Cas02] A. Casimiro (ed.): "Preliminary Definition of the CORTEX System Architecture", http://cortex.di.fc.ul.pt/deliverables/WP3-D4.pdf

[CKI02] CAN Kingdom, http://www.can-cia.org/cankingdom/

[CMV01] António Casimiro, Pedro Martins, Paulo Veríssimo and Luís Rodrigues, "Measuring Distributed Durations with Stable Errors", Proceedings of the 22nd IEEE Real-Time Systems Symposium, London, UK, December 2001.

[FMD02] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, M. Walther: "Time Triggered Communication on CAN", http://www.can-cia.org/can/ttcan/fuehrer.pdf

[HLS97] T.H. Harrison, D.L. Levine, D.C. Schmidt. The Design and performance of a Real-time CORBA Event Service. Proc. of the 12th Ann. Conference on Object-oriented Programming, Systems, Languages and Applications, OOPSLA, Atlanta, USA, 1997.

[KaL99] J. Kaiser, M. A. Livani, "Achieving Fault-Tolerant Ordered Broadcasts in CAN", *Proc. of the Third European Dependable Computing Conference (EDCC-3),* Prague,Sep. 1999

[KaM99] Kaiser, J. and M. Mock : "Implementing the Real-Time Publisher/Subscriber Model on the Controller Area Network (CAN)", *2nd Int'l Symposium on Object-Oriented Distributed Real-Time Computing Systems, San Malo, May 1999*

[KIM00] K. Kim, G. Jeon, S. Hong, T. Kim, and S. Kim. Integrating Subscription-Based and Connection-Oriented Communications into the Embedded CORBA for the CAN Bus. In IEEE Real-time Technology and Application Symposium, May 2000.

[KVA02] KVASER: "CAN on Bluetooth", http://www.kvaser.com/

[LKJ99] M.A. Livani, J. Kaiser, W. Jia : "Scheduling Hard and Soft Real-Time Communication in the Controller Area Network (CAN)", *Control Engineering Practice, Vol. 7 Number 12 pp. 1515-1523, Dec. 1999.*

[MAF97] S. Maffeis: "iBus - The Java Intranet Software Bus", Olsen&Associates, www.olsen.ch, 1997.

[Mor93] K. Mori, " Autonomous decentralized Systems: Concepts, Data Field Architectures, and Future Trends", *Int. Conference on Autonomous Decentralized Systems (ISADS93)*, 1993.

[NSS00] OMG: CORBA Notification Service Specification, V1.0, June 2000

[OPS93] B. Oki, M. Pfluegl, A. Seigel, D. Skeen. The Information Bus® An Architecture for Extensible Distributed Systems. 14th ACM Symposium on Operating System Principles, Ashville, NC, Dec 1993, pp.58-68.

[PAR97] Parado-Castellote, G., Schneider, S., and Hamilton, M. Ndds. The Real-Time Publish Subscribe Network. In IEEE Workshop on Middleware for Distributed Real-Time Systems and Services, San Francisco, CA, 1997.

[RAJ95] R. Rajkumar, M. Gagliardi, L Sha. The Real-Time Publisher/Subscribe Inter-Process Communication Model for Distributed Real-Time Systems: Design and Implementation. IEEE Real-time Technology and Applications Symposium, June 1995.

[RTL02] FSMLabs. The Real Time Linux Project. www.fsmlabs.com.

[SDS96] Smart Distributed Systems, Application Layer Protocol Version 2, *Honeywell Inc, Micro Switch Specification GS 052 103 Issue 3, USA, 1996*

[SIE96] Siemens AG. C167 Derivatives User's Manual 03.96 Version 2.0, 1996.

[TAL02] Talarian Corporation. Smart Sockets®, www.talarian.com.

[TiB94] Tindell, K. and A. Burns : "Guaranteed Message Latencies for Distributed Safety-Critical Hard Real-Time Control Networks", *Report YCS229, Department of Computer Science, University of York, May 1994.*

[TTC02] Time-Triggered CAN (TTCAN), CiA (CAN in Automation), http://www.can-cia.org/can/ttcan/

[VCC02] P. Verissimo, V. Cahill, A. Casimiro, K. Cheverest, A. Friday, J. Kaiser: " CORTEX: Towards Supporting Autonomous an Cooperating Sentient Entities," in *Proc. Of the European Wireless Conference*, Florence, Italy, March 2002

[ZuS95] Zuberi, K.M., and K.G. Shin:"Non-Preemptive Scheduling of messages on the Controller Area Network for Real-Time Control Applications", *Tech. Report, University of Michigan, 1995*

[PBV01] C.E.Pereira, L.B. Becker, C. Villela, C. Mitidieri, J. Kaiser: "On Evaluating Interaction and Communication Schemes for Automation Apllications based on Real-Time Distributed Objects", Proc. of the IEEE 4th International Symp. on Object-Oriented Real-Time Distributed Computing (ISORC 2001), Magdeburg, Germany, May 2001