Interface Design for Smart Transducers

W. Elmenreich, W. Haidinger, and H. Kopetz
Institut für Technische Informatik
TU Vienna, Austria
{wilfried,wolfgang,hk}@vmars.tuwien.ac.at

June 22, 2001

Abstract

This paper describes design issues on smart transducer interfaces to hide the internal node properties and allow a decoupling of applications from communication properties like message timing, flow control and bus access. We present a smart transducer interface that incorporates three different interfaces (real-time service, diagnostic/maintenance, configuration/planning).

Further decomposition of real-time systems can be provided by hiding the sensor properties from the application program. Therefore, a data structure that represents a model of the environment is introduced. This extra interface reduces the complexity of the application and enables reuse of the application code.

Finally, an application of the presented concepts is described in a case study featuring a mobile robot.

Keywords: transducer interface, sensor fusion, real time, time-triggered architecture, interface design.

1 Introduction

Transducers are sensors and actuators in order that a computer system can interact with the physical environment. Today's requirements for transducer networks are multifaceted: The communication service must guarantee transmission of real-time data with predictable timing and small jitter, the interface to transducers should be simple and consistent, the system should be expandable, and, at last, acquisition, operational, and maintenance costs should be low.

To achieve this goals the design of transducer networks has to follow several guidelines:

First, common boundaries between smart transducers and their users have to be introduced. Information exchange across such an interface is only possible if the engaged subsystems share a common background of concepts and a common coding space [1].

Second the application of sensor fusion algorithms on the physical sensor data increases the accuracy of the physical sensor. Thus, the necessary resolution or coverage of a sensor system is achievable with a set of less precise, but cheaper sensors.

A further motivation for sensor fusion is the reduction of system complexity by hiding data streams of physical sensors behind a sensor fusion unit. Since in a traditionally designed system the sensor measurements are fed directly into the application, the application has to cope with a lot of imprecise, ambiguous, and incomplete data streams. In a system where sensor data is preprocessed by fusing methods the input to the application is a more robust and more comprehensive view of the environment [2], which reduces the complexity of the application.

Related work on smart transducer interfaces can be found in [3] and [4] regarding the IEEE P1451:2 standards. This standard has its main focus on generic sensor interfaces for different communication layers.

In our work, we examine a smart transducer interface in a Time-Triggered Architecture (TTA) [5] with a focus on its communication timing behavior.

The rest of the paper is organized as follows: Section 2 explores the abstract properties of an interface. Section 3 describes the specification of a smart transducer interface as part of the Time-Triggered Architecture [5] and Section 4 describes the implementation of the Time-Triggered Architecture for a mobile robot. The paper is concluded in Section 5.

2 Interfaces

An interface is a common boundary between two subsystems. An information exchange across an interface is only possible if the engaged subsystems share a common background of concepts and a common coding system. In the context of a distributed control system, the smallest area of concern is a cluster, consisting of a set of sensors, actuators, and processing nodes connected by a communication medium.

Common Code Spaces

Communication is only possible, if the interfacing subsystems share the concepts and representation of the data items in the interface. To exchange observations across a smart sensor interface, agreement on three code spaces must be provided [1]:

The Name Space is necessary for the meaning of transmitted values.

The Time Space serves for the definition of an instant of an event among the communicating nodes.

The Value Domain provides encoding schemes for the transmitted values.

One key task in the development of a generic smart transducer model is concerned with the specification of these code spaces and the meaning of the referenced elements.

Observations

In the abstract, the purpose of a smart sensor interface is the timely exchange of "observations" of real-time entities between the engaged subsystems across the provided interfaces. An RT entity is a state variable of interest that has a name and a value. An observation [6] is thus an atomic triple:

<RT entity name, instant, value>,

where RT entity name is an element of the common name space of RT entities, instant is point on the "time space" and value is an element of the chosen domain of values. An observation thus states that the referenced RT entity possessed the stated value at the indicated instant.

If the value of an observation does not change over the time interval of interest, we call this observation a timeless observation. Timeless observations can be represented by tuples, consisting only of an entity name and the associated value.

Flow Control

Communication between subsystems exchanges information in two distinct domains, the *time domain* and the *value domain*. In the value domain the message data is transmitted, while in the time domain *control information* is exchanged [7]. Control information allows the generating subsystem to influence the *temporal control flow* [6] of the other subsystem.

Commonly a communication between two subsystems is either controlled by the sender's request (*push* style) or by the receiver's request (*pull* style) [8].

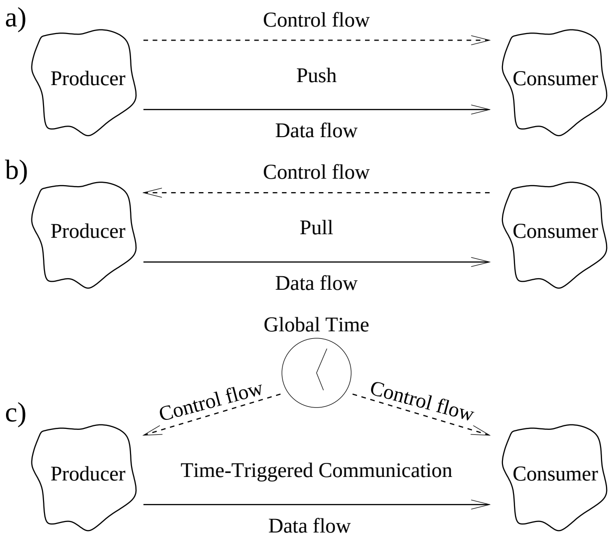

Figure 1: Push-, Pull-, and decoupled Communication

For explanation let us assume two components, A and B that need to exchange data over a network.

Further, without restrictions to generality, we assume the message data to be transmitted from producer A to consumer B.

In order to transfer data between two components, they must agree on the mechanism to use and the direction of the transfer.

Figure 1 a) shows the push method. The supplier A is empowered to generate and send its message at any time. Flow control relies on A. This method is very comfortable for the push supplier (A), but the receiving push consumer (B) has to be watchful for incoming 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 file [9]. The push style communication is the basic mechanism of event-triggered systems.

In Figure 1 b) the flow control is on the consumer (B). Whenever B wants to access the message information, supplier A has to respond on the request. This facilitates the task for the pull consumer B, but the pull supplier A has now to be watchful for incoming data requests. Popular "pull" mechanisms are reading a file, polling, state messages, or shared variables [9]. The pull style communication is the basic mechanism of client-server systems.

Figure 1 c) depicts a communication model where the flow control of A is decoupled from the flow control of B.

Each subsystem possesses a memory object that acts as a data source and sink for communication activities. Components that want to submit data are able to write the data into this memory without hesitation. After transmission of the message data from the memory element in subsystem A to the memory element in subsystem B via a communication system component B accesses this data at any time. The values in the memory are state messages that keep their content until they are updated and overwritten [10].

This approach combines a push supplier interface for component A and a pull consumer interface for component B. The critical ends of the push and the pull communication rely on the memory elements. However, because these elements are usually passive components, the negative effects of the push and the pull communication do not affect the system performance.

To avoid interference between concurrent read and write operations on the memory element, the task of the communication system is done by a time-triggered protocol like TTP/A [11] or TTP/C [12]. Since in the time-triggered architecture all nodes have knowledge about transmission schedules and access to a global time base, the instant when the protocol updates a value in the memory element is known to all components.

Interfaces for Smart Transducers

A 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 [1].

The smart sensor technology offers a number of advantages from the points of view of technology, cost, and complexity management [13]:

- Electrically weak non-linear sensor signals can be conditioned, calibrated and transformed into digital form on a single silicon die without any noise pickup from long external signal transmission lines [14].

- The smart sensor contains a well-specified digital communication interface to a sensor bus, offering "plug-and-play" capability if the sensor contains a reference to its documentation in form of an electronic data sheet as it is proposed in the IEEE 1451 Standard [15].

- It is possible to monitor locally the operation of the sensing element via the network and thus simplify the diagnosis at the system level.

The internal complexity of the smart-sensor hardware and software and internal failure modes can be hidden from the user by well-designed fully specified smart sensor interfaces that provide just those services that the user is interested in.

In order to support complexity management and composability, it is useful to specify distinct interfaces for functional different services [16]. We have identified the following three types of interfaces:

Real-Time Service (RS) Interface: This interface provides the timely real-time services to the component during the operation of the system.

Diagnostic and Maintenance (DM) Interface: This interface opens a communication channel to the internals of a component. It is used to set parameters and to retrieve information about the internals of a component, e. g., for the purpose of fault diagnosis. The DM interface is available during system operation without disturbing the real-time service. Normally, the DM interface is not time-critical.

Configuration and Planning (CP) Interface: This interface is necessary to access configuration properties of a node. During the integration phase this interface is used to generate the "glue" between the nearly autonomous components. The CP interface is not time-critical.

3 The Approach of the Time Triggered Architecture

In this section, the principles of operation of the Time-Triggered Protocol, the ideas behind the Interface File System and the different views of a transducer node are described.

Time-Triggered Communication

TTP/A [11] is a time-triggered master-slave communication protocol for fieldbus applications that uses a time division multiple access (TDMA) bus arbitration scheme.

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 strictly TDMA schedule for

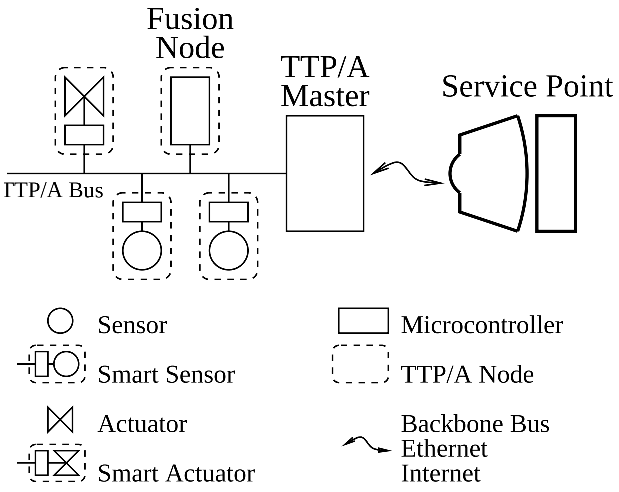

Figure 2: TTP/A Cluster

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 selects one of 8 rounds with a preconfigured communication layout.

The Interface File System

For unique addressing of the slave's internals all relevant data of a TTP/A node like round definitions, application specific parameters and I/O properties are organized into a structure called Interface File System (IFS) [1]. The IFS is structured in a record-oriented format. Each record is addressable separately by master-slave rounds.

The Interface File System was introduced for two reasons:

- Provide a consistent view of the transducer properties.
- Decouple subsystems from the point of view of temporal control.

The addressing scheme of the IFS serves as the common name space for the smart transducer interface as described in Section 2. Time space and value domain are also standardized in the TTP/A protocol specification.

All nodes contain several files that can be accessed over the TTP/A protocol in a unified manner. The minimal set-up for a smart transducer is:

Round Descriptor List (RODL) File: Each node contains at least one and up to six RODL files that contain TDMA schedules for the TTP/A multipartner rounds.

TTP/A Configuration File: This file contains at least an 8 bit alias which is the slave's name when it is addressed via master-slave rounds.

Documentation File: This file consists of the node's serial and series number, two 32 bit identifiers that are assigned invariably to each node. The series number identifies the node's type while the serial number distinguishes nodes of equal type. Optionally this documentation file contains the ASCII text of an uniform resource locator (URL) pointing to a file containing the node's data sheet. Documentation files are read-only from the master's viewpoint.

TTP/A Round Types

A multipartner round (see Figure 3) consists of a configuration dependent number of slots and an assigned sender node for each slot. 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.

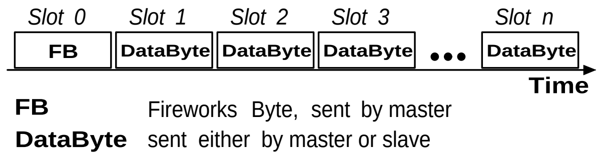
Slot 0
Slot 1
Slot 2
Slot 3
Slot n
FB
DataByte
DataByte
DataByte
DataByte
Time
FB Fireworks Byte, sent by master
DataByte sent either by master or slave

Figure 3: A TTP/A Multipartner Round

A master-slave round establishes a connection between the master and a slave for reading/writing monitoring or configuration data, e. g. the RODL information. The action and the memory addressing is encoded in three parameter bytes of a master-slave round. In a further part the addressed data bytes are transmitted between master and slave. A master-slave round has a fixed layout. The address scheme is derived from the Interface File System that is explained in the next Section.

At startup the master uses master-slave (MS) rounds for determining the types of the connected nodes and configuring them. The multipartner (MP) round is intended to establish a periodical, predictable and efficient real-time communication. To support a diagnosis and maintenance access concurrent to the real-time traffic it is recommended to schedule multipartner rounds and master-slave rounds interleaved (see Figure 4).

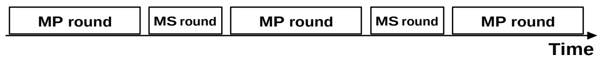

Figure 4: Recommended TTP/A Schedule

To support ultra-low-cost implementations of TTP/A slave nodes, it is also possible to omit the implementation of the master-slave rounds and the file system and hard-code the TDMA schedule for the TTP/A multipartner rounds. Such a node would not respond to any master-slave round and does not support configurability. It is possible to build heterogeneous networks with ultra-lost-cost nodes and configurable nodes together but this might have an negative effect for the system overview because the maintenance program is "blind" on the ultra-low-cost nodes.

4 Case Study

For evaluation purposes we worked out the concept of a mobile robot ("smart car") equipped with a suit of pivoted distance sensors, an electric motor and a steering unit. The distance sensors and the DC motors for sensor pivoting, the driving and steering are all equipped with a smart TTP/A transducer interface implemeted on a low-cost microcontroller. The fieldbus network also contains a master node and a data processing node. The distance sensors are swiveled around by stepping motors so they are able to scan the area in front of the robot. The sensors return a value that correspondents to the distance of the object they are aimed at.

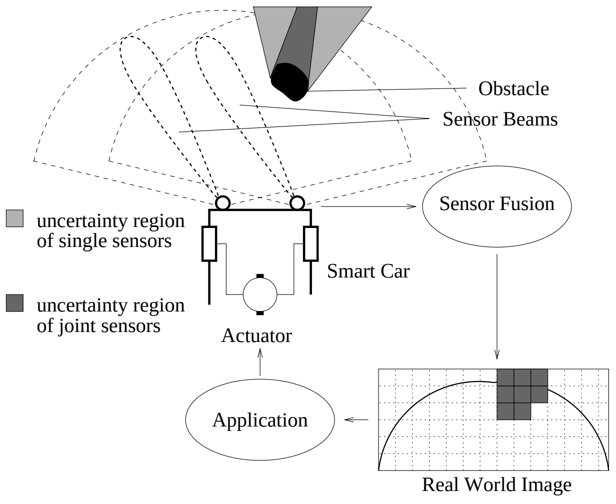

Figure 5: Smart Car

The data stream that is provided by the distance sensors is taken over by the data processing node, that fuses the perceptions from the distance sensors 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. Figure 5 depicts the scanning range of the distance sensors, the fusing of sensor data into a real world image and the control application that makes decisions about direction and speed of further movement on basis of this real world image. Fusing the data from two sensors leads to the following benefits [17]:

Robustness: In case one sensor fails, the other one is sufficient to update a constricted sector of the real world image.

Extended Temporal Coverage: One sensor can perform a measurement when the other cannot, thus the effective sampling rate increases.

Extended Spatial Coverage: The real world image presents an area that is expanded in comparison to the scanning sector of a single sensor.

Increased Confidence: In the overlapping sector in front of the robot, measurements of one sensor are confirmed by measurements of the other sensor.

Reduced Uncertainty: Combining information reduces the region of uncertainty about the environment. E. g. the area behind an object cannot be observed by the sensors, but this area of uncertainty is much smaller, when observations from different angles and sensors are joint (see Figure 5).

Although the smart car is able to move autonomously through its environment, it also comprises an interface to a service point. As depicted in Figure 2 the TTP/A master also acts as a gateway, that communicates over a backbone network. Via a service point, the user gets access to the configuration/planning and diagnostic/maintenance interface as described in section 2.

5 Conclusion

We introduced a smart transducer interface that hides the internal node properties and allows a decoupling of applications from communication properties like message timing, flow control, and bus access. A smart transducer needs three different interfaces for real-time service, diagnostic/maintenance and configuration/planning.

A further decomposition of a real time system can be achieved by hiding the sensor properties behind a data structure that represents a model of the environment the sensors are observing. This data structure can be serviced by a sensor fusion node and reduces the complexity of the application.

These concepts can be used in real-time applications like mobile robots.

Acknowledgments

We would like to give special thanks to our colleagues Günther Bauer, Raimund Kirner, Michael Paulitsch, Philipp Peti, Stefan Pitzek, and Lukas Schneider for constructive comments on earlier versions of this paper. 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. *Proceedings of the 3rd International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, March 2000.

[2] E. Bosse, J. Roy, and D. Grenier. Data fusion concepts applied to a suite of dissimilar sensors. *Canadian Conference on Electrical and Computer Engineering, 1996*, 2:692–695, May 1996.

[3] B. Travis. Sensors smarten up. *EDN Access*, pages 76–86, March 1999.

[4] K. B. Lee and R. D. Schneeman. Distributed measurement and control based on the IEEE 1451 smart transducer interface standards. *IEEE Transactions on Instrumentation and Measurement*, 49(3):621–627, June 2000.

[5] C. Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, and C. Temple. Time-Triggered Architecture (TTA). *Advances in Information Technologies: The Business Challenge, IOS Press*, 1997.

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

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

[8] Alcatel Corp., Fujitsu Ltd., IBM, NEC Corp., NTT Corp., and IONA Tech. Management of event domains. *OMG TC Document telecom/2000-01-01*, Jan. 2000. Available at http://www.omg.org.

[9] R. Deline. *Resolving Packaging Mismatch*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, June 1999.

[10] Information Society Technologies IST. Preliminary version of conceptual model. Technical Report 20, IST-1999-11585 Dependable Systems of Systems (DSoS), 2000. Available at http://www.vmars.tuwien.ac.at.

[11] 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.

[12] H. Kopetz. *Specification of the TTP/C Protocol*. TTTech, Schönbrunner Straße 7, A-1040 Vienna, July 1999. Available at http://www.ttpforum.org.

[13] H. Kopetz. Do current technology trends enforce a paradigm shift in the industrial automation market? *Closing Keynote at the 7th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 99), Barcelona, Spain*, October 1999.

[14] P. Dierauer and B. Woolever. Understanding smart devices. *Industrial Computing*, pages 47–50, 1998.

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

[16] H. Kopetz. Software engineering for real-time: A roadmap. *IEEE Software Engineering Conference, Limmerick, Ireland*, 2000.

[17] E. Waltz and J. Llinas. *Multisensor Data Fusion*. Artech House, Norwood, Massachusetts, 1990.