Managing fieldbus systems

S. Pitzek
Institut für Technische Informatik
Technische Universität Wien, Austria
pitzek@vmars.tuwien.ac.at

W. Elmenreich
Institut für Technische Informatik
Technische Universität Wien, Austria
wil@vmars.tuwien.ac.at

Abstract

Digital fieldbus systems are increasingly becoming popular. The configuration and management of such a system, mainly carried out manually, is an expensive and error-prone task. Many existing fieldbusses provide means like "plug-and-play" that assist the user in these tasks. However the existing solutions cover merely parts of the possible configuration and maintenance tasks.

In this paper we will describe configuration and management aspects in the area of dependable real-time fieldbus systems. First we identify means for improving fieldbus management. In order to reduce interface complexity we introduce different user-dependent views.

In the following we discuss ways for organizing the management information. We will examine existing solutions from other fieldbusses and discuss their suitability for our purpose.

We plan to use the described concepts and mechanism in a tool-set for the time-triggered fieldbus TTP/A.

1. Introduction

Using digital fieldbus technology is increasingly becoming popular for a wide area of applications. From factory instrumentation to car body electronics many competing protocols promise safer and less expensive solutions that are easier to develop and maintain compared to the former technologies applied in the respective sectors (like point-to-point wiring of analog sensor and actuator devices, discrete logic, ...).

Nonetheless, the configuration and management of these fieldbus networks remains a complex task. Sensor devices and micro-controller of different architectures and from different vendors have to interact via fieldbus protocols to form the distributed application of the network.

To ease the management of fieldbus systems all major protocols provide mechanisms, formalisms and tools that support this process.

Most of these approaches were developed independent of each other and specifically tailored for the respective fieldbus system and its technical attributes. Most notable exceptions are *CAN Kingdom* [1], *OSEK* [2], and IEEE 1451 [3] in their target of being in principle independent of the underlying physical fieldbus.

But the central problem remains in that there is no single model that tries to cover all aspects of the management process of a fieldbus system.

A general management model would have to address the same problems most vendor-specific solutions were developed for, in that it should lead to a less expensive system that is faster deployed and contains as few errors as possible. In addition it should be largely independent of the underlying fieldbus protocol and preferably easy to understand.

In this paper we will examine aspects of the management process. We will look at ways how to improve this management process and the requirements on the infrastructure of a fieldbus to achieve these improvements. We will also examine how a distributed configuration approach matches with current mechanisms for configuration and management.

The paper is structured as follows. In section 2 we will identify general methods on how to improve the management process. Section 3 deals with the infrastructure required for these improvements. In section 4 we present a summary of this paper while section 5 contains an outlook on our further plans.

2. Ways to Improve Fieldbus Management

To get knowledge about what infrastructure is required for successful management of a fieldbus we first have to identify central approaches for improvements of the management process.

2.1. Automation of Operations

Automating re-occurring manual tasks usually leads to significant time-saving and can eliminate many potential errors. We distinguish two central application areas for automation.

Plug-and-play Plug-and-play capabilities of a system supersede the manual configuration process that would be required when integrating nodes in a running system, thereby unburdening system integrators and operators. Most commercially available fieldbus systems provide plug-and-play mechanisms to some extent.

Generation of applications and communication schedules If a protocol uses a priori created communication schedules, the creation of these schedules should not

be burdened on the application developer, but instead created automatically by an appropriate software tool.

2.2. Abstraction

Usually in fieldbus systems we have a variety of different nodes, sensors and local node applications. Differences stem from varying hardware architectures, vendors, sensor types and so on. In order to guarantee their inter-operability we have to introduce abstraction mechanisms that hide these inherent differences from the other nodes in the network.

However, in real-time systems too much abstraction is not always appropriate because the real-time properties may be lost. Therefore it is necessary to define interfaces to transducer hardware, that provide information in a structured way in value *and* time domain.

Of course the protocol itself is already such a mechanism, since it provides uniform rules on how nodes have to communicate and in general controls the communication and interaction of the participating nodes in the network.

With that in mind we will focus on abstractions that specifically support an easier and faster system development and management process. Some protocols like *LON* [4] include sophisticated abstraction mechanisms in the protocol. For example all application messages flowing between network nodes in *LON* are transmitted as so-called network variables, which are members of predefined network variable types, thus forming an abstraction over the physical data transport.

2.3. Multiple Views on a System

If different user groups access the system for different purposes they should only be provided with the information relevant for their respective purpose [5]. Following we will discuss different views (with their respective interfaces) a system could provide:

Application development The perception of the network by the application developer depends on the configuration mechanism provided for the protocol but usually is service-centered. When possible (physical) network details and mechanisms that have no direct relevance for an application (fault-tolerance/sensor-fusion) should be hidden from the application developer, e. g. by introducing an intermediate data structure that contains mappings from all sensor measurements of interest. Such a data structure will be independent of the sensor configuration but fully temporally specified in order to maintain real-time behavior [6].

System integration The system integrator decides on the structure of the network and, if required, integrates sensor-fusion and fault-tolerant components in the network. Since the system integrator is responsible for the connection between the physical nodes and the application he requires information about the physical structure of the network, outputs from the sensor/actuator nodes and inputs to the application.

Maintenance Maintenance access usually takes place from a maintenance device that connects to the network and supports the maintenance engineer in diagnosing errors, debugging the system and performing function checks. In order to identify faulty components the maintenance engineer needs exact knowledge about the physical structure of the network.

Monitoring The perception of the network largely depends on the information the monitoring process should yield. Examples range from data throughput analysis on the network, checking individual senors in a fault-tolerant/sensor-fusion unit to examining communication patterns within an application. A system operator might be interested in getting a view of the system that can be adjusted to these different degrees of abstraction.

By providing each accessing user group with tailored interfaces, the amount of information each user has to deal with is reduced significantly. An example for a fieldbus protocol that provides different views on a system is the *Foundation Fieldbus* [7].

After looking at the central ways of improving the management of fieldbus systems we will now examine the infrastructure a fieldbus system has to provide in order to support these mechanisms.

3. Organizing the Management Information

The implementation of the proposed methods depends on a formal description of fieldbus properties. In this section we will talk about these parts of a system that need formalization and how this information is organized.

As a first step let us find the elements of a fieldbus systems that require a formal description.

Physical nodes For every fieldbus protocol there usually is a range of different transducer nodes and sensors, with different properties (memory size, processing power, measuring type, ...), that are part of the same installation. To provide application developers with a uniform view on these nodes we require some way to formalize these properties so that support tools can hide these differences from the user. An example for such a description is the *device description language (DDL)*. A description language that was originally created for the HART-fieldbus [8] but soon was adopted for the *Foundation Fieldbus*.

Local node applications Every transducer node contains

local node applications that provide services to the distributed network application. Examples for such services are transmitting sensor values or controlling actuators. Since many of these local node applications are in principle identical when looking at different fieldbus devices, we can organize them in a structured way, to provide a service-centered view of the system.

Application The overall application of the fieldbus system is distributed over the nodes in the network. Data communicated between different nodes in the network has to be converted to a generally accepted form, to be of use to other nodes. Some protocols introduce data type hierarchies for this kind of data (e.g. network variable types in *LON*).

Protocol specific information Protocol-internal information is usually of no concern for the end-users of fieldbus systems, but essential for node developers that produce devices that have to stick to the standards provided by the target fieldbus protocol. When creating abstraction mechanisms we have to include ways to still be able to access this information.

Fault-tolerance/sensor-fusion For convenience reasons fault-tolerance and sensor fusion mechanism should be transparent for the application developer. But in case a maintenance engineer is accessing the system we still require a representation of the concrete implementation of these mechanisms in the system (number of nodes in a fault tolerant unit, failure modes, ...). To our knowledge there are no fieldbus solutions available that include fault tolerance/sensor fusion mechanisms on the level of services and device descriptions (independent of the application).

Many existing systems integrate predefined fieldbus nodes into the network. Each fieldbus node is assigned a description, whereas this description can be simple human-readable printed data sheets, machine-readable for computer-aided or automated configuration or a combination of both. An example of the last case is the IEEE 1451 standard that defines multiple classes of transducer electronic data sheets (TEDS) and divides them in optional/required and machine-readable/human-readable subclasses.

However, it is assumed, that the information required for the integration of a single component is *static*, i. e. does not change over the lifetime of the system. The configuration data that controls the system is often stored centrally in a master, control node, or gateway node. This data is *dynamic*. An example for an architecture that follows this approach is the LIN-fieldbus [9]. However, actual requirements on real-time communication and fault-tolerance may lead to a shift to distributed systems, where we are confronted with distributed configuration information and the configuration of a single node becomes an important part of the overall system. An example for an architecture with a need for distributed configuration is TTP/A [10], where communication schedules have to be configured consistently in all nodes in order to enable a predictable real-time communication. When such node configurations are changeable in-system, part of the node's configuration information becomes *dynamic*, since the node needs to be synchronized to the new schedules.

In general time-triggered systems require an increased effort in the design phase in comparison to other event-triggered systems. However this brings the advantage of easier verification, predictability and deterministic temporal behavior [11].

With these requirements in mind, we will look at how we can organize the management organization in a way that supports this distributed configuration.

3.1. Distributed Configuration Management

Much of the support functionality for fieldbus management lies in software tools located outside the fieldbus. They provide extensive additional services to support users in their respective tasks.

Thus machine processabilty of management descriptions is a central requirement. In case of distributed configuration this requirement is pronounced, since manually integrating a new node in a running network is a very difficult task which includes the adaption of all node schedules in the network to the new node. We presented a solution for the automatic integration of new nodes (plug-and-play) in a fieldbus system with distributed configuration [12].

In our opinion this distributed configuration fits well to a more general view of the system as a distributed application. While current solutions do provide abstractions for services in the network, there is still a strong focus on individual nodes, since usually a node in the application context is defined by the services it provides. This focus on the network nodes causes some problems. For example, in current solutions we have found no way to include fault-tolerance or sensor-fusion services on a level other than the distributed application. Since such services require the interaction of multiple nodes, we need an abstraction for services that is above the node level.

We introduced this explicit distinction between *static* and *dynamic management information* mainly for the reason of being able to partition the management information in those parts mostly decided at design time and those parts that can change during operation of the system. Up to this point we used this distinction solely in the context of protocol schedules, however we can also extend it to the overall management information of a fieldbus system. In this case the dynamic information set represents the running state of a

system. This information includes the schedules on the nodes, configuration values that can be set at runtime and the nodes dynamic application data (measurements, actuator positions).

On the other hand the set of static information contains those properties that we must not necessarily keep in the cluster. Some approaches like IEEE 1451 require that all the configuration and management information has to be kept on the nodes. We find this requirement overly restrictive since information that is not required for the run-time system (the *static information*) could also be stored outside the network (e.g. in a centralized Internet data base, or on CD-ROM). Since access to fieldbus devices is performed via software tools, these tools can easily retrieve this extended information, if they get a hint from the field device where to find this information (e.g. unique identifier or URL for the location of the device description). For example in case of the *Foundation Fieldbus* (FF) the FF protocol group regularly publishes CD-ROMs with an up-to-date database of device descriptions of supported field devices.

The TTP/A standard already supports this hybrid storage of static and dynamic configuration data. We will focus on the TTP/A fieldbus with the implementation of a formal device description supported by appropriate tools.

4. Summary

With the increasing capabilities of digital fieldbus systems, management and configuration of such systems become more complex and error-prone. Many existing fieldbusses provide means like ”plug-and-play” that assist the user in these tasks. However existing solutions cover merely parts of the possible configuration and maintenance tasks.

In this paper we have described configuration and management aspects in the area of dependable real-time fieldbus systems. In order to reduce interface complexity we introduced different user-dependent views. Furthermore we analyzed ways for organizing the management information, with a special focus of the impact distributed configuration has on the configuration and management process.

5. Future Work

We plan to implement the concepts described in this paper in a smart transducer case study [13] instrumented with the TTP/A fieldbus. We use an autonomous mobile robot that comprises three different types of sensor nodes and four types of actuator nodes. Furthermore there are a master and a control node with a user-defined application. As a proof of concepts we plan to derive formal device descriptions of all nodes in the network. In addition we will implement tools that provide appropriate views for the different user groups.

6. Acknowledgments

We would like to give special thanks to our colleague Wilfried Steiner for constructive comments on an earlier version of this paper. This work was supported in part by the Austrian Ministry of Science, project TTSB and by the European IST project DSoS under contract No IST-1999-11585.

7. References

[1] L.-B. Fredriksson. CanKingdom and dependable CAN systems. available at http://www.cankingdom.org.

[2] BOSCH. OSEK/VXD operating system - version 2.1 revision 1. available at http://www-iiit.etec.uni-karlsruhe.de/ osek/, Dec. 2000.

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

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

[5] A. Ran and J. Xu. Architecting software with interface objects. In *Proceedings of the Eighth Israeli Computer Systems and Software Engineering*, pages 30–37, 1997.

[6] W. Elmenreich and S. Pitzek. The time-triggered sensor fusion model. In *5th IEEE International Conference on Intelligent Engineering Systems (INES), Helsinki-Stockholm*, pages 297–300, Sep. 2001.

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

[8] R. Bowden. *HART - A Technical Overview*. Fisher-Rosemount, Aug. 1997.

[9] 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.

[10] R. Schlatterbeck and W. Elmenreich. TTP/A: A low cost highly efficient time-triggered fieldbus architecture. *SAE World Congress 2001, Detroit, Michigan, USA*, March 2001.

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

[12] W. Elmenreich, W. Haidinger, P. Peti, and L. Schneider. New node integration for master-slave fieldbus networks. In *Proceedings of the 20th IASTED International Conference on Applied Informatics (AI 2002)*, pages 173–178, Feb. 2002.

[13] W. Elmenreich et al. A smart sensor LIF case study: Autonomous mobile robot. *DSoS Project (IST-1999-11585) Deliverable PCE3*, Apr. 2002.