Configuration and Management of a Real-Time Smart Transducer Network

Stefan Pitzek
Institut für Technische Informatik
Technische Universität Wien
Karlsplatz 13, Vienna, Austria
pitzek@vmars.tuwien.ac.at

Wilfried Elmenreich
Institut für Technische Informatik
Technische Universität Wien
Karlsplatz 13, Vienna, Austria
wil@vmars.tuwien.ac.at

October 20, 2003

Abstract

Smart transducer technology supports the composability, configurability, and maintainability of sensor/actuator networks. The configuration and management of such networks, when carried out manually, is an expensive and error-prone task. Therefore, many existing fieldbus systems provide means of "plug-and-play" that assist the user in these tasks.

In this paper we describe configuration and management aspects in the area of dependable real-time fieldbus systems. We propose a configuration and management framework for a low-cost real-time fieldbus network. The framework uses formal XML descriptions to describe node and network properties in order to enable a data representation that comes with low overhead on nodes and enables the easy integration with software tools. The framework builds on the infrastructure and interfaces defined in the OMG smart transducers interface standard. As a case study, we have implemented a TTP/A configuration tool operating on a basic framework using the described concepts and mechanisms.

1 Introduction

A fieldbus is a distributed system of nodes connected via a shared digital communication line. Fieldbus applications cover different domains (e. g., home automation, factory automation, car body electronics) and are used to fulfill different tasks (e. g., monitoring of sensors, controlling actuators, control feedback loops).

Regardless of the application, the configuration and management of modern digital fieldbus systems are very complex tasks. In order to support users and implementors of fieldbus systems in dealing with this complexity a variety of support mechanisms is necessary. This support infrastructure generally consists of hard- and software tools together with formal abstractions for relevant system properties. An important target of such an infrastructure is the support of the many different tasks and processes during implementation, setup, and runtime of a system, such as system configuration, application creation, diagnosis, or maintenance. The quality of the support infrastructure is a major factor for the acceptance and success of a system.

In this paper we want to present a configuration and management framework for the OMG smart transducers interface standard. This standard was ratified by the Object Management Group (OMG) in January 2003 [1]. The smart transducers interface specification defines a distributed smart transducer system with an in-system configuration service, a hard real-time service, and a diagnosis and management service that can be accessed concurrently to the real-time service. Our proposed framework models the configuration information in formally specified XML structures called *descriptions*. This formal description of fieldbus properties is the basis for the implementation of generic and interoperable software tools.

The paper is structured as follows: Section 2 will present some related work on configuration and management of fieldbus systems. Section 3 states requirements at the fieldbus level and requirements on configuration and management. Section 4 presents the system architecture of the proposed framework for the smart transducers interface standard. In section 5 a case study implementation of the proposed architecture is presented. Section 6 compares our approach to the IEEE 1451 smart transducer standard, whereas section 7 summarizes the main ideas of this paper and gives an outlook on future work.

2 Related work

Many current fieldbus systems come with some kind of support infrastructure. For example, the support framework of the Foundation Fieldbus [2] provides predefined function blocks for creating applications and a uniform access model for nodes with virtual field devices. Field devices are described by using a variant of the device description language (DDL). The DDL has first been used for the HART bus and later has also been adopted for the Profibus. This development finally culminated in the standardization of the Electronic Device Description Language (EDDL) in the IEC 61804-2 standard [3]. LonWorks [4] and the European Installation Bus (EIB) [5] both come with detailed specifications called functional profiles (LonWorks), respectively functions (EIB) for typical services from their application domain (home and building automation). Other protocols supporting a similar approach are the Profibus [6] and CANopen [7]. Profiles are usually not interchangeable between different fieldbuses, but there are some efforts for defining common profiles for easing the transition to a future common IEC fieldbus standard, which is currently in development [8].

The IEEE 1451 standards define a general smart transducer interface that is independent of any particular physical protocol. The sub-standard IEEE 1451.2 defines a digital interface for the connection of transducers to a microcontroller and proposes a universal description of devices by specifying so-called transducer electronic data sheets (TEDS) [9]. IEEE 1451 also includes definitions for a general application model (IEEE 1451.1) [10], definitions of digital communication and TEDS formats for distributed multi-drop systems (IEEE P1451.3), and mixed-mode communication protocols and TEDS formats (IEEE P1451.4) [11]. IEEE 1451 covers many important aspects of smart transducer networks, but for the sake of interoperability it also places rather rigid requirements on the designers of fieldbus devices.

The *Local Interconnect Network (LIN)* [12] is an example for a fieldbus following a completely different design approach. *LIN* bus [12] has a very tight integration with (and depends on) an external support infrastructure (i. e., software tools), which leads to a very low management overhead on the nodes, which is important for low-cost applications.

3 Architectural Requirements

In this section we define the requirements for the proposed configuration and management framework. First we list some requirements of the communication service that are given by the final fieldbus application. Thereafter, we discuss the requirements on a configuration and management framework, which must be established without affecting the application requirements.

3.1 Application Requirements

This section subsumes requirements on embedded real-time applications.

- *Real-Time Communication:* Many embedded systems feature control algorithms that need periodical updates of control values where the update times must have low jitter. In a distributed architecture the update of measurement values should be performed deterministically within a bounded time.

- *Composability:* In order to reduce complexity it is often necessary to split a system into interacting subsystems, that can be implemented and tested separately. Such a *partitioning* of a system can be applied to hard- and software [13]. To put these subsystems to work, they must be able to communicate with each other. The design of the communication interfaces is a critical task since the borderlines between subsystems have to be well-defined interfaces to enable *composability*, i. e., if each subsystem implements well-defined interfaces in the temporal and value domain, it can be *a priori* guaranteed that the subsystem provides its specified service also in the composite system.

- *Reusability:* Software reuse and the integration of legacy systems, i. e., autonomous systems that have been developed according to their own rules and conventions [14], can often become an important factor. If a legacy system does not provide an appropriate interface it may violate the composability principle. In this case, the introduction of an extra component, an *interface system*, may resolve this mismatch. The interface system will then negotiate between the legacy subsystem and the other system parts. Since the introduction of this interface system incurs an additional overhead on the system, care must be taken not to violate the real-time requirements of the system.

- *Limited resources:* Embedded systems usually provide only a very limited amount of memory and computational power. For example, the usage of software architectures like Java and Jini in the fieldbus devices, as proposed in [15], will not be

possible for most applications. Therefore, we have to find ways to minimize the resource requirements at the fieldbus devices.

3.2 Requirements on Configuration and Management

The requirements on a configuration and management framework are driven by different factors than the above stated application requirements. We have identified the following points:

- *(Semi-)Automatic configuration:* The requirement for a plug-and-play-like configuration can be justified by three arguments: First, an automatic or semi-automatic configuration saves time and therefore leads to better maintainability and lower costs. Second, the necessary qualification of the person who sets up the system may be low if the overall system is easier to configure. Third, the number of configuration faults will decrease, since monotone and error-prone tasks like looking up configuration parameters in heavy manuals are done by the computer.

  A fully automatic configuration will in most cases only be possible if the functionality of the system is reduced to a manageable subset. For more complex applications consulting a human mind is unavoidable. Thus, we distinguish two use cases, (i) the automatic set-up of simple subsystems and the (ii) computer-supported configuration of large distributed systems. The first use case mostly deals with systems that require an *automatic* and *autonomous* (i. e., without human intervention) reconfiguration of network and communication participants in order to adapt to different operating environments. Usually, such systems either use very sophisticated (and often costly) negotiation protocols or work only on closely-bounded and well-known application domains. Creating low-cost fieldbus systems that provide such functionality is an interesting research topic on its own; but in this paper we will primarily focus on the more general second use case that focuses on the cooperation of user and support tools.

- *Comprehensible interfaces:* In order to minimize errors, all interfaces will be made as comprehensible as possible. This includes a uniform representation of data at the various interfaces and a reduction of an interface to the type of data that is necessary for the user of an interface. The comprehensibility of an interface can be expressed by the *mental load* that is put onto the user that uses the interface. Different user types will need different specialized interfaces, each with a minimum of mental load. For example, an application developer will mostly have a service-centered view of the system. Physical network details and other properties not relevant for the application should be hidden from the developer [16].

- *Uniform data structures:* The configuration and management of fieldbus systems requires representations of system properties that are useable by software tools. In order to avoid a situation where each application deals with the required information in its own way, we require a generic, highly structured, and exactly specified representation of system properties. This representation is then used by generic and/or specific tools.

- *Low overhead on embedded system:* In contrast to the embedded software of network nodes, most configuration and management tools will execute on a standard desktop or laptop computer running standard operating systems such as Windows or Linux. Such systems provide far more resources of memory and processing power than the average embedded system. The design of a configuration and management tool must take care that there is as little overhead on the embedded system nodes as possible.

- *Use of standard software/hardware:* Computers running standard Windows or Linux operating systems do not provide guaranteed response times for programs and most hardware interfaces are controlled by the operating system. Since this might violate the special timing requirements of a fieldbus protocol it is often not possible to directly connect a configuration host computer to the fieldbus network using the fieldbus protocol itself. Instead a configuration tool must use some other means of communication, such as standard communication protocols or interfaces like TCP/IP, RS232, USB or standard middleware like CORBA. Since fieldbus nodes might not be powerful enough to implement these mechanisms, communication will often occur over dedicated gateway nodes. In order to reduce the complexity of the involved conversion and transformation steps, the interface to and from the fieldbus node must be comprehensible, structurally simple and easy to access.

  In order to minimize the effort for using a tool, it should work without further extensions to the system, e. g., as an applet in a web browser.

4 System Architecture

In this section we will present a support infrastructure for the TTP/A fieldbus system. The TTP/A fieldbus is standardized *smart transducers interface standard* [1] by the OMG.

In the following sections we will present the various parts of the architecture in greater detail.

4.1 Fieldbus Communication Protocol

The information transfer between a smart transducer and its communication partners is achieved by sharing information that is contained in an internal interface file system (IFS), which is situated 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 [17]. 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.

A time-triggered sensor bus will perform a periodical time-triggered communication by sending data from IFS addresses to the fieldbus and writing received data to IFS addresses at predefined points in time. 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.

Communication is organized into rounds consisting of several TDMA (Time Division Multiple Access) 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 and is a reference signal for clock synchronization. The protocol supports eight different firework bytes encoded in a message of one byte using a redundant bit code [18] supporting error detection.

Generally, there are two types of rounds:

1. *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 1.

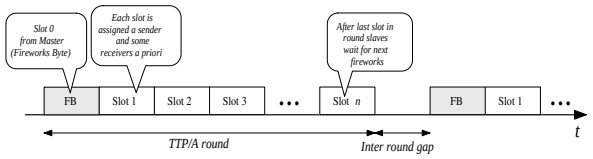

Figure 1: TTP/A Multipartner Round

2. *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 using a hierarchical IFS address and specifies an action like reading of, writing on, or executing that record.

The multipartner (MP) round establishes a real-time communication service with predefined access patterns. Master/slave (MS) rounds are scheduled periodically between multipartner rounds, whereas the most commonly used scheduling scheme consists of MP rounds alternating with MS rounds. The MS rounds allow maintenance and monitoring activities during system operation without a probe effect. The master/slave rounds enable random access to the IFS of all nodes, which is required for establishing two conceptual interfaces to each node, a *configuration and planning* (CP) interface and a *diagnosis and management* DM interface. These interfaces are used by remote tools to configure node and cluster properties and to obtain internal information from nodes for diagnosis.

4.2 Descriptions for system properties

Formal descriptions of system properties are an important part of the support infrastructure. We subsume the formal representations of system properties under the term *descriptions*. For a uniform representation of all system aspects, we have chosen *XML* [19] as the primary representation mechanism. Together with related standards, such as XML Schema or XSLT, it provides advanced structuring, description, representation, and transformation capabilities. XML enjoys extensive support throughout the industry, concerning its use as a description language as well as the support for processing.

XML already has been used for several applications in the fieldbus domain [20, 21, 22].

4.3 Description Meta-Information

Due to the uniform representation of the descriptions, it is possible to integrate *meta-elements* into the descriptions. Meta-elements are valid sub-elements for all non-simple description elements (i. e., elements that may contain subelements). The content of the meta-elements may be any valid element-subtree. We distinguish three different kinds of description meta-elements:

1. *Documentation:* Documentation elements contain additional information for a particular element. In many cases this will be a simple text string that acts as a comment for the content of an element.

2. *Meta information:* Meta information elements contain meta-data from software tools. These elements provide a well-defined space so that different tools can store their data without conflicting with the primary description content or with each other. By reducing the places where information concerning the system is stored, it becomes easier to keep the external representation synchronized. The structure of the data within the meta-elements and how it is processed is in the responsibility of the respective tool.

3. *Views:* Views allow the definition of multiple interfaces to an object [23]. We apply this notion to the elements in the descriptions in a very general way. A typical use for views is the definition of multiple different style sheets (e. g., defined with XSL) for the transformation of an element's content for different accessing devices.

Figure 2 shows an exemplary use of the meta information element. Each `MetaInformation` element requires an accessor identifier, which uniquely identifies the software tool that is responsible for the content of the element. In the given example, a *Configuration and Planning Tool* (see section 5) stores the color and the short name for representing a node in the graphical user interface.

4.4 Smart Transducer Description

As the name implies, the *smart transducer descriptions* (STD) describe the properties of nodes in the smart transducer network. We distinguish two types of STDs:

1. *Static STDs* describe the node properties of a particular field device family. Static STDs contain node properties that are fixed at node creation time and act as a documentation of the nodes' features. In addition, they are also used as a template for the definition of dynamic STDs.

2. *Dynamic STDs* describe individual nodes, as they are used in a particular fieldbus application. Besides the information from the corresponding static STD, these descriptions include dynamic properties, such as configuration values or the logical (local) name of a node. [24]

```
<ClusterSTDRef ref="1" nodeid="0000000000010000">
  <MetaInformation
    AccessorID="VENUS:VMARS:TUWIEN:AC:AT:CPTOOL">
    <nodecolor>140140220</nodecolor>
    <shortname>MASTE</shortname>
  </MetaInformation>
            .
            .
</ClusterSTDRef>
```

Figure 2: Example for description meta information

The properties described in STDs can be divided into the following categories:

- *Microcontroller information:* This block holds information on the microcontroller and clock of the smart transducer (e. g., controller vendor, clock frequency, clock drift).

- *Node information:* This block describes properties that are specific for a particular node and mostly consist of identification information, such as vendor name, device name/version, and node identifiers (serial number, local name).

- *Protocol information:* This block holds protocol specific information, such as version of the communication protocol, supported baud rates, UART types, and IFS layout.

- *Node service information:* The information in this block specifies the behavior and the capabilities of a node. In the current approach, a service is a basic representation of a functional unit (similar to functional profiles [25]). Such functional units are especially important for supporting the creation of applications, since they can be considered as the building blocks of an application model. Services consist of a service identifier (e. g., name), in- and output parameters, configuration parameters, and monitoring parameters. Parameters are specified

by data-type and multiple constraints (range, precision, minimum interval time, maximum run time).

Figure 3 shows the description of a file in the IFS, consisting of the name of the file, its length (in records), and the location of the data (e. g., `data` specifies that a file is mapped into the internal RAM of the microcontroller). The prefix `rodl:` is a shorthand for an XML namespace. Namespaces allow the reuse of element definitions in multiple places. For example, the elements from the `rodl` (round descriptor list) namespace are once defined separately and used in smart transducer descriptions as well as in cluster configuration descriptions.

```
<SmartTransducerDescription xmlns="...>
     ...
  <ProtocolBlock>
       ...
    <IFSFile>
      <rodl:fileName>16</rodl:fileName>
      <FileLength>12</FileLength>
      <FileStorage>data</FileStorage>
    </IFSFile>
       ...
  </ProtocolBlock>
</SmartTransducerDescription>
```

Figure 3: Example STD element

4.5 Cluster Configuration Description

The cluster configuration description (CCD) contains descriptions of all relevant properties of a fieldbus cluster. It acts as the central structure for holding the meta-information of a cluster. With help of a software tool capable of accessing the devices in a smart transducer network it is possible to configure a cluster with the information stored in the CCD. A CCD consists of the following parts:

- *Cluster description meta information:* This block holds information on the cluster description itself, such as the maintainer, name of the description file, or the version of the CCD format itself.
- *Communication configuration information:* This information includes round sequence lists as well as round descriptor lists, which represent the detailed specification of the communication behavior of the cluster. Other properties important for communication include the UART specification and minimum/maximum signal run times.
- *Cluster node information:* This block contains information on the nodes in a cluster. These nodes are represented either by a list of dynamic STDs or by references to static STDs.

Figure 4 outlines a fragment of the communication configuration block in the CCD. Most of the elements should be self-explanatory. The `record` entry in the `RODLFormatVersion` element specifies the type of the RODL entry format, which in the standard case uses one record per entry. For nodes with very low resources of RAM memory there is a more compact but less flexible short format that uses one byte per entry. The `InterframeGapLength` designates the length of silence (in bit cells) on the bus between successive frames.

```
<ClusterConfigurationDescription xmlns="...>
    ...
  <ConfigurationBlock>
    <ClusterRODLBlock>
      <RODLFormatVersion>
        record
      </RODLFormatVersion>
      <rodl:rodl name="0">
          ...
      </rodl:rodl>
        ...
    </ClusterRODLBlock>
    <BaudRate>9600</BaudRate>
    <BusInterface>ISO-K</BusInterface>
    <UARTFormat>
      <ParityBits>1</ParityBits>
      <StopBits>1</StopBits>
    </UARTFormat>
    <InterframeGapLength>2</InterframeGapLength>
  </ConfigurationBlock>
</ClusterConfigurationDescription>
```

Figure 4: Example fragment from CCD

5 Case Study

As a first example for the proposed concepts we implemented a case study consisting of various description mechanisms and a set of software tools for TTP/A networks (see figure 5 for an overview on the overall system). Communication between the components is performed with CORBA (Common Object Request Broker Architecture) [26], whereas the smart transducer network is accessed via the interfaces specified in the OMG

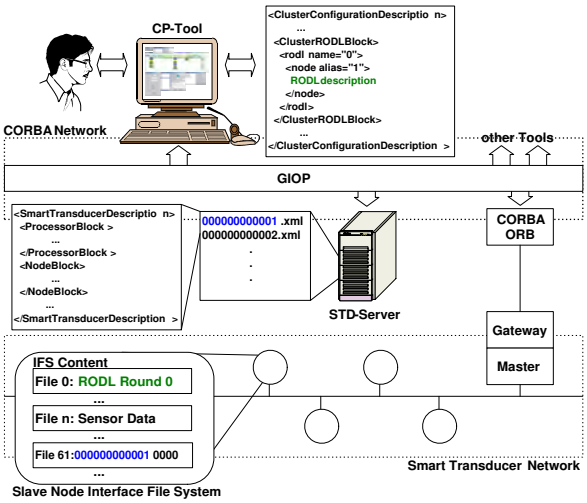

Figure 5: Overview on the structure of the case study

standard for smart transducer interfaces. The central parts of the support framework are:

- *Configuration and planning tool (CP-tool):* This tool uses the CP interface to access the cluster. The tool allows the visual creation and manipulation of communication schedules for the underlying smart transducer network. Figure 6 gives a screen-shot of the tool. The large window contains a representation of the schedule, whereas each column consists of the nodes that send, receive, or execute in one slot. The other windows contain the round list, a node list, the detail view of a single operation, and a window that holds cluster-specific information. The visual representation is created from the RODL-descriptions, either extracted from the CCD or by collecting the local schedules on the nodes in the cluster, enhanced with meta-information gained from the static STDs stored on the STD-server [24] and additional properties from the CCD.

- *Smart transducer description server:* CORBA server that stores the actual XML-representations of the static STDs [24]. During retrieval the data sheets are identified via the series part of the serial number, which is stored in a mandatory IFS file on each node.

- *Corba ORB gateway:* Gateway node that connects the fieldbus network to the CORBA network [27].

- *Smart transducer network:* The smart transducer network used for the case study is an autonomous mobile robot, *the smart car* [28], that is instrumented by a TTP/A network. The robot uses the perception from several infrared and ultrasonic sensors in order to perform navigation around objects.

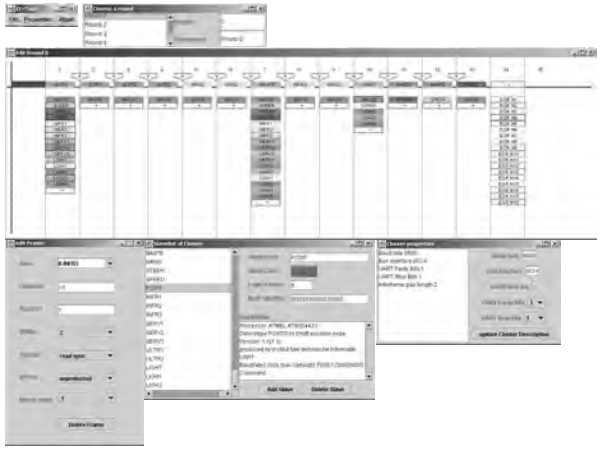

Figure 6: Screenshot of the configuration and planning tool

Primary focus of the case study has been the support of network configuration by providing semi-automatic creation of time-triggered communication schedules. The manual creation of time-triggered communication schedules has shown to be rather tedious and error-prone and the tool helps in the creation, analysis, and modification of schedules with consistent results. Currently this support consists of visualizing the schedule and performing early integrity checks on the validity of changes to the schedule. Fully automated schedule creation will be dealt with in the future. Other areas of the support infrastructure (e. g., tools for creating applications on a higher level of abstraction) received less attention in this particular case. Nonetheless the existing implementations of the descriptions have been developed with such future use in mind.

6 Discussion

The proposed architecture aims at a tight integration of the tool framework with the embedded network, i. e., the tools depend on the infrastructure with the smart transducer description CORBA server and store information about nodes and clusters outside the fieldbus cluster.

For example, imagine a dealer's garage making maintenance at the TTP/A system of a car. The tools of the maintenance personal would not work without access to the dynamic descriptions and access to the CORBA

smart transducer description server. A similar tight integration of external tools with the fieldbus cluster is also realized in LIN networks. In contrast, the IEEE 1451 standard aims at self-contained nodes that store all their dynamic information in persistent memory locally at the smart transducer. These oppositional philosophies lead to different design decisions. Storing all the configuration information on the embedded nodes can cause significant overhead at the smart transducer. In order to keep that overhead as small as possible, the description format for such a system must be very compact. IEEE 1451 achieves this goal by providing a large set of predefined transducer types and modes. Thus, while the memory requirements for minimal implementations of IEEE 1451.2 standard can be quite low (in the order of few hundred bytes [29]) using extended data sheets is out of the question for such basic devices. On the other hand, the STD and CCD descriptions for the TTP/A protocol are very generic and need much more memory. For example, the uncompressed CCD for the smart car case study had a size of 124 Kbytes. However, this information is stored at a desktop computer, while the smart transducer nodes contain only some configuration parameters that are necessary for operation and a unique identification number. The unique identification number serves as reference to the according information that is stored as STDs and CCD outside the cluster. This approach comes with two advantages:

- First, the overhead at the nodes is very low. Current low-cost microcontrollers provide RAM or EPROM memory of around 128 bytes. This will not suffice to store more than the most basic parts of data-sheets according to the IEEE 1451.2 standard without extra hardware. In our architecture, only the ROM memory for storing the identification number is necessary. The framework only indirectly causes an overhead on the nodes since it depends on some advanced features of the TTP/A protocol (e. g., node baptizing), which need not be implemented for every TTP/A node. Trödhandl presents some results on the implementation effort of the protocol and these features in [30].

- Second, instead of implicitly representing the node-information with many predefined data structures mapped to a compact format, we have an explicit representation of the information in a well-structured and easy to understand way. Since the system hosting the configuration tool usually provides memory for several megabytes of data, we can use generic XML constructs for the specification of transducers and services, which improves the openness of the system for future extensions of transducer or service types.

Naturally, for actual production systems the external descriptions must be stored in a reliable way (like other valuable electronic documents). Availability of the external descriptions might actually be a greater problem than reliability, especially if the external descriptions are distributed on a world-wide scale. It should be noted that the actual fieldbus application executing in a fieldbus network (the real-time service) does *not* depend on the presence of the external representations

7 Conclusion and Outlook

In this paper we presented a support infrastructure for the configuration and management of the low-cost real-time smart transducer network TTP/A. We introduced a system model with narrow interfaces with a special focus on structure and representation of (meta-)information on system properties. Information in the proposed infrastructure is uniformly represented with XML and stored in a way that incurs minimal overhead on nodes. The proposed support infrastructure consists of descriptions for smart transducer nodes and cluster configuration, as well as software tools for configuring the network.

Special care has been taken not to influence the real-time behavior of the underlying fieldbus protocol by designing the system around the appropriate interfaces of the OMG smart transducers interface standard. The presented tools depend on the infrastructure with the smart transducer description CORBA server and store information about nodes and clusters externally. This has been a design decision in order to achieve a small overhead on the smart transducer nodes and to keep the system open to new transducer types and services. Since the tools only use the Configuration and Planning (CP) interface of the OMG STI standard, it is guaranteed that the real-time services are not influenced by the configuration process. Since configuration and planning are not considered time-critical, no additional assumptions on the real-time behavior of the cluster are made.

The presented case study integrates the descriptions and corresponding software tools together with a smart transducer network into an overall fieldbus configuration infrastructure.

In the future we will focus on further parts of the configuration and management framework, such as the representation of a layered system model supporting the separation of sensor fusion and fault tolerance from the application as proposed in [31].

8 Acknowledgments

This work was supported in part by the Hochschuljubiläumsstiftung der Stadt Wien via project CoMa (H-965/2002) and by the European IST project DSoS under contract No IST-1999-11585.

References

[1] Smart transducers interface v1.0, January 2003. Available: http://doc.omg.org/formal/2003-01-01 as document ptc/2002-10-02.

[2] Fieldbus technical overview - understanding FOUNDATION fieldbus technology. available at http://www.fieldbus.org/.

[3] CENELEC: Function blocks for process control - part 2: Specification of fb concept and eddl. IEC/-PAS 61804-1, ed. 1, August 2002.

[4] Control network specification. eia standard 709.1, March 1998.

[5] European Installation Bus Association (EIBA). *EIBA Handbook Series - Release 3.0*, 1999. Available: http://www.eiba.com.

[6] CENELEC: General purpose field communication system. standard EN 50170, vol. 2/3 (Profibus), December 1996.

[7] CAN in Automation e.V. Canopen - communication profile for industrial systems. available at http://www.can-cia.de/downloads/.

[8] M. Felser. The fieldbus standards: History and structures. *Technology Leadership Day 2002, Organised by MICROSWISS Network*, October 2002.

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

[10] J. Warrior. IEEE P1451 network capable application processor information model. In *Proc. Sensors Expo Anaheim*, pages 15–21, April 1996.

[11] S.Chen and K.Lee. A mixed-mode smart transducer interface for sensors and actuators. *Sound & Vibration*, April 1998.

[12] Audi AG, BMW AG, DaimlerChrysler AG, Motorola Inc. Volcano Communication Technologies AB, Volkswagen AG, and Volvo Car Corporation. LIN specification and LIN press announcement. SAE World Congress Detroit, http://www.lin-subbus.org, 1999.

[13] W. H. Wolf. Hardware-software co-design of embedded systems. *Proceedings of the IEEE*, 82(7):967–989, July 1994.

[14] C. Jones, M.-O. Killijian, H. Kopetz, E. Marsden, N. Moffat, D. Powell, B. Randell, A. Romanovsky, R. Stroud, and V. Issarny. Final version of the DSoS conceptual model. *DSoS Project (IST-1999-11585) Deliverable CSDA1*, October 2002. Available as Research Report 54/2002 at http://www.vmars.tuwien.ac.at.

[15] W. Kastner and C. Krugel. A new approach for Java in embedded networks. In *Proc. 3rd International Workshop on Factory Communication Systems*, pages 19–26, 2000.

[16] S. Pitzek and W. Elmenreich. Managing fieldbus systems. In *Proceedings of the Work-in-Progress Session of the 14th Euromicro International Conference*, June 2002.

[17] H. Kopetz, M. Holzmann, and W. Elmenreich. A universal smart transducer interface: TTP/A. *International Journal of Computer System Science & Engineering*, 16(2):71–77, March 2001.

[18] 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, Vienna, Austria, 2000.

[19] Extensible markup language (XML) 1.0 (second edition), October 2000. Available: http://www.w3.org.

[20] M. Wollschlaeger. A framework for fieldbus management using XML descriptions. In *Proc. IEEE International Workshop on Factory Communication Systems 2000 (WFCS 2000)*, pages 3–10, September 2000.

[21] S. Eberle. XML-basierte Internetanbindung technischer Prozesse. In *Informatik 2000 Neue Horizonte im neuen Jahrhundert*, pages 356–371. Springer-Verlag, Berlin Heidelberg, September 2000.

[22] Dieter Bühler. The CANopen Markup Language – Representing fieldbus data with XML. In *Proc. 26th IEEE International Conference of the IEEE Industrial Electronics Society (IECON 2000)*, Nagoya, Japan, October 2000. IEEE.

[23] B. Hailpern and H. Ossher. Extending objects to support multiple interfaces and access control. In *IEEE Trans. Software Eng.*, volume 16, pages 1247–1257, November 1990.

[24] S. Pitzek. Description mechanisms supporting the configuration and management of TTP/A fieldbus systems. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[25] D. Loy, D. Dietrich, and H.-J. Schweinzer (Eds.). *OPEN CONTROL NETWORKS*. Kluwer Academic Publishing, Oct. 2001.

[26] The common object request broker: Architecture and specification - revision 2.6.1, May 2002.

[27] W. Elmenreich, C. Trödhandl, and T. Losert. An embedded system hosting a CORBA and TTP/A service. Research Report 17/2001, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2001.

[28] W. Elmenreich, W. Haidinger, H. Kopetz, T. Losert, R. Obermaisser, M. Paulitsch, and C. Trödhandl. Initial demonstration of smart sensor case study. *DSoS Project (IST-1999-11585) Deliverable PCE3*, April 2002.

[29] P. Conway, D. Heffernan, B. O'Mara, D. P. Burton, and T. Miao. IEEE 1451.2: An interpretation and example implementation. In *Proc. IEEE Instrumentation and Measurement Technology Conference*, Baltimore, USA, May 2000.

[30] C. Trödhandl. Architectural requirements for TTP/A nodes. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[31] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.