Abstract
This paper deals with sensor networking frameworks, namely interconnected fieldbuses and IEEE 1451 smart transducer interface architecture. The contribution begins with interconnected fieldbuses mediating an access from an Intranet to sensors; particularly, coupling architectures are treated both from the viewpoint of more classical interconnections of wide-area and local-area networks, and from the viewpoint of fieldbus domain. Next section of the paper reviews some concepts aimed at connecting sensors and actuators through low-level fieldbuses to Intranets or the Internet. The following section introduces the IEEE 1451 smart transducer interface for sensors and actuators as an emerging, standard-based networking framework. The kernel of the paper deals with the application of the discussed frameworks for a computer-based pressure measurement system as a real-world project, stressing the Intranet connectivity issues.

1. Introduction

A framework represents a set of constraints on components and their interaction, and a set of benefits that derive from those constraints [Le00]. For the embedded, computer-based systems domain, components can be any kind of hardware/software building blocks. This paper compares two sensor networking frameworks that demonstrate contemporary alternatives for design of sensor-based industrial applications.

Industrial distributed computer-based systems typically encompass, at their lowest level, various digital actuator/sensor-to-controller connections. Those connections, usually based on low-level fieldbuses, constitute the bottom, vendor-dependent segments of hierarchical communication systems that typically include higher-level fieldbus or LAN backbones and, more recently, an access to and from Intranets/Internet. Hence, the systems must comprise suitable interconnections of incident higher and lower fieldbus segments, which intermediate not only top-down commands and bottom-up responses but also remote system management. Nevertheless, only sporadic attention has been devoted to the fieldbus interconnectivity issues; see e.g. [AF97], [DN95]. The first part of this contribution dedicates attention to the multiple fieldbus interconnecting architectures that fit hierarchical structures with sensors on low-level segments.

A special case of the previous problem appears connecting a fieldbus to an Intranet. While for the majority of high-level fieldbuses the solutions are commercially available, they offer, as a rule, a proprietary approach. Similarly to fieldbus-to-fieldbus coupling, it is possible only to map the broad variety of concepts applicable to fieldbus-to-Intranet interconnections.

Current industry is migrating away from proprietary hardware and software platforms mentioned above in favor of open and standardized approaches. Internet technologies such as Java, WWW, TCP/IP, and also Ethernet are rapidly becoming the platforms of choice for building next generation distributed measurement and control systems. The second considered framework, which can support the current trend, is the IEEE 1451 smart transducer interface architecture that enables to unify not only interconnecting smart sensors with various fieldbuses but also direct coupling to the Ethernet-based Intranet. The proposed standards include an object-oriented information model targeting software-based, network independent, transducer application environments with a standard digital interface and a communication protocol for accessing sensors and actuators.

2. Interconnected Fieldbuses

This section aims at interconnecting low-level fieldbuses. The framework focuses on the fact that various smart sensors and actuators are equipped with different fieldbus interfaces, and that their internetworking requires
interconnecting the fieldbuses applied, see [VS97], [SV99], [ZS99].

2.1. General taxonomy

According to the ISO Open Systems Interconnection vocabulary, two or more subnetworks are interconnected using an equipment called as intermediate system whose primary function is to relay selectively information from one subnetwork to another and to perform protocol conversion where necessary, see [JA90]. A bridge or a router provides the means for interconnecting two physically distinct networks, which differ occasionally in two or three lower layers respectively. The bridge converts frames with consistent addressing schemes at the data-link layer while the router deals with packets at the network layer. Lower layers of these intermediate systems are implemented according to the proper architectures of interconnected networks. When subnetworks differ in their higher layer protocols, especially in the application layer, or when the communication functions of the bottom three layers are not sufficient for coupling, the intermediate system, called in this case as gateway, contains all layers of the networks involved and converts application messages between appropriate formats.

An intermediate system represents typically a node that belongs simultaneously to two or more interconnected networks. The backbone network interconnects more intermediate systems that enable to access different networks. If two segments of a network are interconnected through another network, the technique called tunneling enables to transfer protocol data units of the end segments nested in the proper protocol data units of the interconnecting network.

2.2. Fieldbus interconnection architectures

The architecture of fieldbus interconnections covers both the network topology of an interconnected system and the structure of its intermediate system, which is often called in that domain as coupler or bus coupler. On the other hand, the term gateway sometimes denotes an accessory connecting PC or PLC to a fieldbus. For this paper, the expression "gateway" preserves its original meaning according to the ISO-OSI terminology as discussed above.

The first item to be classed appears the level ordering of interconnected networks. A peer-to-peer structure occurs when two interconnected networks interchange commands and responses through a bus coupler in both directions so that no one of the subnetworks can be distinguished as a higher level, see Fig. 1. In case of the master/slaves or single master configuration, the coupler consists of two interconnected masters (two interconnected slaves can fit very special applications). If two interconnected subnetworks arise hierarchically ordered, see Fig. 2, the master/slaves configuration appears usual at least for the lower-level network.

The second point of view for couplers classification stems from the protocol profiles involved. In this case, the standard taxonomy using the general terminology mentioned above can be employed: bridge, router, and gateway. Also, the tunneling and backbone networks can be distinguished in a standard manner. Nevertheless, the most common architecture in this case appears to be a gateway because of usually different application layers of the interconnected fieldbusses. In that case, the gateway has to translate namely between appropriate messaging protocols.

The next, refining items to be classed include internal logical and physical architectures of the coupler, such as routing strategy (source or adaptive) and routing and relaying algorithms (more detailed specification) embedded in communication tasks, I/O drivers and task synchronization and communication services of a supporting real-time operating system kernel, number of processors and the type of their connection (direct serial or parallel, indirect by FIFO queue or by dual-port RAM). Of course, the complete information about a coupler can be offered only by the detailed description of a concrete implementation, see e.g. [SV00].

Fig. 1. Peer-to-peer interconnection configuration

Fig. 2. Hierarchical interconnection configuration
3. Fieldbus-to-Intranet Coupling

To join a fieldbus or an interconnected system of fieldbuses to an Internet-compatible Intranet/Internet, it is necessary to develop the full gateway that has to adhere to the adjacent fieldbus protocol profile on one side and to the selected Internet protocol profile on the other side, and has to translate between messaging protocols of application layers in both domains. While the fieldbus side architecture of the gateway is usually predetermined by a concrete fieldbus protocol suite, the Intranet side offers free choice among many possible profiles even if we focus our attention on WWW-based technologies, see e.g. [BK00], [SL97].

Evidently, the interconnections of sensors with Internet/Intranets by mediating fieldbus systems result in multitude of proprietary solutions. Those interconnections cannot be standardized internationally because of the cancellation of the related IEC Fieldbus standardization initiative. Hence, that methodology appears useful only to meet special application or implementation requirements. On the other hand, there is a possibility how to manage that problem utilizing the sensor/actuator-based standardized approach. The next section reviews the family of IEEE 1451 standards as a framework for sensors coupling to various fieldbuses or directly to the Ethernet-based Intranet.

4. IEEE 1451 Network Interface

The IEEE 1451 consists of the family of standards for a networked smart transducer interface that include namely (i) a smart transducer information model, 1451.1 [IE96], targeting software-based, network independent, transducer application environments, (ii) and a standard digital interface and communication protocol, 1451.2 [IE97], for accessing the transducer via the microprocessor modeled by the 1451.1. The next two standards, 1451.3 and 1451.4, extend the possible single-attached configurations to embedded distributed multidrop systems and to mixed-mode communication protocols for analog transducers.

The 1451.1 information model [LS96] deals with an object-oriented definition of a network capable application processor (NCAP), which is the object-oriented embodiment of a smart networked device. This model includes the definition of all application-level access to network resources and transducer hardware. The object model definition encompasses a set of objects classes, attributes, methods, and behaviors that provide a concise description of a transducer and a network environment to which it may connect. The standard brings a network and transducer hardware neutral environment in which a concrete implementation can be developed.

The standard uses block and base classes to describe the transducer device. The 1451.1 defines four component classes offering patterns for one Physical Block, and one or more Transducer Blocks, Function Blocks, and Network Blocks. Each block class may include specific base classes from the model. The base classes include Parameters, Actions, Events, and Files, and provide component class.

All classes in the model have an abstract or root class from which they are derived. This abstract class includes several attributes and methods that are common to all classes in the model and provide a class definition facility to be used for instantiation and deletion. In addition, methods for getting and setting attributes within each class are also provided.

4.1. Block Classes

Block classes form the major blocks of functionality that can be plugged into an abstract card-cage to create various types of devices. One Physical Block is mandatory as it defines the card-cage and abstracts the hardware and software resources that are used by the device. All other blocks and component base classes can be referenced from the Physical Block.

A Physical Block represents the card-cage and contains all the logical hardware and software resources in the model. These resources determine the basic characteristics of the device being assembled. Information contained in the Physical Block as attributes include the manufacturer’s identification, serial number, hardware and software revision information, and more importantly, data structures that provide a repository for other class components. As previously mentioned, the Physical Block is the logical container for all components in the device model; therefore, it must have access to and be able to locate all available resources instantiated by the device.

The data structures provided by the Physical Block house pointers (Instance_ID) to these components thereby offer easy indirect access to them. In order for the Physical Block to resolve address queries from the network (i.e., a remote NCAP requests an attribute from the Physical Block), a hierarchical naming/addressing scheme based on unique Tags (ASCII descriptions of the block or component name) that can be concatenated together to form fully qualified addresses is used to communicate with the device or device object across the network. The Physical Block is the centralized logical connector or backplane that the other blocks plug into. Therefore, for the Physical Block to find other components in the system, it must provide a Locate method.

The Transducer Block abstracts all the capabilities of each transducer that is physically connected to the NCAP I/O system. During the device configuration phase, the description is read from the hardware device what kind of sensors and actuators are connected to the system. This information is used by the Physical Block to create and configure the related type of transducer block. The Transducer Block includes an I/O device driver style
interface for communication with the hardware. The I/O interface includes methods for reading and writing to the transducer from the application-based Function Block using a standardized interface (i.e., io_read and io_write). The I/O device driver paradigm provides both plug-and-play capability and hot-swap feature for transducers. This means any application written to this interface should work interchangeably with multiple vendor transducers.

In a similar fashion the transducer vendors provide an I/O driver to the network vendors with their product that supports this interface. The driver is integrated with the transducer’s application environment to enable access to their hardware. This approach is identical to the interface found in device drivers for UNIX.

The Function Block equips a transducer device with a skeletal area in which to place application-specific code. The interface does not place any restrictions on how an application is developed. In addition to a State variable (which all block classes maintain), the Function Block contains several lists of parameters that are typically used to access network-visible data or to make internal data available remotely. That is, any application-specific algorithms or data structures are contained within these blocks to allow separately for integration of application-specific functionality using a portable approach.

The Network Block is used to abstract all access to the network by the block and base classes employing a network-neutral, object-based programming interface. The network model provides an application interaction mechanism based on the remote procedure call (RPC) framework for client-server distributed computing settings. The RPC mechanism supports both a client-server and a publisher-subscriber paradigm for event and message generation. In support of the two types of application interaction, a communication model that stems from the notion of a port is defined in the specification. This means, if a block wishes to communicate with any other block in the device or across the network, it must first create a port that logically binds the block to the port name. Once enough information about the addressing of the port is known, the port can be bound to a network-specific block address. At this point the logical port address has been bound to the actual destination address by the underlying control network technology. Any transducer application’s use of the port name is now resolved to the endpoint associated with the logical destination. This scheme allows a late binding effect on application uses of the ports so that addresses are not hard-coded or dependent upon a specific architecture. The port capability is similar to the TCP/IP socket programming interface where a socket is created and bound using an application specific port number and IP address. Once bound, the socket can be used for data transfer.

### 4.2 Base Classes

Base classes represent the basic building blocks used by the block classes. They are generally used within block classes to provide application functionality. The base classes include: Actions, Events, Parameters, and Files.

Actions provide a model for control interactions between the various block classes that define a system. Essentially, all actions are called using an Invoke method and may be either blocking or nonblocking in their communication of the action.

Events model the generation of asynchronous communication of signals in the system. That is, if an application wishes to have a certain occurrence of something to happen at a given time in the system, then the designer simply creates an event with a certain time period. The underlying event generation and control mechanisms provided by the network will be used to support this capability.

The Parameter class represents network-visible variables in the model. Parameters have two methods associated with this class for reading and writing to these network accessible data storage locations. Parameters are typically found in the Function blocks to give access to network variables to executing applications.

Files provide a means for applications to up and download information to the device. The kinds of transfers of information are not specified while only either byte or record-oriented data streams are considered. The specification defines a minimal file transfer state machine.

### 5. Case Study

The case study deals with the real application of a pressure analyser consisting of a group of smart pressure sensors interconnected by the Ethernet-based Intranet with supporting nodes, enabling also to join to components on various fieldbuses, see Fig. 3. That application respects both of the above introduced frameworks: while the architecture applied is based on the IEEE 1451 framework, it enables also to include interconnected fieldbuses.

Each smart sensor contains one of two NCAP implementations based either on 16-bit ARM 7 or on 8-bit Rabbit microprocessors that provide both realization of the 1451.1 object model for Intranet interconnectivity and the access to the 1451.2 interface for joining pressure sensors with reflected laser beam and diffractive lens. Read-out sensing system applies a combination of two principles. Large displacements are measured by the position of the reflected focused laser beam. Small position changes are measured by one-side layer diffractive lens principle. Sensor output signal is conditioned in digital by ADuC812 one-chip microcomputer, which provides the IEEE1451.2 interface as one of its communication ports.
The networking solution is a Java-based architecture with Web server in the role of a half gateway. The Java applet represents the client-side of the client-server communications model. The software half gateway provides the key server-side capabilities allowing Java clients to connect, subscribe, and communicate to the smart sensors. Java can directly support client-server application architectures as the core Java specification includes a TCP/IP networking API. The developed Java applet uses the core java.net package to implement the client-server application distribution. This allows the Java applet client to access the smart sensors and supporting nodes. The Java applet consists of a series of object classes, including multi-threaded applet environment, animation, and TCP/IP-based network client communications.

6. Related Work

The research group at the NIST developed demonstration architectures for direct coupling 1451-compatible devices to fieldbuses or Internet-based Intranets [LS96], [SL97], [Sc99].

7. Conclusions

This paper presents, in frame of the pressure analyzer case study, a real-world application that can offer a design pattern for 1451 sensors-based embedded systems.

Acknowledgments

The authors appreciate contributions of their colleagues from the Department of Computer Science and Engineering, the Department of Microelectronics, and the Department of Automatic Control and Measurement Techniques at the Brno University of Technology, namely Vladimir Musil, Frantisek Zezulka and Petr Benes, to the presented work.

This research is supported by Grand Agency of the Czech Republic under of the contract GACR 102/00/0938 TLAKAN and by the Czech Ministry for Education in frame of the Research Intentions JC MSM 262200012: Research in information and control systems, and JA MSM 262200011: Research in electronic communication systems and technologies. The presented work is sponsored also by the Faculty of Electrical Engineering and Computer Science of the Brno University of Technology.

References


Fig. 3. Pressure analyser