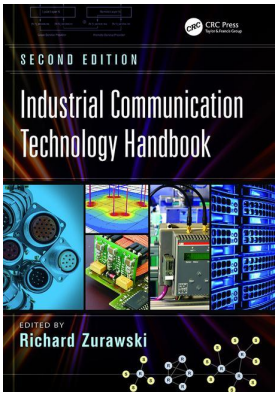

Industrial Communication Technology Handbook

Richard Zurawski

Configuration and Management of Networked Embedded Devices

Publication details
https://test.routledgehandbooks.com/doi/10.1201/b17365-5
Wilfried Elmenreich, Andrea Monacchi
Published online on: 26 Aug 2014

3

Configuration and Management of Networked Embedded Devices

Wilfried Elmenreich
University of Klagenfurt

Andrea Monacchi
University of Klagenfurt

3.1 Introduction

e advent of embedded microcontrollers and the possibility of equipping microcontrollers with small, fast, and low-cost network interfaces has allowed for the creation of smart distributed systems consisting of a set of networked embedded devices. e main reasons for building a system in a distributed manner are the possibilities of having redundancy and exploiting parallelism. In the context of embedded networks, a distributed system also supports the following applications:

- *Distributed sensing* with smart transducers. A smart transducer is the combination of sensor/actuator, processing element, and network interface. Smart transducers perform local preprocessing of the analog sensor signal and transmit the measurement digitally via the network.
- Systems that integrate *legacy systems*, which cannot be extended or changed due to legal or closed system issues. e unchanged legacy system is thus integrated as a subsystem into a network, establishing a larger overall system.
- *Complexity management* by dividing the overall applications into several subsystems with separate hardware and so ware. A distributed network of embedded devices also comes with increased complexity for typical con guration and management tasks such as system setup (no hyphen/space), diagnosis, repair, monitoring, calibration, and change. In order to handle this complexity, computer-automated mechanisms can perform tasks such as registering new devices, auto-con guring data ow, and detecting con guration mismatches.

In this chapter, we examine state-of-the art mechanisms for handling these tasks. A considerable number of mature solutions for con guration and management exist in the context of eldbus systems and service-oriented architectures (SOAs). A eldbus system is a network for industrial manufacturing plants. e eldbus system conducts eldbus nodes comprising sensors, actuators, valves, console lights, switches, and contactors. Challenges for eldbus systems are interoperability, real-time communication, robustness, support for management, and con guration. us, eldbus systems have evolved a set of interesting concepts for supporting setup, con guration, monitoring, and maintenance of embedded devices connected to a eldbus. Twenty years ago, a typical process automation plant consisted of various eld devices from a half-dozen vendors. Each device had its own setup program with di erent syntax for the same semantics. e data from the devices o en di ered in format and in routine to interface each device [1]. Since that time, many eldbus con guration and management methods have been devised.

e following sections are organized as follows: Section 3.2 gives de nitions for the concepts and terms in the context of con guration and management of networked embedded systems. Section 3.3 investigates requirements for con guration and management tasks. Section 3.4 analyzes the necessary interfaces of an intelligent device and proposes meaningful distinctions between interface types. Section 3.5 discusses pro les and other representation mechanisms of system properties for embedded devices. Section 3.6 gives an overview of application development methods and their implications for con guration and management of distributed embedded systems. Section 3.7 examines the initial setup of a system in terms of hardware and so ware con guration. Section 3.8 deals with approaches for system management, such as application download, diagnosis, and calibration of devices. Section 3.9 presents maintenance methods for recon guration, repair, and reintegration of networked devices. Section 3.10 concludes this overview.

3.2 Concepts and Terms

e purpose of this section is to introduce and de ne some important concepts and terms that are used throughout this chapter.

3.2.1 Configuration versus Management

e term *configuration* is used for a wide range of actions. Part of con guration deals with setting up the hardware infrastructure of a network and its nodes, that is, physically connecting nodes (cabling) and con guring nodes in a network (e.g., by using switches and jumpers). Con guration also involves setting up the network on the logical (i.e., so ware) level. erefore, a particular con guration mechanism o en depends on the network issues such as topology or underlying communication paradigm. For example, the TTP-Tools [2] are designed for the time-triggered architecture [3]. TTP-Tools provide various design and con guration mechanisms for engineering dependable real-time systems, but it is assumed that the system is based on a time-triggered paradigm [3].

In contrast to con guration, *management* deals with handling of an established system and includes maintenance, diagnosis, monitoring, and debugging. As with con guration, di erent systems can di er greatly in their support and capabilities in these areas.

Con guration and management are o en di cult to separate because procedures such as plug and play (see Section 3.2.3) involve both con guration and management tasks.

3.2.2 Smart Devices

e term *smart* or *intelligent* device was rst used in this context by Ko and Fung in [4], meaning "a sensor or actuator device that is equipped with a network interface in order to support an easy integration into a distributed control application."

In the context of networked embedded systems, an *intelligent* device supports its con guration and management by providing its data via a well-de ned network interface [5] and/or o ering a self-description of its features. e description usually comes in a machine-readable form (e.g., as an XML description) that resides either locally at the embedded device (e.g., IEEE1451.2 [6]) or at a higher network level, referenced by a series number (e.g., Object Management Group [OMG] Smart Transducer Interface [7]).

3.2.3 Plug and Play versus Plug and Participate

Plug and play describes a feature for automatic integration of a newly connected device into a system without user intervention. Although this feature works well for personal computers within an o ce environment, it is quite di cult to achieve this behavior for automation systems because without user intervention, the system would not be able to ascertain which sensor data should be used and which actuator should be instrumented by a given device. erefore, in the automation domain, the more correct term *plug and participate* should be used, which describes the initial automatable con guration and integration of a new device. For example, a er connecting a new sensor to a network, the sensor could be automatically detected, given a local name, and assigned to a communication slot. e task of a human system integrator is then reduced to deciding on further processing and usage of the sensor data.

3.2.4 State

Zadeh states that " e notion of state of a system at any given time is the information needed to determine the behavior of the system from that time on" [8, p. 3]. In real-time computer systems, we distinguish between the *initialization state* (i-state) and the *history state* (h-state) [9].

e i-state encompasses the static data structure of the computer system, that is, data that are usually located in the static (read-only) memory of the system. e i-state does not change during the execution of a given application, for example, calibration data of a eldbus node. e h-state is the "dynamic data structure […] that undergoes change as the computation progresses" [9, p. 91]. Examples of an h-state are the cached results of a sequence of measurements that are used to calculate the current state of a process variable.

e size of the h-state at a given level of abstraction may vary during execution. A good system design will aim at having a *ground state*, that is, when the size of the h-state becomes zero. In a distributed system, this usually requires all tasks to be inactive and no messages to be in transit.

3.3 Requirements for Configuration and Management

e requirements for a con guration and management framework are driven by several factors. We have identi ed the following points:

- (*Semi*)*automatic configuration*: e requirement for a plug-and-play-like con guration can be justi ed by three arguments: Firstly, automatic or semiautomatic con guration saves time and therefore leads to better maintainability and lower costs. Secondly, the necessary quali cation of the person who sets up the system may be less if the overall system is easier to con gure. irdly, the number of con guration faults will decrease because monotonous and error-prone tasks such as looking up con guration parameters in manuals are done by the computer. In most cases, a fully automatic con guration will be possible only if the functionality of the system is reduced to a manageable subset. For more complex applications, consulting a human mind is unavoidable. We thus distinguish two use cases: (1) the automatic setup of simple subsystems and the (2) computer-supported con guration of large distributed systems. e rst case mainly concerns systems that require an *automatic* and *autonomous* (i.e., without human intervention) recon guration of network and communication participants in order to adapt to di erent operating environments. Usually, such systems either use very sophisticated (and o en costly) negotiation protocols or work only on closely bounded and well-known application domains. e second case is the usual application.
- *Comprehensible interfaces*: In order to minimize errors, all interfaces will be made as comprehensible as possible. is includes the uniform representation of data provided by the interfaces and the capability for selectively restricting an interface to the data required by the interface's user. e comprehensibility of an interface can be expressed by the *mental load* that it puts on the user. Di erent users need di erent specialized interfaces, each with minimal mental load. For example, an application developer mostly has a service-centered view of the system. Physical network details and other properties not relevant for the application should be hidden from the developer [10].
- *Uniform data structures*: Con guration and management of distributed embedded systems requires representations of system properties that are usable by so ware tools. In order to avoid a situation in which each application deals with the required information in its own way, these representations should be generic, highly structured, and exactly speci ed.
- *Low overhead on embedded system*: Typical embedded hardware is restricted by requirements for cost, size, power consumption, and mechanical robustness. us, embedded hardware usually provides far less memory and processing power than average desktop systems. Currently, typical microcontrollers provide about several hundred bytes of RAM and less than 128 kB of Flash ROM. Clocked by an internal oscillator, these microcontrollers provide only a few MIPS of processing power. e local application will consume most of the available resources, and consequently, the designers of con guration and management mechanisms must ensure that there is no excessive overhead on the embedded system nodes. is requirement drives design decisions where con guration data are compressed or even stored on a separate repository outside the embedded network.
- *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. Lack of a guaranteed response time might violate the special timing requirements of an embedded real-time protocol, and thus, it is o en not possible to directly connect a con guration host computer to the eldbus network without a gateway. Instead, a con guration tool must use some other means of communication, such as standard communication protocols or interfaces such as TCP/IP, RS232, USB, or standard middleware like Common

Object Request Broker Architecture (CORBA). O en, the system uses a dedicated gateway node that routes the con guration and management access to the sensor/actuator nodes; thus, the timing of the embedded network is not disturbed by con guration tra c, and the embedded nodes do not need to implement a USB or CORBA interface and can be kept slim. In order to reduce the complexity of the involved conversion and transformation steps, the interface to and from the embedded node must be comprehensible, structurally simple, and easy to access.

3.4 Interface Separation

If di erent user groups access a system for di erent purposes, users should only be provided with an interface that presents information relevant for their respective purposes [11].

Interfaces for di erent purposes may di er in accessible information and temporal behavior of the access across the interface.

Kopetz, Holzmann, and Elmenreich have identi ed three interfaces to a transducer node [5]:

e con guration and planning (CP) interface allows the integration and setup of newly connected nodes. e CP interface is used to generate the *glue* in the network that enables network components to interact in the intended manner. e CP interface is not usually time critical.

e diagnostic and management (DM) interface is used for parameterization and calibration of devices and for collecting diagnostic information to support maintenance activities. For example, a remote maintenance console can request diagnostic information from a certain sensor. e DM interface is not usually time critical.

e real-time service (RS) interface is used to communicate the application data, for example, sensor measurements or set values for an actuator. is interface usually has to ful ll timing constraints such as a bounded latency and a small communication jitter. e RS interface has to be con gured by the use of the CP (e.g., communication schedules) or DM interface (e.g., calibration data or level monitors).

e TTP/A eldbus system in [12] uses time-triggered scheduling to provide a deterministic communication scheme for the RS interface. A speci ed part of the bandwidth is reserved for arbitrary CP and DM activities. erefore, it is possible to perform CP tasks while the system is in operation without a probe e ect on the RS [13].

3.4.1 Interface File System Approach

e concept of an Interface File System (IFS) was introduced in [5]. e IFS provides a unique addressing scheme to all relevant data belonging to nodes in a distributed system. us, the IFS maps real-time data, many kinds of con guration data, self-describing information, and internal state reports for diagnostic purposes.

e IFS is organized hierarchically as follows: e *cluster name* addresses a particular eldbus network. Within the cluster, a speci c node is addressed by the *node name*. e IFS of a node is structured into *files* and *records*. Each record is a unit of 4 bytes of data.

e IFS is a generic approach that has been implemented with the TTP/A protocol as a case study for the OMG Smart Transducer Interface. e IFS approach supports the integration and management of heterogeneous eldbus networks well. e IFS provides the following bene ts:

- e IFS establishes a well-de ned interface between network communication and local application. e local application uses application programming interface functions to read and write data from/into the IFS. e communication interface accesses the IFS to exchange data across the network.
- e IFS provides transparency on network communication because a task does not have to discriminate between data that are locally provided and data that are communicated via the network.

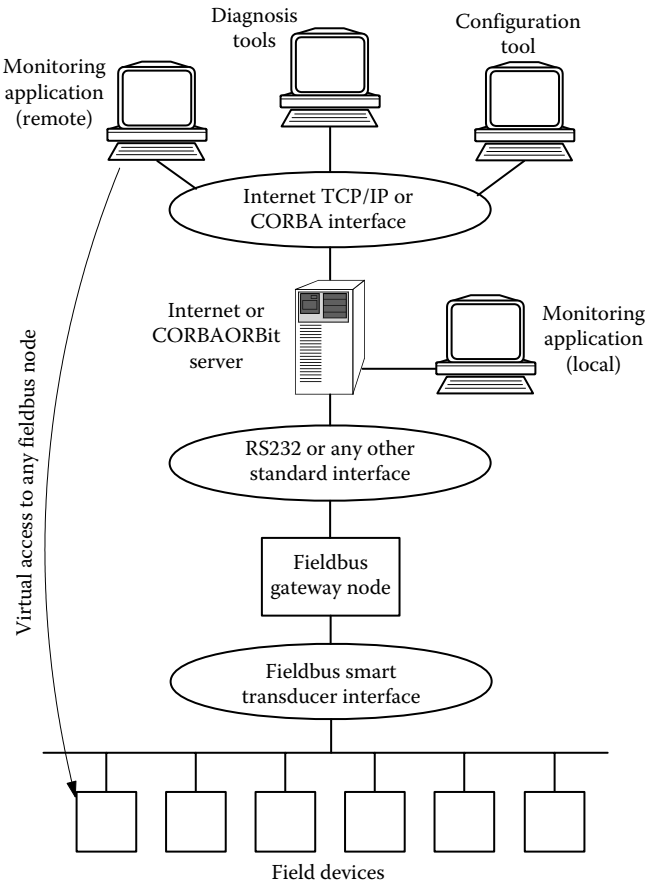

FIGURE 3.1 Architecture for remote con guration and monitoring.

- Because the con guration and management data are mapped into the IFS, the IFS supports access from con guration and management tools from outside the network well. Figure 3.1 depicts an architecture with con guration and management tools that access the IFS of a eldbus network from the Internet.

e IFS maps di erent data domains such as RS data, con guration data, and management data using a consistent addressing scheme. In fact, the management interface can be used to de ne the RS data set dynamically (e.g., to select between a smoothed measurement and a measurement with better dynamics as the output from a sensor). Although real-time guarantees for communication of real-time data are necessary, access to con guration and management data is not time critical. is enables the employment of web-based tools for remote maintenance.

Tools that operate on the IFS have been implemented using CORBA as middleware. CORBA is an object model managed by the OMG that provides transparent communication among remote objects. Objects can be implemented in di erent programming languages and can run on di erent platforms. e standardized CORBA protocol IIOP (Internet Inter-ORB Protocol) can be routed over TCP/IP, thus supporting worldwide access to and communication between CORBA objects across the Internet.

Alternatively, it is possible to use web services as management interface for embedded devices. A case study that implements web services on top of the IFS is described in [14].

3.4.2 Service-Oriented Device Architectures

In a network of smart transducers, nding a model that can suit each sensor is necessary for accessing data in the same way and for enabling cheaper development and maintenance of the network.

SOA concerns the organization of self-contained and distributed computing components in services so that complex applications can be developed as compositions of loosely coupled components.

e SOA approach is independent of any technology because architectures specify a common language for the de nition of services. e Web Service Description Language (WSDL) provides a means for the speci cation of functionalities o ered by a web service so that the implementation of the service can be separated from the component's interface. Services are described in terms of actions, I/O interface, and constraints on data, as well as communication protocols. In order to interact with other web services, a central service registry may be deployed. e Universal Description, Discovery, and Integration registry is a platform-independent registry that enables components to discover each other. Provider components can publish their WSDL descriptions on the registry, enabling client components to reach the provider components using Simple Object Access Protocol (SOAP) messages. SOAP relies on XML to provide an abstract and structured representation for messages.

Because complex services can be assembled from individual services, interaction in the process needs to be coordinated. Synchronization can be controlled by a central orchestrator or directly managed by the distributed entities according to their roles (service choreography). Such coordination may be hard-coded into the procedures used to access the service. However, this is di cult to apply in networks of numerous distributed services and multiple interleaved business processes. Approaches dealing with this issue are the Business Process Execution Language for Web Services, the Web Services Choreography Interface, and the Web Services Choreography Description Language.

3.4.3 Resource-Oriented Architectures

An interesting characteristic of communication between components concerns the information used for requesting a service. A typical distinction is between remote procedure call (RPC) and the Representation State Transfer (REST) model. Remote procedure call is an interprocess communication mechanism that allows processes to call procedures (which are turned into methods on object-oriented platforms such as Java RMI) on di erent address spaces. SOAP-based services implement this approach. A module called *stub* manages this mechanism. *Stub* takes care of data serialization (i.e., marshaling) and hides network mechanisms from developers. e main di erence between RPC and REST is that in RPC, the sender sends part of its state to the receiver to perform the request and delegates the receiver to carry out the procedure. In contrast, in the REST model, the sender considers the receiver a collection of resources identi ed by URIs, and only representations of those resources are sent back to the requester. erefore, REST web services are manipulations of resources (addressable by URI) rather than procedure or method calls to components. REST is an architectural style de ning the behavior of web services in terms of client–server interactions. Accordingly, a client requests a resource to a server. e server processes the request and returns a representation of the resource, consisting of a document describing the state of the resource. is requires resources to be identi ed with a global identi er (e.g., URI) and means that network components use a standard interface and protocol to exchange such representations (e.g., HTTP). Consequently, the user can progress through a web application by navigating links, which represent state transitions where a di erent state of the application is returned to the user, who is unaware of the actual network infrastructure (e.g., topology).

erefore, REST does not provide any mechanism for device discovery, as users are supposed to start their navigation at an index page and receive a meaningful list of resources available on the site (server), as well as connections to other sites. Due to its resource-centered approach, REST is usually considered a resource-oriented architecture. is approach was rst described in Fielding's PhD thesis [15] and during the de nition of the HTTP protocol, although a RESTful approach can also be implemented in other application-level protocols (e.g., SOAP). To enable service interoperability, a typical solution adopted by REST designs is

to describe the payload of HTTP messages in the XML or JSON (Javascript object notation) format. is means that REST can directly take advantage of HTTP, using a uniform interface to access resources (i.e., GET, POST, PUT, and DELETE operations). On the contrary, SOAP-oriented designs use HTTP only as a means for complex XML messages, describing both the protocol and the data to be exchanged when interacting with the service. Moritz et al. [16] examine possible solutions for IP-based smart cooperating objects. In particular, the author provides a comparison of RESTful architectures with the DPWS and underlines necessary requirements when using REST architectures. Moreover, he claims that DPWS can be easily used to model a RESTful application, but a RESTful design can model only a restricted subset of the DPWS design.

What is missing in RESTful architectures is support for asynchronous messaging and eventing, as well as dynamic discovery of resources and semantics.

e interface of REST web services can be described by means of the Web Application Description Language (WADL), a machine-readable and XML-based description of HTTP-based web applications. erefore, WADL is the parallel of WSDL for RESTful web services, as WADL focuses only on describing the interface without any support for semantics. Kamilaris et al. [17] propose a service discovery mechanism using WADL to advertise REST web services when joining the network. e approach is then evaluated in a smart home scenario, consisting of a network of smart meters. In WSDL de nitions, semantic annotations can be added using the Semantic Annotation for WSDL. Another solution is the OWL-S description language, which uses an RDF ontology to assign a semantic to the web service functionality, although OWL-S does not provide any means to express its relationships to other services [18]. However, as the original RDF speci cation does not include quanti ers, Verborgh et al. [18] aim at providing a semantic method to express RESTful web services. In particular, because RESTful services strongly depend on their URIs, a descriptive semantic explanation of the relationship between a resource and its URI should be provided. To enable automated agents to consume REST web service, it is necessary to describe the meaning of services and their relationships with other services. RESTdesc is proposed as a valid solution for describing the semantics of RESTful web services according to the functionalities they implement, rather than their input–output interface. RESTdesc allows the semantic description of web services as implications in the Notation 3{n3}* format, which means that services are de ned by the pre- and post-conditions needed to execute a request. is gives a reasoner (e.g., EYE {eye}†) the possibility of performing inferences by dynamically evaluating preconditions and applying given axioms, thus providing a exible solution to handle complex situations. Indeed, the authors suggest that RESTdesc descriptions can be used for automatic context-aware discovery of services, as well as service composition and execution. An example use case illustrating the power of these semantic descriptions is presented in [19]. e authors show that a service composition is a logic entailment binding initial and goal con guration. Accordingly, each service is seen as implication, and service composition is a chain of implications corresponding to a dependency relationship between services. Beyond the formal demonstration, the authors also show how the plan can be implemented by an executor component. Certain rules involve physical data, and thus, these rules are context dependent and are considered only during plan execution.

3.5 Profiles, Datasheets, and Descriptions

In order to support computer-aided con guration, dedicated computer-readable representations of network and node properties are required. is information plays a similar role to that of manuals and datasheets for a computer-based support framework during con guration and management of a system. ese computer-readable representations allow the establishment of common rule sets for developing and con guring applications and for accessing devices and system properties (for con guration as well as management functions). In the following section, we will discuss the pro le approach as well as several mechanisms following an electronic datasheet approach: Electronic Device Description

* http://www.w3.org/Teamsubmission/n3/
† http://eulersharp.sourceforge.net

Language (EDDL), Field Device Tool/Device Type Manager (FDT/DTM), transducer electronic datasheets (TEDS), smart transducer descriptions (STD) of the IFS, Devices Pro le for Web Services (DPWS), and constrained application protocol (CoAP).*

3.5.1 Profiles

Pro les are a widely used mechanism to create interoperability in distributed embedded systems. We distinguish between several types of pro les, namely, application, functional, or device pro les. Heery and Patel propose a very general and short pro le de nition that we adopt for our discussion. In short, "… pro les are schemata, which consist of data elements drawn from one or more namespaces,[1] combined together by implementers, and optimized for a particular local application" [20].

In many cases, a pro le is the result of the joint e ort of a group of device vendors in a particular area of application. Usually, a task group is founded that tries to identify reoccurring functions, usage patterns, and properties in the device vendors' domain and then creates strictly formalized speci cations according to these identi ed parts, resulting in the so-called pro les.

More speci cally, for each device type, a pro le de nes exactly what kind of communication objects, variables, and parameters have to be implemented so that a device will conform to the pro le. Pro les usually consist of several types of variables and parameters (e.g., process parameters, maintenance parameters, and user-de ned parameters) and provide a hierarchical conformance model that allows for the de nition of user-de ned extensions of a pro le. A device pro le need not necessarily correspond to a particular physical device, for example, a physical node could consist of multiple *virtual* devices (e.g., multipurpose I/O controller), or a virtual device could be distributed over several physical devices.

Protocols supporting device, functional, and application pro les are CANopen [21], Pro bus, and LON [22] (LonMark functional pro les). Figure 3.2 depicts, as an example, the visual speci cation of a LonMark[2] functional pro le for an analog input object. e pro le de nes a set of network variables (in this example, only mandatory variables are shown) and local con guration parameters (none in this example). e arrow speci es that this pro le outputs a digital representation of an analog value, whereas the structure of this output is de ned with the standardized (in LON) network variable type `SNVT_lev_percent` (−163.84% to 163.84% of full scale). In addition, a pro le also speci es other important properties, such as timing information, valid range, update rate, power-up state, error condition, and behavior (usually as a state diagram).

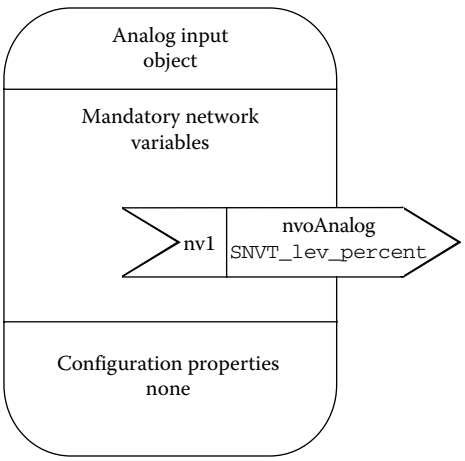

FIGURE 3.2 Functional pro le for an analog input.

* http://tools.ietf.org/html/dra -ietf-core-coap-18

Although the approach for creating pro les is comparable in di erent protocols, the pro les are not always interchangeable between the various eldbuses, although advancements (at least for process control–related eldbuses) have been made within the IEC 61158 [23] standard. Block- and class-based concepts such as function blocks, as de ned for the Foundation Fieldbus (FF) or the Pro bus DP or component classes in IEEE 1451.1 [24] can be considered implementations of the functional pro le concept.

3.5.2 Electronic Device Description Language

e EDDL was rst used with the HART communication protocol in 1992 [25]. In 2004, EDDL was extended by combining the device description languages of HART, PROFIBUS, and Fieldbus Foundation, resulting in an approved international standard IEC 61804-2 [26]. An additional standard IEC 61804-3 extends EDDL by an enhanced user interface, graphical representations, and persistent data storage [27]. In 2004, the OPC Foundation joined the EDDL Cooperation Team. Subsequently, EDDL is used in the open OPC Uni ed Architecture.

In EDDL, each eldbus component is represented by an electronic device descriptor (EDD). An EDD is represented in a text le and is operating system independent. Within the basic control and database server, an EDDL interpreter reads the EDD les corresponding to the devices present at the eldbus system.

Although EDD les are tested against various eldbus protocols by the vendors, there is no standardized test process for assuring that an EDD works with every available EDDL interpreter. Another weak point of EDDL is that the functionality that can be described with EDDL is limited by the basic functions required by the IEC 61804-2 standard. us, device functionality that cannot be described by these functions is o en modeled via additional proprietary plug-ins outside the standard. With eldbus devices becoming increasingly sophisticated, it will become di cult to adequately describe devices with the current EDDL standard.

3.5.3 Field Device Tool/Device Type Manager

e FDT/DTM is a manufacturer-spanning con guration concept for eldbus devices [28]. FDT is supported by the FDT group, which currently consists of over 50 members (vendors and users). e FTD group is responsible for the certi cation of DTMs.

A DTM is an executable so ware that acts as a device driver. An FDT/DTM con guration system consists of the following parts: an FDT frame application for each distributed control system, a communication DTM (comm-DTM) for each eldbus system, and a device DTM for each eld device type. Device functionality is fully encapsulated into the DTM so FDT/DTM comes with no limits regarding the functionality of more complex future devices. In contrast to EDDL, FDT/DTM comes with higher costs for the device manufacturers because manufacturers have to provide the DTM device drivers. On the other hand, EDDL puts greater strain on system manufacturers.

e FDT so ware has to be executed on a server separate from the basic control and database server. Currently, DTM can run only on Microso Windows, which is a drawback due to Microso 's upgrading and licensing policies (Windows versions evolve rather fast in comparison to the lifetime of an automation plant, and it is not possible to obtain support and extra licenses for outdated Windows versions).

3.5.4 Transducer Electronic Datasheet

e TEDS was developed to establish a generic electronic datasheet format as part of the smart transducer-related IEEE 1451 standards family. e IEEE 1451.2 [6] standard speci es the TEDS including the digital interface to access datasheets, read sensors, or set actuators.

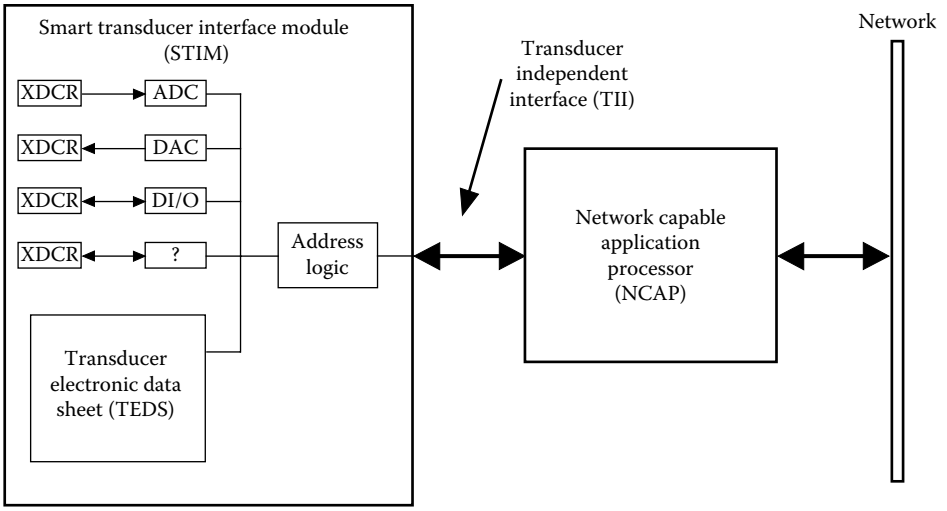

FIGURE 3.3 Smart transducer interface module connected to NCAP.

Figure 3.3 depicts the TEDS in context with the system architecture as de ned in the IEEE 1451 standard:

- *Smart transducer interface module* (*STIM*): A STIM contains between 1 and 255 transducers of various prede ned types together with their descriptions in the form of the corresponding TEDS.
- *Network capable application processor* (*NCAP*): e NCAP is the interface to the overall network. By providing an appropriate NCAP, the transducer interface is independent of the physical eldbus protocol.
- *Transducer independent interface* (*TII*): e TII is the interface between the STIM and the NCAP. e TII is speci ed as an abstract model of a transducer instrumented over 10 digital communication lines.

TEDS describe node-speci c properties, such as the structure and temporal properties of devices and transducer data. IEEE 1451 de nes self-contained nodes with the TEDS stored in a memory directly located at the node. is causes an overhead on the nodes, so the representation of the con guration information for such a system must be compact. IEEE 1451 achieves this goal by providing a large set of prede ned transducer types and modes based on enumerated information, in which identi ers are associated with more detailed prespeci ed descriptions (similar to error codes). Although the standard de nes some possibilities for parameterization of transducer descriptions, this approach restricts the device functionality expressiveness in a similar way as in EDDL.

3.5.5 Interface File System/Smart Transducer Descriptions

STDs as de ned in [29] are a method for formally describing properties of devices that follow the CORBA Smart Transducer Interface standard (the descriptions themselves are currently not part of the standard).

e STD uses XML [30] as the primary representation mechanism for all relevant system aspects. Together with related standards, such as XML Schema or XSLT, XML provides advanced structuring, description, representation, and transformation capabilities. XML is becoming the de facto standard for data representation and has extensive support throughout the industry. Some examples where XML has already been used for applications in the eldbus domain can be found in [31–33].

As the name implies, *STDs* describe the properties of nodes in the smart transducer network. e STD format is used for both describing *static* properties of a device family (comparable to classic datasheets) and devices that are con gured as part of a particular application (e.g., if the local STD also contains the local node address). e properties described in STDs cover microcontroller information (e.g., controller vendor, clock frequency, and clock dri), node information (e.g., vendor name, device name/version, and node identi ers), protocol information, and node service information specifying the behavior and the capabilities of a node. In the current approach, a service plays a similar role as a functional pro le (see Section 3.5.1) or function block. e functional units align to the interface model of the CORBA STI standard [34].

It is not always possible to store all relevant information outside the node, but by focusing on reducing the amount of required information on the node to the minimum, extensive external meta-information can be used without size constraints. e reference to this external information is the unique combination of series and serial number of the node. e series number is identical for all nodes of the same type. e serial number identi es the instance of a node among all nodes of a series.

e advantages of this approach are twofold:

- Firstly, the overhead at the node is very low. Current low-cost microcontrollers provide internal RAM and EEPROM memory of around 256 bytes. is is not su cient for storing more than the most basic parts of datasheets according to standards such as the IEEE 1451.2 standard without extra hardware such as external memory. With the proposed description approach, only 8 bytes of memory are required for storing the series and serial numbers.
- Secondly, instead of implicitly representing the node information with many prede ned data structures mapped to a compact format, it is possible to have an explicit representation of the information in a well-structured and easy-to-understand way. A typical host computer running the con guration and management tools can easily deal with very extensive generic XML descriptions. Furthermore, XML formats are inherently easy to extend, so the format is open for future extensions of transducer or service types.

3.5.6 Devices Profile for Web Services

e Universal Plug and Play (UPnP) standard is a rst step toward the implementation of service-oriented device architectures, and UPnP is already di used in common consumer electronic devices such as network-attached storage systems and media servers. Other frameworks are commonly used in distributed and service-oriented settings, such as OSGi, Jini, and CORBA. However, these frameworks are usually not applicable to very resource-constrained devices.

e DPWS is a collection of standards that allow embedded devices to support secure web services. DPWS does not rely on a global service registry to store and retrieve service descriptions because DPWS uses dynamic service discovery mechanisms to allow devices to advertise themselves and discover their peers. Moreover, due to metadata description of devices, provided services can be accessed alike. Communication among devices is granted by an event-based mechanism. is provides a way to notify state changes according to a publish/subscribe policy.

To meet constraints of deeply embedded devices, Moritz et al. [35] developed uDPWS, a lightweight version that can be run by 8-bit microcontrollers. In this variant, the authors use a table-driven implementation (consisting in a simple string comparison) to avoid parsing the whole incoming message as XML le.

Since SOAP representations rely on XML, data management introduces a certain overhead. A solution to overcome the heavyweight XML representation is to use a binary format. A potential candidate is the E cient XML interchange (EXI), recommended by W3C since 2011. EXI provides an alternative compact representation of structured data that can be easily integrated into existing XML-based services by means of special converters. A comparison between di erent formats is addressed by Sakr [36].

EXI is shown to provide the greatest compression and compactness compared to other representations. erefore, EXI enhances deeply constrained embedded devices with an e cient way of processing XML information, enabling devices to be integrated in SOAs. An implementation of EXI that can be run by deeply embedded systems is proposed by Kyusakov [37].

3.5.7 Constrained Application Protocol

Using HTTP is the straightforward solution to implement a REST approach. However, as HTTP provides functionalities that can be considered unnecessary in constrained environments, the IETF constrained RESTful environments working group (CoRE) has proposed the CoAP{coap} as an alternative that can be run by deeply embedded systems (e.g., 8-bit microcontrollers). To maintain an easy mapping with HTTP, CoAP uses the same methods (i.e., GET, POST, PUT, and DELETE). However, instead of using TCP, CoAP uses UDP along with a simple retransmission mechanism in which each GET request is associated with a unique identi er. Moreover, CoAP provides an eventing mechanism to the architecture.

In HTTP, data are transferred from a server to a client a er a synchronous request. However, in networks of resource-constrained devices, such as sensor nodes, this means that a node should perform polling on another peer in order to monitor the peer's state, thus increasing the network usage and power consumption. To overcome this limitation, CoAP allows a node to subscribe to updates of a certain resource. erefore, CoAP provides an asynchronous eventing mechanism, in which nodes can become observers of a certain resource and receive noti cation of status changes. Another characteristic of REST is that REST does not provide any mechanism for device and resource discovery. In HTTP and the web, discovery of resources takes place by accessing an index page listing links to resources on the same or di erent servers. erefore, links de ne relationships between web resources. is means that giving a meaning to links enables machines to automatically calculate how to use resources on a server. For this purpose, a CoRE link format (RFC6690) has been proposed as the format to be used in M2M applications within the CoAP protocol. Accordingly, links are de ned by a list of attributes. A resource-type attribute, *rt*, de nes the semantic type of a resource, which may be expressed by a URI referencing a speci c concept in an ontology.

e interface description attribute, *if*, can be used to describe the service interface, for instance, by URI referencing a machine-readable document such as a WADL de nition. Finally, a maximum size attribute, *sz*, can be used to return an indication of the maximum expected size of the resource representation.

According to this approach, a client can contact the server using a GET to a prede ned location (i.e., */.well-known/core*), which returns the list of resources exposed by the server and their media type. To provide a basic ltering mechanism, the query can be associated to certain standardized attributes. In particular, *href* lters resources according to their path and type, allowing retrieval of certain Multipurpose Internet Mail Extensions types. Similarly, the attributes *rt* and *if* can be used to select resources according to their application-speci c meaning and permitted operations. In addition, it is possible to deploy a resource directory listing links to resources stored on other services. In order to be visible to clients, resource providers can POST their resources to the */.well-known/core* position of the chosen directory node.

3.6 Application Development

In the following section, we examine several application development approaches and how they in uence system con guration.

Model-based development is a widely used approach in distributed systems. e basic idea is to create a model of the application that consists of components that are connected via links that represent the communication ow between the components. Di erent approaches usually di er in what constitutes

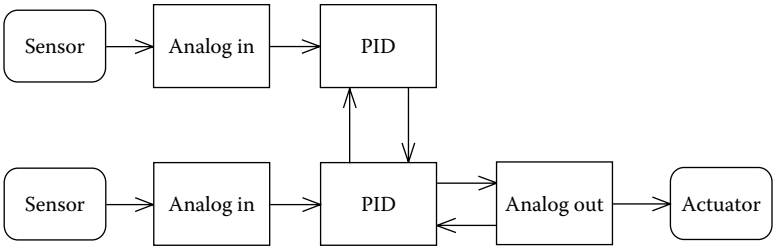

FIGURE 3.4 Example of an application model.

a component (e.g., function blocks, subsystems, services, functional pro les, and physical devices) and the detailed semantics of a link. Many approaches support the recursive de nition of components, which allows grouping multiple lower-level components into one higher-level component. Figure 3.4 depicts a model of a typical small control application consisting of two analog inputs receiving values from two sensors, two PIDs, and one analog output controlling an actuator.

e model-based approach is not the only application design approach. Another approach used by multiple eldbus con guration tools is the ANSI/ISA-88.01-1995 procedural control model [38].

is modeling approach enforces a strictly modular hierarchical organization of the application (see Figure 3.5). ere should be little or no interaction between multiple process cells. Interaction between components in a process cell is allowed. To make best use of this approach, the structure of the network site and the application should closely correspond to the hierarchy speci ed by this model.

The modeling approach conceptually follows the typical hierarchy of process control applications, with multiple locally centralized Programmable Logic Controllers that drive several associated control devices. This eases transition from predecessor systems and improves overall robustness because this approach provides fault containment at the process cell level, the downside being that the coupling between the physical properties of the system and the application is rather tight. An example of a fieldbus protocol that supports this modeling approach is the Profibus PA protocol, which supports this model by providing a universal function block parameter for batch identification [39].

Another design approach is the *two-level design approach* [40], which originated in the domain of safety critical systems. In this approach, the communication between components must be con gured before con guring the devices. Although this requires many design decisions to be taken very early in the design process, this approach greatly improves overall composability of the components in the system.

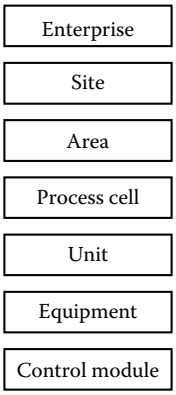

FIGURE 3.5 ANSI/ISA-88.01-1995 hierarchical model.

Abstract application models provide several advantages for application development:

- e modular design of applications helps to deal with complexity by applying a *divide-and-conquer* strategy. Furthermore, modular design supports reuse of application components and physical separation.
- e separation of application logic from physical dependencies allows hardware-independent design that enables application development before hardware is available, as well as easing migration and possibly allowing the reuse (of parts) of applications.

For con guring a physical eldbus system from such an application model, we must examine (1) how this application model maps to the physical nodes in the network and (2) how information ow is maintained in the network.

In order to map the application model to actual devices, eldbuses o en provide a model for specifying physical devices, for example, in Pro bus DP, the physical mapping between function blocks and the physical device is implemented as follows (see Figure 3.6): A physical device can be subdivided in several modules that take the role as *virtual* devices. Each device can have from one (in case of simple functionality) to many slots, which provide the mapping from physical devices to function blocks. A function block is mapped to a slot, whereas slots may have associated physical and transducer blocks. Physical and transducer blocks represent physical properties of a eldbus device. Parameters of a function block are indexed, and the slot number and parameter index cooperatively de ne the mapping to data in the device memory.

In contrast, the *FF* follows an object-oriented design philosophy. us, all information items related to con guring a device and the application (control strategy) are represented with objects. is includes function blocks, parameters, as well as subelements of parameters. ese objects are collected in an object dictionary (OD), whereas each object is assigned an index. is OD de nes the mapping to the physical memory on the respective device. In order to understand the methods for controlling the communication ow between the application components, we rst examine some recurring important communication properties in eldbus applications:

- Use of state communication as primary communication mechanism for operating a eldbus [41]. State communication usually involves cyclically updating the associated application data.
- Support for asynchronous/sporadic communication (event communication) in order to perform management functions and deal with parts of the application that cannot be performed with state communication.

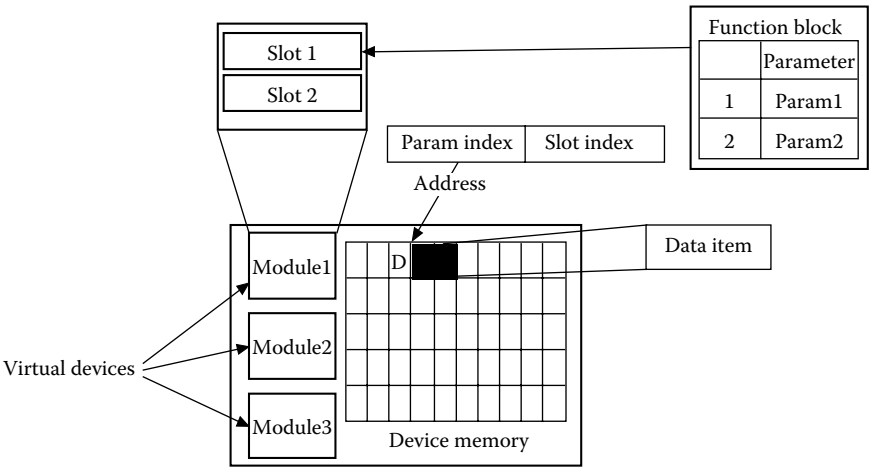

FIGURE 3.6 Mapping of function blocks to physical device in Pro bus DP.

A common method to achieve these properties is by scheduling. ere are many scheduling approaches with vastly di erent e ects on con guration. e following are some commonly adopted approaches in eldbus systems:

Multicycle polling: In this approach, communication is controlled by a dedicated node that authorizes other nodes to transmit their data [42]. is dedicated node, typically called the *master node*, polls the other nodes in a time division multiplexing scheme. us, the cyclic polling also takes care of bus arbitration. is approach is taken, for example, in WorldFIP, FF, and ControlNet. For con guring the devices in such a network, the master nodes require at least a list of nodes to be polled, that is, in a con guration with a single master, only one node need be con gured with time information to control the whole cluster.

Time-triggered: In a time-triggered communication model, the communication schedule is derived from the progression of physical time. is approach requires a prede ned collision-free schedule that de nes a priori when a device is allowed to broadcast its data and an agreement on a global time, which requires synchronization of the local clocks of all participating devices [3]. Examples of protocols that support time-triggered communication are TTP/A [12], TTP/C [43], and the synchronous part of the Flexray protocol [44]. In order to con gure the communication in these systems, the schedules must be downloaded to all nodes in the network.

Event-triggered: Event-triggered communication implements a push model in which the sender decides when to send a message, for example, when a particular value has changed more than a given *delta*. Collisions on the bus are solved by collision detection/retransmission or collision avoidance, that is, bitwise arbitration protocols such as CAN [45]. Event-triggered communication does not depend on scheduling because communication con icts are either resolved by the protocol at the data link layer (e.g., bit-wise arbitration) or must be resolved by the application.

Scheduling information is usually stored in dedicated data structures that are downloaded to the nodes in the network to be available for use by the network management system functions of the node. e TTP/A protocol integrates con guration information for application data ow and the communication schedule. us, the local communication schedules (named *round descriptor lists*) as well as the interfaces of application services [34] are mapped into the same interfacing mechanism, the *IFS* (see Section 3.4.1).

For representation of the overall system, cluster con guration description (CCD) format was developed, which acts as a central uniform data structure that stores all the information pertinent to the eldbus system. is information includes the following:

- *Cluster description meta-information*: is description block holds information on the cluster description itself, such as the maintainer, name of the description le, or the version of the CCD format itself.
- *Communication configuration information*: is information includes round sequence lists as well as round descriptor lists, which represent the detailed speci cation of the communication behavior of the cluster. Additionally, this part of the CCD also includes (partially physical) properties important for communication, such as the UART speci cation, line driver, and minimum/maximum signal run times.
- *Cluster node information*: is block contains information on the nodes in a cluster, whereas nodes are represented with the STD format.

3.7 Configuration Interfaces

In the previous section, we focused on the relationship between application and con guration. In this section, we examine parts of system con guration that are mostly independent of the application. We brie y look at the physical con guration of networked embedded systems, how nodes are recognized by the con guration system, and how application code is downloaded to nodes.

3.7.1 Hardware Configuration

e hardware con guration encompasses the setup of the plugs and cables of the network. For instance, several eldbus systems implement means to avoid mistakes, such as connecting a power cable to a sensor input, which would cause permanent damage to the system or even harm people. Moreover, hardware con guration interfaces such as plugs and clamps are o en subject to failure in harsh environments, for example, on a machine that induces a lot of vibration.

For hardware con guration, the following approaches can be identi ed:

- Usage of special jacks and cables that support a tight mechanical connection and avoid mistakes in orientation and polarity by their geometries, for example, the actuator–sensor interface[3] (AS-i) speci es a mechanically coded at cable that allows the connection of slaves on any position on the cable by using piercing connectors. AS-i uses cables with two wires transporting data and energy via the same line. e piercing connectors support simple connection, safe contacting, and protection up to class IP67.
- Baptizing devices in order to obtain an identi er that allows addressing the newly connected device. is could be done by explicitly assigning an identi er to the device, for example, by setting dip switches or entering a number over a local interface, or implicitly by the cabling topology, for example, devices could be daisy-chained and obtain their name subsequently according to the chain.
- Alternatively, it is possible to assign unique identi ers to nodes in advance. is approach is taken, for example, with Ethernet devices for which the MAC address is a worldwide unique identi er, or in the TTP/A protocol that also uses unique node IDs. However, such a worldwide unique identi er will require many digits making it unpractical to have the number printed somewhere on the device. To overcome this problem, machine-readable identi ers in the form of barcodes or RF tags are used during hardware con guration.
- Simple con guration procedures, which can be carried out and veri ed by nonexpert personnel.

3.7.2 Plug and Participate

Hardware con guration is intended to be simple, and consequently a networked system should behave intelligently in order to relieve human personnel of error-prone tasks.

In the case of plug and participate, the system runs an integration task that identi es new nodes, obtains information about these nodes, and changes the network con guration in order to include the new nodes in the communication.

Identi cation of new nodes can be supported with manual baptizing as described in the previous section. Alternatively, it is also possible to automatically search for new nodes and identify them as described in [46].

If there can be di erent classes of nodes, it is necessary to obtain information on the type of the newly connected nodes. is information will usually be available in the form of an electronic datasheet that can be obtained from the node or from an adequate repository.

e necessary changes of the network con guration for including the new node greatly depend on the employed communication paradigm. In the case of a polling paradigm, only the list of nodes to be polled has to be extended. In the case of a time-triggered paradigm, the schedule has to be changed and updated in all participating nodes. In the case of an event-triggered paradigm, only the new node has to be authorized to send data; however, it is very di cult to predict how a new sender will a ect the timing behavior of an event-triggered system. In all three cases, critical timing might be a ected due to a change in response time, that is, when the cycle time has to be changed. us, in time-critical systems, extensibility must be taken into account during system design, for example, by reserving at rst unused bandwidth or including spare communication slots.

3.7.3 Service Discovery Mechanisms

Service discovery de nes how network entities can represent and advertise services, so that they can be detected and used by other members. is is a naming problem: assigning each entity a persistent identi er that can be used for retrieving resources. A simple naming system may use broadcasting or multicasting to advertise or retrieve resources (according to push or pull semantics). For instance, this is what happens in the address resolution protocol. Another solution is to use centralized name servers (registries), which can be static nodes or dynamically elected by the community. is approach presents single points of failure that can be improved by organizing nodes in a hierarchy of registries (e.g., domain name service).

Large-scale distributed systems usually use logical overlays, and as such, topologies are set up depending on certain criteria, such as physical distance between nodes. We refer to [47] for a complete overview of resource naming.

Dargie [48] classi es naming systems in three generations: name services (e.g., DNS), directory services (e.g., CORBA and LDAP), and service discovery systems. Although name service o ers just mapping between identi ers and resources, directory services enable attribute-based queries because each entry is de ned by a set of attributes. However, the author suggests that the rst two generations may not be enough in dynamically changing environments such as mobile networks, where entities join and leave the network, thus continuously modifying the network topology. According to Dargie [48], service discovery systems can provide self-con guration and self-healing properties to networks of mobile embedded devices. ese systems are indeed able to update the network con guration by discovering new services and detecting failures and disconnected entities (e.g., using leasing on services). Dargie [48] reports on state of the art of service discovery systems. Network size a ects bandwidth, so he divides systems into small and large systems. Small systems (e.g., Jini, UPnP, SLP, and Bluetooth) can be implemented as directory based or using multicast. On the other hand, service discovery for large systems (e.g., Ninja SDS, Twine, Jxta, and Ariadne) has to take into account scalability issues. Consequently, systems can be implemented as registries organized in multiple hierarchies or using logical overlays connecting nodes in a peer-to-peer manner.

3.7.4 Application Configuration and Upload

Some frequently recurring applications, such as standard feedback control loops, alert monitoring, and simple control algorithms can o en be put in place like building bricks because these applications are generically available (e.g., PID controllers).

For more complex or unorthodox applications, however, it is necessary to implement user-de ned applications. ese cases require code to be uploaded onto target devices.

About 15 years ago, the most common method of reprogramming a device was to have an EPROM memory chip in a socket that was physically removed from the device, erased under UV radiation, and programmed using a dedicated development system, that is, a PC with a hardware programming device, and then put back into the system.

Today, most memory devices and microcontrollers provide an interface for in-system serial programming of Flash and EEPROM memory. e hardware interface for in-system serial programming usually consists of a connector with four to six pins attached either to an external programming device or directly to the development PC. ese programming interfaces are o en proprietary to particular processor families, but there also exist some standard interfaces that support a larger variety of devices. For example, the JTAG debugging interface (IEEE Standard 1149.1) also supports the upload of application code.

While the in-system serial programming approach is much more convenient than the socketed EPROM method, both approaches are conceptually reasonably similar because it is still necessary to establish a separate hardware connection to the target system. A more advanced approach for uploading applications is in-programming. In this approach, it is possible to program and con gure a device without taking the device out of the distributed target system and without using extra cables and hardware interfaces.

In-system configuration is supported by state-of-the-art flash devices, which can reprogram themselves in part by using a bootloader program. Typically, whenever a new application has to be set up, the programming system sends the node a signal causing it to enter a dedicated upload mode. During the upload phase, the node's service is usually inactive. Failures that lead to misconfiguration must be corrected by locally connecting a programming tool.

Alternatively, application code could be downloaded via the network into the RAM memory at startup. In this case, only the bootloader resides in the persistent memory of the device, and the user-defined application code has to be downloaded at startup. This approach has the advantage of being stateless, so that errors in the system are removed at the next startup, thus engineers could handle many faults by a simple restart of the system. On the other hand, this approach depends on the configuration instance at startup—the system cannot be started if the configuration instance is down. Moreover, the restart time of a system may be considerably longer.

3.8 Management Interfaces

The possibility of performing remote management operations on distributed devices is one of the most important advantages of these systems. Wollschläger states that "in automation systems, engineering functions for administration and optimization of devices are gaining importance in comparison with control functions" [49, p. 89].

Typical management operations are monitoring, diagnosis, or node calibration. Unlike primary fieldbus applications, which often require cyclical, multidrop communication, these management operations usually use a one-to-one (client–server) communication style. For this reason, most fieldbus systems support both communication styles.

A central question is whether and how this management traffic influences the primary application, a problem known as probe effect [13]. System management operations that influence the timing behavior of network communication are critical for typical fieldbus applications (e.g., process control loops) that require exact real-time behavior.

The probe effect can be avoided by reserving a fixed amount of the bandwidth for management operations. For example, in the FF and WorldFIP protocols, the application cycle (macro cycle) is chosen to be longer than strictly required by the application, and the remaining bandwidth is free for management traffic.

In order to avoid collisions within this management traffic window, adequate mechanisms for avoiding or resolving such conflicts must be used (e.g., token-passing between nodes that want to transmit management information and priority-based arbitration).

In TTP/A, management communication is implemented by interleaving real-time data broadcasts (implemented by multipartner rounds) with the so-called master–slave rounds that open a communication channel to individual devices.

If management traffic is directly mingled with application data, such as in CAN, LonWorks, or Profibus PA, care must be taken that this management traffic does not influence the primary control application. This is typically achieved by analyzing network traffic and leaving enough bandwidth headroom. For complex systems and safety-critical systems that require certain guarantees on system behavior, this analysis can become very difficult.

3.8.1 Monitoring and Diagnosis

In order to perform passive monitoring of the communication of the application, it usually suffices to listen on the network. However, the monitoring device must have knowledge of the communication scheme used in the network in order to be able to understand and decode the data traffic. If this scheme is controlled by the physical time, as is the case in time-triggered networks, the monitoring node must also synchronize itself to the network.

Advanced networked embedded devices often have built-in self-diagnostic capabilities and can disclose their own status to the management system. It depends on the capabilities of the system how such information reaches the management framework. Typically, a diagnosis tool or the diagnosis part of the management framework will regularly check the information in the nodes. This method is called *status polling*. In some fieldbus protocols (e.g., FF), devices can also transmit status messages by themselves (*alert reporting*).

In general, restrictions from the implementation of the management interface of a fieldbus protocol also apply to monitoring because in most fieldbus systems, the monitoring traffic is transmitted using the management interface.

For systems that do not provide this separation of management from application information at the protocol level, other means must be taken to ensure that monitoring does not interfere with the fieldbus application. Status polling usually is performed periodically, and thus, it should be straightforward to reserve adequate communication resources during system design so that the control application is not disturbed. In case of alert reporting, the central problem without adequate arbitration and scheduling mechanisms is discerning how to avoid overloading the network in case of *alarm showers*, when many devices want to send their messages at once. It can be very difficult to give timeliness guarantees (e.g., the time between an alarm occurring and the alarm being received by its respective target) in such cases. The typical approach for dealing with this problem (e.g., as taken in CAN) is to provide much bandwidth headroom.

For in-depth diagnosis of devices, it is sometimes also desirable to monitor operation and internals of individual field devices. This temporarily involves greater data traffic, which cannot be easily reserved a priori. Therefore, the management interface must provide some flexibility on the diagnosis data in order to dynamically adjust to the proper level of detail using some kind of *pan and zoom* approach [50].

3.8.2 Calibration

The *calibration* of transducers is an important management function in many fieldbus applications. There is some ambiguity involved concerning the use of this term. Berge strictly distinguishes between *calibration* and *range setting*:

"Calibration is the correction of sensor reading and physical outputs so they match a standard" [39, p. 363]. According to this definition, calibration cannot be performed remotely because the device must be connected to a standardized reference input.

Range setting is used to move the value range of the device so that the resulting value delivers the correctly scaled percentage value. *Range setting* does not require any input and measurement of output, thus *range setting* can be performed remotely. In the HART bus, this operation is called *calibration*, whereas calibration is called *trim*.

Network technology does not influence the way calibration is handled, although information that is required for calibration is stored as part of the properties that describe a device. Such information could be the minimum calibration span limit, this being "the minimum distance between two calibration points within the supported operation range of a device." Additionally, calibration-related information, that is, individual calibration history can be stored in the devices themselves. This information is then remotely available for management tools in order to check the calibration status of devices. Together with the self-diagnosis capabilities of the field devices, this allows performance of a focused and proactive management strategy.

3.9 Maintenance in Fieldbus Systems

Maintenance is the activity of keeping the system in good working order. Typical networked embedded systems (e.g., fieldbuses) provide extensive management features, such as diagnosis and monitoring, which help greatly in maintaining systems. There are several different maintenance schemes that

in uence the way these steps are executed in detail. Choice of a particular maintenance scheme is usually motivated by the application requirements [39]:

- *Reactive maintenance* is a scheme in which a device is xed only a er it has been found to be broken. Reactive maintenance should be avoided in environments where downtime is costly (such as in factory applications). us, designers of such applications will usually choose more active maintenance strategies. Nonetheless, eldbus systems also provide advantages for this scheme because eldbus systems support fast detection of faulty devices.
- *Preventive maintenance* is a scheme in which devices are serviced in regular intervals even if devices are working correctly. is strategy prevents unexpected downtime, thus improving availability. Due to the associated costs, this approach will only be taken in safety-related applications such as in aviation, train control, or where unexpected downtime would lead to very high costs.
- *Predictive maintenance* is similar to preventive maintenance, di ering in a dynamic service interval that is optimized by using long-time statistics on devices.
- *Proactive maintenance* focuses on devices that are expected to require maintenance.

Maintenance mainly comprises the following steps:

- Recognizing a defective device
- Repairing (replacing) the defective device
- Reintegrating the serviced device

In eldbus systems, faulty devices will usually be recognized via the network. is is achieved by monitoring the eldbus nodes and the application or with devices that are capable of sending alerts (also refer to Section 3.8).

A er the source of a problem has been found, the responsible node must be serviced. is o en means disconnecting the node from the network, thus strategies are required to allow the system to deal with *disconnection of* the node, as well as reconnecting and *reintegrating* the replacement node.

If the whole system has to be powered down for maintenance, the faulty node can simply be replaced, and integration of the replacement node occurs as part of the normal initial startup process. If powering down the whole system is undesirable or even impossible (in the sense of leading to severe consequences, as in the case of safety-critical applications), this process becomes more complicated. In this case, we have several options:

- *Implementation of redundancy*: is approach must be taken for safety- or mission-critical devices, where the respective operations must be continued during replacement a er a device becomes defective. A detailed presentation of redundancy and fault-tolerant systems can be found in [51].
- *Shut down part of the application*: In the case of factory communication systems that o en are organized as multilevel networks and/or use a modular approach, it might be feasible to shut down a local subnetwork (e.g., a local control loop, or a process cell as de ned in the ANSI/ISA-88.01-1995 standard).

e replacement node must be con gured with individual node data, such as calibration data (these data usually di er between replaced and replacement nodes), and the *state* of a node. e state information can include the following:

- Information accumulated during run-time (the history state of a system): is information must be transferred from the replaced node to the replacement node.
- Timing information so that the node can synchronize with the network: In networks that use a distributed static schedule (e.g., TTP/A), each node must be con gured with its part of the global schedule in order to establish a network-wide consistent communication con guration.

One alternative approach for avoiding this transfer of system state is to design (and create) a stateless system. Bauer proposes a generic approach for creating stateless systems from systems with state in [52]. Another possibility is to provide well-de ned reintegration points where this state is minimized. Fieldbus applications typically use a cyclical communication style so the start of a cycle is a *natural* reintegration point.

3.10 Conclusion

Con guration and management play an important role for distributed embedded systems. e need for con guration and management in the domain of industrial eldbus systems has led to interesting mechanisms evolving during the last 20 years. Most of these mechanisms can also be applied in the domain of embedded systems.

e con guration phase can be subdivided into a part that requires local interaction such as connection of hardware and setting dip switches, and a part that can be performed remotely via the eldbus system. A good design requires the local part to be as simple as possible in order to simplify the interactions of local personnel. e other part should be supported by tools that assist the system integrator in tedious and error-prone tasks such as adjusting parameters according to the datasheet of a device. Examples of systems with such a con guration support are, among others, the IEEE 1451, the FDT, and the EDDL, which all employ machine-readable electronic datasheets in order to support con guration tools.

Management encompasses functions such as monitoring, diagnosis, calibration, and support for maintenance. In contrast to the con guration phase, most management functions are used concurrently to the RS during operation. Some management functions, such as monitoring, may require real-time behavior for themselves. In order to avoid a probe e ect on the RS, scheduling of the eldbus system must be designed to integrate management tra c with real-time tra c.

Acknowledgments

We would like to thank Lizzy Dawes for contributing her English language skills. is work was supported by the Austrian FWF project TTCAR under contract no. P18060-N04, by Lakeside Labs, Austria, by the European Regional Development Fund (ERDF) and the Carinthian Economic Promotion Fund (KWF) under grant KWF 20214-23743-35470 (Project MONERGY: http://www.monergy-project.eu/).

References

1. J. Powell. e “pro le” concept in eldbus technology. Technical article, Siemens Milltronics Process Instruments Inc., Peterborough, Ontario, Canada, 2003.
2. TTTech Computertechnik. Utilizing TTPTools in by-wire prototype projects. White paper, Vienna, Austria, 2002.
3. H. Kopetz and G. Bauer. e time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, January 2003.
4. W.H. Ko and C.D. Fung. VLSI and intelligent transducers. *Sensors and Actuators*, 2:239–250, 1982.
5. 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.
6. Institute of Electrical and Electronics Engineers, Inc. IEEE Std 1451.2-1997. Standard for a smart transducer interface for sensors and actuators—Transducer to micro-processor communication protocols and transducer electronic data sheet (TEDS) formats, New York, September 1997.
7. OMG. Smart transducers interface V1.0. Available speci cation document number formal/2003-01-01. Object Management Group, Needham, MA, January 2003. Available at: http://doc.omg.org/formal/2003-01-01.

8. L.A. Zadeh. *The Concept of System, Aggregate, and State in System Theory*, Inter-University Electronics Series, vol. 8, pp. 3–42, McGraw-Hill, New York, 1969.
9. H. Kopetz. *Real-Time Systems—Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, Boston, MA/Dordrecht/London, U.K., 1997.
10. S. Pitzek and W. Elmenreich. Managing eldbus systems. In *Proceedings of the Work-in-Progress Session of the 14th Euromicro International Conference*, Vienna, Austria, June 2002.
11. A. Ran and J. Xu. Architecting so ware with interface objects. In *Proceedings of the Eighth Israeli Conference on Computer-Based Systems and Software Engineering*, Washington, DC, 1997, pp. 30–37.
12. H. Kopetz et al. Speci cation of the TTP/A protocol. Technical report, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002. Version 2.00. Available at: http://www.vmars.tuwien.ac.at/ttpa/.
13. J. Gait. A probe e ect in concurrent programs. *Software Practice and Experience*, 16(3):225–233, March 1986.
14. M. Venzke. Spezi kation von interoperablen Webservices mit XQuery. PhD thesis, Technische Universität Hamburg-Harburg, Hamburg-Harburg, Germany, 2003.
15. R.T. Fielding. Architectural styles and the design of network-based so ware architectures. PhD thesis, 2000. AAI9980887.
16. G. Moritz, E. Zeeb, S. Prüter, F. Golatowski, D. Timmermann, and R. Stoll. Devices pro le for web services and the rest. In *Proceedings of the Eighth IEEE International Conference on Industrial Informatics* (*INDIN*), Shanghai, China, 2010, pp. 584–591.
17. A. Kamilaris, A. Pitsillides, and V. Trifa. e smart home meets the web of things. *International Journal of Ad Hoc and Ubiquitous Computing*, 7(3):145–154, May 2011.
18. R. Verborgh, T. Steiner, D. Van Deursen, J. De Roo, R. Van de Walle, and J.G. Vallés. Description and interaction of RESTful services for automatic discovery and execution. In *Proceedings of the FTRA 2011 International Workshop on Advanced Future Multimedia Services*, Jeju, South Korea, 2011.
19. R. Verborgh, V. Haerinck, T. Steiner, D. Van Deursen, S. Van Hoecke, J. De Roo, R. Van de Walle, and J.G. Vallés. Functional composition of sensor Web APIs. In *Proceedings of the Fifth International Workshop on Semantic Sensor Networks*, Boston, MA, November 2012.
20. R. Heery and M. Patel. Application pro les: Mixing and matching metadata schemas. Ariadne, 25, September 2000. Available at: http://www.ariadne.ac.uk.
21. CAN in Automation e.V. CANopen—Communication Pro le for Industrial Systems, 2002. Available at: http://www.can-cia.de/downloads/.
22. D. Loy, D. Dietrich, and H.-J. Schweinzer (eds.). *Open Control Networks*, Kluwer Academic Publishing, Norwell, MA, October 2001.
23. International Electrotechnical Commission (IEC). Digital data communications for measurement and control—Fieldbus for use in industrial control systems—Part 1: Overview and guidance for the IEC 61158 series, April 2003.
24. Institute of Electrical and Electronics Engineers, Inc. Standard for a smart transducer interface for sensors and actuators—Network capable application processor (NCAP) information model, IEEE Std 1451.1-1999. June 1999.
25. Borst Automation. Device description language. e HART book, 9, May 1999. Available at: http://www.thehartbook.com/.
26. International Electrotechnical Commission (IEC). Function Blocks (FB) for process control—Part 2: Speci cation of FB concept and Electronic Device Description Language (EDDL). IEC Standard 61804-2:2004, 2004.
27. International Electrotechnical Commission (IEC). Function blocks (FB) for process control—Part 3: Electronic Device Description Language (EDDL). IEC 61804-3:2006, 2006.
28. J. Riegert. Field device tool—Mastering diversity & reducing complexity. *PROCESSWest*, pp. 46–48, 2005.

29. S. Pitzek and W. Elmenreich. Con guration and management of a real-time smart transducer network. In *Proceedings of the Ninth IEEE International Conference on Emerging Technologies and Factory Automation*, Lisbon, Portugal, September 2003, vol. 1, pp. 407–414.
30. World Wide Web Consortium (W3C). *Extensible Markup Language* (*XML*) *1.0*, 2nd edn., October 2000. Available at: http://www.w3.org.
31. M. Wollschläger. A framework for eldbus management using XML descriptions. In *Proceedings on the 2000 IEEE International Workshop on Factory Communication Systems* (*WFCS 2000*), Porto, Portugal, September 2000, pp. 3–10.
32. S. Eberle. XML-basierte Internetanbindung technischer Prozesse. In *Informatik 2000 Neue Horizonte im neuen Jahrhundert*, pp. 356–371, Springer-Verlag, Berlin, Germany, September 2000.
33. D. Bühler. e CANopen Markup Language—Representing eldbus data with XML. In *Proceedings of the 26th IEEE International Conference of the IEEE Industrial Electronics Society* (*IECON 2000*), Nagoya, Japan, October 2000.
34. W. Elmenreich, S. Pitzek, and M. Schlager. Modeling distributed embedded applications using an interface le system. *The Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Vienna, Austria, 2004.
35. G. Moritz, S. Prüter, D. Timmermann, and F. Golatowski. Real-time service-oriented communication protocols on resource constrained devices. In *International Multiconference on Computer Science and Information Technology* (*IMCSIT*), Wisla, Poland, IEEE, 2008, pp. 695–701.
36. S. Sakr. Xml compression techniques: A survey and comparison. *Journal of Computer and System Sciences*, 75(5):303–322, August 2009.
37. R. Kyusakov. Towards application of service oriented architecture in wireless sensor networks. PhD thesis, Luleå University of Technology, Luleå, Sweden, 2012.
38. ANSI/ISA-88.01. *Batch Control Part 1: Models and Terminology*, ANSI/ISA, Research Triangle Park, NC, December 1995.
39. J. Berge. *Fieldbuses for Process Control: Engineering, Operation, and Maintenance*, e Instrumentation, Systems, and Automation Society, Research Triangle Park, NC, 2002.
40. S. Poledna, H. Angelow, M. Glück, M. Pisecky, I. Smaili, G. Stöger, C. Tanzer, and G. Kroiss. TTP two level design approach: Tool support for composable fault-tolerant real-time systems. *SAE World Congress 2000*, Detroit, MI, March 2000.
41. P. Pleinevaux and J.-D. Decotignie. Time critical communication networks: Field buses. *IEEE Network*, 2(3):55–63, May 1998.
42. S. Cavalieri, S. Monforte, A. Corsaro, and G. Scapellato. Multicycle polling scheduling algorithms for eldbus networks. *Real-Time Systems*, 25(2–3):157–185, September–October 2003.
43. TTAGroup. *TTP Specification Version 1.1*, TTAGroup, Vienna, Austria, 2003. Available at: http://www.ttagroup.org.
44. T. Führer, F. Hartwich, R. Hugel, and H. Weiler. Flexray— e communication system for future control systems in vehicles. *SAE World Congress 2003*, Detroit, MI, March 2003.
45. BOSCH. *CAN Specification Version 2.0*, Robert Bosch GmbH, Stuttgart, Germany, 1991.
46. W. Elmenreich, W. Haidinger, P. Peti, and L. Schneider. New node integration for master-slave eldbus networks. In *Proceedings of the 20th IASTED International Conference on Applied Informatics* (*AI 2002*), Innsbruck, Austria, February 2002, pp. 173–178.
47. A.S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*, 2nd edn., Prentice-Hall, Inc., Upper Saddle River, NJ, 2006.
48. W. Dargie. *Context-Aware Computing and Self-Managing Systems*, 1st edn., Chapman & Hall/CRC, Boca Raton, FL, 2009.
49. M. Wollschläger, C. Diedrich, T. Bangemann, J. Müller, and U. Epple. Integration of eldbus systems into on-line asset management solutions based on eldbus pro le descriptions. In *Proceedings of the Fourth IEEE International Workshop on Factory Communication Systems*, Vasteras, Sweden, August 2002, pp. 89–96.

50. L. Bartram, A. Ho, J. Dill, and F. Henigman. e continuous zoom: A constrained sheye technique for viewing and navigating large information spaces. In *ACM Symposium on User Interface Software and Technology*, Pittsburgh, PA, 1995, pp. 207–215.
51. S. Poledna. *Fault-Tolerant Real-Time Systems—The Problem of Replica Determinism*, Kluwer Academic Publishers, Boston, MA/Dordrecht, the Netherlands/London, U.K., 1995.
52. G. Bauer. Transparent fault tolerance in a time-triggered architecture. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2001.