Habilitationsschrift

Time-Triggered Transducer Networks

eingereicht an der Technischen Universität Wien
Fakultät für Informatik

von
Dipl.-Ing. Dr.techn. Wilfried Elmenreich

zur Erlangung der Lehrbefugnis
für das Fach Technische Informatik

Wien, im September 2007

Für Claudia und Gretchen

Acknowledgments

First of all I would like to thank my supervisor Prof. Dr. Hermann Kopetz for his support and advice on my work. As head of the Institute of Computer Engineering he provided me with the fruitful environment of our research group and the resources that were necessary to write this habilitation thesis.

Thanks to Andreas Steininger, Peter Puschner and Christian El Salloum for proofreading and comments on the introduction. All remaining mistakes are mine.

I also have to thank my colleagues at the institute for the friendly atmosphere and many discussions that had a positive influence to the quality of my work. Furthermore, I would like to thank my students for their interest and dedication for my research topics. Beyond science I owe a big thanks to our technician Leo and the administrative support from Maria and Traude.

Finally, I am indebted to my wife Claudia for her support, endurance and patience.

Contents

List of Figures

List of Tables

List of Acronyms

ARQ	Automatic Repeat Request
CAN	Controller Area Network
CCDL	Communication-Computation Description Language
CORBA	Common Object Request Broker Architecture
COSMIC	Cooperating Smart Devices
CP	Configuration and Planning
CWA	Confidence-Weighted Averaging
CWA	Confidence-Weighted Averaging
DECOS	Dependable Embedded Components and Systems
DM	Diagnostic and Management
DSoS	Dependable Systems of Systems
DSP	Digital Signal Processor
FDCML	Field Device Configuration Markup Language
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
IFS	Interface File System
ISMA	Idle Signal Multiple Access
LIN	Local Interconnect Network
MEMS	Micro-Electro-Mechanical Systems
MFM	Modified Frequency Modulation
MP	Multipartner
MS	Master/Slave

MTTF	Mean-Time-To-Failure
NCAP	Network-Capable Application Processor
OMG	Object Management Group
PID	Proportional-Integral-Derivative
PWM	Pulse-Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
RODL	ROund Descriptor List
ROM	Read Only Memory
RS	Real-time Service
SPLIF	Service Providing Linking Interface
SRLIF	Service Requesting Linking Interface
STI	Smart Transducer Interface
STIM	Smart Transducer Interface Module
TDMA	Time Division Multiple Access
TEDS	Transducer Electronic Data Sheet
TII	Transducer Independent Interface
TTA	Time-Triggered Architecture
TTCAN	Time-Triggered Controller Area Network
TTP	Time-Triggered Protocol
TTP/A	Time-Triggered Protocol for SAE class A applications
UART	Universal Asynchronous Receiver Transmitter
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WCAN	Wireless Controller Area Network
WCET	Worst Case Execution Time
WSDL	Web Services Description Language
WSN	Wireless Sensor Network
XML	eXtended Markup Language

Chapter 1

Introduction

The number of applications for a transducer network, that is a set of interconnected sensors and actuators that measure and/or interact with the environment, is growing. In the future, applications of transducer networks will intersect our daily life in many applications as for example environmental monitoring [Mai02], automotive and traffic applications [Kos05], civil infrastructure monitoring [Kim05], and health care [Shn05].

Whenever a system interacts with a real environment through its sensors and actuators, the aspect of time has to be considered in order to achieve a correct and meaningful behavior. Although soft real-time systems may cope with this problem by just providing enough processing and communication power, there is a fundamental difference in designing architectures that are required to guarantee hard real-time properties.

The time-triggered approach is an established paradigm for designing hard real-time systems with predictable and guaranteed behavior. The time-triggered approach has already been successfully applied to safety-critical applications in cars [Rus03], railway control systems [Hei98] and for flight-critical functions in aircraft and aircraft engines [Rus03].

We think that the application of the time-triggered approach to the area of transducer networks is beneficial from the viewpoint of (i) establishing predictable real-time response times within the network, (ii) reducing system complexity and, therefore, the design of real-time applications, by introducing an architecture with a global notion of time and well-defined interfaces, (iii) facilitating the processing of sensor data, since measurements can be synchronized and interpreted on a global timescale, and (iv) enable the coordination of timely correlated actions of actuators.

The following sections give a brief introduction to the basic concepts of the time-triggered approach and elaborate on the goals and problems encountered in transducer networks. The introduction is followed by an overview of the papers that are included in this habilitation thesis. The overview discusses the relevance of each paper and shows how each paper relates to the problem domains of time-triggered sensor networks.

1.1 The Time-Triggered Paradigm

There are two major design paradigms for constructing real-time systems, the *event-triggered* and the *time-triggered* approach. In principle, an event-triggered system follows the principle of reaction on demand. In such systems the environment enforces temporal control onto the system in an unpredictable manner (interrupts), with a lot of undesirable problems of jitter, missing precise temporal specification of interfaces and membership, scheduling etc. On the other hand, the event-triggered approach is well-suited for sporadic actions and data, low-power sleep modes, and best-effort soft real-time systems with high utilization of resources. Event-triggered systems do not ideally cope with the demands for predictability, determinism, and guaranteed latencies – requirements that must be met in a hard real-time system.

Time-triggered systems derive control from the global progression of time, thus the concept of time that appears in the problem statement appears also as basic mechanism for the solution:

Definition 1 *A real-time system is time-triggered if the control signals, such as sending and receiving of messages or recognition of an external state change are derived solely from the progression of a (global) notion of time (cf. [Kop97]).*

This approach supports a precise temporal specification of interfaces and the implementation of "temporal firewalls" to protect error propagation via control signals. The Time-Triggered Architecture (TTA), a computing infrastructure for the design and implementation of dependable distributed embedded systems [Kop03], implements the time-triggered paradigm and supports membership identification, interoperability, and replica determinism.

A basic concept in the time-triggered paradigm is the *global time*. For most real time applications it is sufficient to model time according to Newtonian physics without regarding relativistic effects [Kop02b]. Unlike the concept of logical clocks [Lam78], the global time is thus bound to physical time with a given accuracy. In order to establish a meaningful global time with a given granularity, an ensemble of physical clocks must be synchronized with a precision better than the intended granularity.

The global time is used to define the instances when communication and computation of tasks take place in a time-triggered system. Communication is realized by message send and receive operations of particular nodes. Typically, sending messages takes place in a broadcast manner. The message length and message sender are known *a priori* according to the predefined message schedule.

Computation is realized by the execution of *Simple Tasks*, that is tasks that consume their input at task start and provide their output with task completion. Simple Tasks do not have synchronization points within the task, and, therefore, cannot be blocked. For each task an *a priori* known upper bound for their Worst Case Execution Time (WCET) [Pus00] is assumed.

Using a static scheduling algorithm, the tasks and messages are scheduled to form a collision-free communication pattern where it is guaranteed that all tasks can finish in time before their results are used.

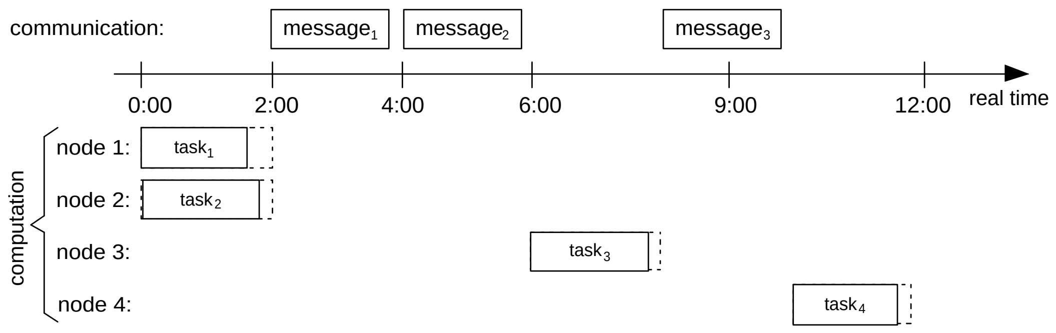


Figure 1.1: Time-triggered scheme for communication and computation

Such a communication and computation pattern forms so-called *rounds* which are periodically repeated.

1.1.1 Advantages of the Time-Triggered Approach

With respect to embedded real-time systems, the time-triggered approach has proven to show the following advantages:

- The low jitter for message transmission and task execution is especially advantageous for distributed control loops.

- The predictable communication scheme simplifies diagnosis of timing failures. Furthermore, a timing failure of a node can be confined from the bus using the concept of a bus guardian.
- The periodically transmitted messages enable a short and bounded error detection latency for timing and omission errors.
- The principle of resource adequacy guarantees the nominative message throughput independent of the network load. Problems like increasing delays at message floods or thrashing [Den68] are avoided by design.
- Due to the predefined schedule, it is possible to derive the message ID and the message sender from the time, when a message was received. Using this information enables high protocol efficiency.
- The time-triggered paradigm avoids bus conflicts using a Time Division Multiple Access (TDMA) scheme, making an explicit bus arbitration obsolete.
- By using a *sparse time base*, replica determinism between time-triggered components can be achieved without the need of complex agreement protocols.
- The TTA supports temporal composability, i. e., the constructive design of dependable distributed real-time systems out of previously validated components while retaining the previously validated properties [Kop02a].

1.1.2 Disadvantages of the Time-Triggered Approach

A time-triggered system is a specialization of an event-triggered system where only the time is used as a trigger. Therefore, there are problems, for which an event-triggered approach is better suited than a strict time-triggered scheme:

- When a system is required to achieve a low energy consumption over time, as it is the case for wireless sensor networks. In event-triggered systems, messages are created on demand, i. e., on the occurrence of respective events. In the time-triggered scheme messages and computations are periodically triggered which causes a permanent constant energy consumption even during minimum load situations.[1]

[1]However, for wireless systems with a low duty cycle the time-triggered approach can be less energy consuming since the knowledge of message transmission instants allows to suspend the receiving units for the duration between two transmissions.

- When the average response time of the system is of concern. Event-triggered systems may outperform time-triggered systems in that aspect since the latter are designed to have a constant response time independent of the system load, thus the average response time equals the worst-case response time.

- Time-triggered systems have to plan for an upper bound for the execution time of each task, in contrast an event-triggered approach can work with weaker assumptions such as a global time budget for a set of tasks.

- When it is difficult to fit messages with differing periods into a static schedule. The length of the static schedule is defined by the least common multiple of the message periods, which can lead to a very extensive static schedule causing memory problems in embedded systems.

- In wireless scenarios where a considerable rate of link failures cannot be handled by the standard time-triggered approach. If the node connectivity is also dynamic, a dynamic re-routing algorithm does not allow for a static time-triggered approach.

- When it is not possible to establish the required precision of the global time, e.g., in state-of-the-art chip design with clock frequencies of several Gigahertz, it is very difficult to distribute the clock signal across the chip [Res98, Mut99].

Most of the problems listed above appear in applications, where non-real-time or soft real-time requirements are prevalent over dependability issues such as reliability and safety.

Some system architectures come with a hybrid approach in order to overcome the disadvantages of either the time-triggered or the event-triggered approach. For example, Time-Triggered Ethernet [Kop05], Flexray [Fle05], and COSMIC [Kai05] support time-triggered as well as event-triggered traffic. For the TTA, a layered approach has been proposed to extend the system by event-triggered channels [Elm03]. An implementation of this idea is given in the DECOS architecture [Sal06].

1.2 Scope of Transducer Networks

A transducer is a device for converting energy from one form to another for the purpose of measurement of a physical quantity or for information transfer. Thus, a transducer is a generic term for both sensor and actuator.

By means of small low-cost embedded microcontroller devices, the concept of a *smart transducer* can be realized:

Definition 2 *An intelligent or smart transducer is the integration of an analog or digital sensor or actuator element, a processing unit, and a communication interface. In case of a sensor, the smart transducer transforms the raw sensor signal to a standardized digital representation, checks and calibrates the signal, and transmits this digital signal to its users via a standardized communication protocol. [Kop01, p. 71]*

The standard communication interface allows us to build a network of transducers, typically including a gateway component that provides access to the transducer data for a system outside the transducer network domain. Additionally, it is also possible to use the local processing power of the transducer nodes for data processing and control within the transducer network. Typical applications are sensor data fusion and closed control loops.

In the industrial automation domain, transducer networks can be found in the form of sensor/actuator networks or fieldbus systems. While the former usually do not provide computation capabilities within the network, a fieldbus forms a control and instrumentation system architecture in which each device has its own intelligence.

Transducer networks are also used in building automation and transportation systems such as modern automobiles, railway and avionic systems. While in building automation mostly event-triggered protocols are used, in transportation systems time-triggered protocols such as Time-Triggered Protocol (TTP) [TTA03], Flexray [Fle05], and Local Interconnect Network (LIN) [Aud99] can be found.

Another application of smart transducer networks are wireless sensor networks:

Definition 3 *A Wireless Sensor Network (WSN) is a wireless network consisting of spatially distributed autonomous devices using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at different locations. [Röm04]*

In most wireless networks, the event-triggered approach is prevalent, an exception will be given by one of the papers that is included in this habilitation thesis.

1.2.1 Goals of Smart Transducer Networks

Smart transducer networks are expected to bring the following benefits:

- Easy configuration and deployment of smart transducer networks based on local intelligence and self-describing functions of the nodes providing automated tool support for plug-and-play functionality.
- Reusability of smart transducer nodes with a standard interface in different applications and, in consequence, cost benefits through mass production.
- Standardized protocol and time models, allowing the system integrator to abstract over the particular transducer node implementations.
- Low noise pickup for measurements due to short analog transmission lines to local A/D converters. Data transmission takes place in the digital domain enhanced with error-correcting features if necessary.
- Possibility for data processing within the network, e. g., by algorithms for signal conditioning, calibration, or sensor fusion, thus reducing complexity, network bandwidth (if the processed data is more dense than the raw data), and processing requirements of the central system.

In the opinion of the author, none of these possibilities have been already exploited to its full extend, which leads to several research possibilities in the area of smart transducers.

1.2.2 Problems of Smart Transducer Networks

In the following we sketch some current problems in smart transducer networks, however the list is not expected to be exhaustive.

Defining Standard Architectures

One problem in smart transducer networks is that there are no standard architectures that support a broad majority of applications. Certainly, this goal is difficult to achieve, since each application domain comes with different sets of requirements. However, for the Internet, such a standard has been established by the TCP/IP protocol, which provides a set of standardized architectural services used by a vast majority of applications.

In contrast, for smart transducer applications, there exist different specific solutions depending on the required bandwidth, the targeted costs, the required dependability, the real-time requirements, size and energy constraints, and existing legacies in a particular domain.

In the field of industrial automation, the struggle for a standard has become popular under the term *fieldbus wars* [Fel02]. The reluctance of vendors to agree on a single standard has led to the standardization of a set of eight existing fieldbusses [DKE02].

For wireless sensor networks, often proprietary protocol solutions depending on the the application requirements are used besides standards like IEEE 802.15.4 (ZigBee) and S-MAC. Radio standards like Bluetooth are usually disregarded because of cost and energy constraints. Above protocol level, a widely used platform for wireless sensor networks is given by the Berkeley Motes and the TinyOS middleware.

The first paper included in this thesis (see Chapter 2) deals with a standard for a smart transducer interface that provides an open interface to both, the physical layer and the application layer by introducing a flexible and simple interface concept named the Interface File System (IFS).

Automatic Configuration

Automatic configuration, often called Plug-and-Play, describes the configuration of a system without manual intervention. Due to complexity and cost reasons such an approach is desirable in all domains where systems with several devices have to be assembled.

The IEC 61804 and IEEE 1451 already describe standardized ways to enhance devices with self-describing features. For the Object Management Group (OMG) Smart Transducer Interface such an approach has been proposed in [Pit02]. However, these approaches provide rather a plug-and-participate functionality than a true automatic configuration, because for the latter case the configuration tool requires a semantic understanding of the application goals [Pit05]. Future configuration approaches are expected to be more sophisticated in order to disburden the system integrator of tasks that can be solved automatically by integrating and exploiting data from the employed transducer nodes with knowledge about the application by the configuration tool.

The papers in Chapter 3 and 4 contribute to this topic: The paper in Chapter 3 presents the architecture of three different smart transducer approaches and discusses their principles of operation and configuration support. The paper in Chapter 4 establishes a platform-independent application model for distributed embedded applications that enables an automatic configuration for the deployment of the application onto a time-triggered transducer network.

Data Processing in Transducer Networks

Although there exist several well-explored distributed algorithms for the aggregation of sensor data in wireless sensor networks, many problems like sensor data fusion cannot be easily mapped to an arbitrary transducer network without cross-checking and adjusting several protocol-dependent parameters.

Due to a lack of effective standard architectures, many algorithms presented in the literature rely on particular properties of the transducer network like specific timing models, data types or even require a specific media access control method.

Therefore, data processing in time-triggered transducer networks require further research regarding

- the analysis of the underlying assumptions and architectures of existing algorithms and methods for the purpose of using them in other architectures such as time-triggered transducer networks,
- the construction of algorithms that exploit the properties of time-triggered transducer networks in order to provide better performance.

The papers in Chapter 5 and 6 describe new algorithms and the extension of an existing approach for sensor fusion in a time-triggered transducer network.

1.3 Overview on Selected Papers

The following paragraphs give a short overview of the papers that are included in this thesis. For each paper, the research contribution and the relevance to the problem domains of time-triggered transducer networks are stated.

1.3.1 Time-Triggered Smart Transducer Networks

Wilfried Elmenreich. IEEE Transactions on Industrial Informatics, Volume 2, Number 3, 2006, pages 613–624.

This paper generally describes the application of the time-triggered approach for smart transducer networks and gives a specific case study that implements the OMG Smart Transducer Interface Standard. The case study uses the TTP/A protocol, a real-time-capable time-triggered master-slave protocol that supports low-cost embedded hardware such as small 8bit microcontrollers with a few hundred bytes of working memory and a few kilobytes of program memory. The TTP/A protocol establishes interfaces to a smart transducer node for real-time operation, maintenance, and configuration. All interfaces are based on a unique concept named Interface File System (IFS).

The approach is demonstrated by a case study implementation of a smart transducer and the design of a configuration infrastructure based on Common Object Request Broker Architecture (CORBA).

1.3.2 Interface Design for Real-Time Smart Transducer Networks – Examining COSMIC, LIN, and TTP/A as Case Study

Wilfried Elmenreich, Hubert Piontek, and Jörg Kaiser. Proceedings of the 15th International Conference on Real-Time and Network Systems (RTNS'07), Nancy, France, 2007, pages 195-204.

This paper analyzes the interface models of the three real-time smart transducer networks COSMIC, LIN, and TTP/A with respect to their support for real-time interaction, configuration and abilities for self-description.

Although some basic constraints lead to common structures among the different solutions, like differentiating between configuration and planning, diagnostic and management, and real-time service, the three approaches apply different interaction patterns (publish/subscribe, pull, time-triggered) to provide their real-time communication service.

The paper discusses also interface models that are apt to interconnect networks using different interaction patterns, where the shared memory concept of the TTP/A network interface is the most promising.

1.3.3 Modeling Distributed Embedded Applications on an Interface File System

Wilfried Elmenreich, Stefan Pitzek and Martin Schlager. IEEE International Symposium on Object-Oriented Real-time Distributed Computing, Vienna, Austria, May 2004, pages 175–182.

Based on the concepts which were introduced in the previous paper, this paper describes a platform-independent model of services in a distributed real-time application. The implementation of each service is encapsulated behind its interfaces for real-time operation, maintenance, and configuration. The IFS takes the central role of acting as a generic interface for all three interface types. The proposed model supports the description of a distributable application by defining data flow and real-time constraints between services. Both, the service properties and the application constraints are expressed in a machine-readable eXtended Markup Language (XML) format that support automatic processing by tools for task and message scheduling as well as for the verification of the end-to-end real-time constraints of the application.

The paper illustrates the concept by a case study consisting of an application with five services that is mapped to a distributed TTP/A network. The presented approach enhances the basic configuration approach by introducing a platform-independent model of an application that can be mapped to a time-triggered smart transducer network.

1.3.4 Fusion of Continuous-Valued Sensor Measurements using Confidence-Weighted Averaging

Wilfried Elmenreich. International Journal of Vibration and Control. To appear.

This paper presents a sensor fusion algorithm that can be used in smart transducer networks in order to perform data refinement and reduction within the network.

The algorithm works without knowledge of the application or target system and is stateless (unlike the Kalman filter), which supports easy integration and recovery. In contrast to intersection-based methods the algorithm is based on a statistical model which provides a more accurate model of measurement errors. Each measurement is represented by a value, a time instant, and its uncertainty. Since the approach is stateless, it requires synchronous measurements, a property that is well supported by time-triggered systems.

In order to support an efficient implementation, the paper also sketches a way, how the uncertainty can be efficiently represented in TTP/A messages. Thus, the presented method contributes to the domain of data processing in the network by introducing a new method that exploits the properties of the time-triggered paradigm.

1.3.5 Fault-Tolerant Certainty Grid

Wilfried Elmenreich. 11th International Conference on Advanced Robotics, Coimbra, Portugal, June–July 2003, volume III, pages 1576–1581.

This paper elaborates on fusion of sensor data into a generic representation of obstacles in a flat environment. The generic representation is implemented in the form of a certainty grid that maps the position of obstacles and free space around a mobile robot.

In order to generate such a grid from unreliable sensor data, two new approaches for robust fusion named Fault-Tolerant Certainty Grid and Robust Certainty Grid are described. Both approaches are based on the existing standard implementation of the certainty grid, but instead using Bayesian inference the presented methods use different algorithms to integrate measurements into the certainty grid.

The proposed algorithms are more robust against faulty measurements and thus applicable to low-cost networks of unreliable sensors. A case study shows the application of the method using commercial-off-the-shelf distance sensors. Both approaches are evaluated by simulation and implemented in a mobile robot with a distributed smart transducer network.

References

[Aud99] Audi AG, BMW AG, DaimlerChrysler AG, Motorola Inc. Volcano Communication Technologies AB, Volkswagen AG, and Volvo Car Corporation. LIN Specification and LIN Press Announcement. SAE World Congress Detroit, http://www.lin-subbus.org, 1999.

[Den68] P. J. Denning. Thrashing: Its Causes and Prevention. In *Proceedings AFIPS Fall Joint Computer Conference*, volume 33, pages 915–922, 1968.

[DKE02] DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE. *Information about the International Fieldbus Standards Series IEC 61158 and 61784*, Feb. 2002. Available at http://www.dke.de.

[Elm03] W. Elmenreich, R. Obermaisser, and P. Peti. A Model for Reactive Systems Supporting Varying Degrees of Synchrony. In *Proceedings of IEEE International Conference on Computational Cybernetics*, pages 275–280, Aug. 2003.

[Fel02] M. Felser and T. Sauter. The Fieldbus War: History or Short Break Between Battles? In *Proceedings of the 4rd IEEE International Workshop on Factory Communication Systems*, pages 73–79, Västerås, Sweden, Aug. 2002.

[Fle05] Flexray Consortium. *FlexRay Communications System Protocol Specification Version 2.1*, 2005. Available at http://www.flexray.com.

[Hei98] G. Heiner and T. Thurner. Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems. In *Proceedings of the The Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing*, pages 402–407, 1998.

[Kai05] J. Kaiser, C. Brudna, and C. Mitidieri. COSMIC: A Real-Time Event-Based Middleware for the CAN-bus. *Journal of Systems and Software*, 77(1):27–36, July 2005.

[Kim05] S. Kim. Wireless Sensor Networks for Structural Health Monitoring. Master's Thesis, University of California at Berkeley, Department of Electrical Engineering and Computer Sciences, 2005.

[Kop97] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[Kop01] H. Kopetz, M. Holzmann, and W. Elmenreich. A Universal Smart Transducer Interface: TTP/A. *International Journal of Computer System Science & Engineering*, 16(2):71–77, Mar. 2001.

[Kop02a] H. Kopetz and R. Obermaisser. Temporal Composability. *IEE's Computing & Control Engineering Journal*, 13(4):156–162, Aug. 2002.

[Kop02b] H. Kopetz and N. Suri. Compositional Design of RT Systems: A Conceptual Basis for Specification of Linking Interfaces. Research Report 37/2002, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[Kop03] H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1):112–126, Jan. 2003.

[Kop05] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer. The Time-Triggered Ethernet (TTE) Design. In *Proceedings of the 8th International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, pages 22–33, Seattle, WA, USA, May 2005.

[Kos05] T. Kosch. Technical Concept and Prerequisites of Car2Car Communication. In *Proceedings of the 5th European Congress and Exhibition on ITS*, Hanover, Germany, June 2005.

[Lam78] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[Mai02] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 88–97, 2002.

[Mut99] J. Muttersbach, T. Villiger, H. Kaeslin, N. Felber, and W. Fichtner. Globally-Asynchronous Locally-Synchronous Architectures to Simplify the Design of On-chip Systems. In *Proceedings of the twelfth*

Annual IEEE International ASIC/SOC Conference, pages 317–321, Washington DC, USA, Sep. 1999.

[Pit02] S. Pitzek. Description Mechanisms Supporting the Configuration and Management of TTP/A Fieldbus Systems. Master's Thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[Pit05] S. Pitzek and W. Elmenreich. Plug-and-Play: Bridging the Semantic Gap Between Application and Transducers. In *Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA05)*, volume 1, pages 799–806, Catania, Italy, Sep. 2005.

[Pus00] P. Puschner and A. Burns. A Review of Worst-Case Execution-Time Analysis. *Journal of Real-Time Systems*, 18(2/3):115–128, May 2000.

[Res98] P. J. Restle and A. Deutsch. Designing the Best Clock Distribution Network. In *Proceedings of the Symposium on VLSI Circuits Digest of Technical Papers*, pages 2–5, 1998.

[Röm04] K. Römer and F. Mattern. The Design Space of Wireless Sensor Networks. *IEEE Wireless Communications*, 11(6):54–61, Dec. 2004.

[Rus03] J. Rushby. A Comparison of Bus Architectures for Safety-Critical Embedded Systems. Technical Report NASA/CR-2003-212161, National Aeronautics and Space Administration, Langley Research Center, Mar. 2003.

[Sal06] C. El Salloum, R. Obermaisser, B. Huber, H. Kopetz, and N. Suri. Supporting Heterogeneous Applications in the DECOS Integrated Architecture. In *Proceedings of the International DECOS Workshop at the Mikroelektroniktagung 2006*, Oct. 2006.

[Shn05] V. Shnayder, B. Chen, K. Lorincz, T. R. F. Fulford-Jones, and M. Welsh. Sensor Networks for Medical Care. Technical Report TR-08-05, Harvard University, Apr. 2005.

[TTA03] TTAGroup. *Specification of the TTP/C Protocol V1.1*, 2003. Available at http://www.ttagroup.org.

Chapter 2

Time-Triggered Smart Transducer Networks

Wilfried Elmenreich. IEEE Transactions on Industrial Informatics, Volume 2, Number 3, 2006, pages 613–624.

The time-triggered approach is a well-suited approach for building distributed hard real-time systems. Since many applications of transducer networks have real-time requirements, a time-triggered communication interface for smart transducers is desirable, however such a time-triggered interface must still support features for monitoring, maintenance, plug-and-play, etc.

The approach of the OMG Smart Transducer Interface consists of clusters of time-triggered smart transducer nodes that contain special interfaces supporting configuration, diagnostics, and maintenance without affecting the deterministic real-time communication. This paper discusses the applicability of the time-triggered approach for smart transducer networks and presents a case study application of a time-triggered smart transducer network.

2.1 Introduction

With the advent of modern microcontrollers it became feasible to build low-cost smart transducers by equipping sensors and actuators with a microcontroller and a standard network interface. Several smart transducers are connected to a cluster using a standard or non-standard fieldbus network. Many applications that interact with the environment via transducers have real-time requirements, that is to make correct actions at the right time. So there is a need for an appropriate real-time communication interface for smart transducers.

There are two major design paradigms for implementing distributed real-time systems, the event-triggered and the time-triggered approach. Simplified, an event triggered system follows the principle of reaction on demand. In such systems, the environment enforces temporal control onto the system in an unpredictable manner (interrupts), with all the undesirable problems of jitter, missing precise temporal specification of interfaces and membership, scheduling etc. On the other hand, the event-triggered approach is well-suited for sporadic action/data, low-power sleep modes, and best-effort soft real-time systems with high utilization of resources. Event-triggered systems do not ideally cope with the demands for predictability, determinism, and guaranteed latencies – requirements that must be met in a hard real-time system. Time-triggered systems derive all triggers for communication, computation, sensing and control by the global progression of time. In this approach the concept of time is prevalent in the problem statement as well as in the provided solution.

The objective of this paper is to present a time-triggered approach for smart transducer networks that supports the hard real-time requirements of embedded applications while still providing features for maintenance and plug-and-play.

The remaining parts of this paper are structured as follows: The next section identifies basic requirements for smart transducer networks. Section 2.3 depicts the generic model of a time-triggered system. Section 2.4 describes the OMG Smart Transducer Standard, which incorporates a time-triggered communication interface to smart transducers while fulfilling the requirements for a smart transducer. Section 2.6 gives an overview on related work on time-triggered smart transducer networks. Section 2.7 concludes the paper.

2.2 Requirements for Smart Transducers

We make the following assumption: The sensors and actuators in the system are distributed and implemented as smart transducers with a network interface. The network connects these smart transducers to a communication system with broadcast characteristics (e. g., bus or star topology). The network also contains local intelligence, e. g., for feedback control purposes or sensor information processing. This intelligence is either implemented in the processing unit of the smart transducers or provided by separate control nodes.

Large transducer networks will be divided into clusters where each cluster connects a set of transducers via a bus. A gateway node exports the interfaces of the nodes in the cluster to a backbone network.

We have identified the following requirements for smart transducers in such a network:

Real-Time Operation: Most applications for transducers, especially in the fields of process automation, automotive and avionic networks, require timely actions, e. g., for information gathering, sensor processing and actuator setting. Thus, a smart transducer should provide a real-time interface that allows for such a coordination.

Complexity management: The number of sensors and actuators employed in a typical system has drastically increased in the last two decades. Thus, a smart transducer should provide means to manage the system complexity when composing or changing a network of transducers, e. g., by supporting electronic datasheets. Electronic datasheets contain a machine-readable self-description of the transducer which can be used to support a plug-and-play-like computer aided configuration (cf. [Ecc98]).

Maintenance support: Systems which are in operation for an extended period of time usually require maintenance access to smart transducers, e. g., for reading sensor logs, calibration or trimming of the sensor's output.

Often, the information to be monitored is not fully covered by the data exchanged via the real-time interface. Therefore, the monitoring operation requires an extra data channel for communication of these additional data. In this case, it is required that the real-time traffic among the transducers is not affected by the monitoring operation in order to avoid a "Probe Effect" [Gai86, McD89] on the system.

An appropriate interface is essential for supporting effective maintenance methods such as Condition-Based Maintenance [Ben04] where affected components are repaired or replaced before their failure causes greater costs.

Deterministic Behavior: A system is deterministic, if a given set of inputs always leads to the same system output. Determinism is especially important if replicated transducers are used to enhance the dependability of an application. For real-time systems, a deterministic system must, for a given set of inputs, always produce the same output with regard to values and timing.

Since transducers operate on the borderline to the analog process environment, determinism is difficult to achieve. For example, in the case

Time	[0:00,3:00)	[3:00,5:00)	[5:00,8:00)	[8:00, 12:00)
Node A	send	receive	receive	execute task
Node B	receive	send	receive	send
Node C	receive	receive	execute task	receive
Node D	receive	receive	idle	receive

Table 2.1: Example for a time-triggered schedule

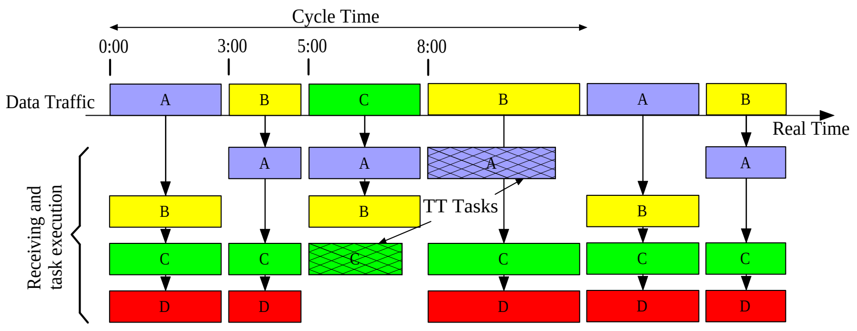


Figure 2.1: Execution of time-triggered schedule

of a sensor, the input comes from the process environment, which is an analog system. Thus, even a digital sensor will not be exactly value deterministic due to digitalization and intrinsic sensor errors. If a system contains such sources of indeterminism, consistency must by achieved by mechanisms like inexact voting [Par92], sensor fusion [Elm01a], and sparse time [Kop03].

2.3 The Time-Triggered Approach

The core mechanism of a time-triggered system is very simple. A global schedule defines for each node which action it has to take at a given point in time.

This schedule is executed periodically. A prerequisite for time-triggered systems is that all communication partners agree on the current execution state of the schedule and that the duration of all communication and computation activities are bounded and an upper bound for this duration is known.

An example for such a schedule is given in Table 2.1. Within a cycle, depicted by the *cycle time*, each message is scheduled at a predefined point in time – in this example, a message from node A is scheduled at time 0:00, a message from node B at 3:00, a message from node C is scheduled at 5:00 and another message from node B is scheduled at 8:00. Figure 2.1 depicts the execution of this schedule.

For a network of transducers, the time-triggered approach comes with the following advantages:

Global time: The global synchronized time is a requirement and a feature in time-triggered systems. The global time must be established by periodic clock synchronization in order to enable time-triggered communication and computation [Kop97, p. 52].

In case of a smart transducer, the global time provides each measurement with a timestamp that can be globally interpreted.

Conflict-free bus arbitration: Time-triggered systems need no explicit arbitration mechanism for bus access, since all communication actions are scheduled at predefined points in time. This simplifies communication design by making it possible to omit message retry mechanisms and special data encodings for detecting data collisions on the bus. Moreover, the electrical specification of the bus system is not required to cover the case of multiple partners concurrently accessing the communication medium as senders.[Kop97, p. 176]

Synchronization of distributed actions: The time-triggered schedule allows a precise coordination of actions in the network. Examples for such synchronization actions are:

- Synchronous measurements by several distributed sensors: If the measured variable is a fast moving real-time value, unsynchronized measurements from multiple sensors will lead to significantly different results.
- Cascaded measurements to avoid interference: Sensors that emit an active signal, for example ultrasonic distance sensors, may interfere with each other if the measurement is started concurrently.
- Synchronous actuating: Applications where two or more actuators are manipulating the same process might require synchronous action. An example for this case is an application with multiple servos applied to the same shaft, where an unsynchronized execution leads to increased electrical current flow and load for the servo that actuates first.

Determinism: Because of the time-triggered coordination, sources of indeterminism like race conditions are removed by design. Time-triggered systems are therefore deterministic in the value and in the time domain. This factor is especially important in the case of replicated systems where voting is used on the outputs to enhance dependability (cf. replica determinsm [Pol94]).

Note that time-triggered communication alone, i. e., using a TDMA communication scheme, is not sufficient to establish a time-triggered architecture. For example in the LIN system [Aud99], the master follows a time-triggered communication schedule, while the interface to the LIN nodes operates on a polling principle. Thus, LIN does not support synchronized measurements of multiple sensors within one cluster.

The most prominent example for a time-triggered approach is the Time-Triggered Architecture [Kop03], which provides a highly dependable real-time communication service with a fault-tolerant clock synchronization scheme and error detection of faulty nodes. This architecture is suitable to build ultra-dependable computer systems for safety-critical applications, where a Mean-Time-To-Failure (MTTF) of better than 10^9 hours is required [Sur95, Kop04].

2.4 OMG Smart Transducer Standard

In December 2000 the OMG called for a proposal of a Smart Transducer Interface (STI) standard [OMG00]. In response, a new standard has been proposed that comprises a time-triggered transport service within the distributed smart transducer network and a well-defined interface to a CORBA environment. The key feature of the STI is the concept of an IFS that contains all relevant transducer data. This IFS allows different views of a system, namely a real-time service view, a diagnostic and management view, and a configuration and planning view. The interface concept encompasses a communication model for transparent time-triggered communication. This STI standard has been finalized by the OMG in January 2003 [OMG03].

The STI standard defines a smart transducer system as a system comprising of several clusters with transducer nodes connected to a bus. Via a master node, each cluster is connected to a CORBA gateway. The master nodes of each cluster share a synchronized time that supports coordinated actions (e. g., synchronized measurements) over transducer nodes in several clusters. Each cluster can address up to 250 smart transducers that communicate via a cluster-wide broadcast communication channel. There may be redundant shadow masters to support fault tolerance. One active master controls the communication within a cluster (in the following sections the term master refers to the active master unless stated otherwise). Since smart transducers are controlled by the master, they are called slave nodes. Figure 2.2 depicts an example for such a smart transducer system.

It is possible to monitor the smart transducer system via the CORBA interface without disturbing the real-time traffic.

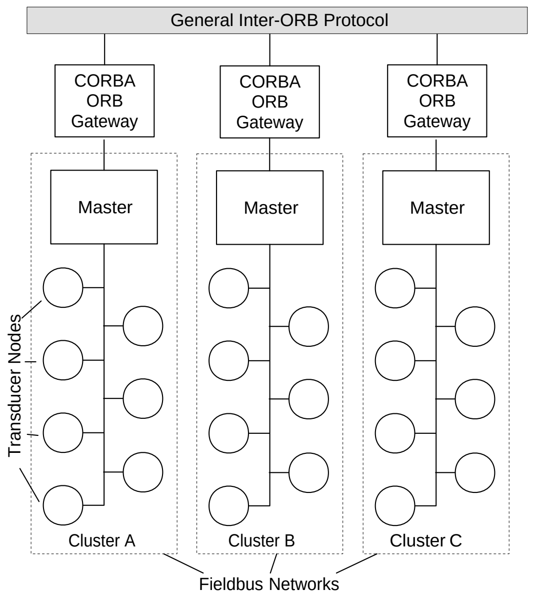


Figure 2.2: Multi-Cluster Architecture with CORBA Gateway

The STI standard is very flexible concerning the hardware requirements for smart transducer nodes, since it only requires a minimum agreed set of services for a smart transducer implementation, thus supporting low-cost implementations of smart transducers, while allowing optional implementation of additional standard features.

The information transfer between a smart transducer and its client is achieved by sharing information that is contained in an internal IFS, which is encapsulated in each smart transducer.

Interface File System. The IFS [Kop01] provides a unique addressing scheme to all relevant data in the smart transducer network, i. e., transducer data, configuration data, self-describing information, and internal state reports of a smart transducer. The values that are mapped into the IFS are organized in a static file structure that is organized hierarchically representing the network structure (Table 2.2).

Communication via temporal firewalls. A time-triggered sensor bus will perform a periodical time-triggered communication to copy data from the IFS to the fieldbus and to write received data into the IFS. Thus, the IFS is the source and sink for all communication activities. Furthermore, the IFS acts as a temporal firewall that decouples the local transducer application from

Element	Size	Description
Cluster name	8 bit	Identifies a particular cluster. Native communication (without routing) among nodes is only possible within the same cluster.
Node alias	8 bit	The node alias or logical name selects a particular node. Some values have an associated special function, e. g., alias 0 addresses all nodes of a cluster in a broadcast manner
File name	6 bit	The file name addresses a certain file within a node. A subset of files, the system files, have a special meaning in all nodes. Each service of a node is mapped onto a file containing sections for the service providing and service requesting linking interface as well as for configuration/planning and diagnosis/management data.
Record number	8 bit	Each file has a statically assigned number of records. The record number addresses the record within the selected file. Each record contains 4 data bytes. Note that each file contains only the necessary number of records, thus, the number of addressable records is statically defined for each file.

Table 2.2: Hierarchical structure of an IFS address

the communication activities. A temporal firewall [Kop97b] is a fully specified interface for the unidirectional exchange of state information between a sender/receiver over a time-triggered communication system. The basic data and control transfer of a temporal firewall interface is depicted in Figure 2.3, showing the data and control flow between a sender and a receiver. The IFS at the sender forms the output firewall of the sender and the IFS of the receiver forms the input firewall of the receiver. Thus, small timing errors at the sender do not propagate through the communication channel (significant timing errors are recognized as a failure).

Flow control using information push and pull paradigms. The sender deposits its output information into its local IFS according to the information *push* paradigm, while the receiver must *pull* the input information out of its local IFS (non-consumable read) [Elm01b]. In the information push model the sender presses information on the receiver. It is ideal for the sender, because the sender can determine the instant for passing outgoing information to the

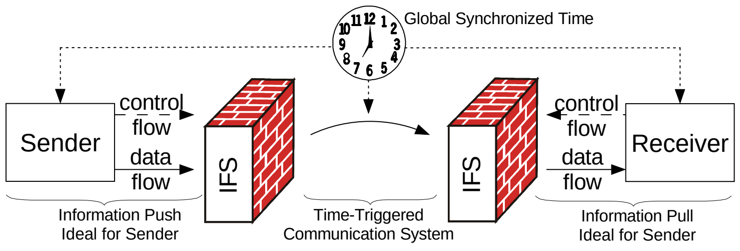

Global Synchronized Time
Sender
control flow
data flow
IFS
Time-Triggered Communication System
IFS
control flow
data flow
Receiver
Information Push Ideal for Sender
Information Pull Ideal for Sender

Figure 2.3: Temporal Firewall

communication system. The information pull model is ideal for the receiver, since tasks of the receiver will not be interrupted by incoming messages. The transport of the information is realized by a time-triggered communication system that derives its control signals autonomously from the progression of time. The instants when typed data structures are fetched from the sender's IFS and the instants when these typed data structures are delivered to the receiver's IFS are common knowledge of the sender and the receiver. A predefined communication schedule defines time, origin and destination of each communication activity. Thus, the IFS acts as a *temporally specified interface* that decouples the local transducer application from the communication task.

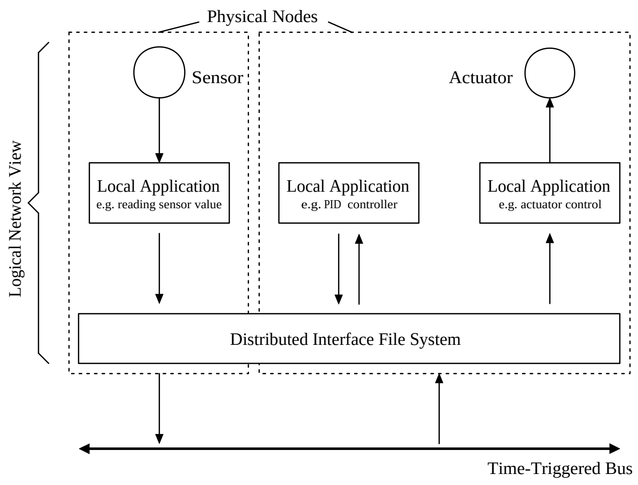


Figure 2.4: Logical network view of a distributed smart transducer application

2.4.1 Interface Separation

If different user groups access the system for different purposes, they should only be provided with an interface to the information relevant for their respective purpose [Ran97]. Therefore, interfaces for different purposes may differ by

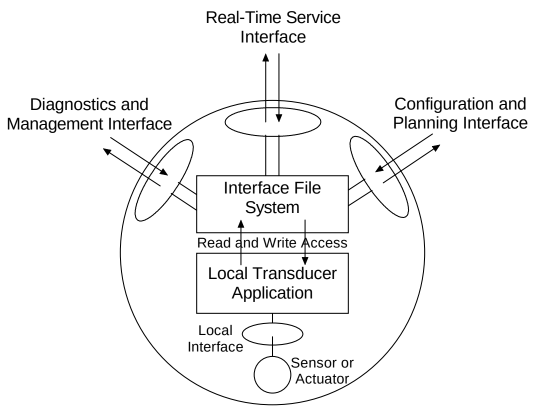

Real-Time Service Interface
Diagnostics and Management Interface
Configuration and Planning Interface
Interface File System
Read and Write Access
Local Transducer Application
Local Interface
Sensor or Actuator

Figure 2.5: The three interface types to a Smart Transducer Node

the accessible information and in the temporal behavior of the access across the interface. As depicted in Figure 2.5, the STI specifies three different interface types to a smart transducer:

DM interface: This is a *diagnostic and management* interface. It establishes a connection to a particular smart transducer node and allows reading or modifying of specific IFS records. Most sensors need parametrization and calibration at startup and continuously collect diagnostic information to support maintenance activities. For example, a remote maintenance console can request diagnostic information from a certain sensor. The DM interface is usually not time-critical.

CP interface: The *configuration and planning* interface allows the integration and setup of newly connected nodes. It is used to generate the "glue" in the network that enables the components of the network to interact in the intended way. Usually, the CP interface is not time-critical.

RS interface: The *real-time service* interface performs a periodic communication with predictable timing behavior among the smart transducer nodes. Communicated data is usually data from sensors and for actuators, but may also involve communication to and from processing nodes. This view employs sensors for producing periodic observations of real-time entities in the environment. For example, a temperature sensor periodically sends the observed and locally preprocessed sensor value to the temporal firewall of the master. Since in a time-triggered system the time interval between sensing the environment and presenting the sensor value at the temporal firewall [Kop97b] of the master is known a priori, it is possible to perform a feed-forward state estimation of the sensor value at the sensor node in such a way, that the delivered sensor value is a good estimate of the real-time entity's actual state at the point in time of delivery.

Naming and addressing. Each transducer can contain up to 64 files in its IFS. An IFS file is an index sequential array of up to 256 records. A record has a fixed length of four bytes (32 bits). An IFS record is the smallest addressable unit within a smart transducer system. Every record of an IFS file has a unique hierarchical address (which also serves as the global name of the record) consisting of the concatenation of the cluster name, the logical name, the file name, and the record name.

Besides access via the master node, the local applications in the smart transducer nodes can also execute a clusterwide application by communicating directly with each other.

Figure 2.4 depicts the network view for such a clusterwide application. Note that the actual communication between physical nodes becomes transparent for the local applications since they exchange their data only via the IFS.

The IFS of each smart transducer node can be accessed via the RS interface, the DM interface, and the CP interface for different purposes. All three interface types are mapped onto the fieldbus communication protocol, but with different semantics regarding timing and data protection.

2.4.2 Fieldbus Communication Protocol

A time-triggered transport service following the specification of the STI has been implemented in the time-triggered fieldbus protocol TTP/A [Kop02].

The bus allocation is done by a TDMA scheme. Communication is organized into rounds consisting of several TDMA slots. A slot is the unit for transmission of one byte of data. Data bytes are transmitted in a standard Universal Asynchronous Receiver Transmitter (UART) format. The first byte of a round is a message from the master called fireworks byte, since this message acts as a signal to all nodes for triggering a communication round.

The fireworks byte defines the type of the round. The protocol supports eight different firework bytes encoded in a message of one byte using a redundant bit code [Hai00] supporting error detection.

Generally, there are two types of rounds:

Multipartner (MP) round: This round consists of a configuration dependent number of slots and an assigned sender node for each slot. The configuration of a round is defined in a data structure called ROund Descriptor List (RODL). The RODL defines which node transmits in

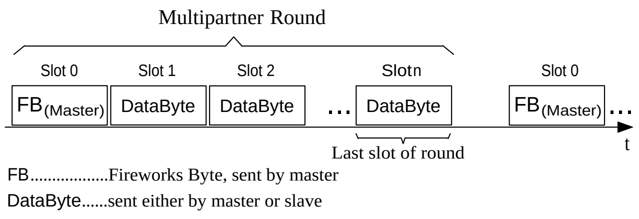


Figure 2.6: A TTP/A Multipartner Round

a certain slot, the operation in each individual slot, and the receiving nodes of a slot. RODLs must be configured in the slave nodes prior to the execution of the corresponding multipartner round. An example for a multipartner round is depicted in Figure 2.6.

Master/slave (MS) round: A master/slave round is a special round with a fixed layout that establishes a connection between the master and a particular slave for accessing data of the node's IFS, e. g., the RODL information. In a master/slave round the master addresses a data record in the hierarchical IFS address and specifies an action that is to be performed on that record. Supported actions are either reading, writing, or executing a record.

The master/slave rounds establish the DM and the CP interface to the transducer nodes. The RS interface is provided by periodical multipartner rounds. Master/slave rounds are scheduled periodically between multipartner rounds as depicted in Figure 2.7 in order to enable maintenance and monitoring activities during system operation without a probe effect.

2.4.3 Integrating New Nodes into the Network

New transducer nodes that are connected to a cluster must be first configured before they can take part in the communication. A plug-and-play configuration consists of at least three tasks: to identify the new nodes, to obtain the documentation, and to download the configuration.

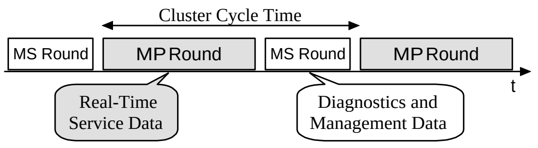


Figure 2.7: Recommended TTP/A Schedule

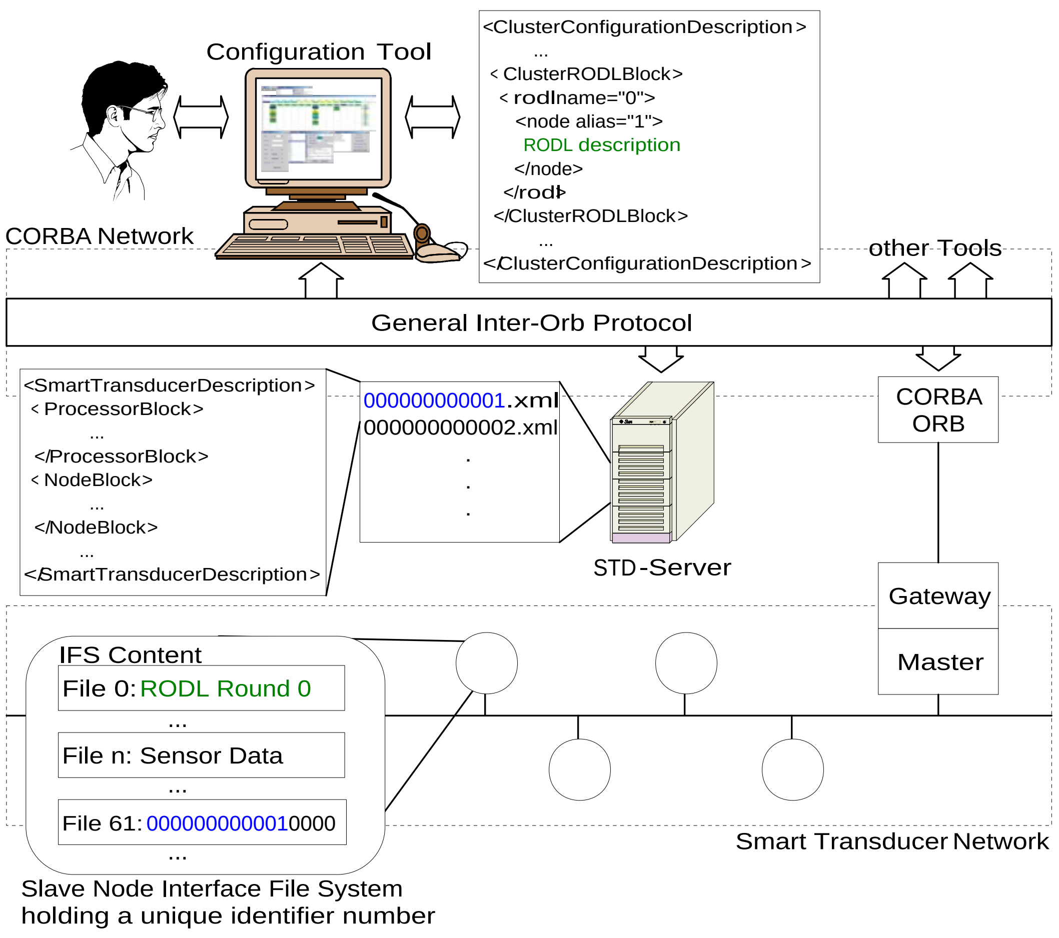


Figure 2.8: Accessing a node's datasheet

While new node identification is trivial for many networks, it is a difficult task in networks where deterministic behavior is achieved by master-slave addressing. The time-triggered smart transducer network uses a baptizing method for identification and configuration of new nodes that does not affect the determinism of the real-time communication of the network.

Until a logical name has been assigned to a node, it does not take part in the multi-partner rounds. The baptize algorithm [Elm02b] is executed by the master to see which nodes are connected to the TTP/A bus and to assign each of them a logical name, which is unique in the current TTP/A cluster.

This mechanism performs a binary search on all physical node names. A physical name is unique for every TTP/A node within the entire universe of TTP/A nodes. The identification of a new node takes 64 iterations. After finding the unique identifier of a node, a new logical name must be assigned to this node. The unique identifier of a node consists of a part that describes the generic node type (the *series* number) and a part that is used to distinguish between multiple instances of a transducer type (the *serial* number). The series number establishes a reference to the node's electronic datasheet containing the necessary information for integrating the node into the network. Datasheet information is uniformly represented in XML and can be accessed via a CORBA service. The descriptions [Pit03] consist of a cluster configuration part and a smart transducer description part (cf. device description approaches like electronic datasheets from IEEE 1452.2 [IEE97], the IEC 62390 common automation device profile [IEC05], or the Field Device Configuration Markup Language (FDCML) [FDC05]).

Figure 2.8 depicts an example for accessing a node's datasheet via the CORBA network.

Since the configuration information is not directly stored at the node, there is no overhead on the smart transducers themselves.

2.5 Implementation Experiences

2.5.1 Smart Transducer Nodes

The presented time-triggered smart transducer interface has been implemented on several hardware platforms. The current segment of cheap 8-bit Microcontrollers is best suited for equipping sensors or actuators with a low-cost smart interface. We have made experiences with node implementations on the Atmel AVR family, the Microchip PIC and, especially for the master, the 32-bit ARM RISC microcontrollers.

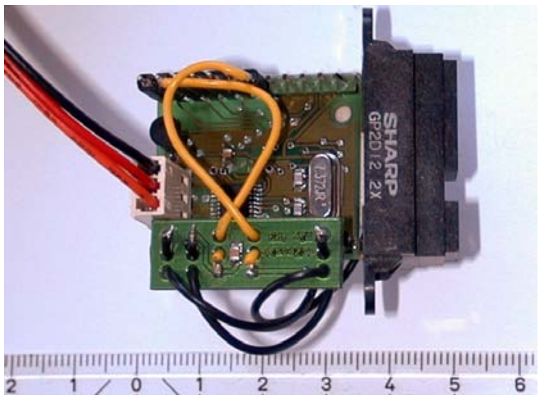


Figure 2.9: Smart transducer based on Atmel 4433 microcontroller with distance sensor attached (scale in centimeter)

Figure 2.9 depicts the hardware of a smart transducer implementation based on an Atmel AVR AT90S4433 microcontroller and an attached distance sensor. This type of controller offers 4K Byte of Flash memory and 128 Byte of SRAM. The physical network interface has been implemented by an ISO 9141 k-line bus, which is a single wire bus supporting a communication speed up to 50 kBps. The wires to the left of the photo contain the bus line and the power supply.

Table 2.3 gives an overview on the resource requirements for smart transducer implementations in Atmel AVR, Microchip PIC and ARM RISC microcontrollers. Since the time-triggered approach follows the *resource adequacy* principle [Kop97, p. 15], the performance and current workload at the controller does not influence the specified real-time behavior of the network, however, a controller that supports only a particular communication speed may not be used in networks that specify a higher communication rate. All three implementations held the timing requirements with a Baud Rate of 19.2 kbps. As physical layer, an ISO 9141 k-line bus had been used. For the Atmel AT90S4433 a maximum performance of 58.8 kbps had been tested on an RS485 physical layer.

Microcontroller	Used Program Memory	Used RAM Memory	Clock Speed	Tested Baud Rate
Atmel AT90S4433	2672B	63B	7.3728 MHz	58.8 kbps
Microchip PIC	2275B	50B	8.0 MHz	19.2 kbps
ARM RISC	8kB	n.k.	32.0 MHz	19.2 kbps

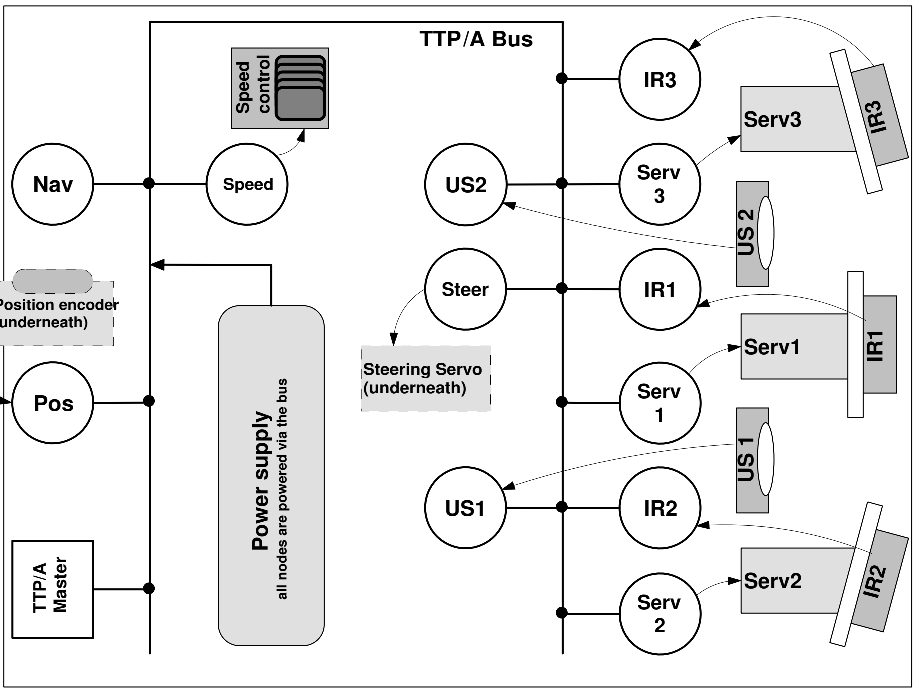
Table 2.3: Resource requirements and performance of time-triggered smart transducer interface implementations (from [Trö02]).

The implementations on these microcontrollers show that due to the low hardware requirements of the time-triggered smart transducer interface it should be possible to implement the protocol on nearly all available microcontrollers with similar features like the Atmel or Microchip microcontroller types, that is 4KB of Flash ROM and 128 Byte of RAM memory.

2.5.2 Application Case Study

As an example for a time-triggered smart transducer application, an autonomous mobile robot consisting of a four-wheeled model car with a smart transducer network for instrumenting a set of sensors, actuators and a navigation module, has been designed and implemented at the Vienna University of Technology.

The robot basically uses its sensors to locate objects in the driving direction and sets its steering and speeding actuators in order to pass the obstacles.



IR1	Middle infrared sensor	US1	Right ultrasonic sensor
IR2	Right forward infrared sensor	US2	Left ultrasonic sensor
IR3	Left forward infrared sensor	Pos	Position encoder sensor
Serv1	Servo for IR1	Speed	Speed control actuator
Serv2	Servo for IR2	Steer	Steering control actuator
Serv3	Servo for IR3	Master	Synchronization and gateway
Nav	Coordination and navigation		

Figure 2.10: Smart transducer nodes on the autonomous robot

Figure 2.10 depicts the layout of the smart transducer nodes employed in the smart transducer network. Each of the 6 sensor nodes and 5 actuator nodes is implemented using an Atmel AT90S4433 microcontroller. Due to the larger memory requirements of the master/gateway and navigation applications, the master and navigation nodes are implemented on the more powerful microcontrollers AT90S8515 and ATmega128 from the Atmel AVR series.

The smart transducer network has to fulfill several real-time tasks:

- The three infrared distance sensors IR1, IR2, and IR3 are mounted on servos Serv1, Serv2, and Serv3, which swivel the sensors for scanning the area in front of the car. Therefore, whenever a measurement from a distance sensor is read, the corresponding position of the respective servo must be exactly known in order to process the measurement correctly. The time-triggered schedule supports this coordination of servo position and measuring time and allows to minimize the time for a sensor sweep.

- The two ultrasonic sensors US1 and US2 are active sensors, i. e., they emit an ultrasonic ping when performing a measurement. Since both sensors face the same direction (the front of the car), the measurements must be coordinated in order to avoid mutual interference from the ultrasonic pings. This requirement can be conveniently fulfilled by defining an appropriate phase offset in the time-triggered schedule.
- The control of the robot's speed and the measurement of the covered distance are done by a feedback control loop. Using standard control theory, the duration between measurement and setting a new speed value has to be constant and known. The time-triggered schedule fulfills this requirement at the outset.

Due to the predictable timing of the time-triggered system, the communication and action schedule could be implemented very tight and efficient, since no time is wasted for repeating messages in case of a busy channel, etc. The whole application runs sufficiently at a bus speed of 9600 bit/sec with a cluster cycle of 30 ms.

Each smart transducer node was designed independently from the overall application, most of them have been reused in other smart transducer applications. The robot has been used as a demonstrator for composable development in the Dependable Systems of Systems (DSoS) project (IST Research Project IST-1999-11585). A report describing the robot implementation in detail can be found in [Elm02a].

2.6 Related Work

Real-time distributed networks for interconnection of sensors and actuators represent a well-established research area in the scientific community. Good overview papers on real-time communication systems including time-triggered communication are [Rus03, Zur05, Ber00, Jor95].

However, there is not a lot of research literature that discusses the application of hard real-time capable time-triggered network interfaces for smart transducer networks. Most notable exceptions are the extension of the IEEE 1451 smart transducer standard with a time-triggered interface and, to a lesser extend, the LIN fieldbus, which has a time-triggered polling scheme.

2.6.1 IEEE 1451 with Time-Triggered Communication

An Irish research group has developed a time-triggered smart transducer system that incorporates the IEEE Smart Transducer Interface Standard (IEEE 1451.2) and a Time-Triggered Controller Area Network (TTCAN) communication protocol [Doy02].

The IEEE 1452 Smart Transducer Interface Standard [Con00] proposes a point-to-point communication interface between a Smart Transducer Interface Module (STIM) and a Network-Capable Application Processor (NCAP). It supports a broad range of sensor and actuator models including those for buffered, time-triggered, data-sequence and event-sequence models. The standard uses an object representation for measured variables that resolves the problem of handling physical units. Additionally, it defines the Transducer Electronic Data Sheet (TEDS), which abstracts the properties of sensors and actuators enabling their dynamic definition via contextual information associated with data.

The IEEE 1451 standard comes also with a Transducer Independent Interface (TII) that supports the multiplexing of messages, however this TII does not support the hard real-time requirements regarding determinism. Therefore in this project the IEEE 1451 standard has been only partially implemented, since the TII protocol has been replaced by the deterministic time-triggered TTCAN protocol.

2.6.2 Local Interconnect Network (LIN)

LIN is basically a polling protocol, where a central master issues request messages to the slave nodes. The master node acts also as a gateway to a higher network. The slave nodes are smart transducers which are listening to specific messages in order to set a control value or to send a measured value on reply. The master issues request messages on a predefined schedule, while the slave nodes are not aware of a global time or the current state of the schedule. This simplifies the implementation of the slave nodes, but does not support coordinated actions like synchronized measurements.

Due to the polling principle (requesting a value involves the request message, a “thinking time” for the slave node and sending the reply message), the effective bandwidth of a LIN network supports only applications with low bandwidth requirements, such as less critical body electronic functions in cars.

2.7 Conclusion

The static structure of time-triggered communication is an advantage and a disadvantage at the same time. On the one hand, it enables guaranteed deterministic timing and supports hard real-time constraints, on the other hand, it makes it difficult to efficiently access sparsely changing values or maintenance facilities. In the proposed architecture for time-triggered smart transducer networks, this problem has been overcome by a separate implementation of virtual communication interfaces: the real-time service interface provides the timely communication of fast changing real-time values, like measurements or control values, while the configuration and planning and the diagnostics and management interfaces allows for flexible access to schedules, sensor logs, trimming and calibration parameters, etc. Moreover, since the time-triggered communication does not need to explicitly address frames in their messages and avoids collisions by design, it is much more efficient for periodic data exchange than event-triggered or polling protocols.

The implementation of time-triggered smart transducers on several platforms has shown that hard real-time requirements can be fulfilled with resources of typical low-cost 8-bit microcontrollers. Implementations of various sensors and actuators have proven to be efficient and reusable.

Recent work in adapting standards like IEEE 1451 to time-triggered communication interfaces has underlined the relevance and appropriateness of the time-triggered approach for smart transducer networks.

2.8 Acknowledgments

I would like to thank the anonymous reviewers for their valuable comments on an earlier version of this paper. Furthermore thanks to Wolfgang Haidinger and Stefan Pitzek for proofreading the paper. This work was supported in part by the European IST project DECOS (Dependable Embedded Components and Systems) under contract number IST-511764 and the Austrian FIT-IT project SDSTI under contract No 808693.

References

[Aud99] Audi AG, BMW AG, DaimlerChrysler AG, Motorola Inc. Volcano Communication Technologies AB, Volkswagen AG, and Volvo Car Corporation. LIN Specification and LIN Press Announcement. SAE World Congress Detroit, http://www.lin-subbus.org, 1999.

[Ben04] M. Bengtsson. Condition Based Maintenance System Technology – Where is Development Heading? In *Proceedings of the 17th European Maintenance Congress*, Barcelona, Spain, May 2004.

[Ber00] J. Berge and S. Mitschke. Building Better Open Networks Using Foundation Fieldbus and OPC. *Sensors Magazine*, Feb. 2000.

[Con00] P. Conway, D. Heffernan, B. O'Mara, D. P. Burton, and T. Miao. IEEE 1451.2: An Interpretation and Example Interpretation. In *Proceedings of the Instrumentation and Measurement Technology Conference*, pages 535–540, Baltimore, MD, USA, May 2000.

[Doy02] P. Doyle, D. Heffernan, and D. Duma. A Time-Triggered Transducer Network Based on an Enhanced IEEE 1451 Model. *Microprocessors & Microsystems Journal*, Dec. 2002.

[Ecc98] L. H. Eccles. A Brief Description of IEEE P1451.2. *Sensors Expo*, May 1998.

[Elm01a] W. Elmenreich. An Introduction to Sensor Fusion. Technical Report 47/2001, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2001.

[Elm01b] W. Elmenreich, W. Haidinger, and H. Kopetz. Interface Design for Smart Transducers. In *IEEE Instrumentation and Measurement Technology Conference*, volume 3, pages 1642–1647, Budapest, Hungary, May 2001.

[Elm02a] W. Elmenreich, W. Haidinger, H. Kopetz, T. Losert, R. Obermaisser, M. Paulitsch, and C. Trödhandl. Initial Demonstration of Smart Sensor Case Study. *DSoS Project (IST-1999-11585) Deliverable PCE3*, Apr. 2002.

[Elm02b] W. Elmenreich, W. Haidinger, P. Peti, and L. Schneider. New Node Integration for Master-Slave Fieldbus Networks. In *Proceedings of the 20th IASTED International Conference on Applied Informatics (AI 2002)*, pages 173–178, Feb. 2002.

[FDC05] fdcml.org. *Field Device Configuration Markup Language FDCML 2.0 Specification*, 2005. Version 1.0.

[Gai86] J. Gait. A Probe Effect in Concurrent Programs. *Software Practice and Experience*, 16(3):225–233, Mar. 1986.

[Hai00] W. Haidinger and R. Huber. Generation and Analysis of the Codes for TTP/A Fireworks Bytes. Research Report 5/2000, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2000.

[IEC05] International Electrotechnical Commission,. *IEC TR 62390, Common automation device - Profile guideline*, 2005. First edition 2005-01.

[IEE97] Institute of Electrical and Electronics Engineers, Inc. *IEEE Std 1451.2-1997, Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Micro-processor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*, 1997.

[Jor95] J. R. Jordan. *Serial Networked Field Instrumentation.* John Wiley & Sons, 1995.

[Kop97a] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[Kop97b] H. Kopetz and R. Nossal. Temporal Firewalls in Large Distributed Real-Time Systems. *Proceedings of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '97)*, pages 310–315, 1997.

[Kop01] H. Kopetz, M. Holzmann, and W. Elmenreich. A Universal Smart Transducer Interface: TTP/A. *International Journal of Computer System Science & Engineering*, 16(2):71–77, Mar. 2001.

[Kop02] H. Kopetz et al. Specification of the TTP/A Protocol. Research Report 61/2002, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, Sep. 2002. Version 2.00.

[Kop03] H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1):112–126, Jan. 2003.

[Kop04] H. Kopetz. On the Fault Hypothesis for a Safety-Critical Real-Time System. In *Proceedings of the Automotiove Workshop San Diego*, CA, USA, Jan. 2004.

[McD89] C. E. McDowell and D. P. Helmbold. Debugging Concurrent Programs. *ACM Computing Surveys*, 21(4):593–622, Dec. 1989.

[OMG00] Object Management Group (OMG). *Smart Transducers Interface Request for Proposal*, Dec. 2000. Available at http://www.omg.org as document orbos/2000-12-13.

[OMG03] Object Management Group (OMG). *Smart Transducers Interface V1.0*, Jan. 2003. Specification available at http://doc.omg.org/formal/2003-01-01 as document ptc/2002-10-02.

[Par92] B. Parhami. Optimal Algorithms for Exact, Inexact, and Approval Voting. *Twenty-Second International Symposium on Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers.*, pages 404–411, July 1992.

[Pit03] S. Pitzek and W. Elmenreich. Configuration and Management of a Real-Time Smart Transducer Network. In *Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2003)*, pages 407–414, Lisbon, Portugal, Sep. 2003.

[Pol94] S. Poledna. *Replica Determinism in Fault-Tolerant Real-Time Systems.* PhD Thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 1994.

[Ran97] A. Ran and J. Xu. Architecting Software with Interface Objects. In *Proceedings of the 8th Israeli Conference on Computer-Based Systems and Software Engineering*, pages 30–37, 1997.

[Rus03] J. Rushby. A Comparison of Bus Architectures for Safety-Critical Embedded Systems. Technical Report NASA/CR-2003-212161, National Aeronautics and Space Administration, Langley Research Center, Mar. 2003.

[Sur95] N. Suri, M.M. Hugue, and C.J. Walter. *Advances in Ultra-Dependable Distributed Systems*. IEEE Press, 1995.

[Trö02] C. Trödhandl. Architectural Requirements for TTP/A Nodes. Master's Thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[Zur05] R. Zurawski, Editor. *The Industrial Communication Technology Handbook*. CRC Press, Boca Raton, FL 33431, USA, 2005.

Chapter 3

Interface Design for Real-Time Smart Transducer Networks – Examining COSMIC, LIN, and TTP/A as Case Study

Wilfried Elmenreich, Hubert Piontek, and Jörg Kaiser. Proceedings of the 15th International Conference on Real-Time and Network Systems (RTNS'07), Nancy, France, 2007, pages 195-204.

This paper analyzes and discusses the interface models of the three real-time smart transducer networks COSMIC, LIN, and TTP/A.

The COSMIC architecture follows a publish/subscribe model, where the producing smart devices broadcast their event data on the basis of a push paradigm. Subscribers receive data in form of a message-based interface.

LIN follows a strict pull principle where each message from a device node is requested by a respective message from a master. Applications have a message-based interface in order to receive and transmit data.

The nodes in a TTP/A network derive its sending instants from predefined instants in time. TTP/A maps communicated data into an Interface File System (IFS) that forms a distributed shared memory.

3.1 Introduction

The availability of cheap microcontrollers and network solutions has enabled distributed architectures with networked smart transducer devices. The hardware for a smart transducer consists of a physical sensor or actuator, a microcontroller or FPGA with on-chip memory and analog I/O, and a network interface. The software in the microcontroller contains transducer-specific routines, like de-noising, linearization, and feature extraction, and a communication protocol establishing a standardized interface to the smart transducer. This interface provides access to the transducer values, like sensor measurements or actuator set values, as well as configuration and management data (calibration data, error logs, etc.). In order to support an automatic (plug-and-play) or semi-automatic configuration, a smart transducer may also host an electronic description, i. e., a machine-readable datasheet describing its features and interfaces.

Real-time and bandwidth requirements make the design of an interface to a smart transducer a difficult task. This paper addresses the specific requirements and issues of interface design for smart transducers and examines three architectures for real-time smart transducer networks, i. e., the Cooperating Smart Devices (COSMIC) middleware, the LIN, and the Time-Triggered Protocol for SAE class A applications (TTP/A).

The paper is structured as follows: Section 3.2 states the basic concepts and requirements for smart transducer interface design. Section 3.3 describes communication model, interface design and node description approach for the COSMIC middleware. Accordingly, Section 3.4 and Section 3.5 examine the LIN and TTP/A approach. Section 3.6 elaborates common features and differences of the three approaches. The paper is concluded in Section 3.7.

3.2 Interface Concepts

A smart transducer interface can be decomposed into several sub-interfaces with different purposes and requirements [Kop01]: The *real-time (RT) service interface* is required for transmitting transducer data such as measurements or set values. The *configuration and planning (CP) interface* provides access to protocol-specific functions like new node identification, obtaining electronic datasheets, and configuration of communication schedules. The CP interface is not time-critical. The *diagnostics and management (DM) interface* is used for accessing sensor and actuator-specific functions like monitoring, calibration, etc. The DM interface is not time-critical, however, some monitoring applications require timestamping.

In the following we discuss real-time requirements, flow control and interaction design patterns which are mostly relevant for the RT service. The DM interface has different requirements and should not interfere with the RT service.

3.2.1 Real-Time Requirements

There are different kinds of real-time requirements for a distributed system of smart transducers. As a common feature, there is always a *deadline* that specifies a point in time when a specific action has to be completed.

Performing some action locally with respect to real time, like generating a particular Pulse-Width Modulation (PWM) signal or making a measurement every 100 ms is relatively easy to achieve if drift of the local clock source is sufficiently low to provide a useful time base.

Timestamping events can be used to temporally relate measurements to each other. In order to create timestamps with a global validity, a synchronized global time is required among the participating nodes. In most cases, clock synchronization has to be done periodically in order to compensate for the drift of the local clocks. Once a global time is established, timestamping does not pose real-time requirements on the communication system, since timestamped events can be locally stored.

Bounded maximum reaction time requires the communication system to deliver messages within a specified time interval. Standard feedback control algorithms also require low message jitter in order to work correctly.

Globally synchronized actions require the synchronized generation of action triggers in different nodes. This can be achieved by a multi-cast message or assigning actions to an instant on the globally synchronized time scale.

Moreover, a real-time requirement can be *hard*, i. e., deadlines must be held under all circumstances or *soft*, the system is still of use if deadlines are violated infrequently.

Many architectures implement a subset of the described features or provide different features with hard or soft real-time behavior. For example the LAAS architecture [Ala98] for component-based mobile robots specifies local hard-real-time such as a locally closed control loop or the instrumentation of an ultrasonic sensor, while at higher levels, e. g., for globally synchronized actions it provides only soft real-time behavior.

3.2.2 Models of Flow Control

Communication between subsystems takes place in the *time domain* and the *value domain*. In the value domain, the message data is exchanged, while in the time domain *control information* is transmitted [Krü97].

The communication partner that generates the control information influences the *temporal control flow* of the other communication partner(s). If a communication is controlled by the sender's request we speak of a *push* model, if communication is requested by the receiver, we speak of a *pull* model.

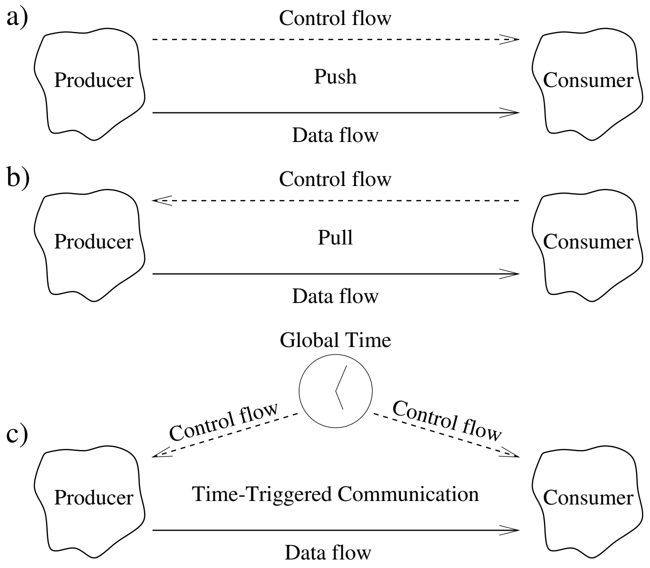


Figure 3.1: Push-, Pull-, and Time-Triggered Communication

For explanation, let us assume that two or more subsystems need to exchange data over a network. Further, without restrictions to generality, we assume message data to be transmitted from a *producer* to one or more *consumers*. Different from the very popular client-server communication pattern it is necessary to support a one-to-many or many-to-many communication pattern in smart transducer networks because in the common case, the sensor readings of a transducer is needed in more than one place in control applications. Therefore, all of considered networks support this property, however, in different ways. In order to transfer data between the subsystems, they must agree on the mechanism to use and the direction of the transfer.

Figure 3.1 a) shows the *push* method. The producer is empowered to generate and send its message spontaneously at any time and is therefore independent of the consumer. This loose coupling enables independence between the supplier and consumers of information [Mor93], [Kai99b], but makes it difficult to enforce temporal predictability in a purely event driven model. Without the option to enforce further temporal constraints, the communication system and the receiving push consumer have to be prepared for data messages at any time, which may result in high resource costs and difficult scheduling. Popular "push" mechanisms are messages, interrupts, or writing to a memory element [DeL99]. The push-style communication is the basic mechanism of event-triggered systems.

In the *pull* model depicted in Figure 3.1 b) the consumer governs the flow control. Whenever the consumer wants to access the message information, the producer has to respond to the consumer's request.. This facilitates the task for the pull consumer, but the pull supplier (producer) must be watchful for incoming data requests. Popular "pull" mechanisms are polling or reading from a memory element [DeL99]. Pull-style communication is the basic mechanism of client-server systems.

Figure 3.1 c) depicts a communication model where the flow control is derived from an external trigger. This can be another physical system or the derivation of the triggers from the progress of physical time. In the latter case, the control signals are known *a priori*, which requires predefined scheduling and error detection in the control domain.

3.2.3 Interaction Design Patterns

We distinguish three basic interaction design patterns for network communication.

In a *master/slave relationship*, at any time one node is considered the *master* while the other nodes are considered to be *slaves*. The master is able to issue synchronization events or to start communications. All slave nodes depend on one master, while the master is independent of a particular slave.

In a *client/server relationship*, a *client* issues a request to a *server*, which has to answer the request. The client and the server are thus tightly coupled via a pull model.

In a *publish/subscribe relationship*, a *publisher* generates data using the push model. A number of nodes may subscribe to a particular publisher but there is no control flow from subscriber to publisher. Depending on the implementation, a publisher may broadcast its data immediately, transmit its data to an intermediary broker, or transmit its data via point-to-point connections to a list of subscribers. Thus, the number of subscribers may influence the time it takes for a publisher to publish its data.

An architecture may hide the communication model by implementing a distributed shared memory on top of the communication. This way, an application uses the same interface to access data locally or remote. However, a memory interface does not transport control data, e. g., in order to launch a particular task upon reception of an event. Such functionality can either be achieved via polling, but that requires an adequate poll frequency and comes with a noticeable overhead [Lan97] or solved via additional features like interrupt generation after update of a specific data field.

3.2.4 Diagnostics and Management

While the RT interface provides only access to a limited data set consisting of the actual needed transducer data, for debugging or monitoring purposes, additional data about the operation of the transducer is of interest.

Transmission of these data typically is not time-critical, but must not interfere with the RT service leading to an unwanted probe-effect [McD89]. Furthermore, monitored RT data should either have time stamps or it must be transmitted before a (typically soft) deadline.

3.2.5 Configuration and Planning

Large smart transducer systems require support by automatic setup facilities in order to keep up with the complexity of setting parameters correctly. Therefore, smart transducer system should be supported by a tool architecture enabling a plug-and-play-like integration of new nodes.

We consider different configuration and planning scenarios:

In the *replacement scenario*, a broken node is to be exchanged by a new one of the same type. Therefore, the new node has to be detected and a backup of the configuration of the broken node has to be uploaded. However, specific parameters like calibration data will have to be created anew.

In the *initial set-up scenario*, a set of nodes is configured in order to execute a particular communication. This action requires a system specification, and, in most cases, a human operator to perform tasks that cannot be solved automatically.

In the *extension scenario*, a distributed application is extended by extra nodes in order to improve the performance and possibilities of the system. In this case the system managing the configuration must have knowledge how to integrate new transducers in order to upgrade the system. An example of such an approach is outlined in [Pit05].

For the purpose of node identification and documentation, a node is assigned a machine-readable description describing the node's features. Example for such descriptions are the Transducer Electronic Datasheets of IEEE 1451.2 [IEE97] or the Device Profiles in CANopen [iAe].

3.3 COSMIC

3.3.1 COSMIC communication abstractions

COSMIC is middleware which is designed for small embedded systems, supporting heterogeneous networks and cross network communication. It provides a publish/subscribe abstraction over different addressing and routing mechanisms as well as it considers different latency properties of the underlying networks. COSMIC provides typed event messages (EM) identified by an event UID which identifies the content of a message rather than a source or destination address. Further, EMs have attributes which define a temporal validity of the EM. It should be noted that the term ”event”does not refer to a specific synchrony class but just denotes a typed message. As indicated previously, in a real–time embedded environment the pure push model creates problems because the consumers of the information must be ready to receive and process this information at any time. This may lead to situations where some of the messages are lost. Therefore, COSMIC introduces the notion of event channels (EC) which allow specifying temporal constraints and delivery guarantees of individual communication channels explicitly. COSMIC supports three event channel classes: A hard real-time event channel (HRTEC) offers delivery guarantees based on a time-triggered scheme. EMs pushed to a soft real-time event channel (SRTEC) are scheduled according to the earliest deadline first (EDF) algorithm. The respective deadline is determined by the temporal validity information in the attribute field of the EM. Because soft real-time EMs which have already missed their transmission deadline may cause further deadline misses of other soft real-time EMs, they are discarded and the respective local application is notified. The application then can decide about re-sending in a lower real-time class or just skip it. Finally, a non real-time event channel (NRTEC) disseminates events that have no timeliness requirements.

ECs are established prior to communication allowing the middleware to reserve the necessary resources and perform the binding to the underlying mechanisms of the communication network.

The COSMIC architecture is not bound to a particular network. An implementation based on Controller Area Network (CAN) [Gmb91] is described in the next section.

3.3.2 COSMIC–on–CAN architecture

Implementing the event model requires to map the abstractions of that model (publisher, subscriber, event channel, event instance) to the elements provided by the infrastructure of the communication system. The respective functional-

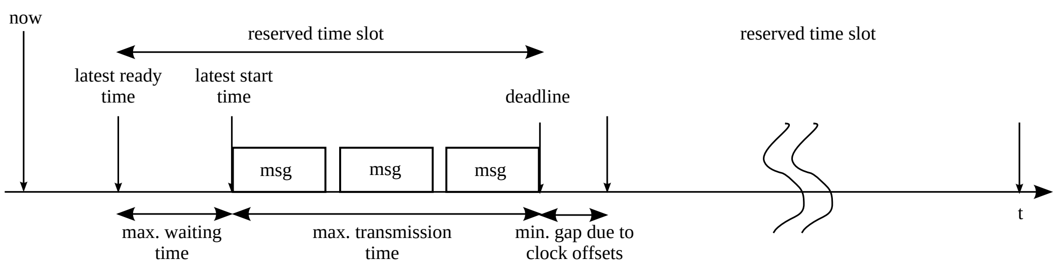


Figure 3.2: Structure of a time slot

ity to perform these mapping in COSMIC is encapsulated in the Event Channel Handler which resides in every node. Given the constraints in bandwidth and in message length on CAN, the implementation of events and event channels has to exploit the underlying CAN mechanisms. To save the rare space in the message body [Fre02] and to reduce the inherent CAN message overhead, significant information is also encoded via the CAN ID.

The implementation is based on the extended 29 bit CAN ID of the CAN 2.0 B specification. The 29-Bit CAN identifier (CAN-ID) is structured into three fields, i. e., an 8 bit priority field used to prioritize messages according to HRTEC, SRTEC, and NRTEC, a 7 bit node- ID ensuring unique identifiers and a 14-Bit event tag. The assignment of an event UID to an event tag is performed dynamically by the COSMIC middleware infrastructure. A description of this infrastructure and the respective binding protocol is described in [Kai02].

3.3.3 Enforcing temporal constraints in COSMIC

HRTECs provide delivery guarantees and use reserved time slots in a TDMA scheme organized in periodic rounds. The intention of the reservation-based scheme is to avoid collisions by statically planning the transmission schedule. Hence, any conflict between HRTECs is avoided. COSMIC is implements clock synchronization based on the algorithm proposed in [Ger94].

Because a CAN message cannot be preempted a non hard real-time message transmission may delay a hard real-time message by the maximum length of one CAN message in the worst case. Furthermore, transient transmission faults may increase the time needed to transmit a hard real-time message. Therefore, a hard real-time slot is extended according to Fig. 3.2. The protocol relies on the fault–handling mechanisms of the standard CAN which has an impact on the fault classes which we can handle. For a message with b bytes of data, the maximum length of the message including header and bit–stuffing is: $Length_{message} = 75 + \lfloor b \cdot 9.6 \rfloor$[1]. Under the assumption of f single transmission failures, the required minimum time–slot length is: $slot\ length = 2 \cdot t_{message} + (t_{message} + 18) \cdot f + 3bittimes$. Assuming a single message failure of an 8 Byte message at 1 Mbit/sec (msg transmission time: 151μsec, fault detection and retransmission 18μsec) and a gap between the slots of 50μsec, approx. 1900 slots/sec can be allocated. If it is necessary to tolerate a permanent controller failure, this number drops down to an approximate number of 350 slots/sec. Compared to a maximum throughput of about 6500 maximum length messages per second, the number of possible HRT slots is low. However, these numbers refer to the number of *guaranteed* HRTECs not to the number of messages which actually can be sent. Unlike in pure time-triggered systems, the CAN priority mechanism can be used to transmit SRT or NRT messages in cases where a HRT message has been received a message successfully by all operational nodes[2]. Thus, time redundancy only costs bandwidth if faults really occur, which may be relatively rare compared to the overall traffic. The priority-based arbitration mechanism is also exploited to schedule SRTECs and NRTECs. HRT messages always reserve the highest priority. The relation between the priorities of HRT, SRT and NRT messages can be expressed by the relation: $P_{HRT} < P_{SRT} < P_{NRT}$ (a lower numerical value represents a higher priority). The assignment enforces that a message of a lower real–time class never will interfere with one of a higher class during bus arbitration. We assume the highest priority (0) for HRT messages and a small number of fixed low priorities for NRT messages. 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 the priority. Mapping deadlines to priorities will cause the problem 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.

3.3.4 Device Descriptions

COSMIC devices are described in an XML-based language called CODES (COSMIC embedded DEvice Specification) [Kai06]. The descriptions are struc-

[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 [Ruf98], inconsistent omissions occur with a probability in the order of 10^{-9}). Kaiser and Livani [Kai99a] describe a transparent mechanism to handle these situations.

tured into three parts. Part one, *General Information* contains the node's name, its type, its manufacturer, its unique identifier, its networking facilities, its supported event channel types, recycling information, and a clear text description of the device. This part also contains version information about the component. The second part contains all *event definitions*, i. e., the description of all events produced or consumed by the device.

For each event is described by a plain text tag and a unique identifier. The event's definition includes a list of attributes giving non-functional details about the event, e. g., the event's expiration time. Whenever an event is disseminated, it is sent as a compact message. This message's data structure is specified in the event definition. For each field in the data structure, its name, data type and byte order are included in the description. This information can be used by tools to automatically create decoder for a compact message. Fields representing a measurement are annotated by the corresponding physical dimension in a machine-readable format. Non-measurement fields are described by lists or state machines. Each field may contain also attributes, e. g., the valid data range. The last part contains the declaration of all *event channels* and their properties. Each event channel definition contains the subject UID linking it to the respective event definition, the class of event channel, the direction of the event channel as seen locally, and again a list of attributes, e. g., the channel's period.

While the greater part of the description contains static information, some elements are not suitable for integration into a static description document, e.g. the period of an event channel, which certainly will vary depending on the application. To overcome this problem, parameterization was introduced. Any non–static element can be marked as a parameter in the static description. The element's actual value is then defined and stored external to the static description. Parameters are stored in path–value–pairs, similar to well–known name–value–pairs. Instead of naming the parameter, it's XPath expression within the static description acts as the identifier. A scheme for mapping this structure down to a binary parameter storage scheme suitable for small devices exists. Whenever the description is used, the parameters are included beforehand. The query service (see below) is a suitable place that will handle this inclusion in running systems. Having each parameter's path expression eases the integration into the description document.

CODES descriptions play a central role for COSMIC components. The life of a COSMIC component starts with the description document created during the component's design phase. It is used during the following implementation phase to generate parts of the component's code [Kai06]. Black–box tests of the component can be assisted, e.g. tests for timing behavior or testing the compliance of disseminated events with their description in terms of the data structure, value ranges, or precision. The descriptions are further useful throughout the component's life–cycle: During the integration phase into a larger system, a number of compatibility checks can be performed automatically. Schedules for the HRT communication can be derived from the respective set of descriptions. While the component is in use, the ready availability of its description forms the basis for dynamic use of formerly unknown components. Currently, this requires a priori knowledge or the interaction with a user. In the long run, the integration of semantic web technology is planned to enable true autonomous dynamic cooperation of components. Whenever a system is in need of maintenance, the availability of the descriptions is beneficial, too. They provide a quick overview of the system, i. e., what components are available, and how they are configured.

The descriptions are stored within the devices themselves. They can be retrieved and queried at run–time: On system start–up, and whenever a new component is added to a system, an automatic configuration is necessary for the component to be able to participate in communication. During this configuration, the components' descriptions and parameters are uploaded to the node running the event channel broker. This node also runs a query service which makes the descriptions accessible from outside the system. The parameters are included in the static part of the description, yielding a single document describing the current configuration of the components. Requests to the query service are given as XSLT transformations [M. 06]. The transformations are applied to the CODES descriptions on the node running the query service, thus enabling even rather low–power nodes to make use of the query service. XSLT transformations represent a suitable technology not only for the query service, but throughout the different application areas of the CODES descriptions. They are e.g. also used for code generation.

3.4 LIN

3.4.1 System Architecture

Each message in LIN is encapsulated in a single message cycle. The message cycle is initiated by the master and contains two parts, the frame header sent by the master and the frame response, which encompasses the actual message and a checksum field. The frame header contains a sync brake (allowing the slave to recognize the beginning of a new message), a sync field with a regular bit pattern for clock synchronization and an identifier field defining the content type and length of the frame response message. The identifier is encoded by 6 bit and 2 bits for protection. Figure 3.3 depicts the frame layout of a LIN message cycle.

The frame response contains up to 8 data bytes and a checksum byte. Since an addressed slave does not know *a priori* when it has to send a message, the response time of a slave is specified within a time window of 140% of the nominal length of the response frame. This gives the node some time to react on the master's message request, for example to perform a measurement on demand, but introduces a noticeable message jitter for the frame response.

The interaction between master and slave is a plain pull mechanism, since the slaves only react on the frame header from the master. It is the master's task to issue the respective frame headers for each message according to a scheduling table. From a data-centric perspective, the communication is defined by messages that are subscribed by particular slaves for reception or transmission. The configuration of the network must ensure that each message has exactly one producer.

In 2003, LIN was enhanced by extra features leading to the LIN 2.0 specification. New features introduced in LIN 2.0 are an enhanced checksum, sporadic and event-triggered communication frames, improved network management (status, diagnostics) according to ISO 14230-3 / ISO 14229-1 standards, automatic baud rate detection, standardized LIN product ID for each node, and an updated configuration language to reflect the changes.

In addition to the unconditional frames (frames sent whenever scheduled according to the schedule table) provided by LIN 1.3, LIN 2.0 introduces *event-triggered frames* and *sporadic frames*.

Similar to unconditional frames, event-triggered frames begin with the master task transmitting a frame header. However, corresponding slave tasks only transmit their frame response if the corresponding signal has changed since the last transmission. Unlike unconditional frames, multiple slave tasks can provide the frame response to a single event-triggered frame, assuming that not all signals have actually changed. In the case of two or more slave tasks writing the same frame response, the master node has to detect the collision and resolve it by sequentially polling (i.e., sending unconditional frames) the involved slave nodes. Event-triggered frames were introduced to improve the handling

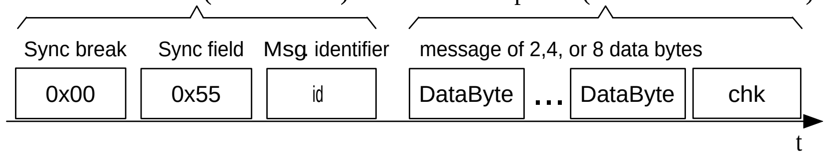

Frame header (from master)
Frame response (from master or slave)
Sync break
Sync field
Msg. identifier
message of 2,4, or 8 data bytes
0x00
0x55
id
DataByte
...
DataByte
chk
t

Figure 3.3: LIN frame format

of rare-event data changes by reducing the bus traffic overhead involved with sequential polling.

Sporadic frames follow a similar approach. They use a reserved slot in the scheduling table, however, the master task only generates a frame header when necessary, i. e., when involved signals have changed their values. As this single slot is usually shared by multiple sporadic frames (assuming that not all of them are sent simultaneously), conflicts can occur. These conflicts are resolved using a priority-based approach: frames with higher priority overrule those with lower priority.

In addition to signal-bearing messages, LIN 2.0 provides *diagnostic messages*. These messages use 2 reserved identifiers (0x3c, 0x3d). Diagnostic messages use a new format in their frame response called *PDU* (Packet Data Unit). There are two different PDU types: *requests* (issued by the client node) and *responses* (issued by the server node).

The LIN 2.0 configuration mode is used to set up LIN 2.0 slave nodes in a cluster. Configuration requests use SID values between 0xb0 and 0xb4. There is a set of mandatory requests that all LIN 2.0 nodes have to implement as well as a set of optional requests. Mandatory requests are:

- Assign Frame Identifier: This request can be used to set a valid (protected) identifier for the specified frame.
- Read By Identifier: This request can be used to obtain supplier identity and other properties from the addressed slave node.

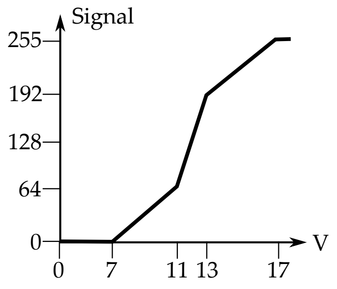
Optional requests are:

- Assign NAD: Assigns a new address to the specified node. Can be used to resolve address conflicts.
- Conditional Change NAD: Allows master node to detect unknown slave nodes.
- Data Dump: Supplier specific (should be used with care).

3.4.2 Device Descriptions

Each LIN 2.0 [Aud03] node is accompanied by a *node capability file* (NCF). The NCF contains:

- The node's name.



```
V_battery {
    logical_value, 0, "under voltage";
    physical_value, 1, 63, 0.0625, 7.0, "Volt";
    physical_value, 64, 191, 0.0104, 11.0, "Volt";
    physical_value, 192, 253, 0.0625, 13.0, "Volt";
    logical_value, 254, "over voltage";
    logical_value, 255, "invalid";
}
```

Figure 3.4: Example for a LIN signal definition

- General compatibility properties, e.g. the supported protocol version, bit rates, and the LIN product identification. This unique number is also stored in the microcontroller's ROM and links the actual device with its NCF. It consists of three parts: supplier ID (assigned to each supplier by the LIN Consortium), function ID (assigned to each node by supplier), and variant field (modified whenever the product is changed but its function is unaltered)

- Diagnostic properties, e.g. the minimum time between a master request frame and the following slave response frame.

- Frame definitions. All frames that are published or subscribed by the node are declared. The declaration includes the name of the frame, its direction, the message ID to be used, and the length of the frame in bytes. Optionally, the minimum period and the maximum period can be specified. Each frame may carry a number of signals. Therefore, the frame's declaration also includes the associated signals' definitions. Each signal has a name, and the following properties associated with it: **Init value** specifies the value used from power on until the first message from the publisher arrives. **Size** specifies the signal's size in bits. **Offset** specifies the position within the frame. **Encoding** specifies the signal's representation. The presentation may be given as a combination of the four choices *logical value*, *physical value*, *BCD value*, or *ASCII value*. Declarations of physical values include a valid value range (minimum and maximum), a scaling factor, and an offset. Optionally, this can be accompanied by a textual description, mostly to document the value's physical unit. An example is given in figure 3.4.

- Status management: This section specifies which published signals are to be monitored by the master in order to check if the slave is operating as expected.

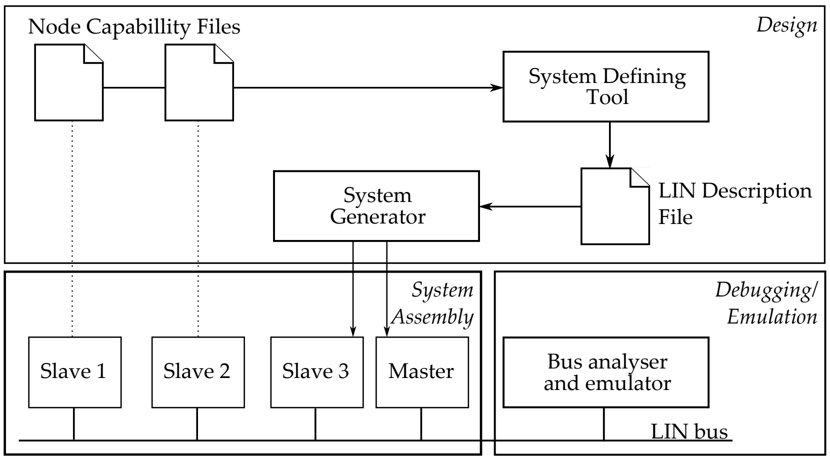


Figure 3.5: Development phases in LIN

- The free text section allows the inclusion of any help text, or more detailed, user–readable description.

The node capability file is a text file. the syntax is simple and similar to C. Properties are assigned using `name = value;` pairs. Subelements are grouped together using curly braces, equivalent to blocks in C.

LIN clusters are configured during the design stage using the *LIN Configuration Language*. This language is used to create a *LIN description file* (LDF). The LDF describes the complete LIN network. The development of a LIN cluster is partitioned into three phases (see figure 3.5). During the *design phase*, individual NCFs are combined to create the LDF. This process is called *System Definition*. For nodes to be newly created, NCFs can be created either manually or via the help of a development tool. From the LDF, communication schedules, and low–level drivers for all nodes in the cluster can be generated (*System Generation*). Based on the LDF, the LIN cluster can be emulated and debugged during the *Debugging and Node Emulation* phase. In the *System Assembly* phase, the final system is assembled physically, and put to service.

In addition to the LIN configuration language and LDF, which are the most important tools to design a LIN cluster, the LIN specification defines a (mandatory) interface to software device drivers written in C. Also, many tools exist that can parse a LDF and generate driver modules by themselves. The LIN C API provides a signal based interaction between the application and the LIN core (core API).

3.5 TTP/A

3.5.1 Communication System Architecture

The information transfer between a smart transducer and its communication partners is achieved by sharing information that is contained in an internal interface file system (IFS), which is situated in each smart transducer. The IFS provides a unique address scheme for transducer data, configuration data, self-describing information, and internal state reports of a smart transducer [Kop01]. It also serves as decoupling element, providing a push interface for producers writing to the IFS and a pull interface for consumers reading from the IFS. Each transducer can contain up to 64 files in its IFS. An IFS file is an indexed array of up to 256 records. A record has a fixed length of four bytes. Every record of an IFS file has a unique hierarchical address (which also serves as the global name of the record) consisting of the concatenation of the cluster name, the logical node name, the file name, and the record name.

A time-triggered sensor bus will perform a periodical time-triggered communication by sending data from IFS addresses to the fieldbus and writing received data to IFS addresses at predefined points in time. Thus, the IFS is the source and sink for all communication activities. Furthermore, the IFS acts as a temporal firewall that decouples the local transducer application from the communication activities.

Communication is organized into rounds consisting of several TDMA slots. A slot is the unit for transmission of one byte of data. Data bytes are transmitted in a standard UART format. Each communication round is started by the master with a so-called fireworks byte. The fireworks byte defines the type of the round and is a reference signal for clock synchronization. The protocol supports eight different firework bytes encoded in a message of one byte using a redundant bit code supporting error detection.

Generally, there are two types of rounds:

Multipartner round: This round consists of a configuration-dependent number of slots and an assigned sender node for each slot. The configuration of a round is defined in a data structure called "RODL" (ROund Descriptor List). The RODL defines which node transmits in a certain slot, the operation in each individual slot, and the receiving nodes of a slot. RODLs must be configured in the slave nodes prior to the execution of the corresponding multipartner round. An example for a multipartner round is depicted in Figure 3.6.

Master/slave round: A master/slave round is a special round with a fixed layout that establishes a connection between the master and a particular slave for accessing data of the node's IFS, e. g., the RODL information. In a master/slave round the master addresses a data record using a hierarchical IFS

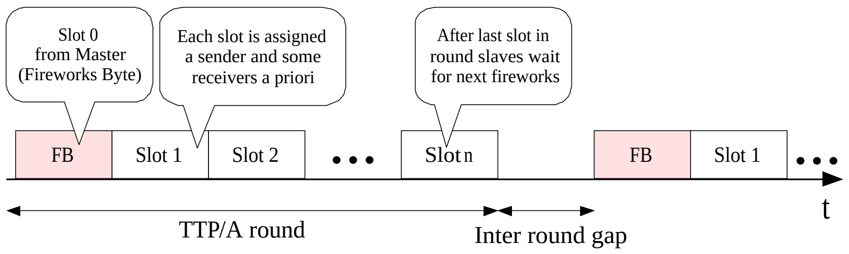


Figure 3.6: TTP/A Multipartner Round

address and specifies an action like reading of, writing on, or executing that record.

The multipartner (MP) round establishes a real-time communication service with predefined access patterns. Master/slave (MS) rounds are scheduled periodically between multipartner rounds, whereas the most commonly used scheduling scheme consists of MP rounds alternating with MS rounds. The MS rounds allow maintenance and monitoring activities during system operation without a probe effect. The MS rounds enable random access to the IFS of all nodes, which is required for establishing two conceptual interfaces to each node, a *configuration and planning* (CP) interface and a *diagnosis and management* DM interface. These interfaces are used by remote tools to configure node and cluster properties and to obtain internal information from nodes for diagnosis.

3.5.2 Smart Transducer Descriptions

For a uniform representation of all system aspects, an XML-based format is used [Pit03]. A *smart transducer descriptions* (STD) describe the node properties.

There are two types of STDs: *Static STDs* describe the node properties of a particular field device family. Static STDs contain node properties that are fixed at node creation time and act as a documentation of the nodes' features. In contrast, *Dynamic STDs* describe properties of individual nodes, as they are used in a particular application.

Instead of storing the STDs directly on a smart transducer, the node contains only a unique identifier consisting of a series and a serial number, whereas the serial number identifies the node type and the serial number differentiates instances of the same node type. This unique identifier is used to access the node's datasheet on an external server. Thus, node implementations keep a small footprint, while the size of the descriptions is not significantly limited.

	LIN	COSMIC	TTP/A
Criticality Levels	HRT, SRT	HRT, SRT	HRT
Flow control model	pull	push	TT, pull
Interaction pattern	master/slave	publish/ subscribe	TT, master/slave
Bounded transmission time	yes	yes	yes
Global Time	no	yes	yes
Synchronized actions	no	no	yes
Middleware abstraction	messages	event messages and channels	IFS
Device Descriptions Language	LIN-specific	XML	XML

Table 3.1: Feature comparison

3.5.3 Cluster Configuration Description

The cluster configuration description (CCD) contains descriptions of all relevant properties of a fieldbus cluster. It acts as the central structure for holding the meta-information of a cluster. With help of a software tool capable of accessing the devices in a smart transducer network it is possible to configure a cluster with the information stored in the CCD. A CCD consists of the following parts:

- *Cluster description meta information:* This block holds information on the cluster description itself, such as the maintainer, name of the description file, or the version of the CCD format itself.

- *Communication configuration information:* This information includes round sequence lists as well as round descriptor lists, which represent the detailed specification of the communication behavior of the cluster. Other properties important for communication include the UART specification and minimum/maximum signal run times.

- *Cluster node information:* This block contains information on the nodes in a cluster. These nodes are represented either by a list of dynamic STDs or by references to static STDs.

3.6 Discussion

Table 3.1 lists the main features of the three transducer networks with respect to the concepts describe in Section 3.2. All three approaches provide a real-time service with hard real-time message guarantees, but use different interaction design patterns. COSMIC comes with a publish-subscribe approach where nodes publish their data using the push principle. LIN is a master-slave network where each message is activated by the master. TTP/A uses a master-slave configuration in order to establish a common time base and then follows a predefined communication schedule based on the physical progression of time.

The basic scheduling mechanisms for hard real-time messages by using a static TDMA scheme is the same in all three approaches. The mechanisms for other data is different – TTP/A provides a polling mechanism via master-slave rounds, LIN 2.0 introduced event messages. COSMIC is the most flexible by providing EDF-scheduled soft real-time messages as well as non real-time messages. However, a full implementation of the SRTC requires substantial software because of the dynamic priorities and the more complex handling of discarded messages. Therefore, it has so far only be implemented on more powerful hardware under RT-Linux. Additionally, COSMIC relies on synchronized clocks while LIN and TTP/A require less effort for the proper protocol operation.

The advantages of COSMIC's publish-subscribe are a loose coupling between producer and consumer which facilitates the configuration of a network. The type of the channel to which EM of a certain type is pushed is defined by the publisher. The subscription and the respective guarantees for delivery at the subscriber side, however, may be of the same or a lower real-time class. This enables reception of a critical hard real-time message also for applications which do not need the respective delivery guarantees, e.g. a navigation task which uses critical messages from an obstacle avoidance system.

The pull principle in LIN makes a node's implementation very simple, but causes an overhead on the network due to the frequent message requests from the master. Moreover, since the LIN slaves do not know the time of a request *a priori*, it becomes difficult to time a measurement adequately or to synchronize measurements.

The time-triggered approach of TTP/A comes with high efficiency, predictability, and the possibility to synchronize actions. However, the configuration effort of a TTP/A network is higher than for a LIN device or COSMIC devices not requiring stringent real-time guarantees. For example, a TTP/A node has to be configured with the correct schedule before it can participate in the RT communication. In contrast, a LIN node or a COSMIC node might be reused in another application without reconfiguration of the node. Anyway, all three approaches depend on an adequate tool support.

The IFS concept of TTP/A is an abstraction mechanism that hides the time-triggered messages from the application. The IFS implements a distributed shared memory that provides a simple interface for applications. Therefore, TTP/A applications are not triggered by the reception of a message, which allows for a separation of communication and computation.

LIN is designed to serve as sub-bus in automobiles and is therefore specified in a very rigid way towards use in a specific end product. This makes the LIN architecture, though the approach is resource efficient and interesting, less suitable for applications which require a higher degree of cooperation between the nodes and also the rather constraint LIN message format restricts larger sensor-actuator systems. Also the LIN device description is rather focussed on the specific LIN application area.

In contrast, COSMIC and TTP/A specify several high-level features, while leaving details of physical and data link layer up to the implementer. The XML-based datasheets of COSMIC and TTP/A are easily extendable in order to support future extensions.

The mechanisms of the three approaches are different, which makes them incompatible in the first place. In order to achieve interoperability between heterogeneous networks, an adequate interface system, whereas the mechanisms of COSMIC and TTP/A are candidates rather than LIN. COSMIC provides a versatile message interface that abstracts over the underlying communication protocol. On the other hand, the IFS approach of TTP/A allows to abstract over the communication by establishing a distributed shared memory. The IFS comes with the main advantage of being easily adapted to a different protocol, however for convenient application development, tools supporting the set up of the distributed communication schedules are required. Thus, it is up to the application developer if a message-based interface (COSMIC) or a memory-based interface (TTP/A / IFS) is preferred.

3.7 Conclusion

The contributions of this paper are threefold: Firstly we have elaborated a set of requirements for different kinds of real-time constraints for a distributed system of smart transducers.

Secondly, we have presented and analyzed the concepts of three different smart transducer interface implementation approaches. Each approach has its specific focus concerning an application area. LIN is the protocol with the lowest hardware and cost requirements, however several design decisions restrict its use to an isolated sub-bus for automotive body electronics or simple control systems in industrial automation. LIN is supported by mature tools from automotive suppliers. TTP/A has a similar resource footprint as LIN but firstly substantially benefits from the strict time-triggered communication scheme and secondly provides a convenient distributed shared memory programming model where consistency problems are solved by the synchrony of the communication system. Parameters such as communication speed can be adapted in a rather flexible way depending on the physical network. This makes TTP/A an interesting choice for all kind of low-cost embedded time-triggered applications with real-time requirements. Additionally, the IFS is standardized by OMG in the Smart Transducer Interface Standard [OMG03]. COSMIC provides flexible real-time support and will integrate well into distributed applications with a publish-subscribe communication scheme. The main objective of COSMIC was interoperability between networks with different real-time properties. Thus, a higher overhead in the nodes may be needed. COSMIC and TTP/A come with different configuration support providing similar features

A third contribution of the paper is the discussion of device description. We think that this is an important issue because it firstly underlines the hardware/software (and probably mechanical) nature of a smart transducer and the intrinsically component-based system structure and secondly is indispensable in a complex reliable control system. Presently, device descriptions are mainly used during system configuration to avoid faults from manual set-up. The LIN NFC and also LDF exactly meet these requirements. Device description of TTP/A and COSMIC go beyond the needs of configuration and also are intended for dynamic use. This can range from diagnostic purposes to dynamic device discovery and use during operation.

Although being quite different, we think that it will be possible to establish methods and tools that operate on a meta-level and can be used to configure an application using different underlying fieldbus systems. In order to achieve this, a generic interface model for transducer data has to be found. The IFS presented with TTP/A seems to be a promising approach for forming a generalized interface, since it is relatively easy to convert transducer data onto an IFS. We will further investigate ways to provide coexistence and cooperation between the different network and programming models.

Acknowledgments

This work was supported in part by the ARTIST2 Network of Excellence on Embedded Systems Design under contract No. IST-004527 and by the Austrian FWF project TTCAR under contract No. P18060-N04. We would like to thank

Christian Paukovits, Stefan Pitzek, Gernot Klingler, Christian El-Salloum and Andreas Pfandler for constructive comments on an earlier version of this paper.

References

[Ala98] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *International Journal of Robotics Research*, 17(4):315–337, Apr. 1998.

[Aud03] Audi AG, BMW AG, DaimlerChrysler AG, Motorola Inc. Volcano Communication Technologies AB, Volkswagen AG, and Volvo Car Corporation. LIN Specification V2.0, 2003.

[DeL99] R. DeLine. *Resolving Packaging Mismatch.* PhD Thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, June 1999.

[Fre02] L.-B. Fredriksson. CAN for critical embedded automotive networks. *IEEE Micro*, 22(4):28–36, 2002.

[Ger94] M. Gergeleit and H. Streich. Implementing a distributed high–resolution real–time clock using the CAN–Bus. In *1st International CAN Conference*, 1994.

[Gmb91] Robert Bosch GmbH. CAN Specification Version 2.0, Sep. 1991.

[iAe] CAN in Automation e.V. CANopen - Communication Profile for Industrial Systems. available at http://www.can-cia.de/downloads/.

[IEE97] Institute of Electrical and Electronics Engineers, Inc. *IEEE Std 1451.2-1997, Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Micro-processor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*, 1997.

[Kai99a] J. Kaiser and M. A. Livani. Achieving Fault–Tolerant Ordered Broadcasts in CAN. In *Proceedings of the Third European Dependable Computing Conference (EDCC–3), Prague*, Sep. 1999.

[Kai99b] J. Kaiser and M. Mock. Implementing the Real–Time Publisher/Subscriber Model on the Controller Area Network (CAN). In *Proceedings of the 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 172–181, Saint Malo, France, May 1999.

[Kai02] J. Kaiser and C. Brudna. A Publisher/Subscriber Architecture Supporting Interoperability of the CAN–Bus and the Internet. In *Proceedings of the 4th IEEE International Workshop on Factory Communication Systems (WFCS 2002), Västerås, Sweden*, 2002.

[Kai06] J. Kaiser and H. Piontek. CODES: Supporting the development process in a publish/subscribe system. In *Proceedings of the fourth Workshop on Intelligent Solutions in Embedded Systems WISES 06*, 2006.

[Kop01] H. Kopetz, M. Holzmann, and W. Elmenreich. A Universal Smart Transducer Interface: TTP/A. *International Journal of Computer System Science & Engineering*, 16(2):71–77, Mar. 2001.

[Krü97] A. Krüger. *Interface Design for Time-Triggered Real-Time System Architectures*. PhD Thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, Apr. 1997.

[Lan97] K. Langendoen, R. Bhoedjang, and H. Bal. Models for asynchronous message handling. *IEEE Concurrency*, 5(2):28–38, Apr.-June 1997.

[M. 06] M. Kay, Ed. W3C XSL Transformations (XSLT) Version 2.0. http://www.w3.org/TR/xslt20, June 2006.

[McD89] C. E. McDowell and D. P. Helmbold. Debugging Concurrent Programs. *ACM Computing Surveys*, 21(4):593–622, Dec. 1989.

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

[OMG03] Object Management Group (OMG). *Smart Transducers Interface V1.0*, Jan. 2003. Specification available at http://doc.omg.org/formal/2003-01-01 as document ptc/2002-10-02.

[Pit03] S. Pitzek and W. Elmenreich. Configuration and Management of a Real-Time Smart Transducer Network. In *Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2003)*, pages 407–414, Lisbon, Portugal, Sep. 2003.

[Pit05] S. Pitzek and W. Elmenreich. Plug-and-Play: Bridging the Semantic Gap Between Application and Transducers. In *Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA05)*, volume 1, pages 799–806, Catania, Italy, Sep. 2005.

[Ruf98] J. Ruffino, P. Verissimo, C. Almeida, and L. Rodrigues. Fault–Tolerant Broadcasts in CAN. In *Proceedings FTCS–28, Munich, Germany*, 1998.

Chapter 4

Modeling Distributed Embedded Applications on an Interface File System

Wilfried Elmenreich, Stefan Pitzek and Martin Schlager. IEEE International Symposium on Object-Oriented Real-time Distributed Computing, Vienna, Austria, May 2004, pages 175–182.

This paper presents a framework for generic modeling of distributed embedded applications. An application is decomposed into services and mapped on a set of distributed nodes, whereas each node hosts one or more services. Each service is described by four interfaces: a real-time input/output, a configuration and planning (CP), and a diagnostic and management (DM) interface. The overall application is described by a cluster configuration description that specifies the interaction of services within and across nodes.

The application requirements, the service properties of a node, and the interaction of the services as well as the application mapping are described formally with XML descriptions. The XML format allows a language-neutral and extensible semantic description of interfaces supporting the implementation of context-aware tools for modeling, scheduling, monitoring, simulation, and validation.

A central concept of the model is the interface file system (IFS) that acts as a distributed shared memory and transparently imple-

ments the interfaces to services from other nodes. In principle, the communication system that updates the data in the IFS data is not bound to a specific implementation as long as it fulfills the given timing requirements. The presented concepts are applied in a case study that uses the time-triggered fieldbus protocol TTP/A for the implementation of a small sensor fusion application.

4.1 Introduction

An embedded system is a computer system designed to perform a dedicated or narrow range of functions with minimal user intervention. By definition, an embedded system can be built either in a monolithic or a distributed way. In this paper we focus on distributed embedded systems. Distributed embedded systems are motivated by several aspects, i. e., (i) the parts of a system, e. g., sensors and actuators, may be situated spatially far apart from each other (as it is the case in factory and building automation), (ii) the need for parallelism (e. g., grid computing, DSP for special tasks), (iii) the need for fault tolerance (e. g., by introducing redundant nodes and channels), and, (iv) the need for a modular design (e. g., extensions for PCI bus).

Such a distributed embedded system can be treated as a composition of several subsystems or components, whereof each component consists of a hardware and software part. The implementation of these components may differ in used programming language (e. g., Assembler, VHDL, C, C++, Java), timing behavior (in terms of throughput, worst-case behavior, response time), I/O types (digital/analog, determinism, ...), and dependability (reliability, availability, maintainability, ...). Finding an adequate model for such systems is difficult, since on the one hand, the model must be rigid in order to describe critical parameters such as timing and dependability with sufficient precision, on the other hand, due to the many different application requirements, the model should be flexible and extensible.

This paper focuses on the modeling of distributed embedded applications based on a description of components in an XML-based specification language. Each service is mapped onto an IFS that forms its input and output interface. Additionally, the IFS contains configuration and maintenance data of the subsystem, thus all *state* information of the subsystem is contained in the IFS. All other aspects, such as the syntactic descriptions of the information in the IFS, are mapped into an external XML description of the component. The IFS acts as a common interface that supports a unified mapping of interfaces, while the extensible XML descriptions provide an extensible semantic description for them.

This paper is structured as follows: The next section outlines important requirements for embedded real-time systems. Section 4.3 depicts the conceptual model of the framework. Section 4.4 shows the modeling via the IFS. Section 4.5 presents the implementation of the framework in the time-triggered fieldbus network TTP/A. Section 4.6 presents a case study application of the framework for real-time simulation of TTP/A functions. Section 4.7 lists related approaches and Section 4.8 concludes the paper.

4.2 Requirements

The following requirements are prevalent in many distributed embedded systems:

Hard real-time support: A hard real-time system is a real-time system in which a guarantee can be given that a certain action will always finish before a given deadline. Many embedded systems require hard real-time behavior in order to avoid damage to man or machine.

Support of low-cost embedded systems: Although state-of-the-art technology provides powerful 32 bit architectures, the presented modeling approach shall also support applications on networks of small 8-bit microcontrollers such as the ATMEL AVR, Microchip PIC, and the Motorola HC08 families. Still, 8-bit microcontrollers have the greatest share in the volume market for microcontrollers.[1]

Two-level design approach: The two-level design approach proposes the separation of the implementation of subsystems and the integration of these subsystem into an overall system. Thus, the implementor of a subsystem focuses on local issues such as interfacing local sensors and actuators, while the system integrator focuses on the interaction of the subsystems.

In order to support a two-level design approach the architecture must support composability [Kop00]. An architecture is composable with respect to a specified property if the system integration does not invalidate this property when it was established at the subsystem level.

Reuse of existing systems: Many projects require that legacy code and legacy components must be included into an application. Thus, the presented modeling approach shall support the reuse of existing solutions.

[1]Source: Gartner Dataquest (August 2003)

4.3 Conceptual Model

A distributed embedded application will be first described *functionally*, i. e., by a set of interconnected real-time *services*. A service is described by its interfaces, its function, and properties like timing behavior or reliability requirements.

The interfaces of a service are divided into the following categories:

Service Providing Linking Interface (SPLIF): This interface provides the real-time service results to other services (cf. [Jon02]).

Service Requesting Linking Interface (SRLIF): A service that requires real-time input information requests these data via the SRLIF (cf. [Jon02]).

Diagnostic and Management (DM): This interface is used to set parameters and to retrieve information about intermediate and debugging data, e. g., for the purpose of fault diagnosis. Access of the DM interface does not change the (a priori specified) timing behavior of the service.

Configuration and Planning (CP): This interface is used during the integration phase to generate the "glue" between the nearly autonomous services (e. g., communication schedules). The CP interface is not time critical.

Local interfaces: The term local interfaces subsumes all kinds of devices, such as sensors, actuators, displays, and input devices, for which the service creates a unified access via the SPLIF or SRLIF services. For example, the service may instrument a physical sensor element by reading the measurement, calibrating the value, and exporting the measurement via its SPLIF.

Figure 4.1 depicts the interfaces of a service in a block diagram.

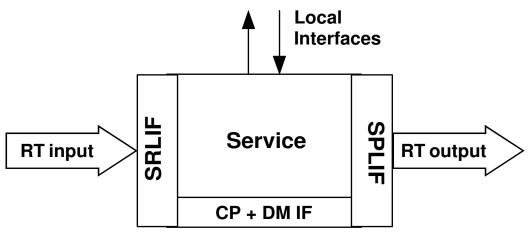


Figure 4.1: Interfaces of a Service

A particular component may comprise only a subset of the above described interface categories. Typically, a *smart sensor* service will implement a SPLIF, CP, DM, and a local interface to the physical sensor. An *actuator*, in contrast, will implement an SRLIF, CP, DM, and a local interface to the physical actuator.

Data flow over the SPLIF and SRLIF is performed using *ports*. A port is specified by a name, a description, and the structure of the data transmitted over the port (e. g., a 16bit measurement value from a sensor). The port structure consists of the data type of the expected input or, respectively, the produced output.

The functional behavior of a service is implemented by a service *task*. The task of a service consumes the data at its SRLIF and produces an output at its SPLIF after termination. The data at the SRLIF must not be changed during task execution and the data at the SPLIF must not be read during task execution. The behavior of a task may also depend on configurations via the DM interface, whereas this configuration is not allowed to change during task execution.

An application consists of one or more services that interact with each other via the real-time interfaces SPLIF and SRLIF. One or more services are implemented per *node*, whereas nodes are loosely coupled via a real-time communication system. Services that require local data exchange communicate via a shared memory interface. Services that are required to exchange data between different nodes communicate via the node's communication interface.

The representation of an application by its services deals with the functional and data flow parts of the application. In order to completely represent an actual application, several non-functional properties must be specified. We distinguish the following two classes:

- Service-specific properties that must be set according to requirements of the application (e. g., the update rate of an actuator).
- End-to-end requirements of the application that can only be specified over the distributed application (e. g., end-to-end signal delay in control loops).

The service-specific properties are modeled in the semantic description of the service. The end-to-end requirements are expressed by so-called *dependencies*. We have identified the following kinds of dependencies:

Connection: This dependency represents the data flow between ports of services. A connection is directed by having a source and a target part. An input port may have only one connection to an output port, while an output may feed several input ports.

Causal: By defining a causal dependency between services, all in-between services (on the directed application graph) must comply to this dependency. In our context causal dependencies always incur timing requirements. An *instant* identifies the timing requirements of participating services. We distinguish the instant before and after execution. The *before* instant of a service is before the service task is executed, i. e., the moments when input data must have arrived. The *after* instant happens after the service task execution, thus a duration of WCET_{TASK} after the before instant, where WCET_{TASK} is the worst case execution time of the service task.

Phase: The phase dependency specifies non-causal time-related dependencies of instants among services.

All the presented properties are used to model the application requirements. All requirements are expressed in XML descriptions, which support interaction among tools (e. g., for modeling, code generation, and verification). A case study that depicts concrete XML descriptions is shown in Section 4.6.

4.4 Interface Implementation

An interface must establish a common boundary between services and their users. In order to exchange information across such an interface the engaged communication partners must share a common background of concepts [Kop01].

Basically, the interface to a service will be modeled as a structured shared memory. All interfaces are modeled upon an Interface File System (IFS) [Kop01] that provides a common name space by a uniform addressing scheme. The values that are mapped into the IFS are organized in a static file structure. The address space is organized hierarchically representing the network structure:

Cluster name: A cluster comprises a network of fully interconnected nodes. The cluster name is an 8 bit integer value that identifies a particular cluster. Native communication (without routing) among nodes is only possible within the same cluster.

Node alias: The node alias or logical name selects a particular node. Node aliases can have values from 0...255, whereas some values have an associated special function, e. g., alias 0 addresses all nodes of a cluster at once, i. e., a broadcast manner [OMG03].

File name: The file name is a 6-bit identifier for addressing individual files in the nodes. A subset of files, the system files, have a special meaning in all nodes. Each service of a node is mapped onto a file containing sections for SPLIF, SRLIF, CP, and DM data.

Record number: The record number is an 8-bit identifier that addresses the record within the selected file. Each record contains 4 data bytes. Since each file has a statically assigned number of records, a file contains only the necessary number of records.

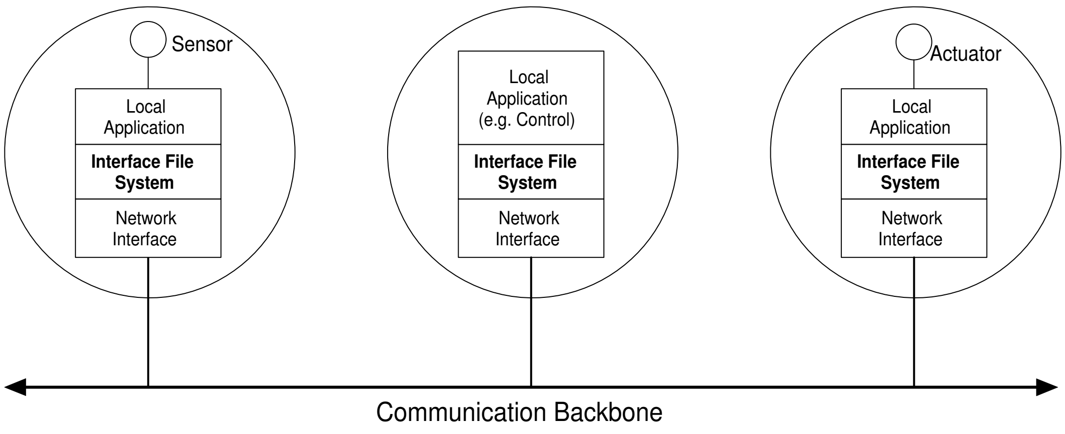


Figure 4.2: Physical Network Topology

Figure 4.2 depicts a possible application with a network of three nodes. Each node may host one or multiple services that all use the local IFS as a data source and sink. The communication system will perform a periodical time-triggered communication to copy data from the IFS to the communication system and write received data into the IFS. Thus, the IFS acts as an interface that decouples the local services from the communication subsystem. It is the task of the communication system to keep consistency among the local copies of the data stored in the IFS. A predefined communication schedule defines time, origin, and destination of each communication action.

As depicted in Figure 4.3, local applications share a common view on the IFS. The application programmer is relieved from considerations concerning low-level communication. Thus, any service perceives the IFS in a logical network structure as depicted in Figure 4.3.

The SPLIF, SRLIF, CP, and DM interfaces of a service are mapped into a file of the IFS on the node that hosts the service. The IFS maps the current state of a all process variables into so-called I/O files. Additionally there may be configuration data, for example communication schedules. In [Pit03] we have described the representation and use of the IFS CP interfaces based on XML-based descriptions. In the following we will describe the structure of the SPLIF and SRLIF, which establish the real-time service.

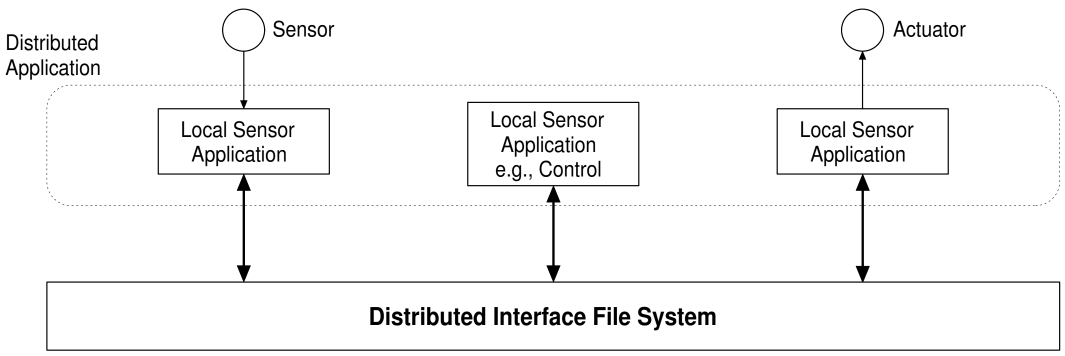


Figure 4.3: Logical Network Structure

Each port in the service file specifies a pointer into the local I/O file of the node. The I/O file hosts the actual input values and the output values of the local services. The input values can be provided either by a local or remote service – in the latter case, the communication system will update this value periodically

Besides the ports, the service file contains service configuration data (e. g., parameters for a PID control algorithm) and diagnostic data such as intermediate results or sensor logs.

4.5 Communication System

The communication system that performs the updates of the IFS data is not bound to a specific implementation as long as it provides deterministic timing behavior in order to fulfill the timing requirements.

We use the time-triggered fieldbus protocol TTP/A, since it meets the timing requirements and supports low-cost implementations of network nodes. TTP/A is standardized as part of the Object Management Group Smart Transducer Interface Standard [OMG03].

4.5.1 Principles of Operation

In TTP/A, the bus allocation is controlled by a Time Division Multiple Access (TDMA) scheme. Communication is organized into rounds consisting of several TDMA slots. A slot is the unit for transmission of one byte of data. Data bytes are transmitted in a standard UART format. Each communication round is started by the master with a so-called fireworks byte. The fireworks byte defines the type of the round and is a reference signal for clock synchronization. The protocol supports eight different firework bytes encoded in a message of one

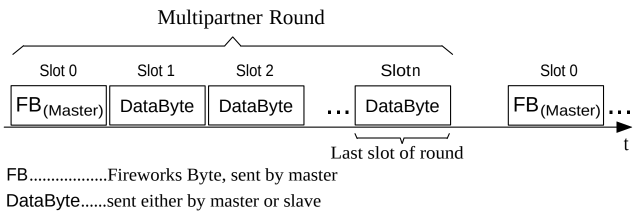


Figure 4.4: Example for a TTP/A multipartner round

byte using a redundant bit code supporting error detection. Generally, there are two types of rounds:

Multipartner round: This round consists of a configuration dependent number of slots and an assigned sender node for each slot. The configuration of a round is defined in a datastructure called "RODL" (ROund Descriptor List). The RODL defines which node transmits in a certain slot, the operation in each individual slot (read, write, execute), and the receiving nodes of a slot. RODLs must be configured in the slave nodes prior to the execution of the corresponding multipartner round. An example for a multipartner round is depicted in Figure 4.4.

Master/slave round: A master/slave round is a special round with a fixed layout that establishes a connection between the master and a particular slave for accessing data of the node's IFS, e. g., the RODL information. In a master/slave round the master addresses a data record using a hierarchical IFS address and specifies one of the following actions: reading from, writing to, or executing a record.

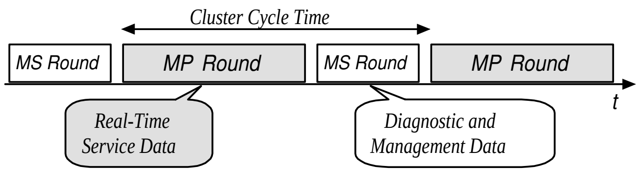
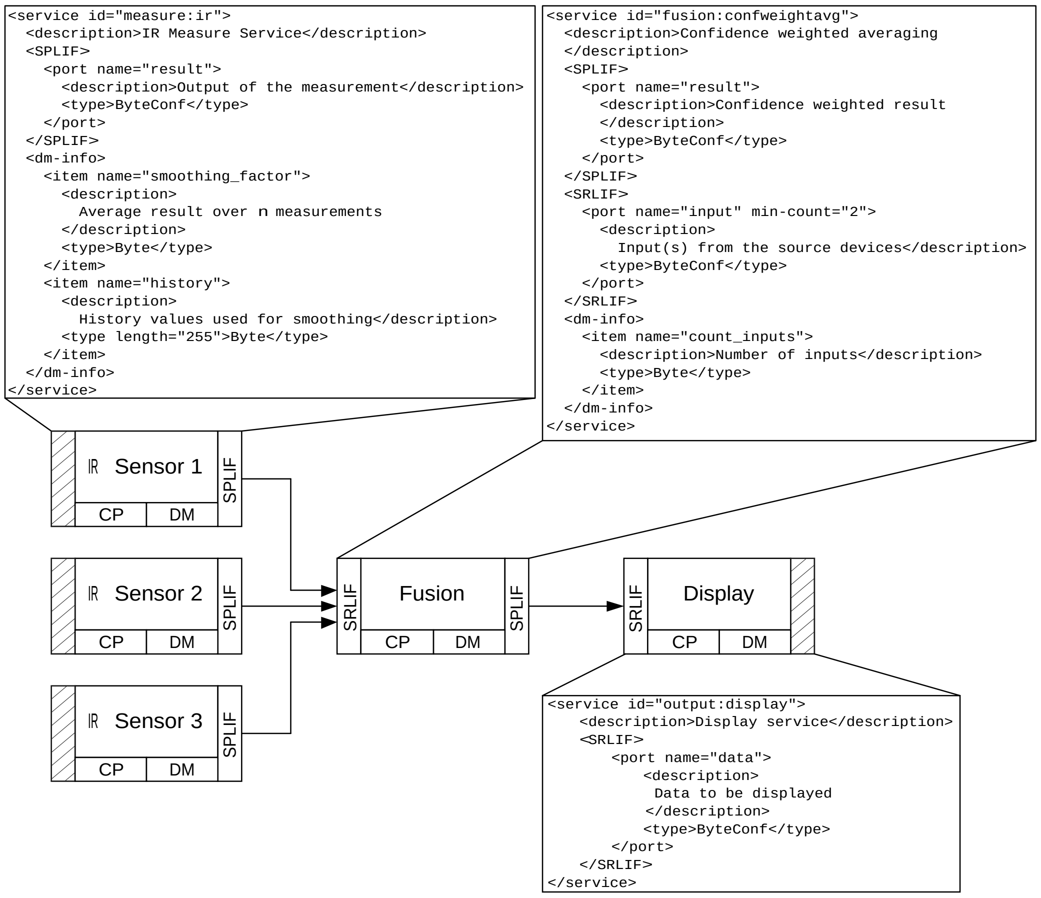


Figure 4.5: Recommended TTP/A Schedule

The master/slave rounds implement the DM and the CP interface to the transducer nodes. The RS interface is provided by periodical multipartner rounds. Master/slave rounds are scheduled periodically between multipartner rounds as depicted in Figure 4.5 in order to enable maintenance and monitoring activities during system operation without a probe effect [Gai86].



```
<service id="measure:ir">
  <description>IR Measure Service</description>
  <SPLIF>
    <port name="result">
      <description>Output of the measurement</description>
      <type>ByteConf</type>
    </port>
  </SPLIF>
  <dm-info>
    <item name="smoothing_factor">
      <description>
        Average result over n measurements
      </description>
      <type>Byte</type>
    </item>
    <item name="history">
      <description>
        History values used for smoothing</description>
      <type length="255">Byte</type>
    </item>
  </dm-info>
</service>
```

```
<service id="fusion:confweightavg">
  <description>Confidence weighted averaging
  </description>
  <SPLIF>
    <port name="result">
      <description>Confidence weighted result
      </description>
      <type>ByteConf</type>
    </port>
  </SPLIF>
  <SRLIF>
    <port name="input" min-count="2">
      <description>
        Input(s) from the source devices</description>
      <type>ByteConf</type>
    </port>
  </SRLIF>
  <dm-info>
    <item name="count_inputs">
      <description>Number of inputs</description>
      <type>Byte</type>
    </item>
  </dm-info>
</service>
```

```
<service id="output:display">
    <description>Display service</description>
    <SRLIF>
        <port name="data">
            <description>
             Data to be displayed
            </description>
            <type>ByteConf</type>
        </port>
    </SRLIF>
</service>
```

Figure 4.6: Conceptual model of case study

4.6 Case Study

The case study implements a distributed application performing a robust distance measurement. We will show how an application can be modeled according to the presented concepts, providing XML-based descriptions for services and applications and defining an actual low-level mapping of services to the IFS.

4.6.1 Application Specification

The case study hardware comprises three infrared (IR) distance sensors and a display. The application shall perform a reliable distance measurement from the three distance sensors, fuse the measurements and display the result onto the display. All of the three distance sensors have to perform synchronous measurements. According to the precision of these measurements, on each sensor a confidence value (0 for lowest confidence, 12 for highest confidence) is

Service	Interface	Description
IR sensor	SRLIF	– (not required)
IR sensor	SPLIF	provide distance and confidence values
IR sensor	CP	configure real-time communication pattern
IR sensor	DM	calibrate IR sensor (history values, precision)
Fusion	SRLIF	receive distance and confidence values
Fusion	SPLIF	provide filtered distance and confidence values
Fusion	CP	configure real-time communication pattern
Fusion	DM	configure fusion algorithm
Display	SRLIF	receive filtered distance and confidence values
Display	SPLIF	– (not required)
Display	CP	configure real-time communication pattern
Display	DM	– (not required)

Table 4.1: Description of Service Interfaces

calculated as a result of fusing subsequent measurements from the local sensor. The fusion service performs a confidence-weighted average fusion [Elm02a] on the three value/confidence data pairs from the distance measurements.

Figure 4.6 depicts the service model of the application. Blocks represent the services and lines between blocks show the data flow in the model. The IR sensor service does not receive messages from other network participants, thus it has no service requesting linking interface (SRLIF). The fusion service takes the outputs from the IR sensor services (i. e., distance and confidence values from the IR sensors) as inputs, executes the fusion algorithm, and produces the fused distance and an according confidence value as outputs. Thus, the fusion service contains both, a service requesting linking interface (SRLIF) and a service providing linking interface (SPLIF). Finally, the display service takes the outputs from the fusion service for displaying it. Since the display service does not send messages to other services, it does not implement a SPLIF.

As indicated in Figure 4.6, each service (IR sensor, fusion, display) is also equipped with a configuration and planning (CP) interface and a diagnostic and maintenance (DM) interface. The CP interface is used at setup time in order to configure the time-triggered communication messages among the nodes. The DM interface can be used for diagnostic and for configuration of local services. Table 4.1 gives an overview on the services and interfaces of the case study.

Figure 4.6 also shows the external XML-based service descriptions that provide a straightforward formal representation of the properties of the service. These descriptions take a similar role as classes in object oriented programming, since they act as generic types of services, which are then instantiated in the application.

Note that the respective elements in the XML representation are intentionally kept simple for brevity and easier implementation, but it would be easy

```
<application>
  <service id="IR1" spec-location="measureIR.xml"/>
    ...
  <service id="DISPLAY" spec-location="display.xml">
    <property name="d_update"><dur bound="maxInclusive">
        0.1<unit>s</unit></dur></property>
  </service>
  <dependency kind="connection">
    <source service="IR1" port="result"/>
    <target service="FUSION" port="input"/>
  </dependency>
    ...
  <dependency kind="connection">
    <source service="FUSION" port="out"/>
    <target service="DISPLAY" port="data"/>
  </dependency>
  <dependency kind="causal">
    <instant service="IR1" type="before"/>
    <instant service="DISPLAY" type="before"/>
    <property name="d_acc"><dur bound="maxExclusive">
        0.05<Unit>s</Unit></dur></property>
  </dependency>
  <dependency kind="causal">
    ...
  </dependency>
  <dependency kind="phase">
    <instant service="IR1" type="before"/>
    <instant service="IR2" type="before"/>
        <property name="phase">0<Unit>ms</Unit></property>
    <property name="precision">
      <dur bound="maxInclusive">1<Unit>ms</Unit></dur>
      <dur bound="maxInclusive">-1<Unit>ms</Unit></dur>
    </property>
  </dependency>
  <dependency kind="phase">
      ...
  </dependency>
    ...
</application>
```

Figure 4.7: Representation of the application with XML

to provide semantically richer specifications for parts of the descriptions, e. g., using XML schema for describing the layout of the IFS elements instead of the currently used proprietary data type formats.

For the application in this case study we can identify the following application-specific requirements:

1. The value shown on the display must be updated at least every 0.1s ($d_{update} = 0.1s$).
2. IR measurements must be synchronized with a precision of $\pm 1ms$.
3. The temporal accuracy d_{acc} of the value received by the display service must be 0.05s.

Figure 4.7 shows fragments from a description of the case-study application in XML. The `service` elements specify the "instantiated" services of the application model. Each `service` element defines an application-wide unique identifier and refers to the location of the associated service description file. Since d_{update} is a property of the display service, it is defined as part of this local service specification. The rest of the application is described using dependency elements.

The `property` elements used in the dependencies specify the relation between the services in detail (e. g., defining the required precision, upper and/or lower bounds for values). For example, the requirement that the three sensor services should be synchronized within 1ms is expressed by specifying that service IR1, IR2, and IR3 must run within a phase of $0 \pm 1ms$.

The third application specific requirement is modeled as a causal dependency between the IR service and the display service, i. e., the time between the start of execution of the IR service and receiving the fused value by the display service must not be greater than 0.05s.

4.6.2 Implementation Model

Figure 4.8 depicts the mapping of services onto network nodes. The network comprises three autonomous nodes hosting the infrared distance sensors, a node for the display, and a master node that hosts the fusion service. The master node acts as a reference clock for the time-triggered communication within the cluster.

All relevant data of a node has been mapped in the node's interface file system (IFS). The IFS forms the SPLIF, SRLIF, CP, and DM interface for each node. All services, like measurement, triggering a fusion process, and updating the displayed data are mapped as tasks in corresponding IFS files.

4.6.3 Evaluation

The resulting system was implemented on five 8-bit low-cost microcontrollers (Atmel AVR8 family) which are interconnected by an ISO 9141 bus system

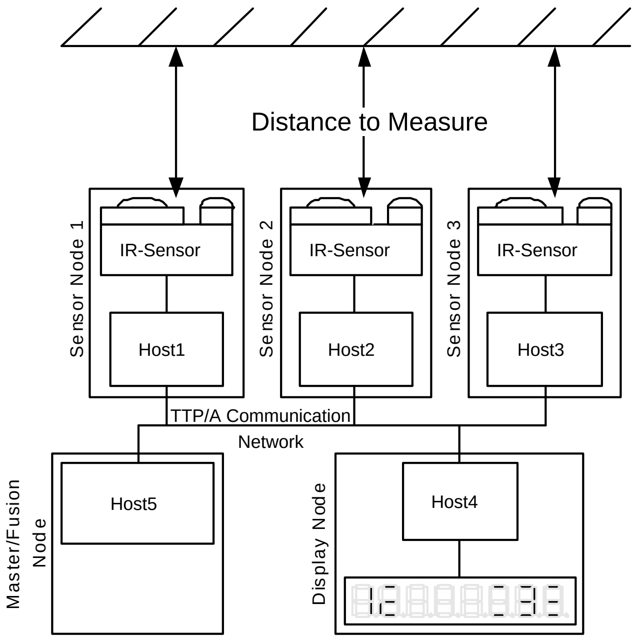


Figure 4.8: Nodes of the case study application

running at a communication speed of 9600 Bit/s. Figure 4.9 depicts the hardware used for the case study, which consists of a master node (implemented on an AT90S8515), display node (ATmega128) with display unit, and three IR sensor nodes (ATmega128) with IR sensors (Sharp GP2D12).

The timing requirements defined in section 4.6.1 are fulfilled as follows:

- The display was updated every 25.73ms. ✓
- The sensor measurements are synchronized within $\pm 0.104ms$. ✓
- The value received by the display service is based on sensor measurements made 13.54ms before the instant of displaying. ✓

4.6.4 Further Aspects of the Case Study

Given a real-time fieldbus system, the above mentioned interface concept eases the seamless integration of simulation aspects. Each functional service is represented as an independent object, that can be mapped to any existing hardware basis. Due to such encapsulation, each node of the distributed system can be emulated through a simulation host with the same functional and temporal behavior.

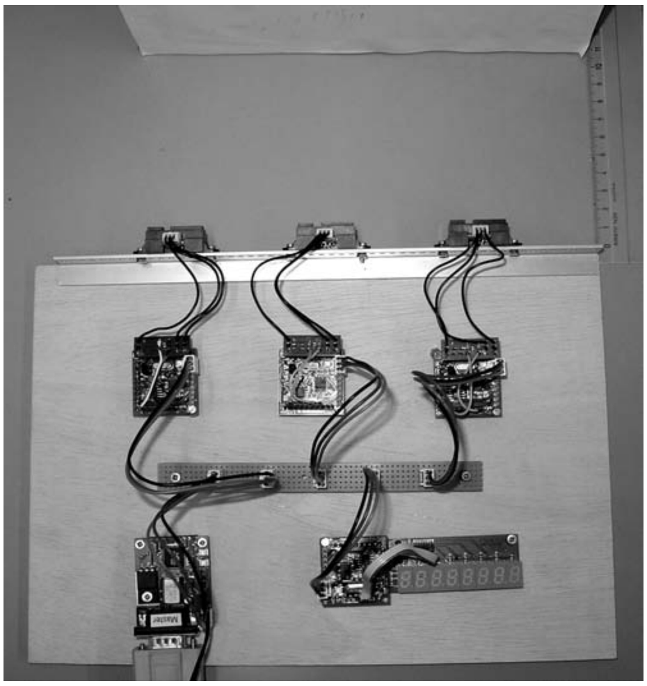


Figure 4.9: Hardware setup of the case study application

In experiments, we emulated two out of the three IR sensors through a simulation node [Sch03]. The simulation unit communicates via its SPLIF and SRLIF with the other network participants.

4.7 Related Work

Many approaches for model- or component-based application modeling only specify components by describing their functional interface. In order to support modeling of real-time applications, additional properties such as timing and other non-functional properties (dependability, quality of service [Ham01a]) must be taken into account. There exist several approaches that address the timing requirements.

Giotto [Hen03] represents a domain-specific high-level programming language for control applications that exactly specifies the real-time interaction between software components and the real world.

The CIP method (described by Fierz in [Fie99]) decomposes the construction of embedded systems into a functional and a connection part.

The BASEMENT^{TM} [Han97] architecture presents a high-level application modeling approach based on software circuits, also taking resource requirements and timing into account.

Even though these approaches are targeted to embedded systems they cannot be easily adopted to ultra-low cost systems. An approach that can be compared to the IFS-based modeling approach is the IEEE1451 standard [Joh01] that specifies an application model and a digital low-level interface for smart transducers. In [Elm03] we discuss the main differences in the design decisions between both approaches. For example, IEEE1451 specifies digital communication lines as common interface mechanisms, whereas the CORBA STI interfacing mechanism is based on a shared memory concept.

XML has already been used for describing interfaces of conventional software components, e. g., by Bramley et al.[Bra00], in the Web Services Description Language (WSDL) [WSD03], and, for the description of communication and computation properties, in the Communication-Computation Description Language (CCDL) for ModelJ components [Ham01b].

4.8 Conclusion and Outlook

The presented modeling approach builds on generic services that are hosted in nodes which are interconnected by a real-time system. The final application is described by a cluster configuration description that specifies the interaction of services within and across nodes.

The basic concept of the model is the interface file system, which acts as a distributed shared memory and transparently maps the interfaces to services from other nodes. The communication system that performs the updates of the IFS data is, in principle, free to choose as soon as it fulfills the requirements of timeliness and determinism that allow a static analysis of the timing properties of the distributed application. In this paper we have used the time-triggered fieldbus protocol TTP/A for the communication system.

The presented model supports a top-down as well as a bottom-up approach. In a top-down approach, the system will first be described by abstract services, which are then grouped into nodes and implemented as concrete services in a node according to the required service specification. The implementation may also be done automatically by employing an embedded code generation tool.

The bottom-up approach will involve existing implementations of nodes that are used as legacy systems within the application. In this case it is necessary to verify that the services of the nodes comply to the application requirements. Since both, the application requirements and the service properties of a node can be described formally in an XML document, there is an inherent support for automated tools that verify the implementation to its specification.

The presented case study has shown a simple application of the modeling approach. Further steps will include cluster emulation for more complex applications.

In the future we plan to extend the node and service XML descriptions by tags describing power consumption, weight, and dependability properties. Thus, it will be possible to add constraints on these properties for the application specification of an embedded system.

4.9 Acknowledgments

We would like to thank our colleagues Wilfried Steiner, Christian Trödhandl and Ingomar Wenzl for their comments on earlier versions of this work. This work was supported in part by the Hochschuljubiläumsstiftung der Stadt Wien via project CoMa (H-965/2002) and by DOC [DOKTORANDENPROGRAMM DER ÖSTERREICHISCHEN AKADEMIE DER WISSENSCHAFTEN].

References

[Bra00] R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko, and M. Yechuri. A Component Based Services Architecture for Building Distributed Applications. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, pages 51–62, Aug. 2000.

[Elm02] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems.* PhD Thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[Elm03] W. Elmenreich and S. Pitzek. Smart Transducers – Principles, Communications, and Configuration. In *Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems*, volume 2, pages 510–515, Assuit – Luxor, Egypt, Mar. 2003.

[Fie99] H. Fierz. The CIP method: component- and model-based construction of embedded systems. In *Proceedings of the 7th European Engineering Conference held jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 375–392. Springer-Verlag, 1999.

[Gai86] J. Gait. A Probe Effect in Concurrent Programs. *Software Practice and Experience*, 16(3):225–233, Mar. 1986.

[Ham01a] D.K. Hammer and M.R.V. Chaudron. Component-Based Software Engineering for Resource-Constraint Systems: What are the Needs? In *Sixth International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'01)*, pages 91–96, Jan. 2001.

[Ham01b] R. Hamouche, B. Miramond, and B. Djafri. ModelJ: Component-based Modeling for Embedded Systems. In *European Conference on Object Oriented Programming (ECOOP'01)*, June 2001.

[Han97] H. Hansson, H. Lawson, O. Bridal, C. Eriksson, S. Larsson, H. Lön, and M. Strömberg. BASEMENT: An Architecture and Methodology for Distributed Automotive Real-Time Systems. *IEEE Transactions on Computers*, 46(9):1013–1027, Sep. 1997.

[Hen03] T. A. Henzinger, C. M. Kirsch, M. A. A. Sanvido, and W. Pree. From Control Models to Real-time Code Using Giotto. *IEEE Control Systems Magazine*, 23(1):50–64, 2003.

[Joh01] R. Johnson, K. Lee, J. Wiczer, and S. Woods. A Standard Smart Transducer Interface - IEEE 1451. Presentation held at the Sensors Expo, Philadelphia, Oct. 2001.

[Jon02] C. Jones, M.-O. Killijian, H. Kopetz, E. Marsden, N. Moffat, D. Powell, B. Randell, A. Romanovsky, R. Stroud, and V. Issarny. Final Version of the DSoS Conceptual Model. *DSoS Project (IST-1999-11585) Deliverable CSDA1*, Oct. 2002. Available as Research Report 54/2002 at http://www.vmars.tuwien.ac.at.

[Kop00] H. Kopetz. Composability in the Time-Triggered Architecture. *SAE World Congress 2000, Detroit, Michigan, USA*, Mar. 2000.

[Kop01] H. Kopetz, M. Holzmann, and W. Elmenreich. A Universal Smart Transducer Interface: TTP/A. *International Journal of Computer System Science & Engineering*, 16(2):71–77, Mar. 2001.

[OMG03] Object Management Group (OMG). *Smart Transducers Interface V1.0*, Jan. 2003. Specification available at http://doc.omg.org/formal/2003-01-01 as document ptc/2002-10-02.

[Pit03] S. Pitzek and W. Elmenreich. Configuration and Management of a Real-Time Smart Transducer Network. In *Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation*, volume 1, pages 407–414, Lisbon, Portugal, Sep. 2003.

[Sch03] M. Schlager. A Simulation Architecture for Time-Triggered Transducer Networks. In *Proceedings of the First Workshop on Intelligent Solutions for Embedded Systems (WISES'03)*, pages 39–49, Vienna, Austria, June 2003.

[WSD03] World Wide Web Consortium (W3C). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, Nov. 2003. (W3C Working Draft 10 November 2003).

Chapter 5

Fusion of Continuous–Valued Sensor Measurements using Confidence–Weighted Averaging

Wilfried Elmenreich. International Journal of Vibration and Control. To appear.

This paper presents a method for fusing measurement samples from multiple sensors into a dependable robust estimation of a variable in the control environment. Each sensor measurement is represented by a measurement value and a confidence marker that corresponds to the respective variance of the measurement. We propose the Confidence-Weighted Averaging (CWA) algorithm for fusing measurements with respect to the estimated variance of the measurement error. For calibrated sensors with uncorrelated error functions this algorithm is optimal for producing a result with minimum mean squared error.

5.1 Introduction

Due to the availability of cheap sensing elements and larger, integrated systems, the number of sensor data sources in typical embedded applications will increase in the future. For example, the DECOS integrated architecture [Kop04] proposes a concept where sensor data from different distributed application subsystems in an automobile is made available to other subsystems via a virtual gateway concept. Furthermore, the advent of sensor networks in the cabled and wireless domain makes it possible to easily instrument a large number of sensors. This allows applications to take advantage of more sensor information about the environment, however requires means to systematically combine sensor information from sensors with different accuracy and reliability. The fused result should be more exact and more dependable than the single sensor measurements.

In this paper we focus on the problem of fusing a sample of several continuous-valued sensor measurements into a more robust and more accurate estimation of the measurand using a statistical approach. Since we do not assume to have a model of the observed process, we do not regard previous measurements (series), but only concurrent measurements from the same real-time entity. By taking advantage of the smart transducer concept [Elm03], we can expect each measurement to be pre-calibrated and assigned with a confidence marker that gives an estimation of the quality of the measurement. Our fusion algorithm combines the measurements with respect to their variance into a more accurate estimation of the measurand and gives an estimation of the result's confidence.

The rest of the paper is structured as follows: Section 5.2 describes the fusion problem to be solved. Section 5.3 gives an overview on related work. Section 5.4 elaborates on a representation of confidence in a digital format. The algorithm and its analysis is presented in Section 5.5. Section 5.6 presents experimental results from a multi-sensor case study. The paper is concluded in Section 5.7.

5.2 The Fusion Problem

Given is a set of sensors that measure the same real-time entity in the process environment. We assume the sensors to be calibrated[1], so that the measurement errors are only of stochastic nature. Furthermore, the correlation between the

[1]Note that there are cases, where a systematic error cannot be removed by calibration, e. g., when values beyond the sensor's measurement range are mapped to a default value.

sensor's error function needs to be insignificant. We will show in Section 5.6 that these assumptions hold for real sensor networks.

Analyses of real sensors have shown that it is difficult to make any assumption on the error distribution of a sensor. Figure 5.1 shows the distribution of measurement errors that has been measured for a particular distance sensor. Therefore, we assume that the probability distribution function of the sensors' measurement errors is uncharacterized.

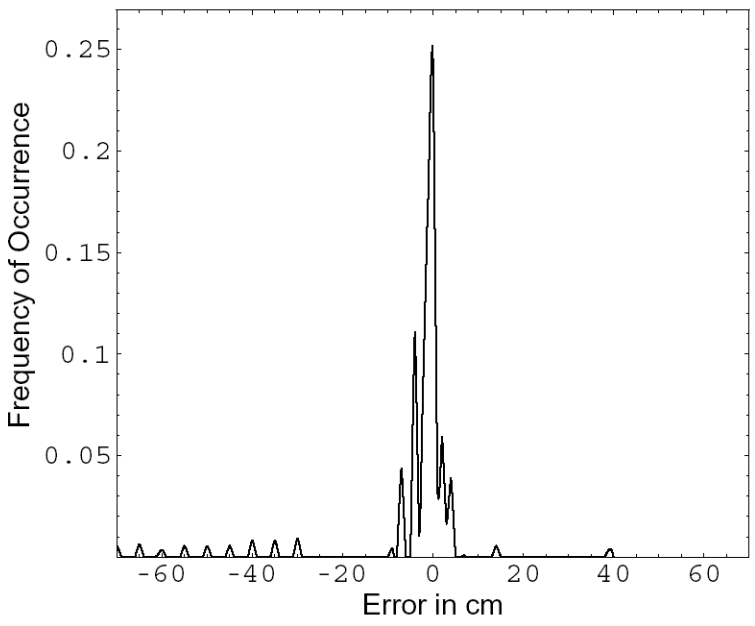


Figure 5.1: Error histogram for Sharp GP2D02 infrared sensor (from [Elm02a])

The sensors will produce a sample of *observations*, where an observation consists of the *measurement value*, the *measurement instant*, a *confidence marker* and the respective name of the measured *entity*.

Furthermore we assume that the observations are taken synchronuously within a time window that is sufficiently small so that the variable to be measured does not change significantly within that interval.

5.3 Related Work

In the literature, several methods can be found for classifier fusion or decision fusion based on sensor information. Some examples are voting mechanisms [Par92], based on reliability of each sensor or classifier ([Ald04]) or, as in [Rao04], also considering correlations between the sources. Other more complex methods include Hidden Markov Models or Neural Networks (e.g.[T.W04]), all with the intention to minimize the expected error of the fused result.

Focusing at fusion of continuous-valued sensor measurements, the fault-tolerant sensor averaging algorithm proposed by Marzullo in [Mar90], is closely related to our approach. Unlike the Kalman filter [Kal60], Marzullo's approach is *stateless*, thus does not require data from previous measurements in the fusion process. We will compare the results from our Confidence-Weighted Averaging (CWA) algorithm to the fault-tolerant sensor averaging algorithm in Section 5.6.

A scheme for confidence markers in digital systems is presented by Parhami in [Par91]. The proposed approach attaches so-called dependability tags to each data object and updates these tags according to operations performed on these data objects.

Another idea that contributed to the work in this paper is given by sensor validation for fieldbus nodes. So-called self-validating sensors are able to provide a standardized digital signal and generate diagnostic information. In the Oxford SEVA system [Hen95], each measurement is delivered as a validated value, together with the validated uncertainty and a measurement value status.

5.4 Representation of Confidence Markers

The confidence measure will be introduced as an integer value between 0 and $conf_{max}$, where 0 is defined to be the lowest confidence and $conf_{max}$ is the highest confidence.

We have chosen the statistical variance as a reciprocal measure for confidence. Also, the *ISO Guide to the Expression of Uncertainty in Measurement* [ISO93] suggests statistical variance as a measure for uncertainty.

In order to enable operations based on the confidence of observations from different sources, the confidence has to be standardized. We assume, that in the best case, variance will be close to 0, thus corresponding to the maximum confidence. In the worst case, a sensor will deliver a random value within its measurement range for the measurement. The worst-case variance can thus be calculated as the variance of a uniformly distributed random function between the limits a and b:

$$\mathbb{V}[S] = \frac{(b-a)^2}{12} \tag{5.1}$$

where a and b are the minimum and maximum values of the expected uniformly distributed random function. It is possible to find a probability distribution function that produces even greater variances, however we assume that all measurements with variances of $\mathbb{V}_{max}$ or greater are nearly useless and therefore are mapped into the same class of minimum confidence.

A standard message format as it is used in the TTP/A sensor bus protocol [Kop02] maps the measurement values to values between 0 and 200. If the sensor measurement is totally random, the variance can be calculated according to equation 5.1. This worst case $\mathbb{V}[S]$ equals $\frac{200^2}{12}$ or 3333.33. On the other hand, a variance of 0 corresponds to best possible case. Without restrictions to generality we have elaborated a mapping between a small number of confidence values and the respective variances.

Using a linear transformation between confidence values and variance would not be feasible, since the variances that indicate exact measurements are of greater interest than those measurements with large variances. Therefore, we decided to use a logarithmic scale to define the confidence values between min_{conf} and max_{conf} (see figure 5.2). Due to the expected computational load when doing logarithmic and exponential operations on embedded systems, we suggest the implementation of look-up tables for the conversion from confidence value to variance. Table 5.1 depicts a conversion table for 16 different levels of confidence.

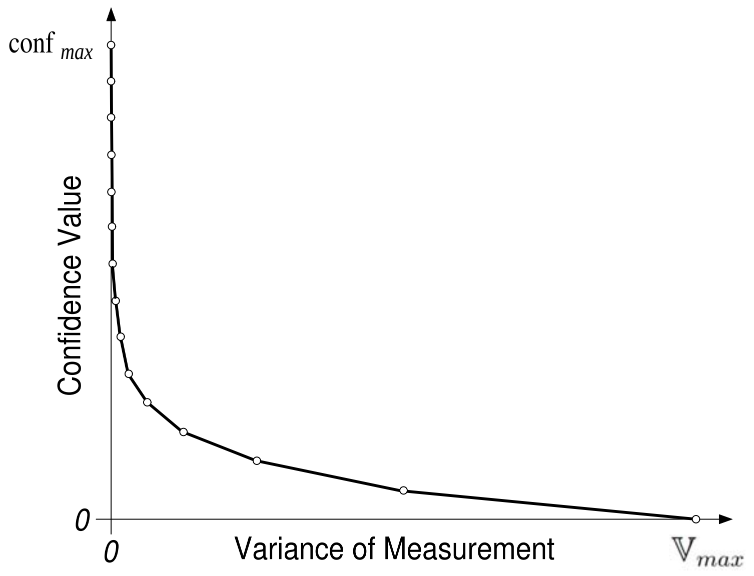


Figure 5.2: Conversion function for confidence/variance values (based on a logarithmic scale)

5.5 Confidence-Weighted Averaging

We suggest an algorithm for fusing data from replicated sensors based on weighted averages. The fused value x_{FUSED} is calculated as the weighted average of all measurement x_i, the weights w_i being derived from the reciprocal of the variance of each sensor S_i.

$$x_{FUSED} = \sum_{i=1}^{n} x_i w_i \tag{5.2}$$

with

$$w_i = \frac{1}{\mathbb{V}(S_i) \sum_{j=1}^{n} \frac{1}{\mathbb{V}(S_j)}} \tag{5.3}$$

where n is the number of observations, x_i represents a measurement taken by sensor S_i and $\mathbb{V}(S_i)$ is the estimated variance associated to that sensor. Under the assumption of independence of errors between sensors and supposing that

Confidence value	Interval for uniformly distributed error	Statistical Variance
0	[-100.0,100.0]	3333.33
1	[-70.2,70.2]	1644.65
2	[-49.3,49.3]	811.47
3	[-34.7,34.7]	400.37
4	[-24.3,24.3]	197.54
5	[-17.1,17.1]	97.47
6	[-12.0,12.0]	48.09
7	[-8.4,8.4]	23.73
8	[-5.9,5.9]	11.71
9	[-4.2,4.2]	5.78
10	[-2.9,2.9]	2.85
11	[-2.1,2.1]	1.41
12	[-1.4,1.4]	0.69
13	[-1.0,1.0]	0.34
14	[-0.7,0.7]	0.17
15	[-0.5,0.5]	0.08

Table 5.1: Conversion table for 16 different levels of confidence

the expected error $\mathbb{E}[x_i - x]$ is equal to 0 , this method minimizes the expected variance of the fused value.

Proof. X_{FUSED} is a weighted average of i independent random variables X_i. The weights w_i should be chosen so that they minimize the mean squared error of the fused variable X_{FUSED}. Furthermore, we require that the fused estimate is unbiased, that is that the average deviation from the true measurement X is equal to 0.

$$\mathbb{E}[X_{FUSED} - X] = \mathbb{E}[\sum_{i=1}^{n} w_i x_i - x] = 0 \tag{5.4}$$

The expected squared error of the fused result can be expressed as

$$\mathbb{E}[(X_{FUSED} - X)^2] = \sigma^2_{FUSED} = \sum_{i=1}^{n} w_i^2 \sigma_i^2. \tag{5.5}$$

Given that $\mathbb{E}[x_i - x] = 0$ and $\mathbb{E}[x] = x$ we deduce that $\sum_{i=1}^{n} w_i = 1$. Looking for the weights w_i that minimize the expression in 5.5, we substitute $w_1 = 1 - \sum_{j=2}^{n} w_j$ and calculate the partial derivative for each weight:

$$\frac{\partial \sigma^2_{FUSED}}{\partial w_i} = -2\sigma_1^2(1 - \sum_{j=2}^{n} w_j) + 2w_i\sigma_i^2 = 0 \tag{5.6}$$

Setting all partial derivatives equal we can derive the expression

$$\sigma_1^2\big(1 - \sum_{j=2}^{n} w_j\big) = w_2\sigma_2^2 = ... = w_n\sigma_n^2 \tag{5.7}$$

We see that all weights $w_i, i = 2...n$ are proportional to the reciprocal of the corresponding σ_i^2. We can therefore express them as

$$w_i = \xi/\sigma_i^2 \tag{5.8}$$

and receive the expression

$$\sigma_1^2\big(1 - \sum_{j=2}^{n} \frac{\xi}{\sigma_j^2}\big) = \frac{\xi}{\sigma_2^2}\sigma_2^2 = ... = \frac{\xi}{\sigma_n^2}\sigma_n^2 \tag{5.9}$$

We can now solve for ξ as follows:

$$\sigma_1^2\big(1 - \sum_{j=2}^{n} \frac{\xi}{\sigma_j^2}\big) = \xi \tag{5.10}$$

$$1 - \sum_{j=2}^{n} \frac{\xi}{\sigma_j^2} = \frac{\xi}{\sigma_1^2} \tag{5.11}$$

$$1 = \frac{\xi}{\sigma_1^2} + \sum_{j=2}^{n} \frac{\xi}{\sigma_j^2} \tag{5.12}$$

$$1 = \xi \sum_{j=1}^{n} \frac{1}{\sigma_j^2} \tag{5.13}$$

$$\xi = \frac{1}{\sum_{j=1}^{n} \frac{1}{\sigma_j^2}} \tag{5.14}$$

Substituting 5.14 into 5.8 we receive 5.15 as the optimal weight for each x_i

$$w_i = \frac{1}{\sigma_i^2 \sum_{j=1}^{n} \frac{1}{\sigma_j^2}} \tag{5.15}$$

To ensure that the solution is in fact a minimum, we derive the second partial derivative which is greater than 0, since σ_1^2 and σ_2^2 are in all cases greater than 0:

$$\frac{\partial^2 \sigma_{FUSED}^2}{\partial w_i^2} = 2\sigma_1^2 + 2\sigma_i^2 > 0 \tag{5.16}$$

The formula for calculating the fused value x_{FUSED} is therefore

$$x_{FUSED} = \frac{\sum_{i=1}^{n} \frac{x_i}{\mathbb{V}(S_i)}}{\sum_{i=1}^{n} \frac{1}{\mathbb{V}(S_i)}} \tag{5.17}$$

The variance of X_{FUSED} being always smaller than any of the input variances and is derived as follows:

$$\sigma_{FUSED}^2 = \sum_{i=1}^{n} w_i^2 \sigma_i^2 = \sum_{i=1}^{n} \frac{\sigma_i^2}{\sigma_i^4 \Big(\sum_{j=1}^{n} \frac{1}{\sigma_j^2}\Big)^2} = \frac{\sum_{i=1}^{n} \frac{1}{\sigma_i^2}}{\Big(\sum_{i=1}^{n} \frac{1}{\sigma_i^2}\Big)^2} = \frac{1}{\sum_{i=1}^{n} \frac{1}{\sigma_i^2}} \tag{5.18}$$

The fused variance of the fusion result, which can be interpreted as a virtual sensor S_{FUSED} is thus

$$\mathbb{V}(S_{FUSED}) = \frac{1}{\sum_{i=1}^{n} \frac{1}{\mathbb{V}(S_i)}}. \tag{5.19}$$

This method is optimal in the sense that it minimizes the expected variance of the fused result.

5.6 Experimental Results

To evaluate the practical applicability of CWA, we have applied the algorithm to data from three infrared sensors of type Sharp GP2D02 and two Polaroid 6500 series ultrasonic sensors.

The infrared sensors are designed for measuring distances within the range of 10-80cm. They show problematic behavior when there is no object within detection range which is as far as about 110 cm. In this case the returned

Fusion sources	Mean squared error (cm^2)	Mean absolute error (cm)	Estimated variance (cm^2)
US 1 + US 2	9.29	1.52	8.54
IR 1 + IR 2 + IR 3 (unfiltered)	129.00	7.29	119.52
US 1 + IR 1 (unfiltered)	7.41	1.66	6.96
US 1 + IR 1 (filtered)	6.98	1.63	6.56
IR 3 + IR 2 + IR 3 (filtered)	55.97	4.88	49.83
US 1+US 2+IR 1+IR 2+ +IR 3 (unfiltered)	6.65	1.37	6.14
US 1+US 2+IR 1+IR 2+ +IR 3 (filtered)	5.32	1.31	4.87

Table 5.2: Performance of the CWA algorithm for the examined sensor configurations

Fusion sources	t	Mean squared error (cm^2)	Mean absolute error (cm)	Estimated variance (cm^2)
US 1 + US 2	**0**	**10.02**	**1.99**	**9.57**
	1	10.99	2.08	10.65
IR 1+IR 2+IR 3 (unfiltered)	0	477.09	10.60	430.41
	1	**130.63**	**7.39**	**113.77**
	2	190.63	10.51	180.11
IR 1+IR 2+IR 3 (filtered)	0	2061.90	26.25	1492.08
	1	**82.95**	**6.72**	**76.87**
	2	100.67	7.33	90.49
US 1 + IR 1 (unfiltered)	0	1129.60	14.32	986.78
	1	**212.35**	**10.87**	**173.71**
US 1 + IR 1 (filtered)	0	1300.40	16.74	1092.96
	1	**212.08**	**11.18**	**181.74**
US 1 + US 2 + +IR 1+IR 2+IR 3 (unfiltered)	0	1646.96	18.84	1376.00
	1	260.00	3.96	257.69
	2	**48.25**	**4.57**	**45.55**
	3	117.55	7.21	101.87
	4	190.63	10.51	180.11
US 1 + US 2 + +IR 1+IR 2+IR 3 (filtered)	0	2387.56	28.88	1680.39
	1	139.74	3.45	138.99
	2	**12.21**	**2.49**	**11.86**
	3	70.08	6.44	63.15
	4	100.67	7.33	90.49

Table 5.3: Performance of Marzullo's fault-tolerant sensor averaging algorithm for the examined sensor configurations. t represents the number of faulty sensors to be tolerated

data is unreliable and may correspond to arbitrary measurements within the range. To detect such erroneous measurements a filtering algorithm was applied locally at each sensor. The filter considers four subsequent measurements of a sensor and determines that there is no object within range if the jitter of these measurements is larger than a particular threshold.

Table 5.2 shows the results of the fusion with the CWA algorithm. The first column indicates which sensor sources have been used for the fusion and if the above described filtering has been applied to the IR sensors. The next three columns contain the error and variance of the fused result.

For comparison we have fused the same data set with the the fault-tolerant sensor averaging algorithm proposed by Marzullo [Mar90].

In Marzullo's algorithm each sensor measurement is modeled by an interval that is expected to contain the real sensor measurement. If a sensor delivers a measurement with the real value outside this interval, the sensor is considered faulty. It is required to parametrize the expected number of faulty sensors at once as t.

Since the behavior of the measurement errors of the employed sensors makes it difficult to select the best t a priori for a given configuration we have performed multiple runs of the fault-tolerant sensor averaging algorithm for each possible t. Table 5.3 lists the results obtained from the different runs using various sensor configurations.

In comparison to the results from the CWA algorithm, the performance is similar for homogeneous sensor configurations while CWA performs much better for heterogeneous sensor configurations. Especially for the homogeneous sensor configurations the sensor errors turned out to be stronger correlated which affected the quality of the fused result. Nevertheless, the CWA algorithm delivers a serviceable result of reduced quality.

5.7 Conclusion and Outlook

We have proposed an algorithm for fusing measurement samples from multiple sensors into a dependable robust estimation of a variable in the control environment. This Confidence-Weighted Averaging (CWA) algorithm takes values annotated with confidence markers as inputs and output. The confidence marker corresponds to the respective variance of the value. We have shown that this algorithm is optimal for producing the minimum possible variance of the average result for calibrated sensors with uncorrelated error functions.

However, CWA is based on the independence of measurement errors, an assumption that cannot generally be made in sensor fusion applications. In the future work we will extend our algorithm by taking correlated error functions into account. Thus, we expect a more accurate estimation of the results quality for fusing measurements taken by the same type of sensors.

5.8 Acknowledgments

This work was supported by the Austrian FWF project TTCAR under contract No. P18060-N04.

References

[Ald04] S.A. Aldosari and J.M.F. Moura. Fusion in sensor networks with communication constraints. In *IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks*, pages 108–115, New York, NY, USA, 2004. ACM Press.

[Elm02] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD Thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[Elm03] W. Elmenreich and S. Pitzek. Smart Transducers – Principles, Communications, and Configuration. In *Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems*, volume 2, pages 510–515, Assuit – Luxor, Egypt, Mar. 2003.

[Hen95] M. Henry. Sensor Validation and Fieldbus. *Computing & Control Engineering Journal*, 6(6):263–269, Dec. 1995.

[ISO93] International Organization for Standardization (ISO), Genève, Switzerland. *Guide to the Expression of Uncertainty in Measurement*, 1st edition, 1993.

[Kal60] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transaction of the ASME, Series D, Journal of Basic Engineering*, 82:35–45, Mar. 1960.

[Kop02] H. Kopetz et al. Specification of the TTP/A Protocol. Research Report 61/2002, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, Sep. 2002. Version 2.00.

[Kop04] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri. From a Federated to an Integrated Architecture for Dependable Embedded Systems. Research Report 22/2004, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2004.

[Mar90] K. Marzullo. Tolerating Failures of Continuous-Valued Sensors. *ACM Transactions on Computer Systems*, 8(4):284–304, Nov. 1990.

[Par91] B. Parhami. A Data-Driven Dependability Assurance Scheme with Applications to Data and Design Diversity. In A. Avizienis and J. C. Laprie, Editors, *Dependable Computing for Critical Applications*, volume 4, pages 257–282. Springer Verlag, Vienna, 1991.

[Par92] B. Parhami. Optimal Algorithms for Exact, Inexact, and Approval Voting. *Twenty-Second International Symposium on Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers.*, pages 404–411, July 1992.

[Rao04] N.S.V. Rao. A Generic Sensor Fusion Problem: Classification and Function Estimation. In T.Windeatt F.Roli, J.Kitler, Editor, *Multiple Classifier Systems: 5th International Workshop, MCS 2004. Proceedings*, volume 3077 of *Lecture Notes in Computer Science*, pages 16–30. Springer-Verlag GmbH, Jan. 2004.

[T.W04] T.W.Lewis and D.M.W. Powers. Sensor fusion weighting measures in Audio-Visual Speech Recognition. In *CRPIT '04: Proceedings of the 27th conference on Australasian computer science*, pages 305–314, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

Chapter 6

Fault-Tolerant Certainty Grid

Wilfried Elmenreich. 11th International Conference on Advanced Robotics, Coimbra, Portugal, June–July 2003, volume III, pages 1576–1581.

World modeling for mobile autonomous robot is usually a process that uses sensor data as input and provides a model of the robot's environment as output. In this paper we investigate on sensor fusion methods for robustness and fault tolerance. We evaluate three methods according to their performance, memory consumption, and required sensor configurations.

The algorithms have been implemented in a four-wheeled autonomous mobile robot that uses a set of three infrared sensors to build the world model. We present a performance analysis of the new algorithms based on simulation and experimental data.

6.1 Introduction

Autonomous mobile robots belong to a class of applications whose operability depends on sensor data. Sensors are physical devices converting a physical property into a measurement that can be interpreted by a computer system. Sensors are affected by several sources of error, like sensor deprivation, limited spatial or temporal coverage, imprecision, cross-sensitivity, and uncertainty. Thus, dependable applications may not rely on a single sensor. In order to overcome these problems, the inputs from several sensors are combined to form a dependable representation of the environment, the *world model*.

For mobile robots, usually the world model is represented as a two-dimensional or, especially for non-flat outdoor environments, a three-dimensional map of the robot's surroundings. Typical approaches to generate such a map are the grid-like division of the environment, like the occupancy grid approach of Elfes [Elf89]. Based on this grid, the navigation and path planning system decides on the robot's actions. Erroneous or ambiguous sensor readings can be detected and resolved by processing redundant sensor information. Redundant information can be evaluated at different levels of abstraction. In general, if the evaluation is performed at sensor level, the system complexity can be kept low at the cost of hardware expenses. If evaluation of redundant information is integrated into the application, the performance is better, since at this level more knowledge about the reasonableness of a particular result is available. However, such sanity checks increase the system complexity since normal processing functions become intertwined with error-detection and fault-tolerance functions [Kop97].

Therefore, we propose the integration of fault-tolerant functions with the sensor fusion part. At fusion level fault-tolerant functions can be efficiently applied with moderate complexity. It is the objective of this paper to investigate on sensor fusion methods for robustness and fault tolerance for a grid-like representation of a mobile robot's world model.

The remainder of the paper is organized as follows: Section 6.2 describes the system architecture of a sensor grid application and examines the possibilities and benefits of applying sensor fusion at particular levels in this model. Section 6.3 presents certainty grid algorithms that can handle faulty measurements. Section 6.4 presents the results from the evaluation, while Section 6.5 discusses the results. Section 6.6 concludes the paper.

6.2 Architectural Considerations

An autonomous robotic system contains at least a set of sensors and actuators and a control application. Sensors and actuators are the interface to the process environment and belong to the *transducer level* of the robotic system, while the control application belongs to the *control level*.

In general, a system is composed out of components, whereas components are subsystems like sensors, actuators, processing nodes, and communication channels. As a matter of fact, every single component of a system will eventually fail [Bau01]. Requirements for highly dependable systems can only be met, if these failures are taken into account.

Some systems, fail in a manner that they still provide a service, however at a degraded level. For example, a sensor may be affected by cross-sensitivity and fail to render a measurement with the specified accuracy – however, the measurement contains still information that can be exploited.

In the following sections we focus on sensor fusion methods in order to exploit the sensors' information under the presence of faults. Fault-tolerant mechanisms handling failures of microcontrollers and communication lines are beyond the scope of this paper.

A dependable system needs a redundant sensor configuration for the purpose of competitive sensor fusion. Competitive sensor fusion can be used to achieve *robustness* or *fault tolerance*. Robustness means that effects of single sensor faults are attenuated in the result. Fault tolerance means that a defined set of faults does not affect the result at all, thus faults are masked out. The defined set of faults is called the *fault hypothesis*. Fault-tolerant systems do not make any guaranty about their behavior when faults occur that are not defined in the fault hypothesis.

For the implementation of the necessary tasks to handle faults, the designer of a system has two main options:

Transducer level: Each transducer can be equipped with a set of redundant sensors and a voting mechanism. Voting can be seen as the simplest method of sensor fusion. The advantage of this approach is that the other parts of the application are not aware of the extra sensors and faults of single sensors are masked out. The disadvantage is the great amount of extra hardware, which results in increased cost, weight, and power consumption, and possible problems of mutual sensor inference and, since the sensors are geometrically not exactly at the same place, parallax errors.

Control level: The application-specific approach uses reasonableness checks that use application knowledge to judge whether a value is correct or not. Since the dependability operations are integrated with (parts of) the application, this approach leads to increased design effort and application complexity.

However, this approach can be more efficient, since dependable behavior could be achieved with less hardware expenses [Pol95].

In order to achieve a high efficiency and keep the system complexity low, we propose a compromise that performs the sensor fusion at an intermediate level between transducer and control level, the fusion/dissemination level. Thus, we propose a system model that decomposes the real-time computer system of a mobile robot into three levels [Elm01a]: First, a transducer level, containing the sensors and the actuators, second, a fusion/dissemination level that gathers measurements, performs sensor fusion respectively distributes control information to the actuators, and third, a control level where a control program makes control decisions based on environmental information provided by the fusion level. The interface between fusion/dissemination level and control level can be made transparent to the transducer configuration, such that the control application can be implemented independently of the employed sensors and actuators [Elm01b].

6.3 Robust World Modeling

We assume a set of sensors (e. g., ultrasonic, infrared, laser) measuring the distance to the nearest obstacle. The sensors are mounted on the robot and detect obstacles at particular angles. The goal is to get a map of the robot's environment containing obstacles and free space.

The sensors are swept around by a motor for each sensor in order to cover a segment of the robot's surroundings over time. The segments overlap partially or fully, hence providing some redundancy in the coverage of the environment. However, although our architecture is capable of synchronizing all sensors and motors, it is not feasible to turn any two sensors into the same or at least approximately the same direction at the same time because of interference problems. Thus, it is not possible to directly compare sensor readings made at the same time.

Furthermore, from the view of hardware architecture it is almost impossible to mount sensors in a way that the viewpoint angle from a sensor to an object is perfectly identical to the angle of the replicated sensor. A replicated sensor will thus always be located slightly offside, thus viewing objects from different angles. Even if two sensors are working correctly, they may produce different results. Figure 6.1 depicts an example for an object that yields ambiguous sensor readings. Although both sensors are working according to their specification, sensor B detects an object for the given region while sensor A does not.

Besides these problems, we assume that a sensor may degrade the quality of its service up to the case where it permanently delivers faulty measurements. For example, one distance sensor could refuse to detect any object and always reports "no object nearby".

Note, that the existing certainty grid method based on Bayesian fusion can handle uncertainty, but not faulty measurements. Although the effects of occasional sensor faults on the grid can usually be neglected [Mar96], permanent faults – as assumed in our fault hypothesis – result in a significant deviation of the representation in the grid from the actual environment.

For the purpose of handling sensor failures where a sensor permanently submits measurements with incorrect values, we have derived a robust certainty grid algorithm (also presented in [Elm02b]) and two fault-tolerant algorithms for grid generation.

The problem is solved by analyzing the redundant parts of the certainty grid. The certainty grid is a two-dimensional array of grid cells that corresponds to the robot's environment. Each grid cell contains a probabilistic value *occ* ranging from 0 to 1 corresponding to the believe that this cell is occupied by an object:

$$cell.occ = \begin{cases} 0 & \text{free} \\ \vdots & \\ 0.5 & \text{uncertain} \\ \vdots & \\ 1 & \text{occupied} \end{cases}$$

Figure 6.2 gives an example for a certainty grid. Figure 6.2(a) depicts a mobile robot and an obstacle, while Figure 6.2(b) depicts the certainty grid. The values in the grid cells are the occupancy values. For regions that are not seen by a sensor, the resulting occupancy value is 0.5, that is unknown territory.

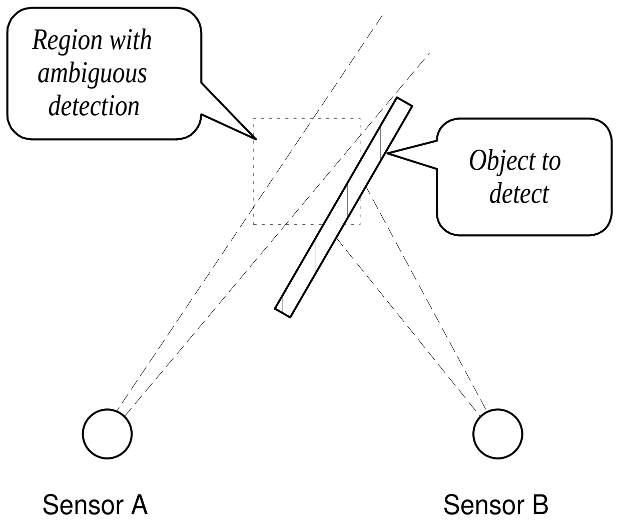
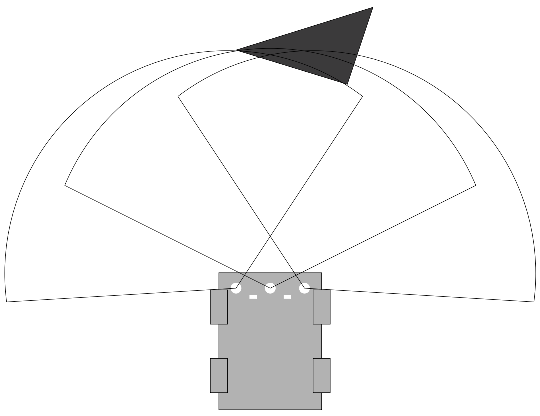
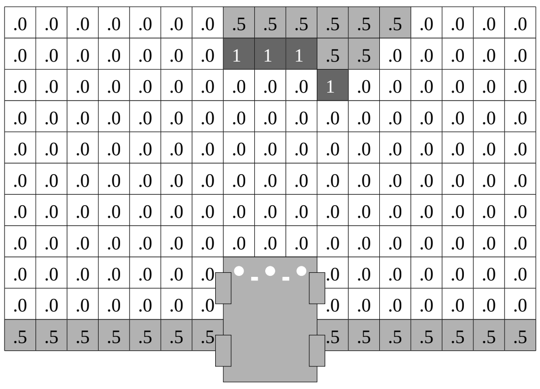


Figure 6.1: Discrepancy between sensor A and sensor B due to object shape


(a) Obstacle in front of robot



.0	.0	.0	.0	.0	.0	.0	.5	.5	.5	.5	.5	.5	.0	.0	.0	.0
.0	.0	.0	.0	.0	.0	.0	1	1	1	.5	.5	.0	.0	.0	.0	.0
.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	1	.0	.0	.0	.0	.0	.0
.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0
.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0
.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0
.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0
.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0
.0	.0	.0	.0	.0	.0	.0				.0	.0	.0	.0	.0	.0	.0
.0	.0	.0	.0	.0	.0	.0				.0	.0	.0	.0	.0	.0	.0
.5	.5	.5	.5	.5	.5	.5				.5	.5	.5	.5	.5	.5	.5

(b) Representation in certainty grid

Figure 6.2: Representation of the robot environment in a certainty grid

6.3.1 Fault-Tolerant Certainty Grid

The fault-tolerant certainty grid uses a separate entry of an occupancy value for each sensor. Thus, each sensor produces its own grid independently of the other sensors. The concise world model is built by fusing all sensor grids. The proposed fault-tolerance algorithm is similar to the Fault-Tolerant Average algorithm for clock synchronization [Lun84].

The fault-tolerant fusion is performed as follows: First the measurements from all sensors for each grid cell are gathered. If the sensors do not update the grid cells simultaneously, the gathering of the measurement will take some time. It is assumed that the environment does not change significantly during this phase.

Then the set of proposed certainty values for each grid cell are sorted and the t lowest and t largest values are removed from the set. t represents the maximum expected number of faulty sensors per measurement. If the set contains at least t faulty measurements, these will either be removed from the set, or the value of the faulty measurements lie between two correct measurements, which will not worsen the result.

The remaining measurements are then fused by Bayesian fusion. Assuming *conditional independence* and *maximum entropy* [Mos02], the fusion formula for n values can be expressed as follows:

$$\frac{1}{\mathbb{P}(cell.occ|S_1,\ldots,S_n)} - 1 = \prod_{i=1}^{n}\left(\frac{1}{\mathbb{P}(cell.occ|S_i)} - 1\right)$$

There must be at least $2t+1$ sensors contributing to each grid cell in order to provide enough data to keep at least one measurement after removing the t extreme values.

The algorithm tolerates at least t faulty measurements at a time, however drops also many correct measurements. This results in an information loss in the fused result, which may degrade the result in the average case.

6.3.2 Fault-Tolerant Median Selection

If the number of expected faults should be maximized, we propose the implementation of fault-tolerance by selecting the median value from the set of measurements for each grid cell. Thus, if a cell is updated by n sensors, at least $\lfloor \frac{n-1}{2} \rfloor$ faulty measurements are tolerated. The median method drops even more correct measurements than the fault-tolerant average algorithm, however has the big advantage of being adaptive to the number of measurements. Thus if the number of contributing grid cells is not constant for all grid cells, the median method is easily applied to this situation, while the fault-tolerant average algorithm is not.

6.3.3 Robust Certainty Grid

The robust certainty grid algorithm fuses the sensor inputs immediately into a single grid. Together with the grid occupancy value *occ* each cell stores the main contributor (i. e., the sensor that updated this cell most recently) of the *occ* value with the cell. This property of each cell is named the current *owner* of the cell:

$$cell.owner = \begin{cases} 0 & \text{unknown} \\ 1 & \text{sensor 1} \\ \vdots & \\ \mathrm{n}_{\text{sensors}} & \text{sensor n} \end{cases}$$

All grid cells are initialized with $cell.occ = 0.5$ and $cell.owner = unknown$. When a new measurement has to be added to the grid, the following *AddToGrid* algorithm is executed (Figure 6.3 lists the algorithm in pseudocode):

- If the particular grid cell has no contributor listed in its owner field or the cell owner is identical with the contributing sensor, the measurement of the sensor is taken as is and the cell stores the index of the sensor as new owner.
- If there is a different contributor, the measurement is first compared to the cell value *cell.occ* by calculating a value named *comparison*. If *comparison* is above a particular *confirmation threshold*, we speak of a *confirmation* of cell value and new measurement. If *comparison* is below a particular *contradiction threshold*, we speak of a *contradiction* of cell value and new measurement. In case of a confirmation, the confidence values of the new sensor and the owner are both increased. In case of a contradiction, the confidence values of the new sensor and the owner are

```
procedure AddToGrid( sensor, cell )
begin
  if (cell.owner = unknown) or (cell.owner = sensor) then
    cell.occ := sensor.measurement;
    cell.owner := sensor;
  else
    comparison := 4*(cell.occ-0.5)*(sensor.measurement-0.5);
    weight1 := abs(cell.occ-0.5)*cell.owner.conf;
    weight2 := abs(sensor.measurement-0.5)*sensor.conf;
    if weight1 = weight2 then
      cell.occ := (cell.occ+sensor.measurement) / 2;
    else
      cell.occ := (cell.occ*weight1+sensor.measurement*weight2)
                        / (weight1 + weight2);
    if comparison > CONFIRMATIONTHRESHOLD then
      inc(cell.owner.conf);
      inc(sensor.conf);
    if comparison < CONTRADICTIONTHRESHOLD then
      dec(cell.owner.conf);
      dec(sensor.conf);
    contribution := 4*(cell.occ-0.5)*(sensor.measurement-0.5);
    if contribution > CONTRIBUTIONTHRESHOLD then
      cell.owner := sensor;
    else
      cell.owner := unknown;
end
```

Figure 6.3: Pseudocode of the AddToGrid algorithm

both decreased. If *comparison* is not significant, it does neither yield a confirmation nor a contradiction.

- The new occupancy value of the cell is calculated as a weighted average between old value and measurement. The weights are derived from the respective confidence values and the significance of the measurement. A measurement is more significant if it has a greater absolute distance to the *uncertain* state (0.5).

- Thereafter, a new owner is selected. Therefore, a value *contribution* is derived. This value is calculated the same way as the comparison value, but it uses the new *cell.occ* value.

- The *contribution* is a measurement of the consistency of the sensor measurement with the new *cell.occ* value. If the *contribution* is above a certain threshold, the contributing sensor becomes the new owner of the cell. Otherwise the *cell.owner* value is reset to *unknown*.

6.4 Evaluation

We used two different test environments to evaluate the proposed algorithms, a real mobile robot and a simulation environment.

We implemented the robust certainty grid in a mobile robot for demonstration purposes. The mobile robot comprises a model car ("smart car") equipped with a suit of pivoted distance sensors, two ultrasonic sensors pointing straight forward, an electric drive, and a steering unit (see Figure 6.4).

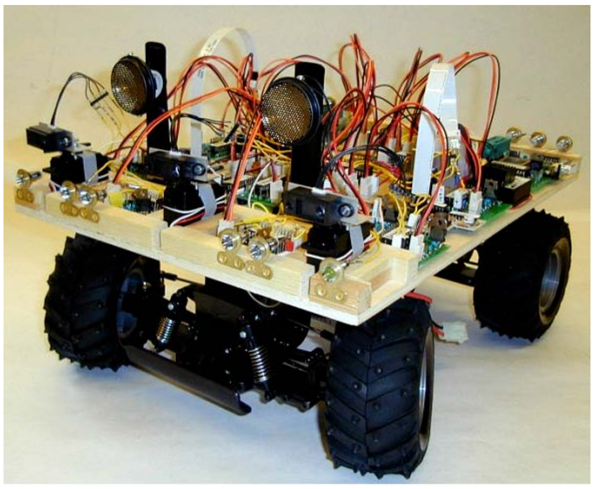


Figure 6.4: Smart Car: Autonomous mobile robot with pivoting sensors

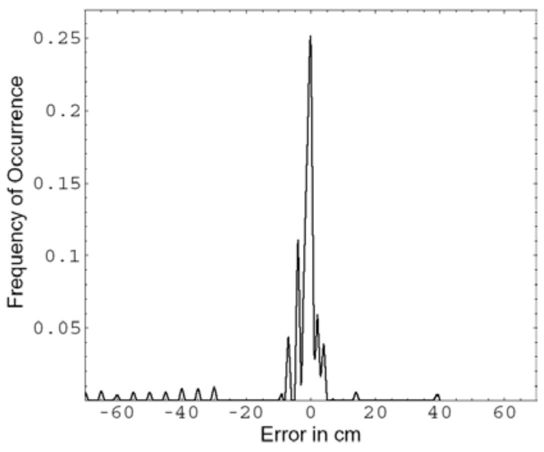


Figure 6.5: Error histogram for Sharp GP2D02 infrared sensor (from [Elm02a]

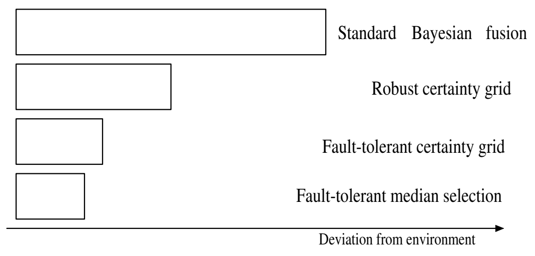


Figure 6.6: Comparison of average deviation of generated grid from environment

The certainty grid is built from the input of the Sharp GP2D02 infrared sensors. These type of sensors provide a rather narrow sensor beam, however it delivers measurements with a significant amount of error. The quality of the infrared sensor data has been analyzed and presented in [Elm02a]. Figure 6.5 depicts the measured distribution of sensor errors for a sensor sample.

The proposed algorithms have been evaluated versus Bayesian fusion in a simulation program that emulates an arbitrary number of sensors that are swept around in order to map a given artificial environment. For each sensor we simulated the measured behavior of the sensor error of a real GP2D02 sensor. We have assumed that there is no correlation between any two measurements of different sensors.

In both test environments, we used 8-bit integer values to express the probability values between 0 and 1. Thus, a value of 0 corresponds to the $free$ state, 128 means the *uncertain* state while 255 is used to express the *occupied* state of a grid cell. The certainty grid had a size of 17 times 11 cells whereas each cell corresponds to a 10 cm times 10 cm square.

Figure 6.6 shows a comparison of the average results from the simulation. The grid has been generated several times using the proposed algorithms. The fault-tolerant median selection achieved the lowest deviation, i. e., best performance. All three proposed algorithms achieved a better performance than the standard Bayesian fusion.

The results obtained from the experiment with the smart car are depicted in Figure 6.7. Figure 6.7(a) shows a photograph of the test environment, while Figures 6.7(b–d) depict the certainty grids generated from the sensor data using different algorithms. Since the scanning ranges of the smart car's sensors do not exactly overlap, the fault-tolerant certainty grid algorithm algorithm was not tested with the smart car.

6.5 Discussion

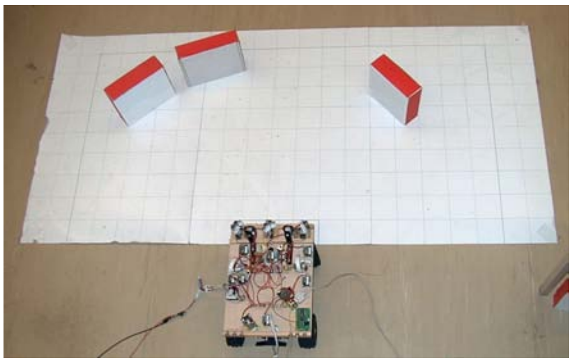
All approaches need at least three sensors in order to compensate a single faulty measurement. In contrast to the fault-tolerant certainty grid algorithm, the robust certainty grid algorithm and the median selection gain extra sensor space, because the sensor views must overlap only partially.

The fault-tolerant algorithms need to separately store the grid for each sensor (the resulting grid may produced on demand), thus require $n_{sensors} \cdot gridheight \cdot gridwidth$ memory elements where each element stores one certainty value. The robust certainty grid needs $gridheight \cdot gridwidth$ memory elements for the certainty values and $\frac{\lceil log_2(n_{sensors}+1)\rceil}{8} \cdot gridheight \cdot gridwidth$ extra bytes of memory for the storage for the owner values. The memory requirements for the confidence values can usually be neglected, since the number of sensors normally is remarkably lower than the total number of cells in the grid. Thus, the memory requirements of the robust certainty grid algorithm are considerable lower than the memory consumption of the fault-tolerant approach.

In contrast to Bayesian fusion and the fault-tolerant algorithms, the *AddToGrid* procedure of the robust certainty grid is sensitive to the ordering of measurements. Thus, when a grid cell is updated by subsequent measurements, the order of updates makes a difference in the result.

6.6 Conclusion

We have presented and evaluated two fault-tolerant and one robust method to handle faulty measurements when building a world model in form of a certainty grid map.



(a) Experiment setup

0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0
0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0
0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5
0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

(b) Bayesian fusion

0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0
0.0	0.0	0.5	0.0	0.4	0.5	0.6	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.5	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0
0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5
0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

(c) Robust certainty grid method

0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0
0.0	0.0	0.5	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0
0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5
0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

(d) Fault-tolerant median method

Figure 6.7: Comparison of algorithms on real data

In the presence of faulty measurements, all methods show a better behavior than the Bayesian fusion. From the examined performance and the resource requirements, the robust certainty grid algorithm and the fault-tolerant median selection are most promising. While the fault-tolerant median selection shows the best performance, the robust certainty grid algorithm needs a significantly less amount of memory to store the certainty grid. However, if the certainty grid is only generated for a limited space while a second data structure is used as a global map, the resource requirements for the certainty grid will not be as stringent.

6.7 Acknowledgments

This work was supported by the Networks of Excellence European IST project ARTIST under contract No. IST-2001-34820.

References

[Bau01] G. Bauer. *Transparent Fault Tolerance in a Time-Triggered Architecture.* PhD Thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2001.

[Elf89] A. Elfes. Using Occupancy Grids for Mobile Robot Perception and Navigation. *IEEE Computer*, 22(6):46–57, 1989.

[Elm01a] W. Elmenreich and S. Pitzek. The Time-Triggered Sensor Fusion Model. In *Proceedings of the 5th IEEE International Conference on Intelligent Engineering Systems*, pages 297–300, Helsinki–Stockholm–Helsinki, Finland, Sep. 2001.

[Elm01b] W. Elmenreich and S. Pitzek. Using Sensor Fusion in a Time-Triggered Network. In *Proceedings of the 27th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 369–374, Denver, CO, USA, Nov.–Dec. 2001.

[Elm02a] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems.* PhD Thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[Elm02b] W. Elmenreich, L. Schneider, and R. Kirner. A Robust Certainty Grid Algorithm for Robotic Vision. In *Proceedings of the 6th IEEE International Conference on Intelligent Engineering Systems (INES)*, pages 25–30, Opatija, Croatia, May 2002.

[Kop97] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[Lun84] J. Lundelius and N. Lynch. A New Fault-Tolerant Algorithm for Clock Synchronization. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 75–88, Vancouver, Canada, Aug. 1984.

[Mar96] M. C. Martin and H. P. Moravec. Robot Evidence Grids. Technical Report CMU-RI-TR-96-06, The Robotics Institute, Carneghie Mellon University, Pittsburgh, PA, USA, 1996.

[Mos02] B. Moshiri, M. R. Asharif, and R. Hosein Nezhad. Pseudo Information Measure: A New Concept for extension of Bayesian Fusion in Robotic Map Building. *Information Fusion*, 3(1):51–68, 2002.

[Pol95] S. Poledna. Fault Tolerance in Safety Critical Automotive Applications: Cost of Agreement as a Limiting Factor. In *Proceedings of the 25th IEEE International Symposium on Fault-Tolerant Computing*, pages 73–82, Pasadena, California, USA, June 1995.