An Architecture supporting Monitoring and Configuration in Real-Time Smart Transducer Networks

Philipp Peti
Vienna University of Technology, Vienna, Austria
pp@vmars.tuwien.ac.at

Roman Obermaisser
Vienna University of Technology, Vienna, Austria
ro@vmars.tuwien.ac.at

Wilfried Elmenreich
Vienna University of Technology, Vienna, Austria
we@vmars.tuwien.ac.at

Thomas Losert
Vienna University of Technology, Vienna, Austria
tl@vmars.tuwien.ac.at

Abstract

A smart transducer network consists of a set of transducer nodes interconnected with a digital bus. Smart transducer technology implicates the development of systems supporting the timely exchange of real-time data. Additional requirements are support for system integration, mechanisms for dynamic reconfiguration, and diagnostic interfaces. Such systems should be composable and ease controlling system complexity, i. e. support the system engineer in understanding the system behavior.
Furthermore, developers expect diagnostic services, which are deterministic, reproducible, and do not interfere with real-time services. This paper describes three interfaces for smart transducer networks, which provide the required services while yielding the mentioned properties. We describe a case study demonstrating the effectiveness of the three interfaces for the proclaimed purpose.

INTRODUCTION

In 1982 Wen H. Ko and Clifford D. Fung introduced the term "intelligent transducer" [1]. An intelligent or smart transducer is the integration of an analog or digital sensor or actuator element and a local microcontroller that contains the interface circuitry, a processor, memory, and a network controller in a single unit. The smart sensor transforms the raw sensor signal to a standardized digital representation, checks and calibrates the signal, and transmits this digital signal via a standardized communication protocol to its users [2].
Smart transducer technology implicates the development of transducer networks, that allow monitoring, plug-and-play configuration and the communication of digitized transducer data. Such a smart transducer network provides a framework that helps to reduce the complexity and cost of large distributed real-time systems.
This paper identifies essential services of a smart transducer network. We elaborate on mandatory requirements for interfaces designed for the provision of these services. The paper argues that the three interfaces, which are proposed in [3], are well-suited for establishing these services while satisfying the presented interface requirements. We present a case study demonstrating the effectiveness of the three interfaces by examining the provision of the identified services and the fulfillment of the interface requirements.

Structure

The paper is organized as follows. Section 2 elaborates on smart transducer interface issues, namely desired services and interface properties of a smart transducer network. It also describes three interfaces developed for this purpose. Section 3 defines the concept of the interface file system (IFS), which is a temporally specified common name space for the three interfaces. Section 4 presents the case study which argues why the three interfaces available through the IFS achieve the desired interface properties. It shows two smart transducer networks employing the fieldbus protocol TTP/A. The paper finishes with a conclusion in Section 5.

SMART TRANSDUCER INTERFACE ISSUES

An interface is a common boundary between two subsystems. A correctly designed interface should provide an understandable abstraction to the interfacing partners. Such an interface provides essential properties and hides irrelevant details. Thereby, it eases the understanding of the component interactions and helps to control the system complexity. It offers only information that is required by the user of the specific services available via the interface. Nevertheless, a properly designed interface has to provide completeness by making all information accessible, which is required for using a component's services. The distinction between relevant and irrelevant properties mainly depends on the purposes of the interactions at the interface. Therefore, different interfaces should be provided for different purposes.

Essential Services

This section describes services for real-time smart transducer networks. In addition to the support for the timely exchange of real-time data, diagnostic services offer insight into the system for a maintenance engineer. The system integration out of autonomously developed components requires configuration services.

Timely Exchange of Real-Time Data

A real-time smart transducer network must guarantee the transmission of real-time data with a predictable timing and low jitter. This information is normally used for control purposes (e.g., periodic execution of a control loop), where the quality of control is degraded by jitter [4].

Diagnostic Service Required

Diagnostic activities are commonly referred to as monitoring. One function of monitoring is the collection of runtime information of a real-time system for testing and debugging. Monitoring is often an effective procedure for locating incorrect system behaviors and serves the purpose of a debugging tool [5]. In a safety critical real-time system it can provide additional confidence for the validity of a static analysis. In non safety-critical real-time systems rare system failures may be accepted for economic reasons. Therefore, developers prefer dynamic debugging and testing instead of a static analysis due to lower cost and complexity.

Support for System Integration Necessary

In many engineering disciplines, large systems are built by the constructive integration of well-specified and pre-tested subsys-

tems, called components. During the system integration phase the system is formed out of the independently developed components. Configuration is the customization of a component to fit into the system (e. g. parameterization of a sensor node, assignment of communication parameters, etc.).

Configuration is aided by electronic datasheets that contain all relevant information of a transducer [6]. This information is assigned statically to each smart transducer. The smart transducer may either contain the electronic datasheet on-chip in persistent memory or the electronic datasheet may be available through a configuration server on the Internet. In this case the smart transducer contains merely a reference to an external electronic datasheet repository. Hence human errors associated with entering sensor parameters manually are completely eliminated. Losing transducer paper data sheets is no longer a concern. Enhanced capabilities like plug and play are possible [7].

Constraint Checking

Monitoring also aids in the automatic detection and handling of anomalous system states. If erroneous conditions described by monitoring constraints occur, a monitoring application can initiate corrective actions. In [8] a monitoring technique has been developed which supports detection and handling of timing errors. Another approach is called forced validity [9]. Upon the detection of constraint violation, the faulty data value is reset to a valid value.

Dynamic Reconfiguration

Distributed real-time systems require the possibility for on-the-fly configuration and maintenance without a system shut down [10]. This allows the accommodation of the system to evolutionary changes, which is especially important for networks with a long expected lifetime.

Desired Interface Properties

This section elaborates on desired properties of interfaces for smart transducer networks.

Temporal Composability

In a composable architecture, the system integration should proceed without unintended side effects. For an architecture to be composable, it must adhere to four principles of composability [11], namely independent development of nodes, stability of prior services, performability of the communication system, and replica determinism.

Control Complexity

This requirement is derived from the characteristics of human cognitive information processing [12]. A smart transducer interface must limit the amount of information that must be dealt with.

Non-Interference with Real-Time Service

The real-time communication services must not be effected by other services. In particular the temporal correctness of the real-time services must be independent of monitoring and configuration activities.

Diagnostic Interface Requirements

Monitoring and debugging of distributed embedded real-time systems differ significantly from debugging and testing programs for desktop computers, because only few interfaces to the outside world are present [13]. In addition, a distributed system contains multiple locations of control and therefore conventional break-point techniques result in an undesired modification of the timing behavior. This indeterministic effect is called the "Probe Effect" [14, 5] or the "Heisenberg Uncertainty" [15] applied to software. Therefore a vital property of a convenient monitoring environment is the absence of disturbances or intrusions on the system behavior. Users expect tools to avoid the probe effect and to incorporate a deterministic and reproducible behavior.

Much research has focused on monitoring, debugging, configuration, and maintenance of distributed real-time systems. However, existing tool sets generally neglect a subset of the major monitoring qualities like reproducibility, determinism and the elimination of the probe effect.

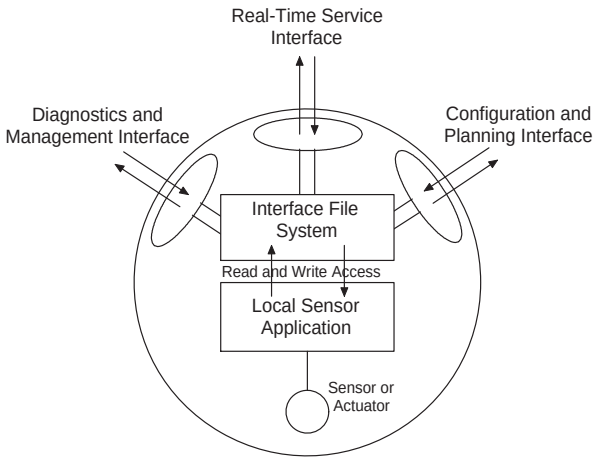


Figure 1. Three Interfaces of a Smart Transducer Node

A useful monitoring interface must satisfy the following qualities [16, 17]:

Elimination of the Probe Effect: The act of observing a distributed real-time system can change its behavior. Existing errors can be hidden and new errors can be introduced. Due to multiple locations of control, the halting of a single control path introduces a temporal distortion. Therefore a vital property of a convenient monitoring environment is the absence of disturbances or intrusions on the system behavior.

Deterministic Monitoring: Deterministic monitoring means that all conditions are observed, which cause a specific system behavior. For deterministic monitoring all entities must be observed with respect to contents, order and timing.

Reproducible Monitoring: Reproducible monitoring is advantageous for debugging, since erroneous conditions can be deterministically reproduced. For sustaining reproducibility in a conventional sequential program, it is sufficient to start at the same initial state and to provide the same inputs. In a real-time system, there is the additional

requirement for a reproducible timing behavior. Therefore, the inputs have to be reproduced with respect to contents, order and timing. Tasks have to be executed in the same order and the probe effect must be eliminated.

Service Provision: Smart Transducer Interfaces

For the provision of the previously described services three distinct interfaces can be applied [3]. This approach helps to control complexity by using different interfaces for different purposes. Figure 1 depicts the three interfaces of a smart transducer node.

Real-Time Service (RS) Interface

The RS interface provides the timely real-time services to the component environment during the operation of the system. In real-time systems it is a time-sensitive interface that must meet the temporal specification of the architecture in all specified load and fault scenarios. The composability of an architecture depends on the proper support of the specified RS interface properties (in the value and in the temporal domain) during the operation. From the point of a user, the internals of the component are not visible, since they are hidden behind the RS interface.

Diagnostic and Management (DM) Interface

The DM interface does not contribute to the composability and is normally not time-critical. The DM interface opens a communication channel to the internals of a component. It is used for setting component parameters and for retrieving information about the internals of the component, e.g., for the purpose of internal fault diagnosis. The maintenance engineer that accesses the component internals via the DM interface must have detailed knowledge about the internal structure and behavior of the component.

Configuration and Planning (CP) Interface

The CP interface is used to connect a component to other components of a system. It is used during the integration phase to generate the "glue" between the nearly autonomous components. The use of the CP interface does not require detailed knowledge about the internal operation of a component.

INTERFACE FILE SYSTEM

The Interface File System (IFS) [2] provides the name space for three interfaces discussed above. The communication among the IFS is fully temporally specified. The IFS is a unique addressing scheme for memory that contains, for example, transducer data (sensor measurements and/or set values for actuators, if necessary a history on former measurements), communication schedules, self-describing information (either locally stored information about the node's type, capabilities, etc. or a reference to a transducer electronic data sheet somewhere outside the fieldbus network, e. g. on the vendors web page), and internal state information for maintenance and diagnostic purposes.

The IFS is the source and sink for all communication activities via the three interfaces. The RS interface accesses the transducer data, the CP interface handles the configuration settings while the DM interface is used to monitor the internals of the node and to manage the sensor settings.

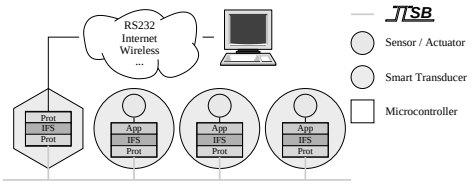


Figure 2. Physical Network Topology

The address space of the IFS is organized hierarchically representing a static file structure:

Cluster name: An 8 bit integer value that selects a cluster which is a network of fully interconnected nodes. 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. Aliases can have values from $0...255$, but some values have a special meaning, e. g. alias 0 addresses all nodes of a cluster in a broadcast manner.

File Name: The file name addresses a certain file within a node. A file name consists of a 6-bit identifier. Some file names have a consistent meaning in all nodes. Each file has a statically assigned number of records, located in ROM or RAM memory or even generated at runtime. The first record of each file is called the header record and contains the file length and a read-only flag.

Record Number: The record number is an 8-bit identifier that addresses the record within the selected file. Addressing a non-existing record of a file yields an error.

Figure 2 depicts the topology of a smart transducer cluster. All transducer nodes are built as smart transducers and contain a physical sensor or actuator, a microcontroller, and a network interface. The local application that accesses the sensor or actuator uses the IFS as a data source and sink. The leftmost node in the figure depicts a gateway node that connects the fieldbus to a different network. The gateway node hosts a second protocol layer that interconnects the fieldbus to other systems.

A time-triggered sensor bus (TTSB) will perform a periodical time-triggered communication to copy data from the IFS to the fieldbus and write received data into the IFS.

It is the task of the protocol to keep consistency among the local copies of the IFS data elements. A predefined communication schedule defines time, origin, and destination of each protocol communication. The instants of updates are specified a priori and known by the communicants. Thus, the IFS acts as a temporally specified interface that decouples the local transducer application from the communication task.

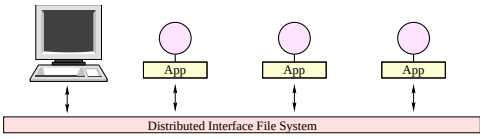


Figure 3. Logical Network Structure

The programmer's view of the network can be simplified by abstracting over the protocol communication. Thus, any ap-

plication "sees" just the IFS in the logical network structure depicted in Figure 3.

CASE STUDY

The focus of the case study lies in showing the effectiveness of the three transducer interfaces (RS,DM,CP) for providing the essential transducer services with compliance to the desired interface properties. The case study applies the time-triggered fieldbus protocol TTP/A for providing the three interfaces.
It introduces a network architecture consisting of three distinct levels (see Figure 4).

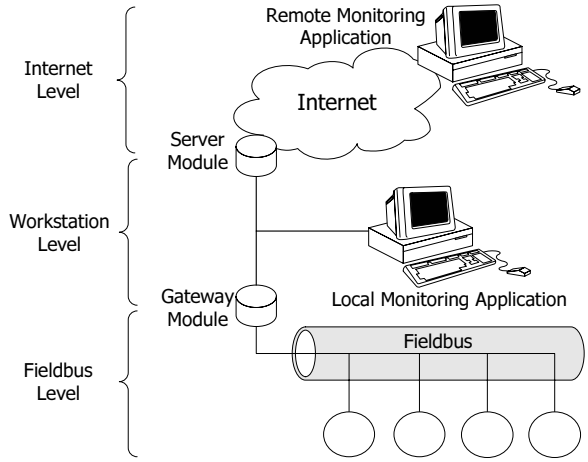
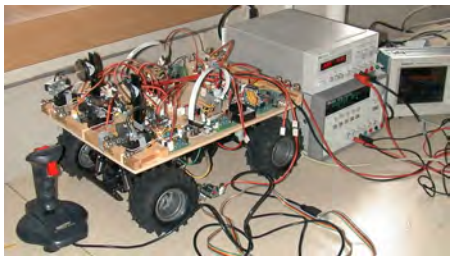


Figure 4. Network Architecture Levels

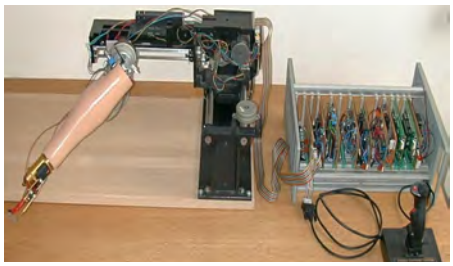
The fieldbus level connects the smart transducer nodes via a TTP/A network. It offers the predictable, timely exchange of real-time data with low jitter.
At the workstation level monitoring and configuration activities take place. Emphasis was set on the realization of the previously described monitoring qualities, namely reproducibility, determinism, and the elimination of the probe effect.
At the Internet level a CORBA server is used to establish availability of DM and CP services via the internet.
The implementation of the case study occurred as part of the development of two TTP/A smart transducer networks. These networks controlled an autonomous mobile robot and a demonstrator with an arm prosthesis [18, 19, 20].

Fieldbus Level: TTP/A

TTP/A is a time-triggered master/slave communication protocol for fieldbus applications that uses a time division multiple access (TDMA) bus arbitration scheme [21]. It is possible to address up to 255 nodes on a bus. One single node is the active master. This master provides the time base for the slave nodes. The communication is organized into rounds. Bus access conflicts are avoided by a strict TDMA schedule for each round. A round consists of several slots. A slot is a 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 round.
A TTP/A round (see Figure 6) consists of a configuration dependent number of slots and an assigned sender node for each slot.



(a) Mobile Car



(b) Arm Prosthesis

Figure 5. Demonstrators

The configuration of a round is defined in the RODL (ROund Descriptor List). The RODL defines which node transmits in a certain slot, the semantics of 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.
A master/slave round is a special round with a fixed layout that establishes a connection between the master and a particular slave for reading/writing monitoring or configuration data, e. g. the RODL information. In a master/slave round the master addresses a data record in the IFS format and specifies an action like reading, writing or executing on that record.

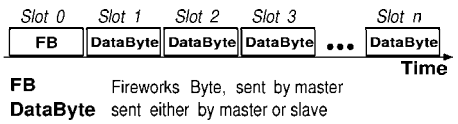


Figure 6. A TTP/A Round

Provision of the Three Interfaces in TTP/A

In TTP/A the RS interface is implemented through multipartner rounds. The master/slave rounds establish the CP and DM interface to the transducer nodes. Master/slave rounds are intended to be scheduled periodically between multipartner rounds as depicted in Figure 7.
They should even be scheduled in the absence of monitoring activities – thereby avoiding a modification to the timing behavior during diagnostic activities. Thus no interference with the real-time service can occur.

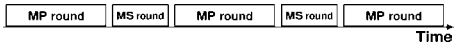


Figure 7. Recommended TTP/A Schedule

Workstation Level

The workstation level consists of both a local monitoring application and a CORBA server module representing the interface to remote monitoring clients. The local monitoring application allows monitoring, maintenance and dynamic configuration without indeterministic delays caused by the Internet. Remote client applications performing monitoring, diagnosis or configuration tasks access the Interface File System of the fieldbus level by sending requests to the CORBA Server Module.

The diagnostic and management (DM) and configuration and planning (CP) interfaces are implemented by providing access to a temporally specified shared memory, namely the IFS. The workstation level supports the dynamic selection of a certain fraction of the IFS, which is of interest to the user of the monitoring application. This part of the shared memory is periodically transferred at a constant bandwidth.

System Integration (Plug 'n' Play)

To illustrate the concept of using the IFS for system integration purposes Figure 8 shows how the user can initiate a scan for new nodes and how the newly connected nodes can be configured according to their transducer electronic datasheets. This approach is based upon the work presented in [22]. As a consequence of the user command the master scans for recently added nodes and automatically assigns a unique node alias (logical name) to each found node without affecting the real-time services.

In addition to the series and serial number (assigned by the hardware manufacturer) the alias of the newly connected node is stored in a record of the IFS providing feedback for the user. As soon as the alias is assigned the node can be configured according to its electronic datasheet using master/slave accesses. The alias is necessary for addressing a particular node during a master/slave access.

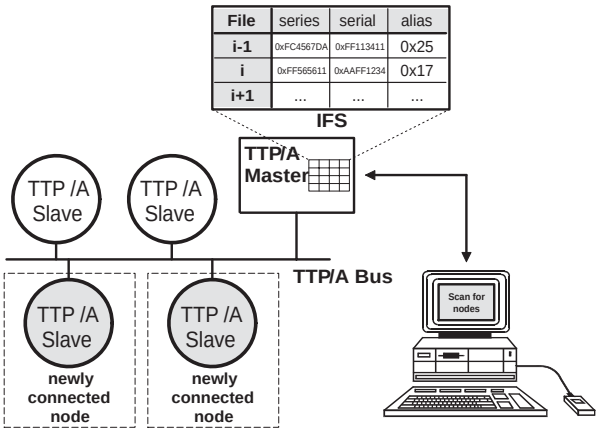


Figure 8. Dynamic Configuration Example: Plug'n'Play

Dynamic Configuration

Dynamic configuration occurs by performing a parameterization of transducer nodes. For example threshold values of an aging sensor can be altered to compensate for sensor wearout. Furthermore, the user can modify the TTP/A communication schedule (RODL files) to accommodate to evolutionary changes of the system (e. g. integration of additional transducer nodes into the TTP/A network).

Local Monitoring

The local monitoring module is superior to remote monitoring applications in terms of temporal predictability. The values of the interface file system are presented to the user in a graphical representation. The tool also offers configuration capabilities.

The local monitoring application serves the purpose of visualization of the real-time data. This visualization takes place with a scalable window of the information space. This approach to adjust the proper level of detail is called "Pan and Zoom approach" [23]. Operators are aided in the comprehension of events on the network. The monitoring functions of testing, debugging, configuration, maintenance and electronic datasheet extraction are enabled.

Establishment of the Monitoring Requirements

The probe effect was eliminated by executing the monitoring code in TTP/A nodes only during idle times. As a consequence no modification of the temporal behavior could occur. The determination of the necessary resources for allowing both time critical real-time operations and monitoring during peak load scenarios was possible by the a-priori knowledge of the processing time required for the execution of the monitoring code. Determinism was established by mapping all relevant data elements into the interface file system. This eases the monitoring task significantly. The IFS provides a fully specified interface in both the value and time domain between the application and the protocol code of the node. Reading this IFS periodically enables the system developer to gain insight of the node's state. By the provision of a global time in all TTP/A nodes events can be time stamped. Therefore all conditions causing a certain system behavior are observable with respect to their values and the point in time of monitoring. In contrast to existing monitoring solutions for fieldbus networks (e. g. [24, 25]) the internal state of nodes need not be broadcast on the bus to be observable.

Reproducibility was ensured by the properties of the TTP/A protocol. The time-triggered architecture results in a predefined execution order of tasks and no need for explicit synchronizing constructs (e. g. semaphores).

CORBA INTERFACE

The CORBA interface is well suited for configuration and diagnostic activities with different hardware platforms, because it enables interoperability in networks of different machine types and between programs written in different languages [26]. In our architecture a CORBA server functions as the server module (see Figure 4) and realizes the three interfaces (RS, DM, CP):

- Real-Time Service (RS) Interface: By using the RS interface a client is enabled to read or write an observation out of the memory of the server module. In addition a client can get information about the instants when the observations were read from the ST and when the next observation will be read as well as when the information will be written to the ST. It allows access to a limited and pre-configured set of records only but with given temporal constraints.
- Diagnostic and Management (DM) Interface: In contrast to the The DM interface is an inherently event-triggered

interface. Accesses via the DM interface cause the retrieval of observations from the fieldbus level. Thus, the DM-Interface is usually used for non-time critical activities only. It allows reading, writing, and executing every record of the IFS (even the internal ones).

- Configuration and Planning (CP) Interface: Due to the similar nature of the DM and CP interfaces (both interfaces do not have to provide hard real-time capabilities and are inherently event-triggered) this interface works in the same way as the DM interface, but restricts access to configuration specific records of the IFS (e. g. TDMA-schedule structures). By controlling visibility and hiding node internals, complexity is effectively reduced.

CORBA is designed to operate on variety of different network protocols. Therefore, the described CORBA solution supports access to the TTP/A smart tranducer network even via wide area networks. However, due to the indeterministic timing behavior of such networks, real-time capabilities strongly depend on a priori knowledge of the temporal characteristics of these networks.

CONCLUSION AND FUTURE WORK

This paper has identified essential services and interface requirements of a smart transducer network. We have argued that three interfaces, namely the real-time service (RS), the diagnostic and management (DM), and the configuration and planning (CP) interface proposed in [3], are well-suited for establishing these services while satisfying the presented interface requirements. Our case study demonstrates the effectiveness of the three interfaces. It examines the provision of the required services and the fulfillment of the interface requirements. The diagnostic service in the case study fulfills the major monitoring requirements (elimination of probe effects, determinism, reproduciblity) and does not effect the timely exchange of real-time data (real-time services).

ACKNOWLEDGEMENT

This work was supported in part by the Austrian Ministry of Science, project TTSB and by the European IST project DSoS under contract No IST-1999-11585.

REFERENCES

[1] W. H. Ko and C. D. Fung. VLSI and intelligent transducers. Sensors and Actuators, (2):239–250, 1982.

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

[3] Hermann Kopetz. Software engineering for real-time: A roadmap. Proceedings of the 22nd International Conference on Future of Software Engineering (FoSE) at ICSE 2000, 4th - 11th June 2000, Limerick, Ireland, Jun. 2000.

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

[5] C. E. McDowell and D. P. Helmbold. Debugging concurrent programs. ACM Computing Surveys, 21(4):593–622, December 1989.

[6] K. Lee. IEEE 1451: A standard in support of smart transducer networking. Proceedings of the 17th IEEE, 2:525–528, 2000.

[7] R. Johnson. Building plug-and-play networked smart transducers. Sensors Magazine, October 1997.

[8] S. E. Chodrow, F. Jahanian, and M. Donner. Run-time monitoring of real-time systems. In Robert Werner, editor, Proceedings of the Real-Time Systems Symposium - 1991, pages 74–83, San Antonio, Texas, USA, December 1991. IEEE Computer Society Press.

[9] M. Hiller. Error recovery using forced validity assisted by executable assertions for error detection: An experimental evaluation. In Proceedings of Euromicro Conference, 1999., University of York, York, England, June 1999. IEEE Computer Society Press.

[10] J. Kramer and J. Magee. Dynamic configuration for distributed systems. IEEE Transactions on Software Engineering, SE-11(4):424–436, 1985.

[11] H. Kopetz. In proceedings of the EMSOFT 2001, Tahoe City California, USA, 8th-10th October 2001.

[12] J. Reason. Human Error. Cambridge University Press, UK, October 1990.

[13] H. Thane. Monitoring, Testing and Debugging of Distributed Real-Time Systems. Phd thesis, Mechatronics Laboratory, Royal Institute of Technology, Stockholm, Sweden, May 2000.

[14] J. Gait. A probe effect in concurrent programs. Software Practice and Experience, 16(3):225–233, March 1986.

[15] C. H. Ledoux and D. Stott Parker. Saving traces for Ada debugging. In Ada in Use (1985 International Ada Conference), pages 97–108, Cambridge, England, May 1985. Cambridge University Press.

[16] C. Glawan. Monitoring von Echtzeitbetriebssystemen. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2000.

[17] M. Mansouri-Samani and M. Sloman. Monitoring distributed systems. IEEE Network, 7(6):20–30, November 1993.

[18] R. Obermaisser. Design and Implementation of a Distributed Smart Transducer Network. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2001.

[19] P. Peti. Monitoring and Configuration of a TTP/A Cluster in an Autonomous Mobile Robot. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2001.

[20] L. Schneider. Real time Robot Navigation with a Smart Transducer Network. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2001.

[21] H. Kopetz et al. Specification of the TTP/A protocol. Technical report, Technische Universität Wien, Institut für Technische Informatik, March 2000. Available at http://www.ttpforum.org.

[22] Wilfried Elmenreich, Wolfgang Haidinger, Philipp Peti, and Lukas 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.

[23] L. Bartram, A. Ho, J. Dill, and F. Henigman. The continuous zoom: A constrained fisheye technique for viewing and navigating large information spaces. In ACM Symposium on User Interface Software and Technology, pages 207–215, 1995.

[24] IXXAT, Inc. CANalyzer – new product descriptions. The Embedded Systems Conference San Francisco, 2001.

[25] A. Rajnak J. Specks. LIN protocol development tools and software interfaces for local interconnection networks in vehicles. 9th International Conference on Electronic Systems for Vehicles, Baden-Baden, 2000.

[26] J. Siegel. CORBA 3: Fundamentals and Programming. John Wiley and Sons, Heidelberg, 2000.