A Smart Transducer Interface Standard for Real-Time Systems

Wilfried Elmenreich, Wolfgang Haidinger, Hermann Kopetz
Thomas Losert, Roman Obermaisser, Michael Paulitsch
Institut für Technische Informatik
Technische Universität Wien
Treitlstrasse 1-3/182-1, Vienna, Austria
{wil,wh,hk,tl,ro,mp}@vmars.tuwien.ac.at

June 27, 2002

Abstract

A universal smart transducer interface should provide a framework that helps to reduce the complexity of large distributed real-time systems by introducing interfaces between smart transducers and their users which are simple and precisely specified within the value domain and in the temporal domain.

In this paper, we present the main concepts of a smart transducer interface that integrates a time-triggered communication protocol with an interface file system. The interface file system provides a unique naming and addressing scheme enabling access to internal transducer data via a CORBA gateway.

This smart transducer standard has been submitted in response to a request for proposal by the Object Management Group. It has been adopted as a world-wide standard in January 2002. Furthermore, we will present an implementation of two case studies of the smart transducer interface.

1 Introduction

The large number of transducers (sensors and actuators) in real-time systems require a generic approach for the organization of sensors and communication between sensors. This approach must features in terms of real-time guarantees, complexity management, and maintainability.

A possibility for reducing complexity is the concept of a *smart transducer*, which 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 transducer 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 [1].

In December 2000 the Object Management Group (OMG) issued a request for proposal (RFP) [2] of a *smart transducer interface* (STI) in order to form a world-wide standard, that satisfies the following needs: (i) real-time characteristics and functionalities for the smart transducer network (ii) online diagnostic service capability (iii) support for start-up and dynamic configuration (iv) a uniform naming and addressing scheme for all relevant data in the smart transducer system (v) a generic interface that enables the smart transducer system to interact with other systems via a CORBA (Common Object Request Broker Architecture) gateway, and (vi) the support of communication interfaces available on current low-cost microcontrollers, e. g. UART ports. In response to this RFP, a time-triggered communication architecture with a well-defined interface to a CORBA environment has been submitted jointly by three companies with support of the Vienna University of Technology. The interface to the smart transducers uses the concept of an interface file system (IFS) that maps all relevant transducer data to a common address scheme. This IFS allows different application-specific 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 that allows accessing the IFS data via a uniform addressing scheme from the CORBA gateway object and provides real-time time-triggered communication among the smart transducers. This proposed STI standard has been adopted by the OMG in January 2002 [3].

It is the objective of this paper to present the STI standard, the design principles that guided the architecture of this standard, and to describe two case study implementations. The rest of the paper is organized as follows: The following section presents related work on fieldbus standards and compares them to the requirements of the OMG RFP. Section 3 explains the design principles of the proposed STI. Section 4.2 addresses the properties of the IFS, which is the key feature of the STI. Section 4.3 depicts the time-triggered fieldbus protocol for the real-time communication in the smart transducer network. In section 5 we will describe two case studies of the STI. The discussion in section 6 concludes the paper.

2 Related Work

A smart transducer interface should conform to a world-wide standard. Such a standard for a real-time communication network has been long sought, but efforts to find one agreed standard have been hampered by vendors, which were reluctant to support such a single common standard in fear of losing some of their competitive advantages [4]. Hence, several different fieldbus solutions have been developed and promoted. Some of these existing solutions have been combined and standardized. In 1994, the two large fieldbus groups ISP (Interoperable Systems Project supported by Fisher-Rosemount, Siemens, Yokogawa, and others) and the WorldFIP (supported by Honeywell, Bailey, and others) joined to form the Fieldbus Foundation (FF). Their Foundation Fieldbus is designed to be compatible with the SP50 standard developed by ISA and IEC. ISA SP50 was the same committee that introduced the 4-20 mA standard back in the 1970s. However, the SP50 protocol does not support low-cost implementations using standard UARTs.

The IEC worked out the IEC 61158 standard, which is based on eight existing fieldbus solutions. However, the IEC fieldbus draft standard was not ratified at the final approval vote, following a set of controversies [5]. The IEC 61158 has the great disadvantage that it still keeps a diversity of eight different solutions, whereof some lack the support for real-time communication.

Meanwhile, another standard for smart transducers has been developed. The IEEE 1451.2 [6] standard deals with the specification of interfaces for smart transducers. An idea proposed by this standard is the specification of electronic data sheets to describe the hardware interface and communication protocols of the smart transducer interface model [7]. While the IEEE 1451 standard includes an adequate naming/addressing scheme and supports the configuration of large transducer systems, it lacks the explicit specification of real-time communication among the smart transducers.

Currently none of the above described systems supports the requirement for uniform access of internal transducer data via a CORBA interface.

3 Design Principles

The architecture of the smart transducer interface has been guided by the following principles:

3.1 Modelling of Observations

On an abstract level, the purpose of a real-time smart transducer interface is the timely exchange of "observations" of real-time entities between the engaged subsystems across the provided interfaces. A real-time entity is a state variable of interest that has a name and a value at a particular instant. An observation [8] is thus an atomic triple:

<name, observation instant, value>,

where name is an element of the common name space of real-time entities, the observation instant is a point in the "time space" and value is an element of the chosen value domain. An observation expresses that the referenced real-time entity possessed the stated value at the indicated instant. It is advantageous to have a synchronized global time among all nodes, enabling a common notion of time.

3.2 Interface Separation

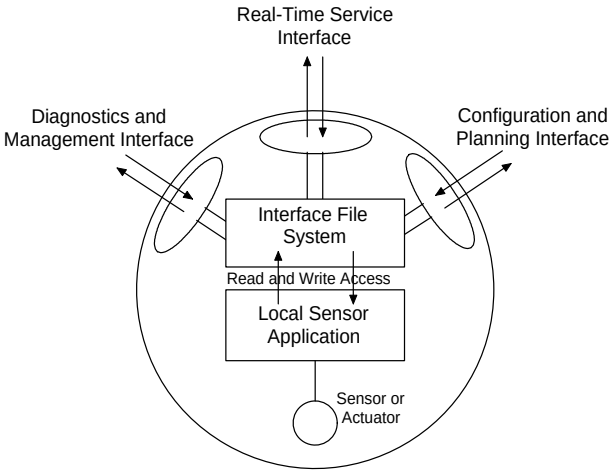

Fig. 1: Three Interfaces to a Smart Transducer Node

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 [9]. Therefore, interfaces for different purposes may differ by the accessible information and in the temporal behavior of the access across the interface. As depicted in figure 1 the STI specifies three different interfaces to a smart transducer:

DM interface: This is a *diagnostic and management* interface. It establishes a connection to a particular ST node and allows reading or modifying of specific IFS records. Most sensors need parameterization and calibration at startup and continuously collect diagnostic information to support the 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* 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 indented function. 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 ST nodes. Communicated data is usually data from sensors and for actuators. 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 TTP/A the time interval between sensing the environment and presenting the sensor value at the temporal firewall [10] 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.

3.3 Temporal Composability

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

During the system design phase this requires a two level design approach. At the overall system design level the system integrator defines the properties and interactions of the overall systems precisely in the value and time domain. The cluster design serves as a requirements definition for the component design where local details can be defined independently [11].

The components are characterized by their physical parameters and the services they provide across well-specified interfaces. In a composable architecture, this integration should proceed without unintended side effects. For an architecture to be composable, it must adhere to four necessary principles with respect to the interfaces of nodes [12]:

Independent Development of Nodes: Nodes can only be designed independently of each other, if the architecture supports the exact specification of all node services provided to the system. The interface data structures must be precisely specified in the value domain and in the temporal domain and a proper conceptual interface model of the node service, as viewed by a user of the node, must be available.

Stability of Prior Services: The stability-of-prior-service principle ensures that the validated service of a node - both in the value domain and in the time domain - is not refuted by the integration of the node into a system.

Performability of the Communication System: The performability-of-the-communication-system principle ensures that if n nodes are already integrated, the integration of the node n+1 will not disturb the correct operation of the n already integrated nodes. A proper configured time-triggered communication system satisfies this requirement.

Replica Determinism: If fault-tolerance is implemented by the replication of nodes, then the architecture and the nodes must support replica determinism. A set of replicated nodes is replica determinate if all the members of this set have the same externally visible state, and produce the same output messages at points in time that are at most an interval of d time units apart (as seen by an omniscient outside observer).

4 Properties of the STI Standard

4.1 Conceptual Model of the STI

A smart transducer system consists of several clusters with transducer nodes connected to a bus. Each cluster is connected to a CORBA gateway via a master node. One CORBA gateway can interface up to 250 clusters. The master of each cluster is connected to the CORBA gateway through a real-time communication network, which provides a synchronized time to each master. Each cluster can contain up to 250 STs 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.

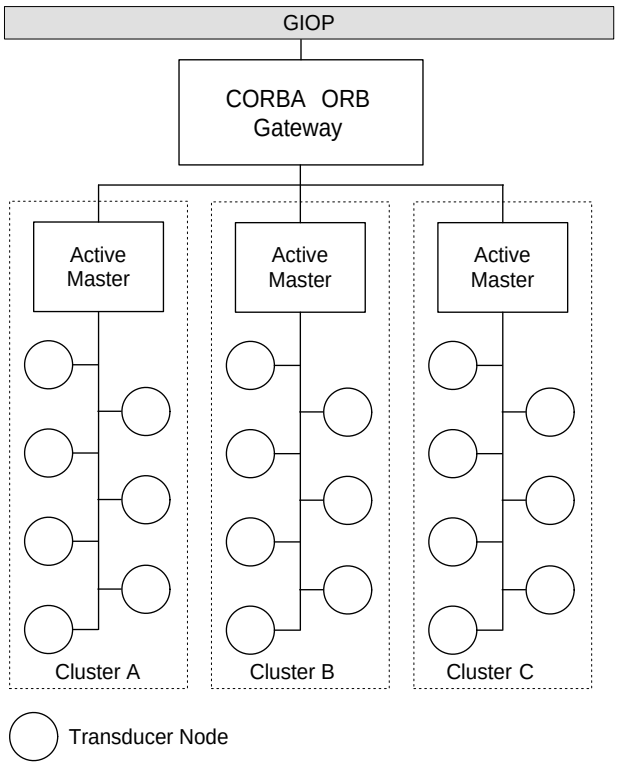

Fig. 2: Multi-Cluster Architecture with CORBA Gateway

Figure 2 depicts an example for such a smart transducer system.

The interface file system is used to provide a unique addressing scheme for the three interfaces. The time-triggered communication protocol establishes the three interfaces. It is possible to monitor the smart transducer system via the CORBA interface without disturbing the real-time traffic.

Although all transducer nodes are built as smart transducers and contain a physical sensor or actuator, a microcontroller, and a network interface, the hardware requirements are very flexible. The STI supports low-cost implementations of smart transducers, by allowing optional implementation of standard features. Thus, it is possible to fit a minimum STI implementation on an embedded microcontroller with 2k flash memory and 64 bytes of RAM memory [13].

4.2 Interface File System

The information transfer between a smart transducer and its client is achieved by sharing information that is contained in an internal interface file system (IFS), which is encapsulated 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 [1].

A time-triggered sensor bus will perform a periodical time-triggered communication to copy data from the IFS to the fieldbus and 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 the communication activities. A temporal firewall [10] 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 3, showing the data and control flow between a sender and a receiver.

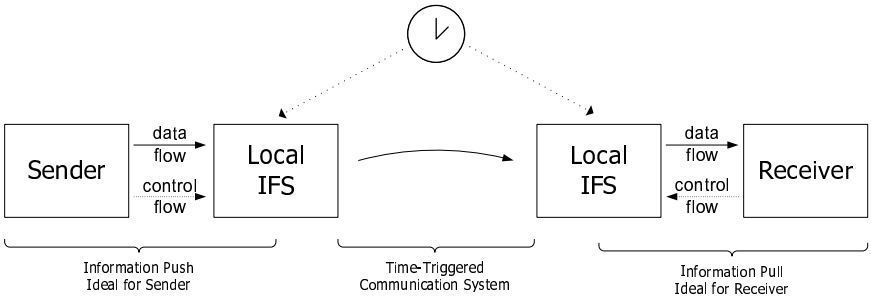

Fig. 3: Temporal Firewall

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. 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) [14]. 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 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 a typed data structures are fetched from the senders IFS and the instant when this typed data structures are delivered to the receivers IFS are common knowledge of the sender and the receiver. A predefined communication schedule defines time, origin, and destination of each protocol communication. Thus, the IFS acts as a *temporally specified interface* that decouples the local transducer application from the communication task.

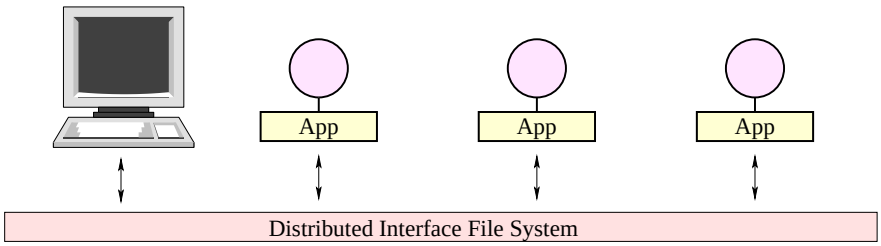

Fig. 4: Logical Network Structure

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 are also able to execute a clusterwide application by communicating directly with each other. Figure 4 depicts the network view for such a clusterwide application.

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 interfaces are mapped onto the fieldbus communication protocol, but with different semantics regarding timing and data protection.

4.3 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.

The bus allocation is done 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.

The protocol supports eight different firework bytes encoded in a message of one byte using a redundant bit code [15] 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, 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 5.

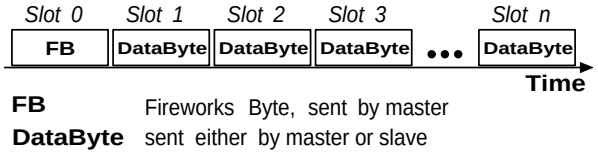

Fig. 5: A TTP/A Multipartner Round

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 in the hierarchical IFS address and specifies an action like reading, writing or executing on that record.

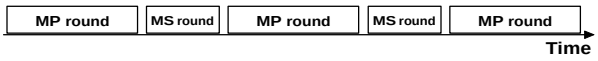

Fig. 6: Recommended TTP/A Schedule

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 6 in order to enable maintenance and monitoring activities during system operation without a probe effect.

5 Case Study Implementations

5.1 Robot Arm

As a demonstrator for the STI we built a system with a robot arm [16]. At the application level a human operator can control a prosthetic arm mounted on top of a linear thrust unit (See Figure 7). Simplicity of control for the user is established by the presence of intelligence in the demonstrator. Smart sensors yield information about the environmental conditions allowing avoidance of operating errors and obtaining precise control. Pressure sensors are present for determining the required

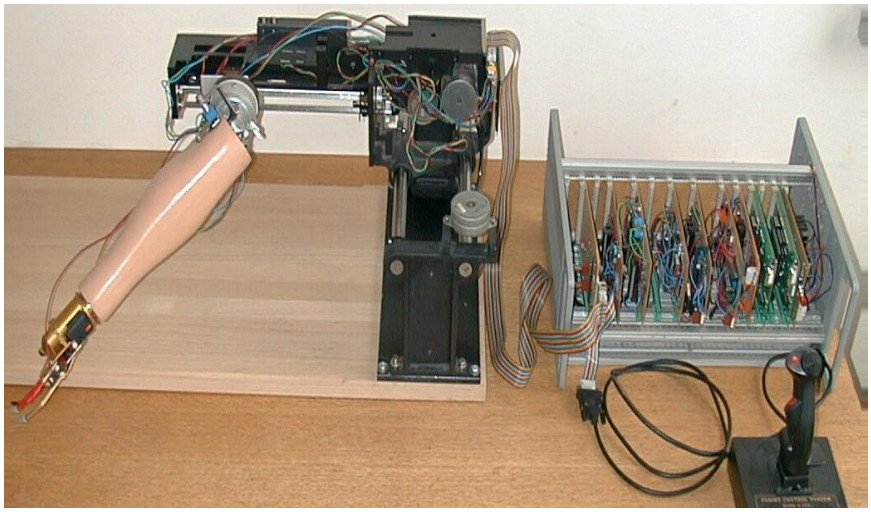

Fig. 7: Robot Arm

grip force to grasp an object. No intervention of the human operator is needed to avoid slipping of an object. Motor actuator nodes implement trapezoid excitation for handling of inertia. The robot arm is equipped with an angle sensor to allow limiting the opening of the elbow.

The demonstrator was also intended to investigate partitioning of nodes into distinct clusters. Intercluster communication was required to preserve temporal predictability and capabilities for monitoring, maintenance and configuration.

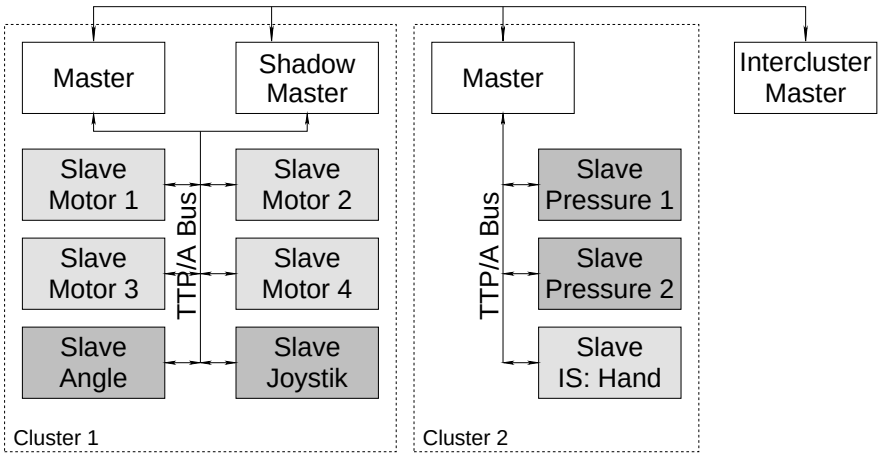

Fig. 8: Architecture of Robot Arm Transducer Network

The demonstrator consists of two clusters (See Figure 8). The first cluster contains the nodes

for controlling the motors of the linear thrust units, the elbow and the wrist. Nodes for retrieving the current angle of the elbow and the joystick commands are also placed in this cluster. A shadow master can take over control in case the primary master fails. Both masters are connected to the intercluster bus and act as intermediate systems. In addition to their TTP/A master role, they are slaves of a time-triggered backbone bus. The second cluster contains a node acting as an interface system for integrating the prosthetic hand into the demonstrator. Two nodes equipped with pressure sensors obtain measurements for grasping objects intelligently.

5.2 Autonomous Mobile Robot

Another implementation of the STI is shown by a model car, that acts as an autonomous robot with sensory inputs [17]. Seventeen nodes were used for building this mobile robot (“smart car”). Some of these nodes are implemented on very small MCUs to demonstrate the possibility of cheap slaves. Other nodes should demonstrate the possibility of complex features like plug & play or reconfiguration.

As depicted in Figure 10 the model comprises a smart car equipped with a suit of pivoted distance sensors, an electric drive and a steering unit. Distance sensors, servo motors for sensor pivoting, driving and steering units are all separate smart transducer nodes. Each node is implemented on a low-cost microcontroller and equipped with an STI.

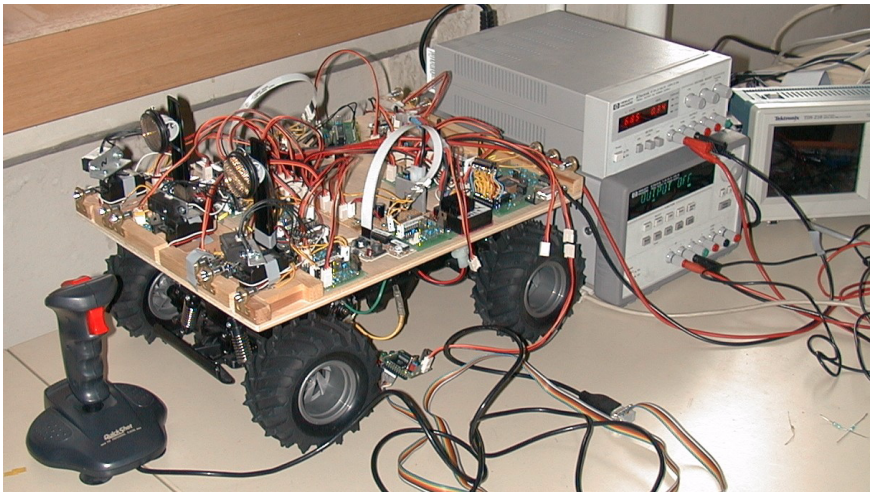

Fig. 9: Smart Car

The STI supports the integration of smart transducer nodes with a predictable timing behav-

ior [18]. It is possible to add extra “light” nodes to the car during operation.

Figure 9 depicts the functionality of the smart car. In order to achieve short efficient opera-

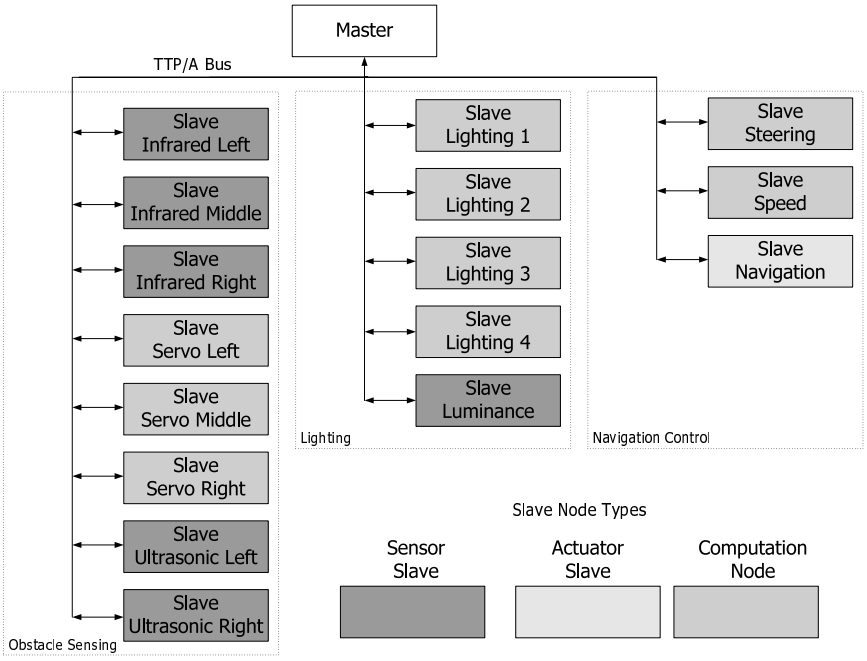

Fig. 10: Architecture of Smart Car Transducer Network

tion two different operation modes are defined. The STI standard supports up to 6 user-definable independent communication modes, which support applications running different modes.

As long as no obstacles are detected within the sensors’ range the car operates in “rabbit mode”. In this mode the car drives straight forward at full speed and two infrared sensors are aimed slightly outward the driving direction. The main detection of obstacles relies on two ultrasonic sensors. These are capable to report obstacles straight ahead of the car within a range of about 5m.

In case an obstacle is detected the car switches to “turtle mode”. In this mode the car uses a communication schedule where all infrared sensors and pivot servos are serviced. The distance sensors are swivelled around by servo motors so that they are able to scan the area in front of the robot. The sensors generate a value that corresponds to the distance of the object they are aimed at. The data stream provided by the distance sensors is taken over by a data processing node that fuses the perceptions from the distance sensors and the directions they are aimed at with a model of the robot environment. In this model the shapes of obstacles are stored and assigned with a probability value, that decreases with the progression of time and increases when the object is re-scanned.

From this data a navigation node calculates the speed and the direction to provide this information to the speed and steering nodes.

5.3 Monitoring and Configuration Tool

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 [19]. 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” [20, 21] or the “Heisenberg Uncertainty” [22] 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.

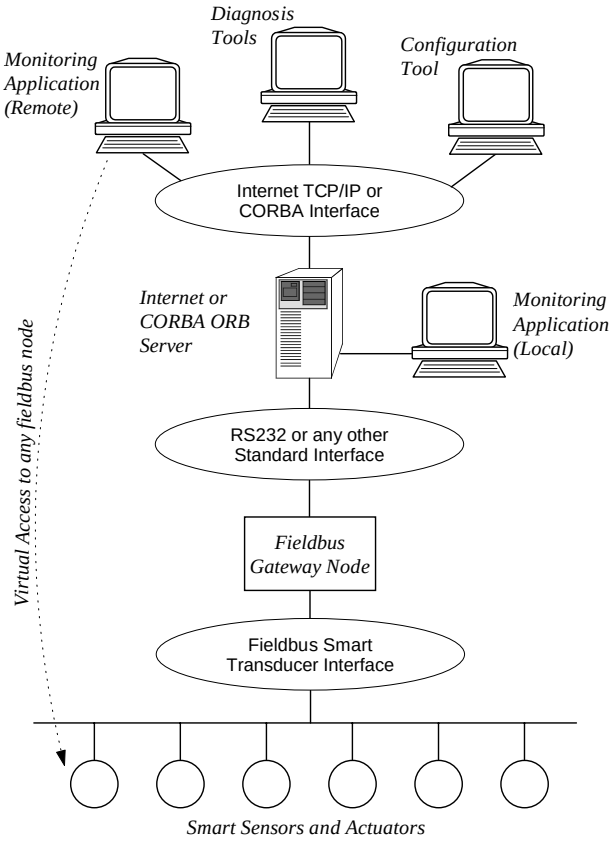

Fig. 11: System Architecture

We have developed a tool set for hard real-time environments, which establishes access via the DM and the CP interface. In contrast to existing tools relevant node internal information is accessible without changing the behavior in either the temporal or value domain of the RT interface. This property allows evolutionary changes of the system during real-time operation [23].

Figure 11 depicts the architecture of the monitoring approach.

The fieldbus gateway node is connected to the fieldbus and contains the software and hardware to establish a connection via a standard PC interface. We used an RS232 serial connection. Other possible connections are RS485, USB, IEEE 1394, or a wireless connection with IrDA or radio signals. The connection between fieldbus and server guarantees real-time behavior. The server runs a CORBA communication module and supports local tools for monitoring and configuration. Because the timing behavior of the communication from the local tools into the fieldbus network provides a deterministic timing behavior, the local tools can support deterministic real-time monitoring. However, compared to access at fieldbus level data is accessible at a lower bandwidth.

The communication module running on the PC will be a CORBA object. The CORBA middleware enables the transparent access of remote service applications to the fieldbus network through, e. g. the Internet. This interface is independent of the employed protocols and physical layers at fieldbus level.

6 Discussion

One requirement stated in the request for proposal by the OMG was real-time capability of the smart transducer interface.

The STI supports hard real-time communication by introducing a time-triggered communication scheme, that is a priori specified before the RS interface of the system is used.

Generally, time-triggered systems require an increased effort in the design phase of the system, but provide an easier verification of the temporal correctness [24]. Since time-triggered systems are designed according to the principle of resource adequacy [25], it is guaranteed that sufficient computing resources are available to handle the specified peak load scenario. On the other hand, time-triggered systems are often blamed for their bad flexibility. The STI overcomes this limitation by introducing means to configure the interaction of the components via the CP interface.

The RS interface provides composability, guaranteed timeliness, and hides components' internals. The DM and CP interfaces involve inherently event-triggered activities, which require an

event-triggered communication service. These interfaces cannot invalidate the temporal behavior of the RS interface and support full access to component internals – as required by a maintenance engineer.

The specification of interfaces should be complete and of minimal cognitive complexity. Cognitive complexity can be minimized by restricting interactions via carefully designed interfaces and by providing access restrictions. The kind of information that must be available via an interface depends on the purpose of the particular interface. For example, a properly designed operational interface hides component internals, thereby allowing a component to form a meaningful abstraction. The corresponding operational interface specification stipulated during architecture design should incorporate a precise specification of a component's inputs and outputs in both the temporal and value domain. A maintenance engineer on the other hand, might require access to intermediate computational results for locating the origin of an incorrect system behavior.

The STI standard meets the requirement for complete interfaces of minimal cognitive complexity by introducing three different types of interfaces of a component. The separation into three interfaces (RS, DM and CP) is done according to the interface purpose, the necessary level of visibility of component internal information, and the type of the temporal interaction patterns. Such a separation minimizes complexity in contrast to a universal interface incorporating support for all possible interactions.

The STI standard also specifies the provision of the three interfaces through a CORBA server. However, currently there is no CORBA architecture for effectively supporting the temporal requirements to establish the RS interface. Current priority-based approaches like Real-time CORBA [26] require complete knowledge about all other service requests and their corresponding priority values in the whole CORBA network. However, the availability of a global notion of time allows to record the instant of the acquisition of a real-time entity's state in each observation.

As a proof of concept, we have implemented two case studies of the STI. The first case study comprises a demonstrator with a robot arm that is instrumented by a smart transducer network partitioned into two clusters. The second case study is an autonomous mobile robot, that shows the integration of new nodes and efficient communication despite of static communication schedules. The results show that the STI standard is an interesting option for various sensor network applications. The STI provides many features that are required by fieldbus applications for automotive or automation industries. Supported features are the real-time capability, the encapsulation of the node's internals, and a universal address space with the interface file system. The STI can be

implemented on low-cost Commercial-off-the-Shelf (COTS) hardware and supports various bus media types.

7 Acknowledgments

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] 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.

[2] Object Management Group OMG. Smart transducers interface request for proposal. *OMG TC Document orbos/2000-12-13*, Dec. 2000. Available at http://www.omg.org.

[3] Object Management Group. Smart Transducers Interface Specification. *Document ptc/2002-05-01*, May 2002. Available at http://www.omg.org.

[4] J. J. Pinto. A neutral instrumentation vendor's perspective. *ISA Proceedings '94 and Intech July '95*, July 1995.

[5] P. Noury. WorldFIP, IEC 61158 and the internet: A new look at fieldbuses, 1999. Available at http://www.worldfip.org/noury02.html.

[6] P. Conway, D. Heffernan, B. O'Mara, P. Burton, and T. Miao. IEEE 1451.2: An interpretation and example interpretation. *Proceedings of the Instrumentation and Measurement Technology Conference*, pages 535–540, 2000.

[7] L. H. Eccles. A brief description of IEEE P1451.2. *Sensors Expo*, May 1998.

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

[9] 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.

[10] 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.

[11] S. Poledna, H. Angelow, M. Glück, M. Pisecky, I. Smaili, G. Stöger, C. Tanzer, and G. Kroiss. TTP two level design approach: Tool support for composable fault-tolerant real-time systems. *SAE World Congress 2000, Detroit, Michigan, USA*, March 2000.

[12] H. Kopetz and R. Obermaisser. Temporal composability. *IEE's Computing & Control Engineering Journal*, 2002.

[13] C. Trödhandl. Architectural requirements for TTP/A nodes. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2002.

[14] W. Elmenreich, W. Haidinger, and H. Kopetz. Interface design for smart transducers. *IEEE Instrumentation and Measurement Technology Conference, Budapest, Hungary*, 2001.

[15] 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, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2000.

[16] R. Obermaisser. Design and implementation of a distributed smart transducer network. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2001.

[17] W. Elmenreich, W. Haidinger, H. Kopetz, T. Losert, R. Obermaisser, M. Paulitsch, and C. Trödhandl. A smart sensor LIF case study: Autonomous mobile robot. *DSoS Project (IST-1999-11585) Deliverable PCE3*, Apr. 2002.

[18] 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.

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

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

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

[22] 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.

[23] Philipp Peti, Roman Obermaisser, Wilfried Elmenreich, and Thomas Losert. An architecture supporting monitoring and configuration in real-time smart transducer networks. In *The First IEEE International Conference on Sensors (IEEE SENSORS 2002)*, Jun. 2002.

[24] Hermann Kopetz. Should responsive systems be event-triggered or time-triggered? *Institute of Electronics, Information, and Communications Engineers (IEICE) Transactions on Information and Systems*, E76-D(11):1325–1332, 1993.

[25] H. W. Lawson. *Parallel Processing in Industrial Real-Time Applications*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[26] D. C. Schmidt and F. Kuhns. An overview of the real-time CORBA specification. *IEEE Computer*, 33(6):56–63, 2000.