End-User Development Framework for Embedded System Applications

Miroslav Sveda
Brno University of Technology, Faculty of Information Technology
Czech Republic
sveda@fit.vutbr.cz

Abstract

This paper presents principles of an end-user development support for embedded system networking. The presented approach offers a design framework consisting of reusable patterns for Internet-based embedded system applications. The framework provides a development environment kernel that can be adapted for various embedded system application domains. It stems from (1) the IEEE 1451.1 smart transducer interface standard, which is an object-based networking model supporting group messaging, and (2) the Internet Protocol multicast communication, mediating efficient and unified access to distributed components through both wired and wireless intranets. As an example, this paper focuses on refinement of those concepts for smart sensor application development support and on their utilization for a gas pipes pressure measurement system as an application example. The paper brings this scheme in form suitable not only for framework builders, but also for end-user developers.

1. Introduction

The design framework, presented in this paper as a flexible design environment kernel stemming from meta-design conception, is rooted in the IEEE 1451.1 standard specifying smart transducer interface architecture. That standard provides an object-oriented information model targeting software-based, network independent, transducer application environments. The framework enables to unify interconnections of embedded system components through wireless networks and Ethernet-based intranets, which are replacing various special-purpose Field-busses in industrial applications [15].

Two additional technologies, namely publish-subscribe messaging and Internet Protocol (IP) multicasting, which offer scalable and traffic-saving solutions important in the context of contemporary Internet, complement the framework providing design patterns reusable for various networked embedded system applications. The schemes discussed can properly interplay with each other and can deliver suitable support for Internet-based embedded systems design. This paper focuses on utilization of this framework for a gas pipe-line pressure measurement system as a real-world application example, which comprises several groups of smart pressure and temperature sensors that clients can access effectively through Internet. Each sensor group is supported by an active web page that provides clients with transparent and efficient access to pressure measurement services.

The paper discusses in the following section metadesign approach utilizable for creating flexible design environment appropriate for various application domains of networked embedded systems. Next section reviews principles of the proposed design framework aiming at Internet-based embedded systems. Its three subsections introduce subsequently IEEE 1451 package of communication standards, client-server and publish-subscribe communication concepts, and IP multicasting as main components of the design pattern forming the kernel of the generic development environment. While the section 4 briefly introduces concepts of the flexible development environment that enhances the framework for networked smart sensor applications design, the section 5 presents in more detail an example of deployment of this framework for creating networked pressure measurement along gas pipes including some implementation details. The example covers network and node configuration and implementation concepts, including smart sensor implementation.

2. Meta-design for end-user development

Component-based development involves multiple roles [9]. Framework builders create the infrastructure for components to interact; developers identify suitable domains and develop new components for them; application assemblers select domain-specific components and assemble them into applications; and end users employ component-based applications to perform daily tasks. There is room for a fifth role in this pipe-line: end-user developers positioned between application assemblers and end users. These end-user developers are able to tailor applications at runtime because they have both domain expertise and technical know-how. They would interact with applications to adjust individual components, and modify existing assemblies of components to create new functionality. Furthermore, they can play a critical role
when component-based systems have to be redesigned for new requirements. End-user development activities can range from customization to component configuration and programming.

Meta-design characterizes objectives, techniques, and processes for creating new environments allowing end users to act as designers [4]. In all design processes, two basic stages can be differentiated: design time and use time. At design time, system developers create environments and tools. In conventional design they create complete systems. Because the needs, objectives, and situational contexts of users can only be anticipated at design time, users often find the system unfit for their tasks at use time. Thus, they require adaptation of the existing environment and tools for new applications. Meta-design extends the traditional notion of system development to include users in an ongoing process as co-designers, not only at design time but throughout the entire life-cycle of the development process. Rather than presenting users with closed development systems, meta-design provides them with concepts and tools to extend the system to fit their needs. Hence, meta-design promotes designing the design process.

Not surprisingly, meta-design relates to design in the same way as meta-modeling relates to modeling [2, 10]. But, while modeling and meta-modeling are identical activities with the only difference of interpretation, designing and meta-designing are targeted differently. Model, which is the object of modeling, remains an abstract notion in the similar sense as meta-model with the only exception of abstraction level. On the contrary, objects of design and meta-design differ: in the former case the process produces an artifact, in the latter case it is design of a design process including related development environment. By the way, model-based design and development – see the domain related papers [1, 6, 18] – create another glue in between designing and modeling.

This paper discusses a deployment of meta-design principles for building up a flexible design framework focused on embedded systems and their components interconnected by Internet. Necessarily under-designed open source tools and techniques create design spaces for end-user developers. The paper demonstrates both the use of this framework for implementation of development environments aimed at various Internet-based smart sensor applications and, concurrently, the utilization of this framework for development of pressure measurement along gas pipes.

3. Design framework

Main components of the framework, which forms kernel of the generic development environment, encompass IEEE 1451 package of communication standards, client-server and publish-subscribe communication patterns, and IP multicasting.

3.1. IEEE 1451 architecture

The IEEE 1451 package consists of the family of standards for a networked smart transducer interface that include namely (see Figure 1) (i) a smart transducer software architecture, 1451.1 [5], targeting software-based, network independent transducer applications, and (ii) a standard digital interface and communication protocol, IEEE 1451.2, for accessing the transducer or the group of transducers via a microprocessor modeled by the 1451.1 standard. The next three standard proposals extend the original hard-wired parallel interface 1451.2 to serial multi-drop 1451.3, mixed-mode (i.e. both digital and analogue) 1451.4, and wireless 1451.5 interfaces.

The 1451.1 software architecture provides three models of the transducer device environment: (i) the model of a network capable application processor (NCAP), which is the object-oriented embodiment of a smart networked device; (ii) the data model, which specifies information encoding rules for transmitting information across both local and remote object interfaces; and (iii) the network communication model, which supports client/server and publish/subscribe paradigms for communicating information between NCAPs. The standard defines a network and transducer hardware neutral environment in which a concrete sensor/actuator application can be developed.

![Figure 1: IEEE 1451 configuration example](image-url)
instantiation and deletion of concrete classes including attributes.

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

The Transducer Block abstracts all the capabilities of each transducer that is physically connected to the NCAP I/O system. During the device configuration phase, the description is read from the hardware device what kind of sensors and actuators are connected to the system. The Transducer Block includes an I/O device driver style interface for communication with the hardware. The I/O interface includes methods for reading and writing to the transducer from the application-based Function Block using a standardized interface. The I/O device driver provides both plug-and-play capability and hot-swap feature for transducers.

The Function Block provides a skeletal area in which to place application-specific code. The interface does not specify any restrictions on how an application is developed. In addition to a State variable that all block classes maintain, the Function Block contains several lists of parameters that are typically used to access network-visible data or to make internal data available remotely.

The Network Block abstracts all access to a network employing network-neutral, object-based programming interface supporting both client-server and publisher-subscriber patterns for configuration and data distribution.

The paper demonstrates use of the IEEE 1451.1 architecture as a building block of the framework, which is aimed at Internet-based smart sensor applications, and utilized for development of pressure measurement along gas pipes.

3.2. Communication patterns

The majority of communication protocols provide a client-server style of communication. In case of sensor communications, the client-server pattern covers both configuration of transducers and initialization actions. If the client wants to call some function on server side, it uses a command execute. On server side, this request is decoded and used by the function perform. That function evaluates the requested function with the given arguments and, after that, it returns the resulting values to the client.

The client-server pattern corresponds to remote procedure call (RPC), which is the remote invocation of operations in a distributed context [3]. To be more precise, the RPC interaction considered in this paper provides a synchronous client-server communication, i.e. the client is waiting for a server’s response before completion the RPC actions related to the current call. Evidently, the client-server communication style relates to point-to-point message passing called as unicast.

The subscriber-publisher style of communication, [3], can provide efficient distributions of measured data. All clients, wishing to receive messages from a transducer, register themselves to the group of its subscribers using the function subscribe. After that, when this transducer generates a message using the function publish, this message is effectively delivered to all members of its subscribing group.

The interaction publish-subscribe relates to point-to-multipoint or multipoint-to-multipoint message passing. While there is a possibility to implement multipoint using multiple point-to-point unicasts, it is much more efficient to utilize elaborate multicast techniques, namely multicast routing. The basic principles of the network layer multicast in the Internet environment are discussed in the following section.

3.3. Multicasting

Traditional network computing paradigm involves communication between two network nodes. However, emerging Internet applications require simultaneous group communication based on multipoint configuration propped e.g. by multicast IP, which saves bandwidth by forcing the network to replicate packets only when necessary. Multicast improves the efficiency of multipoint data distribution by building distribution trees from senders to sets of receivers [8].

The functions that provide the Standard Internet Multicast Service can be separated into host and network components. The interface between these components is provided by IP multicast addressing and Internet Group Management Protocol (IGMP) group membership functions, as well as standard IP packet transmission and reception. The network functions are principally concerned with multicast routing, while host functions can also include higher-layer tasks such as the addition of reliability facilities in a transport-layer protocol.

IP multicasting is the transmission of an IP datagram to a host group, a set of zero or more hosts identified by the single IP destination address of class D. Multicast groups are maintained by IGMP (IETF RFC 1112, RFC 2236). Multicast routing considers multicasting routers equipped with multicast routing protocols such as DVMRP (RFC 1075), MOSPF (RFC 1584), CBT (RFC 2189), PIM-DM (RFC 2117), PIM-SM (RFC 2362), or MBGP (RFC 2283). For Ethernet-based Intranets, the Address Resolution Protocol provides the last-hop routing by mapping class D addresses on multicast Ethernet addresses. Certainly, the routing protocol architecture must
be extended by wireless network routing protocols in case multi-hop sensor networks complement the wired Internet.

The paper demonstrates use of above mentioned communication patterns supported by IP multicast for implementation of communication means aimed at Internet-based smart sensor applications, and describes how to apply them for pressure measurement along gas pipes.

4. Development environment

Development systems have to support important concepts and methods by their tools for complete design and development life cycle of applications belonging to considered application domains. The toolset related to the discussed design framework includes also original tools targeting primarily front-end parts of specification and design, namely formal specifications and rapid prototyping. Those tools and related techniques were discussed in frame of papers that describe the design principles of sensor-based applications in various application domains, such as smart sensor applications [13] and industrial monitoring and measurements [14, 15, 17].

More detailed information about original techniques and tools including supporting principles can be found in the PhD theses, which were prepared in frame of the related research dealing with timed system specifications [7], object-oriented specifications [11], and reverse specifications and modeling [12]. A paper devoted to the flexible development environment and its adaptation for safety and security-critical sensor-based applications is currently under preparation.

5. Design case study

This section demonstrates the above-introduced concepts applied to the design of a gas-pipes pressure measurement system [14, 16]. The developed system comprises several groups of smart pressure and temperature sensors, interconnected by wired intranets and/or wireless networks that clients can access effectively through Internet.

Each sensors group is supported by an active web page with Java applets that, after downloading, provide clients with transparent and efficient access to pressure measurement services. In this case, clients communicate to transducers using a messaging protocol defined by client-server and subscriber-publisher patterns employing 1451.1 Network Block functions. A typical configuration includes a set of smart pressure sensors generating pressure values for the users of those values. To register itself for a specified group of sensors, the user — playing the role of either subscriber or client — opens the related server’s web page with the relevant Java applet. This applet is, after uploading to the subscriber/client site, started on subscriber/client’s computer, which launches communications with a group of transducers allowing Java clients to connect and subscribe to the smart sensors. Java can directly support both client-server and subscriber-publisher application architectures as the core Java specifications include TCP/IP and UDP/IP networking APIs.

The developed Java applet uses the core java.net package to implement both client-server and subscriber-publisher application distribution allowing to access smart sensors and supporting nodes. The applet consists of a series of object classes, including multi-threaded applet environment, animation, and UDP/IP-based subscriber and TCP/IP-based client communications. The subscriber/client software implemented in Java enables applets to be included in a web server HTML page, and run under a regular web browser on subscriber/client side. The subscriber/client communicates with the transducer by standard UDP/TCP sockets employing IP multicast.

The communication scheme applies multicast both for distributing measured values from a transducer to a group of subscribers/clients registered by the web server for this transducer, and for spreading commands of a client to a group of transducers registered for this client.

5.2. System implementation concepts

In the transducer’s 1451.1 object model, basic Network Block functions initialize and cover communication between a client and the transducer, which are identified by unique unicast IP addresses. The client-server style communication, which in this application covers both the configurations of transducers and initialization actions, is provided by two basic Network Block functions: execute and perform.
The standard defines a unique ID for every function and data item of each class. If the client wants to call some function on server side, it uses command `execute` with the following parameters: ID of requested function, enumerated arguments, and requested variables. On server side, this request is decoded and used by the function `perform`. That function evaluates the requested function with the given arguments and, in addition, it returns the resulting values to the client. Those data are delivered by requested variables in `execute` arguments.

The subscriber-publisher style of communication, which in this application covers primarily distribution of measured data, but also distribution of group configuration commands, employs IP multicasting. All clients wishing to receive messages from a transducer, which is joined with an IP multicast address of class D, register themselves to this group using IGMP. After that, when this transducer generates a message by `publish`, this message is effectively delivered to all members of this class D group, without unnecessary replications and repeated transmissions.

The Network Block abstracts all access to a network employing network-neutral, object-based programming interface. The network model provides an application interaction mechanism supporting both client-server and publisher-subscriber paradigms for event and message generation and distribution.

5.3. Node configuration

The primary communication scheme, which is based on publish-subscribe pattern, applies multicast both for distributing measured values from a transducer to a group of clients registered by the www server for this transducer, and for spreading commands of a client to a group of transducers registered for this client. Those commands can specify e.g. individual subgroup’s sampling frequencies and/or events for launching irregular publishing such as a limit value crossing.

A typical node, depicted on Figures 2 and 3, consists of STIM (Smart Transducer Interface Module) connected with PSD sensor for pressure measurements, and with auxiliary temperature sensor for signal conditioning. Of course, NCAP can be either embedded in a complex smart sensor, or shared among more simple smart sensors. On the other hand, from the viewpoint of Internet, only NCAP is directly addressable being equipped by its own IP address. Therefore, we can also denote as smart sensor the device consisting of an NCAP accessing one or more STIMs with connected sensors.

To register itself for a specified group of sensors, the client opens a related server’s web page with the relevant Java applet. This applet is, after uploading to the client site, started on client’s computer, what launches communication with the dedicated group of transducers.

5.4. Smart sensor implementation

This subsection discusses, as an example, the pressure sensors with reflected laser beam and diffractive lens. The sensitive pressure sensor is based on a nitride membrane and an optoelectronic read-out subsystem. Measured pressure values are transformed into related thick-layer nitride membrane deflections. The nitride membrane serves as a mirror for laser beam, and it can move the related reflected laser mark. The mark’s position is sensed using position-sensing device, which is a fotolateral diode. Diode double current signal is amplified and conditioned digitally by the ADuC812 microcontroller. This microcontroller provides also the IEEE 1451.2 interface.

The sensing subsystem combines two principles that provide both high precision and wide range pressure measurements. Large displacements are measured by the position of reflected focused laser beam. Small position changes are measured by one-side layer diffractive lens principle. Sensor output signal is conditioned in digital by the ADuC812 single-chip microcontroller, which pro-
the light spot and converts that position on the measured pressure using an internal table. Figure 3 depicts principles of the implementation of that smart sensor. The STIM contains (1) a PSD sensor with two analog differential transducers (XDCR), (2) a microcontroller ADuC812 with nonvolatile memory containing a TEDS field (Transducer Electronic Data Sheet) that props IEEE 1451.2 storing sensor specifications, (3) a TII (Transducer Independent Interface), (4) a temperature sensor necessary for signal conditioning, (5) an analogue-to-digital conversion units (ADC), and (6) a logic circuitry to facilitate communication between STIM and NCAP.

The ADuC812 microcontroller, the basic building block of the smart pressure sensor electronics, includes on-chip high performance multiplexers, ADCs, DACs, FLASH program and data storage memory, an industrial standard 8052 microcontroller core, and supports several standard serial ports. The microcontroller may also utilize nonvolatile memory containing a TEDS field and ten-wire TII that prop IEEE 1451.2.

### 5.4. Design pattern

In conclusion of this case study it could be helpful to restate the key refinements of the proposed framework through building up a design pattern based on its basic abstract components, i.e. IEEE 1451 architecture, communication procedures and IP multicasting, which introduce a more detailed structure reusable for concrete applications.

The 1451.1 object model provides skeleton supporting individual components. Its Network Block is refined so that it enables to access data-link layer communication services through unicast on IP with client-server procedure for start-up configuration and run-time maintenance, or through IP multicast with publish-subscribe procedure for run-time process measurements on both application data users and transducers sides. This refinement covers also selection of the most appropriate multicast routing protocol for local Internet traffic in case when the relevant parts of the network are accessible through routers.

The Transducer Block includes methods for reading and writing to transducers from the application-based Function Block using the standardized interfaces. The I/O device driver provides both plug-and-play capability and hot-swap feature for each transducer. It enables run-time reconfiguration of sensors that can support robustness of the system or improve measurement efficiency.

The Function Block contains measurement application code. In the current case it prescribes sampling times, data filtering, linearization, conversions and transformations improving measurement accuracy and stability. The Function Block contains lists of parameters used to access network-visible data and, concurrently, to make internal data available remotely. The Function Block enables in this case to compute pressure profiles along the pipeline, pressure or temperature gradients, and the speed of pressure or temperature changes in time.

The measurement system arrangement along a pipeline is depicted on the Figure 4.

![Figure 4. The measurement system network arrangement](image-url)

This figure shows active nodes participating in data exchange on an intranet in frame of client (Client A1…A3) – server (Web server) configuration for client registration, and publisher (Smart Sensor 1…3) – subscribers (a subset of Clients A1…A3 and B) configuration for measured data transmissions.

### 6. Conclusions

This paper presents principles of the end-user development support for embedded system networking in form of the developed design framework. The approach supported offers a reusable pattern for Internet-based embedded system applications. It stems from (1) the IEEE 1451.1 smart transducer interface standard, which is an object-based networking model supporting group messaging, and from (2) the Internet Protocol multicast communication, mediating efficient and unified access to distributed components through both wired and wireless intra-
nets. As an example, this paper discusses adaptations and refinement of this framework for smart sensor application development support and its utilization for a gas pipes pressure measurement system. The paper brings this approach in manner suitable not only for framework builders, but also for end-user developers.

From more general viewpoint, the paper discusses deployment of meta-design principles for creating flexible design environments. Necessarily under-designed open source tools and techniques create design spaces for end-user developers in various application domains of networked embedded systems.

Acknowledgements

The research has been supported by the Czech Ministry of Education in frame of the Research Intention MSM 0021630528: Security-Oriented Research in Information Technology, and by the Grant Agency of the Czech Republic through the grants GACR 102/05/0723: A Framework for Formal Specifications and Prototyping of Information System’s Network Applications and GACR 102/05/0467: Architectures of Embedded Systems Networks. The author appreciates contributions of his colleagues Radimir Vrba, Ondrej Rysavy, Frantisek Scuglik, and Petr Matousek from the Brno University of Technology to this work.

References