THE THREE INTERFACES OF A SMART TRANSDUCER

H. Kopetz *

* *Institut für Technische Informatik*
TU Vienna, Austria
email: hk@vmars.tuwien.ac.at

Abstract: A smart transducer is an intelligent device that comprises in a compact small unit a sensor or actuator element (possibly both), a micro-controller, a communication controller and the associated software for signal conditioning, calibration, diagnostics, and communication. Such a smart transducer can be viewed from three different viewpoints, the real-time service (RS) viewpoint, the diagnostic and maintenace (DM) viewpoint and the configuration and planning (CP) viewpoint, that have different characteristics and are directed to different clients. In order to separate the concerns, this paper argues that these viewpoints should be supported by three different interfaces in order to reduce the complexity of each interface.

Keywords: smart transducer, real-time system, distributed system, sensor bus, interface file system.

1. INTRODUCTION

The impressive advances in the field of computer hardware in the recent years make it economically feasible that the old architectural proverb "form follows function" becomes the guiding principle for the design of distributed real-time computer systems of the future. In nearly all but the high volume applications it will be economically justified to partition a system along functional hardware/software boundaries in order to avoid the extra software design and validation effort needed to cohabitate unrelated functions on the same hardware units. In such an architecture the issue of composability and reusability, how to build systems constructively out of pre-validated software/hardware components that provide well-defined functions, moves to the center of interest.

One natural boundary between nearly autonomous subsystems is the boundary between a smart transducer and the user/provider of the sensor/actuator information. A smart transducer is the combination of a sensor or actuator element and a local microcontroller that contains the interface circuitry, a processing element, memory and a network controller in a single unit. More and more sensor elements are themselves microelectronic mechanical systems that can be integrated on the same silicon die as the associated microcontroller. The smart sensor technology offers a number of advantages from the points of view of technology, cost and complexity management (Dierauer and Woolever, 1998).

(1) Electrically weak non-linear sensor signals that originate from an MEMS sensor can be generated, conditioned, transformed into digital form and calibrated on a single silicon die without any noise pickup from long external signal transmission lines.
(2) It is possible to locally monitor the sensor operation and thus simplify the diagnostics. In some cases it is possible to build smart

sensors that have a single simple external failure mode-fail-silent, i.e., the sensor operates correctly or does not operate at all.

(3) The interface of the smart sensor to its environment can be a simple well-specified digital communication interface to a sensor bus, offering "plug-and-play" capability if the sensor contains pointers to its own documentation on silicon.

(4) The internal complexity of the smart-sensor hardware and software and the internal sensor failure modes can be hidden from the user by a well-designed fully specified smart sensor interface that provides just those services that the user is interested in. Thus, the smart sensor technology can contribute to a reduction of the complexity at the system level.

The focus of this contribution is the reduction of the cognitive complexity of large real-time systems. An approach towards achieving this goal is to apply the principle of "divide and conquer" and partition the large system into nearly autonomous subsystems that interact via well-specified and easy-to-comprehend simple interfaces. One such boundary line is at the interface between a smart transducer and its environment. But even at this boundary line, a complexity reduction can be achieved by providing distinctly different interfaces for the different services a smart transducer is intended to provide. We have identified three different sets of services that are directed to different users and thus justify the introduction of different interfaces: the real-time (RS) service viewpoint, the diagnostic and maintenance (DM) viewpoint and the configuration and planning (CP) viewpoint.

The paper is organized as follows: The following Section 2 discusses the characteristics of an interface and interface complexity. The three different views of a component are the topic of Section 3. Section 4 is devoted to the issue of composability. Section 5 discusses the proposed smart transducer interface. The paper terminates with a conclusion in Section 6.

2. WHAT IS AN INTERFACE

An *interface* is not only a meeting point of a system with its environment, but also a boundary between the system and its environment (Jones *et al.*, 2001). For the purpose of our analysis, we focus on *message interfaces* in a distributed real-time computing system. A distributed real-time computing system consists of a set of nodes that are interconnected by a real-time communication system that is capable to transport messages within given time constraints. A *message* is an atomic data structure that is formed for the purpose of inter-node communication. We assume that the interface contains an *interface memory* that can store messages. The *interface memory* can be accessed by the communication system and by the internal processes of the components. The instant when the sending of a message starts at the sender (by taking the first byte of a message out of the sender's interface memory) is called the *send instant* of the message. The instant when the receiving of a message terminates at the receiver (by storing the last byte of the message into the receiver's interface memory) is called the *receive instant* of the message. In our model, the structure of the interface memory depends on the information contents of the messages, whether a message contains *state information* or *event information.*

2.1 State Information versus Event Information

Any property of a *real-time (RT) entity* (i.e., a *relevant state variable*) that is observed by a node of the distributed real-time system at a particular instant, e.g, the temperature of a vessel, is called a *state attribute* and the corresponding information *state information* (Kopetz, 1997). A *state observation* records the state of a state variable at a particular instant, the *point of observation.* A *state observation* can be expressed by the atomic triple:

<Name of the observed state variable, observed value, time of observation>

For example, the following is a state observation: "*The position of control valve A was at 75 degrees at 10:42 a.m.*" State information is *idempotent* and requires an *at-least once semantics* when transmitted to a client. At the sender, state information is *not consumed on sending* and at the receiver, state information requires an *update-in-place* and a *non-consumable read* from the interface memory. State information is transmitted in *state messages.*

A sudden change of state of an RT entity that occurs at an instant is an *event.* Information that describes an event is called *event information.* Event information contains the *difference* between the state *before* the event and the state *after* the event. An *event observation* can be expressed by the atomic triple

<Name of the observed state variable, value difference, time of event>

For example, the following is an event observation: "*The position of control valve A changed by 5 degrees at 10:42 a.m.*" Event observations requires *exactly-once semantics* when transmitted to a consumer. At the sender, event information

is *consumed on sending* and at the receiver, event information must be *queued* and *consumed on reading* from the interface memory. Event information is transmitted in *event messages*.

Periodic state observations or sporadic event observations are two *alternative* approaches for the observation of a dynamic environment in order to reconstruct *the states and events* of the environment at the observer (Tisato and DePaoli, 1995). Periodic state observations produce a sequence of equidistant "snapshots" of the environment that can be used by the observer to reconstruct those events that occur within a minimum temporal distance that is longer than the duration of the sampling period. Starting from an initial state, a complete sequence of (sporadic) event observations can be used by the observer to reconstruct the complete sequence of states of the RT entity that occurred in the environment. However, if there is no minimum duration between events assumed, the observer and the communication system must be infinitely fast.

2.2 Interface Specification

The specification of a message interface must contain all the information that a user of the component services is required to know in order to make use of the services provided across this particular interface, but not more. In general, a node will contain many different interfaces, each one with its characteristic service specification. The services offered across a particular interface will thus be a subset of the services provided by the node as a whole.

An interface service specification must specify the structure of the data exchanged across this interface, the meaning of the data contained in the messages, and the instants when messages are sent and received. While it is relatively straightforward to specify the structure of a message, e.g., by an interface definition language such as IDL, the formal specification of the meaning of the data, the formal semantics of the data, is an active research topic (de Alfaro and Henzinger, 2001) The specification of the message send and receive instants is facilitated if a global notion of time is available in the distributed system. This topic is discussed in more detail in Section 5.1.

3. THE THREE VIEWS OF A COMPONENT

From the point of view of complexity management and composability, it is useful to distinguish between three different types of interfaces of a component: the real-time-service (RS) interface, the diagnostic and maintenance (DM) interface, and the configuration and planning (CP) interface (Ran and Xu, 1997) (Kopetz, 2000). For the temporal composability, the most important interface is the RS interface. The complexity of the RS interface is thus the determining term for the complexity reduction achieved by the chosen decomposition.

3.1 The Real-Time Service (RS) View

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 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.

A user of the services of a node that are offered across the RS service interface must comprehend the RS service interface specification. The mental effort required to understand this interface specification determines the cognitive complexity of the interface. The cognitive complexity of an interface, depends, among others on the following:

(1) The number of elements that are exposed at the interface
(2) The interactions of the interface elements.
(3) The representation of the interface specification.
(4) The experience of the observer.

A possible (and reasonable) quantitative measure of the cognitive complexity of an interface is given by the amount of time an "average user" requires to understand the given interface specification.

The decomposition of a large system in nearly autonomous nodes should be driven by the concern to minimize the RS service interface complexity. Only if the complexity of the RS service interface is significantly lower than the complexity of the node as a whole, the chosen partitioning will lead to a significant reduction of complexity at the system level.

3.2 The Diagnostic and Maintenance (DM) View

The DM interface opens a communication channel into 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.

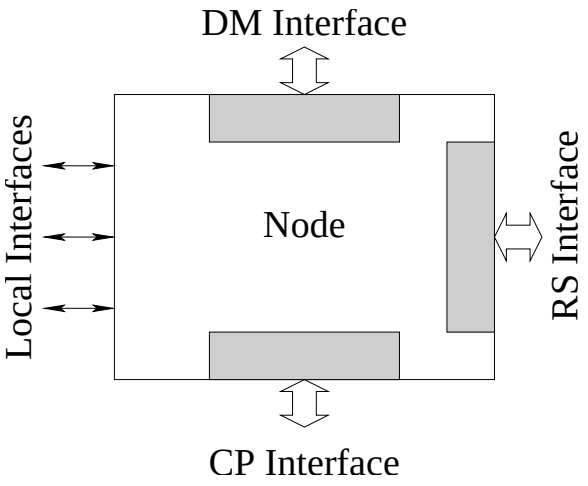

Fig. 1. The Three Interfaces of a Node

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. Since a diagnostic engineer has to comprehend the internals of the complete node, the DM interface complexity will be determined by the complexity of the internals of a node. The DM interface is not contributing to the composability. Normally, the DM interface is not time-critical.

3.3 The Configuration and Planning (CP) View

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. The CP interface is not time critical.

Fig. 1 depicts the different views of a node. In addition to the local interfaces to the node's environment (sensor and actuator interfaces) the three linking interfaces, the service interface, the diagnostic and maintenance interface, and the configuration and planning interface are shown. Only the indicated section of the node has to be understood in order to use the services of the interfaces. The complexity reduction of a given decomposition depends on the quotient of the service interface complexity to the total complexity of a node.

4. THE PRINCIPLES OF COMPOSABILITY

In a distributed real-time system the components interact via a communication system to provide the emergent real-time services. These emerging services depend on the timely provision of the real-time information at the RS interfaces of the components.

For an architecture to be composable, it must adhere to the following four principles with respect to the RS-interfaces:

(1) Independent development of components.
(2) Stability of prior services.
(3) Performability of the communication system
(4) Replica determinism

4.1 Independent Development of Components

A composable architecture must distinguish sharply between architecture design and component design. Principle one of a composable architecture is concerned with design at the architecture level. Components can only be designed independently of each other, if the architecture supports the precise specification of all component services at the level of architecture design. In a real-time system the RS-interface specification of a component must comprise the precise CNI specification in the value domain and in the temporal domain and a proper abstract model of the component service, as perceived by the user of the component. Only then is the component designer in the position to know exactly what can be expected from the environment and what must be delivered by the component.

4.2 Stability of Prior Services

Principle two of a composable architecture is concerned with the design at the component level. A component is a nearly autonomous subsystem that comprises the hardware, the operating system and the application software. The component must provide the intended services across the well-specified component interfaces. The design of the component can take advantage of any established software engineering methodology, such as object based design methods. The stability-of-prior-service principle ensures that the validated RS service of a component-both in the value domain and in the time domain-is not refuted by the integration of the component into a system. For example, the integration of a self-contained component, e.g., an engine controller, into the integrated vehicle control system may require additional computational resources of the component to service the RS-interface, both in processing time and in memory space. Consider the case where the CNI is based on a queue of messages that must be maintained by the host computer: memory space for the queue must be allocated by the component-local operating system and processing time of the host processor for the management of the queue must be made available. In a time-critical component it may happen that these additional resource requirements that are needed

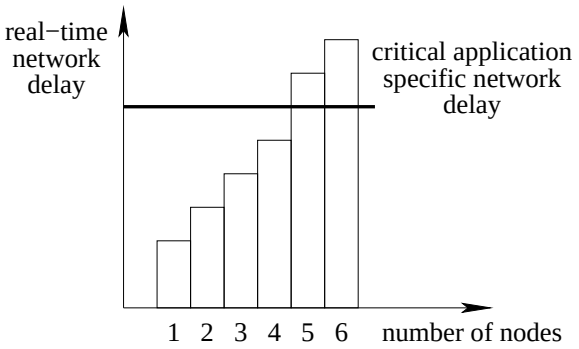

Fig. 2. Maximum network delay at critical instant as a function of the number of nodes.

for the timely interface service, are in conflict with the resource requirements of the application software that implements the prior services of the component. In such a situation, failures in the component services may occur sporadically.

4.3 Performability of the Communication System

Principle three of a composable architecture is concerned with the design of the communication system. Normally, the integration of the components into the system follows a step-by-step procedure, thus gradually increasing the load on the communication system. The performability of the communication system principle requires that if n components are already integrated, the integration of the $n+1$ component may not disturb the correct operation of the n already integrated components. This principle ensures that the integration activity is linear and not circular.

This performability of the communication principle has severe implications for the management of the network resources. If network resources are managed dynamically, it must be ascertained that even at the critical instant, i.e., when all components request the network resources at the same point in time, the timeliness of all communication requests can be satisfied. Otherwise sporadic failures will occur with a failure rate that is increasing with the number of components integrated.

For example, if an RS service requires that the maximum latency of the network must always remain below a critical upper limit (because otherwise a local time-out within the component may signal a communication failure) then the dynamic extension of the network latency by adding new components may be a cause of concern. In a network based on dynamic global load sharing, the message delay at the critical instant (when all components request service at the same instant) increases with the number of components. The system of Fig. 2 will work correctly with up to four components. The addition of the fifth component may lead to sporadic failures.

Other applications, e.g., when a time-sensitive control loop is closed by the network, may require a network of known and constant jitter in order to support this principles of performability of the communication system.

4.4 Replica Determinism

If fault-tolerance is implemented by the replication of components, then the architecture and the components must support replica determinism. A set of replicated components is *replica determinate* (Poledna, 1994) 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 the omniscient outside observer). In a fault-tolerant system, the time interval d determines the time it takes to replace a missing message or an erroneous message from a node by a correct message from redundant replicas. The implementation of replica determinism is simplified if all components have access to a globally synchronized sparse time base.

5. THE SMART TRANSDUCER (ST) INTERFACE

The smart transducer interface (Kopetz *et al.*, 2001), which is currently in the process of standardization by the OMG (OMG, 2001), specifies all three interfaces of a smart transducer, i.e., the service interface, the diagnostic and maintenance interface, and the configuration planning interface. As outlined before, the most important interface for the user of a smart transducer is the service interface.

The service (RS) interface is specification is based on the concept of a temporal firewall (Kopetz and Nossal, 1997).

5.1 Temporal Firewalls

We call a unidirectional data-flow interface *elementary*, if there is only a unidirectional control flow (Kopetz, 1999) across this interface. An interface that supports periodic state messages with error detection at the receiver is an example of such an elementary interface. We call a unidirectional data-flow interface *composite*, if even a unidirectional data flow requires a bi-directional control flow. An event-message interface with error detection is an example for a composite interface. Composite interfaces are inherently more complex than elementary interfaces, since the correct operation of the sender *depends on* the control signals from all receivers. This can be a problem in multicast

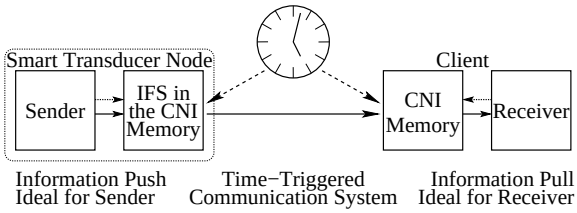

Fig. 3. Data Flow (full line) and Control Flow (dashed line) at an TTA Interface

communication where many control messages are generated for every unidirectional data transfer, and each one of the receivers can affect the operation of the sender. Multicast communication is common in distributed embedded systems.

The service interface of the smart transducer interface is an *elementary* interface. A time-triggered transport protocol carries autonomously-driven by its time-triggered schedule-state messages from the sender's interface memory to the receiver's interface memory. The sender can deposit the information into its local interface memory according to the information *push* paradigm, while the receiver must *pull* the information out of its local interface memory. From the point of view of temporal predictability, *information-push* into a local memory at the sender and *information-pull* from a local memory at the receiver are ideal, since no *unpredictable task delays* that extend the worst-case execution occur during communication. A receiver that is working on a time-critical task is never interrupted by a control signal from the communication system. Since no control signals cross the interface (the communication system generates its own control signals for the *a priori* known *fetch* and *delivery* instants from the progression of the global time as depicted in Fig. 3, control-error propagation is eliminated by design. We call an interface that prevents control-error propagation by design a *temporal firewall* (Kopetz and Nossal, 1997). The integrity of the data in the temporal firewall is assured by the non-blocking NBW concurrency control protocol (Kopetz and Reisinger, 1993).

On input, the precise interface specifications (*receive instants* of messages and the value domain of the messages) are the *pre-conditions* for the correct operation of the host software. On output, the precise interface specifications (*send instants* of messages and the value domain of the messages) are the *post-conditions* that must be satisfied by the host, provided the preconditions have been satisfied by the host environment. The host-software must guarantee that the post-conditions are satisfied, assuming that the pre-conditions are met (*assume-guarantee reasoning*).

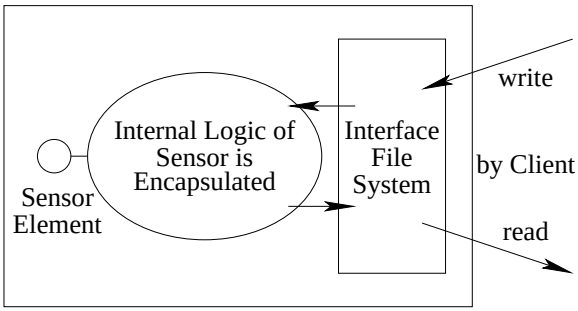

Fig. 4. Interface File System in a Smart Transducer

5.2 The Interface File System

In the smart transducer (ST) interface, the *interface* memory of a smart transducer contains an *interface-file system (IFS)* as depicted in Fig. 4. This IFS is the source and sink of all information exchanged between an ST and its clients across the linking interfaces. It is at the core of the ST interface model, which is thus a data-centric model.

An IFS file is an index sequential file with up to 256 records of a fixed record length of four bytes. An IFS record is the smallest addressable unit. Within a transducer system, every record of every IFS file has a unique hierarchical address (which also serves as the global name of the record) consisting of the concatenation of the *cluster name*, the *logical node name*, the *file name* and the *record name*. Since each name has a length of one single byte, the name of a record is thus also four bytes long and fits itself into a single record. There are three operations defined on a record: *read*, *write*, and *execute*. These operations are described in detail in the OMG standard (OMG, 2001).

The IFS of a node contains up to 64 IFS files. Some of these files are assigned to the real-time service (RS) interface, others to the configuration and planning (CP) interface, while the remaining files can be used by the node internal application processes and can be accessed via the diagnostic and maintenance (DM) interface in order to analyse the internal operation of the node.

In very small STs, the IFS can degenerate to a few records of a few files. Such an ST will only support a limited functionality for a particular mass-market application. In order to become a viable standard for these mass-market applications, this standard proposal suggests a set of services, that starts with a very limited minimum service level. This minimum service level must be provided by any conforming implementation. If more services than these minimum services are provided, this proposal defines services that can be combined like building blocks in order to design an appropriate ST. If a building block is implemented the ST

must provide the full set of services of this building block. The specification of further building blocks will be object of future standards.

5.3 The Transport Service

The smart transducer transport service must support the time-triggered transport of *frames* from one node to all other nodes of a cluster (broadcast transport service within a cluster). A frame consists of one or more bytes sent by a node. Since the instant when a frame is sent is controlled-either directly or indirectly-by a *master*, it is assured that only one sender will access the communication channel at a particular instant. In case the communication is not successful, there is no automatic retransmission. The communication system is thus predictable with a known latency and minimal jitter. Different transport protocols, such as CAN or LIN, or the wireless IEEE 802.11, can be used. For low-cost transducers, a single wire UART transport protocol that uses an ISO standardized physical layer is specified in the OMG standard (OMG, 2001).

6. CONCLUSIONS

The introduction of the smart-sensor technology provides a notable opportunity to reduce the complexity of a large distributed control application. By distinguishing between the three different views of a smart sensor-the service view (RS), the diagnostic and maintenance (DM) view and the configuration and planning (CP) view-and introducing a separate interface for each one of these views, it is possible to reduce the cognitive complexity of the ST interfaces. Since only the service view is relevant from the point of view of composability, such a decomposition will lead to an improved understanding of the system level functions of a large distributed control application.

ACKNOWLEDGEMENTS

This work was supported, in part, by the IST project DSOS and by the IST project FIT.

REFERENCES

de Alfaro, L. and T.A. Henzinger (2001). *Embedded Software*. Chap. Interface Theories for Component-Based Design, pp. 148–165. Vol. 2211 of *Lecture Notes in Computer Science*. Springer Verlag. Heidelberg, Germany.

Dierauer, P. and B. Woolever (1998). Understanding smart devices. *Industrial Computing* **October 1998**, 47–50.

Jones, C., H. Kopetz, E. Marsden, M. Paulitsch, D. Powell, B. Randell and R. Stroud (2001). Revised version of conceptual model. Technical Project Deliverable, Interface Characterization 1 (IC1). IST Programme RTD Research Project IST-1999-11585 "Dependable Systems of Systems (DSoS)". available: http://www.newcastle.research.ec.org/dsos/.

Kopetz, H. (1997). *Real-Time Systems, Design Principles for Distributed Embedded Applications*. 3^{rd} ed.. Kluwer Academic Publisheres. Boston.

Kopetz, H. (1999). Elementary versus composite interfaces in distributed real-time systems. In: *ISADS 1999*. IEEE Press. Tokyo, Japan.

Kopetz, H. (2000). Software engineering for real-time: A roadmap. In: *Software Engineering Conference 2000*. IEEE Press. Limmerick, Ireland.

Kopetz, H. and J. Reisinger (1993). The non-blocking write protocol (NBW): A solution to a real-time synchronization problem. In: *14^{th} Real-Time System Symposium*. Raleigh-Durahm, North Carolina.

Kopetz, H. and R. Nossal (1997). Temporal firewalls in large distributed real-time systems. In: *IEEE Workshop on Future Trends in Distributed Computing*. IEEE Press. Tunis, Tunesia.

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

OMG (2001). Smart transducer interface – initial submission. Technical report. Object Management Group. Needham, Massachusets, USA.

Poledna, S. (1994). Replica determinism in fault-tolerant real-time systems. Technical report. University of Technologie of Vienna.

Ran, A. and J. Xu (1997). Architecting software with interface objects. In: *8th Israeli Conference on Computer Systems and Software Engineering*. IEEE Press.

Tisato, F. and F. DePaoli (1995). On the duality between event-driven and time-driven models. In: *13th IFAC DCCS 1995*. Toulouse, France.