DOCTORAL THESIS

Efficient Web Services for End-To-End Interoperability of Embedded Systems

Rumen Kyusakov

LULEÅ UNIVERSITY OF TECHNOLOGY

Efficient Web Services for End-To-End Interoperability of Embedded Systems

Rumen Kyusakov

EISLAB
Department of Computer Science, Electrical and Space Engineering
Luleå University of Technology
Luleå, Sweden

Supervisors:

Jens Eliasson, Jerker Delsing

Printed by Luleå University of Technology, Graphic Production 2014

ISSN 1402-1544
ISBN 978-91-7583-015-5 (print)
ISBN 978-91-7583-016-2 (pdf)

Luleå 2014

www.ltu.se

To Mimi and Vladi ...

ABSTRACT

As the number of Internet-connected devices rapidly grows, it has become ever more challenging to develop and maintain purpose-made tightly integrated distributed embedded systems. Instead, the Internet of Things (IoT) approach, based on standardized interfaces and open communication protocols, enables support for various applications with the possibility of extension to provide additional services that were not necessarily available at the initial deployment.

This thesis presents methods and tools for the development of standard-based Web services for the Internet of Things. Some of the key challenges in using Web services on resource-constrained devices are due to the overhead of the communication protocols, which leads to the need for greater network bandwidth, processing power, and memory usage. A common solution to these problems is to use gateways that translate between the protocols used on one end of the connection (i.e. low-capability devices) and those on the other end (i.e. powerful Web servers). However, this increases the overall complexity of the system.

The work presented herein answers the following research questions: 1) Is it feasible to deploy standard Web services on IoT systems without using application layer gateways? 2) What are the trade-offs in using Web services for end-to-end interoperability of resource-constrained embedded systems? 3) What levels of efficiency and functionality can be achieved using binary coding schemes for XML data exchange?

The research questions are tested by building and evaluating several prototype IoT systems. These evaluations show that the use of Web services requires more powerful hardware (i.e. CPU and RAM) and a larger form factor in exchange for better interoperability compared to the use of ad hoc application protocols. The main challenge in employing embedded Web services is the large size of the messages, which is due to the use of verbose data formats such as XML. Although it is shown that it is possible to deploy XML-based Web services on low-capability devices without application layer gateways, this approach has severe performance limitations. Using the Efficient XML Interchange (EXI) binary coding scheme overcomes some of these limitations by substantially reducing the size of the XML messages. The main outcome of this thesis is the design and implementation of a software toolkit, called EXIP, for building EXI-based embedded Web services.

The trade-offs in the use of embedded Web services are likely to change in the near future as the importance of application layer interoperability increases and IoT devices become faster, more energy efficient, and equipped with more memory. The dominating importance of interoperability can be seen in highly heterogeneous systems such as energy management systems (i.e. smart grids), where embedded Web services are already in use today. With this in mind, future research directions and extensions of this work include the development of performance optimization strategies for the EXIP toolkit to foster the expansion of embedded Web services to an even wider range of IoT applications.

Contents

Acknowledgments

This thesis would not have been possible without the help, support, and guidance of many people to whom I wish to express my utmost gratitude. First, I would like to thank my supervisors, Associate Professor Jens Eliasson and Professor Jerker Delsing. I am very grateful for all the invaluable help, feedback, guidance, and patience that Jens has given to me throughout the years of my PhD studies. I am also grateful to Jerker for his constant support and guidance.

Next, I would like to thank all my friends and colleagues at EISLAB, as well as in the Department of Computer Science, Electrical and Space Engineering in Luleå, for the inspiring discussions and the pleasant and fruitful working environment. Special thanks go to Pablo Puñal Pereira for the positive and rewarding collaboration, to Blerim Emruli for the stimulating scholarly talks that we have had throughout the years, and to Henrik Mäkitaavola and Jonas Gustafsson for their help and support.

I would also like to thank the members of the W3C Efficient XML Interchange working group for their valuable comments and feedback as well as Robert Cragie for his advice and help on the Smart Energy Profile specification.

I am grateful for the affection, understanding, and unconditional support that I have received from my family and friends. Thank you all!

Finally, I must acknowledge the financial support from the European Commission that made this thesis possible. I would also like to thank the partners from the IMC-AESOP FP7 project and the ARTEMIS Innovation Pilot Project Arrowhead for their support and fruitful collaboration.

Luleå, October 2014
Rumen V. Kyusakov

Part I

Chapter 1

Introduction

In today's technology age, people's lives, as well as business processes and industry, are increasingly dependent on constant Internet connectivity. This dependency is often fueled by the need for rapid innovation and flexibility in various business processes (e.g., e-commerce and the digital economy), higher productivity and quality assurance in industrial sites (e.g., enterprise resource planning), and convenience and efficiency in private users' social interactions (e.g., e-mail, messaging, and social networking). The free flow of information and universal connectivity are what make the Internet so disruptive for our everyday lives and society in general.

Advances in microelectronics have paved the way for extending the reach of the Internet to the everyday objects that surround us (e.g., wearable electronics, smart home appliances, consumer and industrial sensors and actuators) - a concept know as the Internet of Things (IoT). The idea of computing devices seamlessly integrated into our surroundings, i.e. ubiquitous or pervasive computing, is not new and can be traced back to the work of Mark Weiser in the late 1980s and early 1990s [1]. Adding computing capabilities to everyday objects is a foreseeable source of innovation in many fields, such as environmental monitoring, healthcare, agriculture, home and industrial automation. In this context, the Internet of Things is a very broad research area with a focus on using Internet protocols and communication infrastructure to interconnect ubiquitous computing devices - an envisioned disruptive technology of the near future. Already today, the number of connected devices increases rapidly fueled by the demand for consumer electronics and condition monitoring equipment for the industry. According to Vermesan et al. [2], this growth will lead to over 30 billion connected devices by 2020 with significant impact on a large number of industries. The main driving force behind the IoT is the promise of a unified communication interface to make it easy to interconnect distributed embedded systems with the rest of the IT infrastructure.

The interoperability between connected systems strictly depends on the use of standards defining how the systems interact with each other. The description of these interactions should be unambiguous, and preferably open so that the interoperability is not restricted to devices and systems from a particular vendor. Multiple standards developing organizations (SDOs) are leading the effort to define the set of specifications for the future Internet of Things. Because interoperability in the IoT is one of the main research areas in this thesis, the author has followed the standardization activities of the Internet Engineering Task Force (IETF) regarding networking protocols as well as the latest developments in Web service technology of the Organization for the Advancement of Structured Information Standards (OASIS). Moreover, some parts of the thesis work have been performed in collaboration with the binary XML working group of the World Wide Web Consortium (W3C). The motivation behind aligning the thesis work with the latest standardization efforts in the field is to make the research results readily available during the technology evaluation phase of the standardization process and therefore to increase their impact on the development of communication protocols for the IoT.

Interoperability can be achieved on different network layers of the Open Systems Interconnection (OSI) model. It is almost universally agreed today that by providing interoperability on the network layer, IPv6 is one of the enabling technologies for the IoT. IPv6 allows devices to be uniquely addressed and available on the Internet. In the embedded domain, the transition from ad-hoc network protocols to IP and then the convergence to IPv6 did not occur without challenges, because many believed that IP was too large to fit into a memory-constrained device, as noted by Durvy et al. [3]. The limited memory and low bandwidth of some widely adopted low-power wireless technologies required the development of a new adaptation layer and the optimization of the related standards, commonly known as 6LoWPAN [4]. The 6LoWPAN standard enables the efficient use of IPv6 over low-power, low-rate wireless networks on simple embedded devices.

Having network layer interoperability is a big step forward for the IoT ecosystem with embedded devices from various applications being able to exchange data packets with other devices as well as with mainstream Internet hosts. Nevertheless, the use of proprietary or custom-made transport and application protocols implies that further interoperability efforts are needed on the upper OSI layers. Without the unification of these protocols, communication between heterogeneous systems requires gateways to translate between the different IP-based solutions. As the need for cross-application connectivity has increased, and with many application protocols in use even within a single application domain, it has become intractable to develop and maintain application layer gateways capable of translating among various protocols and protocol versions.

Enterprise systems, as well as the more capable embedded distributed systems, often rely on open and loosely coupled architectures such as the Service-Oriented Architecture (SOA) to overcome these challenges. The benefits of using a unified communication interface for the layers above IP include the ease of integration between different hardware and software platforms on one hand and avoiding vendor lock for end users on the other hand.

Web service technology, as a concrete implementation of the SOA principles, has already proven successful in providing application layer interoperability for enterprise applications and in supporting scalability and ease of maintenance through the use of standard Internet protocols. Applying the same technology to the IoT class of devices (sometimes referred to as the Web of Things) would provide immediate benefits when close integration is required between the enterprise and embedded systems. The lack of integration between these systems is a key problem in many industries that has been carefully analyzed in the IMC-AESOP[1] and Arrowhead[2] research projects. The work in this thesis has been performed under the scope of, and following the requirements of, the IMC-AESOP and Arrowhead projects.

A key technology requirement, based on the use cases of the two projects, is support for the cross-layer vertical integration of a large number of devices and systems, as shown in Figure 1.1. The ability to combine services and resources from different layers (e.g., the plant floor, the operational and the enterprise layers) is a key element in the transition from static, single-purpose manufacturing and processing systems to multi-functional and evolvable industrial processes. Moreover, the exchange of information, resources, and services among all layers of an industrial enterprise enables new means of process optimization and efficiency improvement [5].

[1]Architecture for service-oriented process monitoring and control: http://www.imc-aesop.eu/

[2]ARTEMIS innovation pilot project: http://www.arrowhead.eu/

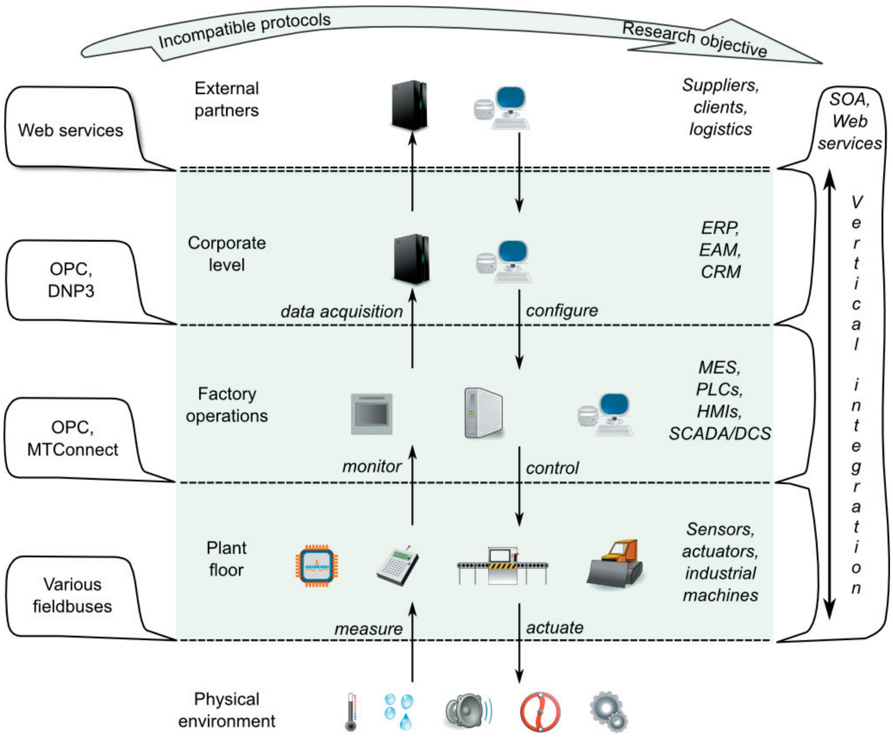


Figure 1.1: A key research objective of the IMC-AESOP and Arrowhead projects is the cross-layer vertical integration of a large number of devices and systems in industrial enterprises.

1.1 Problem Formulation

The use of standard Web technologies for sensor/actuator networks and other IoT applications has been shown to be a beneficial approach for solving many application layer interoperability issues (e.g., see the work of Barisic et al. [6]). However, the overhead of using Web service protocols is significant. As an example, compared to character-separated values (CSV), the Simple Object Access Protocol (SOAP), which is often used for enterprise Web services, uses messages that are at least 5 times larger, take 2.5 times longer to transmit, and overall consume 2.5 times more energy on a sample battery-powered sensor platform, as shown by Groba et al. [7].

The overhead of Web protocols has led to the design of various gateway architectures that mediate interactions with IoT devices (e.g., [8]). This approach shifts the computational and networking load onto more powerful platforms to solve the overhead problem. However, it significantly increases the overall complexity of the system because it violates the fundamental end-to-end principle in computer networking (defined by Saltzer et al. in [9]), which states that application-specific functions ought to reside in the end hosts of a network rather than in intermediary nodes.

The gap between solving the overhead problem and adhering to the end-to-end principle raises the first research question in this thesis:

1. *Is it feasible to deploy standard Web services on IoT systems without using application layer gateways?*

The answer to this question is more complex than a simple yes or no because 1) IoT systems have different requirements that must be taken into account, and 2) many variables affect the outcome, including the different software and hardware components used in each IoT system and the various protocols and data formats under consideration. This leads to the second research question:

2. *What are the trade-offs in using Web services for end-to-end interoperability of resource constrained embedded systems?*

Analyses of the challenges and trade-offs in Web service technologies for IoT have shown that the size of the payload of the exchanged messages has the most significant negative impact on the performance and overall applicability of standard Web protocols. These large payloads are attributed to the use of verbose data formats such as XML, which raises the third question in this thesis:

3. *What levels of efficiency and functionality can be achieved using binary coding schemes for XML data exchange?*

The solution to this research problem should include an analysis of which binary coding scheme is best fitted for use for payload encoding based on a set of criteria extracted from the IoT application requirements. The analysis should highlight any limitations to the use of this binary XML approach and propose concrete methods to overcome the challenges to its implementation in the development of embedded Web services. It is also desired to determine how the properties of this binary XML data format support the specific functionality of Web service technology, such as platform-independent interface, dynamic discovery and composition of services.

1.2 Research Methodology

The work performed in this thesis falls into the category of experimental computer science and engineering (ECSE) research, which can be broadly defined as ”the building of, or the experimentation with or on, nontrivial hardware or software systems” [10]. The research questions were tested by decomposing them into subproblems so that each could be solved separately and reassembled to achieve an overall solution. The goal was to contain the complexity of the research problems without building mathematical models that would require the introduction of oversimplifying assumptions.

The main research methodology that was used to test the decomposed hypotheses was based on building and evaluating small-scale prototypes of IoT systems. The evaluation of these prototypes helped to refine the initial research question (research question **1** of this thesis) and to raise new questions to be solved (research questions **2** and **3**). This methodology follows from the principle for good experimental work stated by Bob Taylor in 1991 at the Workshop on Research in Experimental Computer Science, Palo Alto, CA: *you should build what you design and use what you build as only through the extensive use of an artifact you truly understand the implications of your work* [11]. The main drawback in applying this principle is the substantial engineering work that is required to build even a simple IoT system prototype from scratch. For that reason, the prototypes were assembled from already available hardware and software components to the extent possible, and only the specific parts and designs that were tested in the experiments were built from scratch.

1.3 Thesis Scope

The Internet of Things is a very broad research field with many subdisciplines - from hardware design, energy harvesting, and wireless and network technologies to big data, sensor fusion, and pattern recognition in sensor networks. One of the challenges in performing experimental research in IoT is the need for knowledge and understanding in many of these fields for the development of relevant testbeds reflecting the state of the art in IoT research. To perform deep but narrow exploration in the field, the work in this thesis was performed in close collaboration with other researchers and industrial partners from different IoT subdisciplines.

The focus of this thesis is on application layer protocols for the Internet of Things domain. More specifically, the thesis investigates the use of standard Web protocols and data formats to enable end-to-end interoperability between embedded devices and Web services on the Internet. In this context, relevant research areas that are not within the scope of this thesis include security mechanisms as well as media access control (MAC), networking, and routing protocols for IoT. Furthermore, the main focus is on syntax-level interoperability, and research on semantic schemes beyond the use of statically defined syntax-to-semantic mappings is also beyond the scope of the thesis.

A substantial part of this work involves developing new methods and tools for XML compaction because XML is a key technology in the Web service protocols. Additionally, XML has a major impact on Web service overhead because it inflates the payload of the messages. Consequently, the results from this thesis are relevant to broader IoT fields such as power efficiency, encryption, and scalability, that depend on the size of the messages .

1.4 Thesis Outline

This compilation thesis consists of two parts. Part I introduces the research area, describes the research methodology, and formulates the main research questions of the thesis work. Part II consists of seven appended papers that contain the core research contributions of this thesis. All appended papers have been published or accepted for publication in peer-reviewed scientific journals or conferences. The appended papers have been reformatted to match the layout of this thesis without changing their contents.

The remainder of Part I is structured as follows. Chapter 2 provides an overview and a survey of the state of the art of embedded Web service technologies and their application in the energy management domain. It also provides a high level evaluation of different encoding formats and application protocols for the Internet of Things. Chapter 3 presents the theoretical foundation for several XML compaction techniques and describes their implementation in the Efficient XML Interchange format. Chapter 4 summarizes the research contributions of this thesis and describes how the formulated research questions have been addressed in the appended papers. Chapter 5 concludes Part I of the thesis by evaluating the thesis results against the real-world problem that is the motivation for this research. Additionally, Chapter 5 discusses the impact and implications of future trends in computing hardware on the thesis results and highlights possible future research directions.

CHAPTER 2

Embedded Web Services

A Web service is a well defined and self-contained function or computational resource that is available on the network by means of open network protocols and data encoding standards already in use on the Web. It is by far the most widely adopted implementation of the Service-Oriented Architecture [12].

The Service-Oriented Architecture is an abstract definition of a distributed computing approach for interconnecting heterogeneous systems and devices that promotes loose coupling among the components of the system. According to the SOA principles, the system components should provide vendor- and platform-independent interfaces that support service composition and dynamic discovery of services. The implementation of these principles requires the use of standard means of describing the SOA interface as well as standard protocols for message passing and data serialization. Figure 2.1 shows the abstract components and their roles as defined in the SOA. A service provider is an entity (usually one or more hosts) that makes a computational resource available on the network. The description of how to remotely access this computational resource is contained in the service provider interface through the use of an interface description language (IDL). The interface descriptions for the available services on the network can be stored in a centralized registry, provided on request, or distributed across the network. A service consumer is an entity that accesses the computational resource by following the message exchange patterns and syntactic rules defined in the interface description for that resource.

A key advantage of Service-Oriented Architecture is that it enables syntactic interoperability of remote network resources and functions, which are available for network discovery, reuse, and composition by authorized entities regardless of their geographical location. Based on these principles, predefined generic services can be used to implement a concrete SOA-based development framework, which can further foster the interoperability between interconnected systems. This approach is taken by the Arrowhead framework [13], which defines design patterns, development guidelines, and tools as well as a set of core generic services used for discovery, authorization, configuration, and orchestration. The use of standard Web services even by constrained devices and systems, guarantees conformance to the service specifications defined in the Arrowhead framework without the need of gateways and service buses.

Implementing SOA and Web services for syntactic interoperability of networked embedded systems is challenging due to the constraints of the devices and the network. These constraints necessitate the use of small messages and event-based communication as well as the availability of lightweight

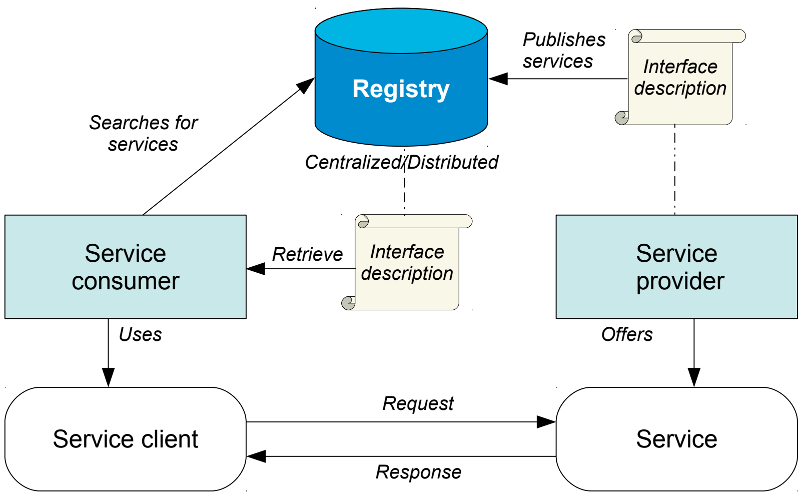


Figure 2.1: Abstract components in the Service-Oriented Architecture

implementations on the end nodes i.e. implementations requiring a small number of CPU cycles and low volatile and non-volatile memory usage. As in the early experiments with IPv6 on sensor/actuator networks, implementing standard Web services on resource-constrained devices can be experienced as impractical due to the inability of the Web service protocols to meet the system requirements. Following the IPv6 analogy with the introduction of 6LoWPAN, complementary networking standards and data formats as well as new implementation strategies are currently under development to achieve end-to-end application layer interoperability.

The following sections define high level requirements of embedded Web service technology, which are derived from IoT application domains, and also introduce the state of the art in IoT networking protocols, Web service standards, and data formats, which are relevant for the application of embedded Web services.

2.1 Requirements

The diversity of Internet of Things applications makes it difficult to generalize the requirements for embedded Web services. Not only the volume and complexity of information exchange varies greatly from application to application, but also the expected battery lifetime, installation topology, communication latency, and real-time properties (i.e. deterministic response time). Many factors affect these requirements, including some specifics of IoT installations (e.g., electromagnetic interference, multihop communications, device proximity). Nevertheless, presenting a summary of the expected properties that communication protocols for the IoT should have, is useful because it highlights some basic guidelines for the design, implementation, and evaluation of concrete embedded Web service solutions.

The requirements presented in this section are divided into two groups: general and application specific. The general requirements are valid for all application domains and are not quantitative, but instead they define non-functional properties that are difficult to measure and test for compliance. The application specific requirements are directly derived from the concrete application needs and can be tested quantitatively. The following list presents the general requirements corresponding to the desired properties of IoT installations (according to Fantana et al. [14]):

Interoperability It reflects the ability to connect different devices and systems (possibly from different vendors) with minimum amount of configuration and human intervention. Interoperability is fostered by conforming to communication standards and protocols and is a key challenge in IoT applications due to the segmentation of the communication technologies.

Reliability and robustness Dependable functionality provided by the Web service technology is required to guarantee continuous operation of the distributed systems or at least heavily minimize the risk of failure. This requires predictable timing and availability that can be verified by testing or using formal validation. Consequently, it is advantageous to reduce the implementation complexity of communication solutions because this will make it easier to validate their design and reliability of software implementations.

Cost The cost of a distributed system based on Web services is a compound property that depends on many parameters, such as hardware, software, licensing fees, system deployment, and maintenance. This makes the cost estimation complex task because there is a trade-off between these parameters that must be weighed against the benefits they provide, e.g., more expensive hardware and software may lead to cheaper deployment and maintenance, and also more capable software that leads to reduction in system maintenance cost might require more expensive hardware.

Security Most IoT application are sensitive to well defined access control, authentication, and authorization, which guarantees the source of the data and its validity. In many application (e.g., wearable electronics and smart home appliances) data confidentiality is also very important.

Development tools A proper tool support, which helps to automate repeating and predictable tasks of development, deployment, and maintenance processes, is required for successful adoption of any communication technology for the IoT.

Flexibility Embedded Web services should be able to model and implement evolvable distributed systems that are capable of being dynamically modified and extended with new functionality in a non predefined manner.

More concrete requirements for different application domains can be found in the literature. The rest of this section summarizes the communication layer requirements for energy management, home- and industrial automation applications.

Energy management: One set of requirements are related to the support of specific functionality and exchange patterns by the communication protocols. As an example, RFC6988 lists the services that the specifications for energy management of connected devices should provide [15]. These services include identification and meta-information exchange for managed entities as well as monitoring and control of their energy consumption. In addition, Bouhafs et al. [16] and Wang and Leung [17] define some general properties of the communication layer for smart grids and advanced metering infrastructure (AMI) applications. These desired properties are near real-time support, scalability, and decentralized data exchange.

Home automation: Kovatsch et al. [18] define a set of requirements for home automation applications that are largely overlapping with the aforementioned general requirements, e.g., future-proof, low cost, low installation overhead, interoperability, and security. Callaway et al. [19] present expected data rates and acceptable message latency in some use cases of home automation. According to that study, the expected data rates vary between 10 kb/s and 115 kb/s and the acceptable latency vary between 15 ms and 100 ms for home automation and consumer electronics applications. The acceptable latency is much higher for heating, ventilation, and air conditioning (HVAC) due to the thermal mass of the buildings and the slow ventilation process.

Industrial automation: The industrial networks are especially sensitive to reliability, robustness, and security requirements. The constraints on the use of embedded Web services in condition monitoring applications are relatively relaxed as the timing is not critically important for process diagnostic. However, the application of Web services in distributed control systems (DCS) poses strict real-time constraints on the communication system with acceptable delays between 10 ms and 1000 ms for continuous process automation [20] and between 5 ms and 50 ms for discrete manufacturing automation [21].

2.2 Network Protocols

The OSI network model defines seven abstraction layers such that each layer provides services to the layer above and receives services from the layer below. The abstraction layers reduce the complexity of the overall architecture by providing generalization of the functionality of the modules in a communication system and hiding the implementation details. The layers in the OSI model, and even in the Internet protocol suite model [22], do not strictly represent the functionality provided by the concrete network protocols but rather are a tool for visualizing the relationship between different components and validating the design of a particular networking stack. Table 2.1 contains a brief description of the functionality provided in each layer and shows how the communication protocols used in the experimental work of this thesis relate to the abstraction layers.

Table 2.1: Network protocols and technologies used in the IoT prototypes developed as part of this thesis, in relation to the OSI layers

OSI model		**Communication protocols**
Layer	**Provided functions**	
Application	Addressing and accessing application resources	SOAP, DPWS, WS-*, CoAP
Presentation	Data representation, encryption/decryption	Plain text, XML, EXI
Session	Managing sessions	
Transport	End-to-end connections, reliability, flow control	TCP, UDP
Network	Path determination and logical addressing	IPv6, 6LoWPAN, RPL
Data link	Physical addressing, medium access control	Bluetooth, IEEE 802.15.4
Physical	Physical media and signal transmission	

In contrast to traditional wired networks, the boundaries between the layers in IoT networks are often blurred, and therefore, information and services can be passed vertically across multiple layers to achieve better performance. This creates cross-layer dependencies that make it difficult to create and evaluate improvements and extensions to certain parts of the networking stack. The limitations and constraints of the underlying networking protocols have direct implications for the properties of different Web service technologies. Additionally, many of the IoT communication protocols are under active development, which can cause interoperability problems between different implementations of the same communication stack, as shown by Ko et al. in [23]. All these factors must be taken into account when performing evaluations of IoT systems. For that reason, the following paragraphs present an overview of important related work in the field of networking technologies for resource-constrained embedded systems.

The main requirements for IoT network protocols are low equipment cost, ease of installation, and high energy efficiency. In some industrial embedded networks, quality of service and real-time guarantees are prioritized over the cost of the equipment. Networking technologies optimized for meeting these requirements have lower throughput and a higher link latency than conventional Internet networks, which are primarily optimized for high throughput and low latency. This relates to the differences between IoT and conventional Internet traffic. In the IoT domain, the data sent over the network consist of short messages such as sensor readings and diagnostics, configuration parameters, and human-machine interface data. Conventional Web traffic is composed of large messages such as database queries, video streaming, and interactive web pages. Consequently, the throughput and latency related to the physical, data link, network and transport layers are the most important network properties for embedded Web services. To achieve end-to-end interoperability among heterogeneous systems (e.g., enterprise systems, hand-held and IoT devices), the performance of the application layer protocols used to implement embedded Web services should not be heavily dependent on the network throughput and latency.

Promising technologies for the IoT in the physical and data link layers are IEEE 802.15.4 wireless radio [24], Bluetooth (classic and low energy) [25], and power-line communication technologies such as G3-PLC [26]. The theoretical data rates for these technologies vary depending on their configuration (e.g., the operating frequency band), but they are in the range of 250 kbit/s for IEEE 802.15.4 and G3-PLC to 1 Mbit/s for Bluetooth. The application layer throughput is usually an order of magnitude smaller than the theoretical data link rate, and the throughput depends heavily on packet fragmentation, electromagnetic interference, and the distance and number of hops between hosts.

The maximum transmission unit (MTU) for IPv6 packets is 1280 bytes. However, IEEE 802.15.4 and G3-PLC frames are 127 and 251 bytes, respectively. This requires the use of an IPv6 adaptation layer to define the methods for IPv6 header compression and packet fragmentation, which are specified in the 6LoWPAN standard [27, 28]. The Internet of Things presents unique challenges for routing protocols, such as lossy links, frequent topology changes (due to sleeping hosts and mobility), limited processing power, and small memory size, that call for a dedicated solution. The effort to build a routing protocol to meet these challenges resulted in the RPL specification [29], which enables IPv6 packet routing in IoT networks.

In the transport layer, TCP and UDP are both viable approaches, but UDP is preferred in most applications because it is better adapted to the IoT requirements. Many of the functions that TCP provides (e.g., reliable delivery, congestion control, and handling of out-of-order and duplicate packets) are implemented in the application layer when UDP is used as a transport protocol. This enables the optimization and simplification of these functions, such as the piggy-backing of acknowledgements onto application packets. An in-depth overview of IoT network technologies, with references to the latest research in the field, is given by Vasseur et al. in the book *Interconnecting Smart Objects with IP: The Next Internet* [30], which also serves as supporting material for the information in this section.

Although not exhaustive, the communication protocols available on the physical, data link, network, and transport layers described herein provide the fundamental communication infrastructure enabling interoperable Web services on the Internet of Things. While the development of the complete IoT networking stack remains an open research field, many of these protocols are already deployed and have proven successful in real-world applications (see the Bluetooth Special Interest Group, ZigBee and G3-PLC alliances for examples). Therefore, the remaining challenges for end-to-end interoperability in the IoT domain are within the presentation and application layers. This is supported by the analysis presented by Shelby [31], which defines two key activities required to achieve efficient embedded Web services: 1) the redesign of the web application transfer protocol and 2) efficient payload encoding. These two topics are presented in separate sections because they are directly related to the objectives of this thesis. The application protocols that are considered for the development of embedded Web services are presented in Section 2.3 and Section 2.4, and the presentation layer standards (i.e. data encoding formats) are discussed in Section 2.5.

2.3 Web Service Technologies

The design of programming interfaces and communication infrastructure for information exchange and sharing of computational resources across the network in distributed systems has evolved from the remote procedure call (RPC) effort, dating back to 1976 [32], to today's Service-Oriented Architecture principles [33]. Many designs and implementations have been proposed, and many are in use today, including CORBA, Protocol Buffers, Apache Thrift, Apache River, and Web services. Web services are different from other distributed technologies in two main aspects: 1) they are based on open and standardized communication protocols, and 2) they use the design principles of the Web architecture. Both of these aspects are essential for providing interoperability on a global scale. It is therefore useful to introduce the design principles of the Web architecture and how these principles are applied to Web service technology.

2.3.1 Web Architecture

The main design goal of the Web was "to be a shared information space through which people and machines could communicate" [34]. To achieve this goal, key aspects of the Web architecture are the definition of a standard way to identify and address information and resources on the Internet and the use of a common resource transfer protocol that supports data format negotiation. The identification and addressing of resources on the Internet is performed by uniform resource identifiers (URIs), which are string-based tokens that have a formally defined structure and semantics [35]. URIs can be further classified as resource locators or resource names. The former are also known as uniform resource locators (URLs). A URL is a URI that identifies a resource via its network access mechanism: a transfer protocol, network address (i.e. IP or DNS name), network port, and absolute or relative path to the resource. The uniform resource name (URN) is an identifier that remains globally unique and persistent even when the resource changes its access mechanism, becomes unavailable, or even ceases to exist.

The common resource transfer protocol used on the Web is the hypertext transfer protocol (HTTP) [36]. It is a request-response application protocol that follows the client-server distributed computing model. HTTP uses URLs to specify the application resources that are requested from the Web servers. These resources can have different representations, or more specifically, they can be delivered using different encoding formats. HTTP specifies a content negotiation mechanism that allows Web clients and servers to agree on a particular resource representation. The negotiation is facilitated by a standard registry of data format identifiers called Internet media types. Examples of widely used media types are *text/html*, which identifies plain text encoding of HyperText Markup Language (HTML), and *application/xml*, which is used for the XML representation of a Web resource.

A formal and systematic study of the fundamental constraints and underlying principles of the Web architecture is presented by Roy Fielding in his doctoral thesis, which defines the representational state transfer (REST) architectural style for distributed systems [37]. He derives REST by identifying and modeling constraints that underpin the Web. The separation of the user interface from data storage improves scalability and simplifies the server components, but leads to the first constraint, the client-server architecture. The visibility, reliability, and scalability requirements of the Web lead to the second constraint, stateless client-server interactions. The network performance of Web interactions is enhanced by adding the third constraint, cacheable resources. The fourth architectural constraint identified by Fielding is the use of a uniform interface between Web components. This constraint mandates the use of a generic way to identify and manipulate resources and to perform application state transitions. Because the interface is generic, it is common to all Web applications. To support load balancing and cache sharing it is beneficial to constrain the architecture further and apply the layered system constraint, which hides information about the nesting of server components from clients. The last architectural constraint in REST is the optional code-on-demand model, which allows the client's functionality to be extended by downloading and executing code. This constraint reduces the visibility of the system in exchange for greater extensibility and simplified client implementations.

The details of how these constraints are reflected in the architectural elements of REST (i.e. the processing, data, and connecting elements) are presented by Fielding et al. [38]. The authors of this study also show how REST can be used to evaluate the design of concrete protocols and Web specifications for their adherence with the principles of the Web architecture. Using this evaluation mechanism, Web service technologies are classified into two distinct groups: SOAP-based Web services and RESTful Web services. The former are based on REST-compliant Web protocols and technologies (URI, HTTP, and standard Internet media types), but the way these protocols are used in SOAP Web services violates some of the architectural constraints identified in REST. In RESTful Web services, however, the use of the same Web protocols and technologies adheres to all of the architectural constraints in REST. The sections that follow provide an overview of SOAP and RESTful Web services from the perspective of their application in embedded distributed systems. A general comparison and trade-off analysis of the two Web service technologies is given in Pautasso et al. [39].

2.3.2 SOAP Web Services

SOAP is an XML-based protocol that defines a message passing framework. SOAP messages are used to define application-specific service requests and responses that carry XML encoded application-defined data types. This is in contrast to the uniform REST interface, in which a generic set of operations is used to implement service requests and responses.

The functionality offered by a SOAP Web service can be defined by an XML-based IDL called Web services description language (WSDL). WSDL provides a machine-readable description of the network access mechanism of the service, what parameters it expects, and what data structures it returns. Because the syntax of WSDL is machine-readable, it is often used for code generation and service composition in Web service development frameworks.

SOAP uses the extensibility features of XML, which enables the addition of new capabilities and functionality to the SOAP protocol without the need to modify the base specification. There are many standards that extend the SOAP functionality, which are commonly labeled as *WS-**. Example extensions that are widely used are WS-Addressing, WS-Discovery, WS-Eventing, WS-Policy, WS-Security, and WS-Management. Some of these extensions decouple the service definitions from the use of a particular transport protocol, thus allowing SOAP to be sent over UDP [40], for example, instead of the more commonly adopted HTTP transport. Because there are incompatible versions of the different transport and WS-* protocols, there might be interoperability issues when connecting different SOAP Web service implementations. Additionally, different application domains have different requirements for the Web service protocols. To address these issues, standardization bodies such as OASIS define Web service profiles that set additional constraints on the protocol versions (e.g., the HTTP and SOAP versions), on the exchange patterns (e.g., the use of distributed or centralized service discovery), and on the set of WS-* protocols that are allowed for the service messages. Web services in enterprise systems conform to the WS-I Basic Profile 1.0 [41], whereas the Devices Profile for Web Services (DPWS) [42, 43] is designed to enable plug-and-play capabilities for embedded networked devices. There are many differences between WS-I Basic Profile 1.0 and DPWS, and these specifications are largely incompatible despite the fact that both are based on SOAP. As an example, enterprise WS-I Web services rely on a centralized registry for service publication and discovery called Universal Description, Discovery and Integration (UDDI), whereas DPWS uses a distributed approach based on WS-Discovery. Moreover, DPWS specifies a set of built-in services also known as device services, that are used for the hosting and advertising of application-specific Web services as well as for providing device-specific metadata, service discovery, and eventing. DPWS was primarily designed for office equipment, but its application in industrial environments and resource-constrained systems has also been investigated [44].

The main challenges of using SOAP and DPWS on resource-constrained devices are related to the large size of the service messages (due to the verbose XML syntax of SOAP control fields and the data payload) on one hand and the use of TCP/HTTP as a transport protocol for SOAP-over-HTTP binding on the other hand (resulting in a lack of push notifications, high overhead for TCP connection establishment, and a relatively large HTTP header size) [7]. The large messages require heavy processing and lead to latency in low-bandwidth networks. This problem is even more severe in battery-powered wireless devices because the large size of the messages results in many transmitted packets and therefore higher energy consumption as well as scalability and robustness problems in multihop communication.

2.3.3 RESTful Web Services

A RESTful Web service can be any Web-accessible resource that conforms to the REST architectural constraints. Although RESTful Web services are implemented with standard protocols and they provide a uniform interface across different applications, there is no standard way to map Web application functionality to that uniform interface. As an example, consider the interface for a Web-accessible e-document storage application. The application developers have the freedom to decide how to expose the application's functionality through the generic RESTful interface using Web resources and operations on these resources. One possibility is to have a single generic */edoc/storage* resource with many operations allowed on it (e.g., for adding new documents and for searching, retrieving, and updating existing documents). Another possibility is to use multiple resources that accept only a small set of operations, such as */edoc/storage/store* to support only the addition of a document, */edoc/storage/browse* to retrieve a list of documents, and */edoc/storage/search* to define filters.

The RESTful interface is defined in terms of CRUD operations (create, read, update, and delete) on a remote resource identified by a URI. The resource itself can be represented by any of the predefined Internet media types, such as XML, JSON, or HTML. Although RESTful Web services are not limited to the HTTP protocol, until recently, HTTP was the only transport protocol used in practice. The CRUD operations are mapped to the HTTP methods GET, PUT, POST, and DELETE. The interface description of a RESTful application consists of a list of available resources with their URIs, which HTTP methods are allowed on these resources, and the names and types of the methods' parameters, as well as the encoding format and the structure of the responses. Because the second version of the WSDL specification provides only limited support for RESTful Web services, the Web application description language (WADL) is often used as a RESTful IDL, although it has not progressed to a formal standard.

Compared to SOAP, RESTful Web services are better suited for use in embedded applications because the service messages are generally smaller. This argument might be mitigated when integration with enterprise systems that are compliant with WS-I Basic Profile 1.0 is required. The challenges related to the use of TCP/HTTP in SOAP (i.e. the lack of push notifications, the high overhead for TCP connection establishment and the large HTTP header size) are also valid for RESTful Web services. For this reason, the charter of the Constrained RESTful Environments (CoRE) IETF working group is to develop the necessary standards to support efficient RESTful interactions in constrained IP networks. The objective of these standards is to provide better support for the specific requirements of resource-constrained networked devices than the HTTP protocol while simultaneously conforming to the REST architectural constraints [31]. The main result of the CoRE working group has been the development of the Constrained Application Protocol (CoAP) [45], which meets the requirements for constrained networks such as support for asynchronous message exchange, multicast capabilities, a lightweight discovery mechanism, very low overhead, and implementation simplicity. This is due to the use of UDP as a transport protocol with optional, reliable unicast support and Datagram Transport Layer Security (DTLS), instead of TCP and TLS. The use of UDP enables the implementation of the CoAP lightweight publish-subscribe mechanism described under the ”Observing Resources in CoAP” specification [46]. This specification supports small-footprint event-based dynamic content delivery using push notifications.

The discovery of resources and services in CoAP is facilitated by the CoRE Link Format standard [47], which defines a string-based syntax for representing a list of Web resources and the available operations on them. Using this format, CoAP hosts can publish the services they provide to a predefined interface called */.well-known/core* and thus allow their discovery over the network.

The CoAP design (such as request methods and error codes) is intentionally similar to HTTP to support the seamless integration of CoRE into the established HTTP-based Web infrastructure. In addition, CoAP is based on REST, and therefore it supports the stateless client-server model, which enables translation between HTTP and CoAP traffic using simple application layer proxies. To limit variations in the HTTP-CoAP translation approaches used by different implementations, which might cause interoperability issues, the mapping between HTTP and CoAP messages is defined as part of the CoAP specification. Although the HTTP-CoAP proxies are simple and standard-based, the use of CoAP RESTful Web services through HTTP-CoAP proxies breaks the end-to-end principle. As suggested by Kovatsch [48], CoAP must become an integral part of the technologies used in the World Wide Web to enable it to satisfy the end-to-end principle and eliminate application-level gateways that simply hide vertical silos. To achieve this goal, standard REST APIs and CoAP Web development toolkits are essential.

2.4 Overview of Application Protocols for IoT

There are different application layer protocols that provide many of the features of SOAP and RESTful Web services but are nevertheless not formally classified as Web service technologies. This section presents two of these protocols that have been proposed for IoT deployments: Message Queue Telemetry Transport (MQTT) [49] and Extensible Messaging and Presence Protocol (XMPP) [50]. The focus in this presentation is to highlight the differences in their architecture without providing a detailed description of their inner workings. Although the investigations presented in the appended papers in this thesis do not involve MQTT and XMPP, understanding their architecture can be useful for assessing the results and the impact of the presented studies on embedded Web service technologies. Therefore, this section also includes a comparison of the protocol stacks of DPWS, CoAP, MQTT, and XMPP.

2.4.1 MQTT

MQTT is a lightweight publish/subscribe messaging protocol designed for machine-to-machine communications constrained by low power, low bandwidth, and high latency network links. MQTT is based on a star topology in which all messages are sent via a central message broker server that manages publish/subscribe requests from the connected clients and sends back event notifications. All publish requests are made to a virtual address, known as the topic. Clients may subscribe to one or more topics to receive notifications for all messages published on these topics. In this way, MQTT supports different message exchange patterns: one-to-one (one publisher, one subscriber), one-to-many (one publisher, many subscribers), many-to-one (many publishers, one subscriber) and many-to-many (many publishers, many subscribers). The topics that are defined in the message broker are hierarchical and resemble the structure and naming conventions used in the file system. Clients can subscribe to a leaf topic or observe all the sub-topics of a particular base topic.

The MQTT protocol is based on TCP/IP and is designed to support small-footprint client code. Normally, the clients establish an always-on TCP connection to the broker, but variable availability and sleeping client deployments are also possible. At the time of writing of this thesis, MQTT is in the process of standardization by OASIS. The standardization effort includes providing guidelines for the use of MQTT topics with commonly available registry and discovery mechanisms.

MQTT security is based on client-supplied username/password or digital certificate authentication and optional TCP connection encryption with TLS for sensitive data.

The use of the TCP protocol in MQTT, as well as the verbose string identifiers used for the subscription topics, makes MQTT impractical for highly constrained devices communicating over lossy network links such as IEEE 802.15.4. This is the motivation behind the MQTT for Sensor Networks (MQTT-SN) protocol, which is based on the same architecture as MQTT but runs on top of UDP instead of TCP. Additionally, MQTT-SN allows the use of short indexes for topic identification instead of verbose string topic names.

Payload encoding of the MQTT(-SN) messages is not defined in the specification and is application specific. MQTT does not provide content negotiation mechanisms, and MQTT consumers must be informed of the format and structure of the payload data in advance using out-of-band mechanisms.

Table 2.2: Network stack comparison for four IoT application layer protocols

DPWS		**CoAP**	**MQTT(-SN)**	**XMPP**
XML payload encoding		Internet Media types	Payload encoding undefined	XML payload encoding
Application services		Application services	Application services	Application services
WS-Discovery, WS-*				
SOAP, WSDL, XML Schema				XEP-0060, XEP-0324, ...
HTTP 1.1	UDP	CoAP	MQTT(-SN)	XMPP Core
TCP[TLS]		UDP [, DTLS]	TCP [, TLS] (UDP)	TCP [, TLS]
IPv4/v6 [, 6LoWPAN]		IPv4/v6 [, 6LoWPAN]	IPv4/v6 [, 6LoWPAN]	IPv4/v6 [, 6LoWPAN]

2.4.2 XMPP

XMPP is an IETF standard initially designed for distributed instant messaging applications. It is based on a client-server architecture and runs on top of a long-lived TCP connection or, optionally, a HTTP connection. XMPP security is based on the Simple Authentication and Security Layer (SASL) and TLS. The communication between clients and servers is based on open-ended XML streams, i.e. XML documents that are incrementally built over time. The messages that are exchanged are XML fragments defining both the control fields and the payload data. The extensibility of XML allows the extension of the XMPP base specification with additional features through XMPP Extension Protocols (XEPs). There are many XEPs defined, such as XEP-0060: Publish-Subscribe and XEP-0030: Service Discovery. Additionally, some of the proposed XMPP extensions are directly targeted for the use of XMPP for IoT communication. These extensions provide support for the provisioning of IoT devices with services, access rights, and privileges (XEP-0324), sensor data transfer (XEP-0323), control of actuators (XEP-0325), device management (XEP-0326), and support for the efficient representation of XML streams over the network using the Efficient XML Interchange Format (XEP-0322).

The main disadvantages in using XMPP for battery-powered wireless devices are the use of an always-on TCP connection and the verbose XML syntax for the control fields and data payload in the XMPP messages.

2.4.3 Protocol Stack Comparison

Table 2.2 presents a compact description and comparison of the communication stacks for DPWS, CoAP, MQTT, and XMPP. The visual representation highlights the complex communication chain in DPWS and XMPP. This is a result of the initial designs of these two technologies to serve as application protocols in different domains (enterprise systems for SOAP and instant messaging for XMPP Core); they were later extended (retrofitted) to meet the IoT communication requirements. To the contrary, both CoAP and MQTT(-SN) were designed for use in constrained networks and thus have built-in support for service discovery and eventing, for example, which require additional layers in DPWS (WS-Discovery, which also requires WS-Addressing and WS-Eventing) and XMPP (XEP-0030 and XEP-0060).

2.5 Payload Encoding

The data format for the application payload can be predefined as in SOAP-based Web services and XMPP, negotiated among a set of standard Internet media types as in RESTful Web services, or left undefined as in MQTT. This section derives the functional and non-functional requirements for the encoding formats used in the Internet of Things. The derivation is based on four desired properties for the application protocols used in the IoT environment. The properties stem from the requirements of the IoT domain (e.g., sleeping nodes, low bandwidth links, and low cost deployment and maintenance) and are listed below:

I. Provide support for end-to-end interoperability

II. Respect the constraints and limitations imposed by the underlying network technologies and computational resources available on the embedded hosts

III. Enable scalability on a global scale

IV. Support a wide range of application data, including various payload sizes and different types (from small scalar and vector values to tabular data and structured hierarchical representations)

2.5.1 Data Format Requirements

This section lists the derived requirements that data formats should meet when used in IoT applications.

Platform independence There should be no dependencies on the hardware architecture or platform-specific features to ensure that different implementations can interoperate (derived from I and III).

Open and standardized The specification of the data format should not be restricted to a particular vendor, and its implementation should not be impeded by software patents or licensing (to fulfill I and III).

Efficiency Based on II and III, the encoding format should be efficient in terms of size (limited or no data redundancy) and processing (simple and fast to parse and serialize i.e. a small code footprint, low memory usage, and low CPU cycles).

Support for data types The use of data types provides many benefits. Narrowly defined data types closely reflect the nature and properties of the represented data, which leads to more compact encoding. Data types are also useful for providing an auxiliary semantic description of the data because they reveal certain properties of the value space of the information they hold. Finally, data types enable the validation of data according to the type constraints. (II and IV)

Capable of representing complex information The type of information exchanged in today's sensor/actuator networks is often very basic - the sensors report simple scalar values for the measured phenomena, and actuators are controlled by boolean or integer processing values. However, as devices become more capable and sophisticated, it will become more common to equip the traditionally simple sensor and actuator platforms with additional diagnostic, logging, and security capabilities and even a human-machine interface. In order to support a variety of use cases and make the technology future-proof, the data formats used in the Internet of Things should support tabular, hierarchical, and structured data. (III and IV)

Formal structure definitions (schema) The support for data types and structured representations in the data formats requires that the exact structure (e.g., *is-a*, *has-a*, and *instance-of* relationships between different data elements) and type definitions be well defined. These definitions should be machine-readable to support code generation and dynamic negotiation of the data structure. (III and IV)

Support for semantic data models A well known approach for addressing the complex semantic interoperability problem is its reduction to a more manageable syntactic interoperability problem by means of domain ontologies or data models. In its basic form, this approach involves formally representing the knowledge within a particular domain by mapping the types, properties, and interrelationships of data elements to syntactic and data type constraints. This mapping is often formally defined within data model standards that aim to provide semantic level interoperability for communications in a particular application domain. Examples of such standards include Bluetooth profiles such as the Human Interface Device Profile (HID), Basic Printing Profile (BPP), the Health Device Profile (HDP), ZigBee profiles such as the Smart Energy Profile 2 (SEP2), IPSO Smart Objects, and specifications for generic sensory data from the Open Geospatial Consortium's Sensor Web Enablement initiative, e.g., Observations and Measurements (O&M) and the Sensor Model Language (SensorML). To support the creation of such data models, the encoding formats should provide formal structure definitions that are expressive and human-readable because the mapping of the domain knowledge to syntactic constraints is performed by humans (i.e. domain experts). (I and III)

2.5.2 Data Format Evaluation

Once derived, the data format requirements can be used to assess IoT conformance and the fitness of different encoding solutions. Based on this approach, Table 2.3 presents a coarse-grained evaluation of several data formats: plain text (not strictly defined; used primarily for scalar values), character-separated values (CSV) (specified by IETF [51]; used for spreadsheets and tabular data), JavaScript Object Notation (JSON) (a widely adopted format in JavaScript and Web applications), Extensible Markup Language (XML), Efficient XML Interchange (EXI), and Abstract Syntax Notation One (ASN.1) using Basic Encoding Rules (BER) and Packed Encoding Rules (PER) for binary encoding.

As shown, EXI and ASN.1 BER/PER are the most suitable candidates of the ones considered. The evaluation presented in Table 2.3 is limited in its scope and comprehensiveness because there are many data formats that are not included. Moreover, the evaluation criteria should be assigned weights (e.g., efficiency has greater significance in the assessment than semantic data model support), and a more detailed analysis should be performed on the extent to which the given requirements are met. The aim of the format comparison is to motivate the choice of the encoding standard for use in the thesis investigations and not to present quantitative measurements that are supported by experimental data. In this thesis, EXI is used for payload encoding for the following reasons:

1. Interoperability with existing Internet- and Web-based solutions is a priority in this work. EXI is better suited for use in these scenarios than ASN.1 because many Web technologies are based on XML, and EXI has been selected by W3C as the most promising XML compaction technology.

Table 2.3: Evaluation of data formats according to the derived requirements.

Requirements	**Data formats under consideration**					
	Plain text	**CSV**	**JSON**	**XML**	**EXI**	**ASN.1 BER/PER**
Platform independence	x[1]	x	x	x	x	x
Open and standardized		x	x	x	x	x
Efficiency	x	x	x		x[4]	x[5]
Data types			x	x	x	x
Complex information		x[2]	x	x	x	x
Schema			x[3]	x	x	x
Semantic data models				x	x	x

[1] UTF-8 solves most issues; possible discrepancies in special characters e.g. *newline*
[2] CSV supports tabular but not hierarchical data
[3] JSON Schema is not formally a standard
[4] EXI reduces data redundancy (more compact representation), improves the processing speed, but has little effect on code size and run-time memory when compared to XML
[5] The binary BER/PER ASN.1 encoding rules are very compact but there is little information available on code footprint, processing speed and run-time memory usage besides the work by Mundy et al. on the use of ASN.1 BER for payload encoding in Web services [52]

2. EXI is more compact than ASN.1 BER/PER in representing XML data [53].
3. Although it is promising, the EXI standard is relatively new, and there have been few research studies investigating its use in developing embedded Web services.

2.6 Applications

This section presents example application areas for embedded Web service technology that can be used to estimate the potential impact of the research. As already discussed, the main driving force behind the use of embedded Web services is the increasing demand for greater interoperability between heterogeneous systems and devices. This demand is often a result of optimizing complex processes that incorporate a multitude of isolated subsystems. The optimization of such processes often requires access to large amounts of data from the interdependent subsystems. Providing access to information in heterogeneous environments can be an extremely laborious task if the subsystems use incompatible communication technologies.

Application areas that are characterized by heterogeneous environments with complex interdependencies include home and industrial automation, smart cities (e.g., waste management and transportation), retail and logistics, and energy management in electrical grids (i.e. smart grids). To illustrate the benefits of embedded Web services in such applications, the following paragraph will present an analysis of the smart grid domain as an example of a complex heterogeneous system.

A smart grid is an electrical power grid with two-way communication infrastructure that can be used to enhance the reliability of electricity distribution, reduce peak demands, and lower total energy consumption and the carbon dioxide footprint. This requires coordination and information exchange between many systems and devices: electricity producers (i.e. power plants and distributed energy resources (DER) for generation (e.g., wind and solar power and industrial co-generation) and storage (e.g., electrical vehicles)); electricity transmission and distribution systems (i.e. electrical substations); and energy consumers (e.g., industrial processes, smart meters, HVAC (heating, ventilation, and air conditioning), and home appliances). It is convenient to take the smart meter as an example to illustrate the interdependencies within such a system. An advanced smart meter has the capability to request dynamic pricing information from the utility company and report back energy consumption information. It might also be equipped with a human-machine interface (e.g., a touch screen, Web interface, or smart phone app) for monitoring and control of energy usage. The device can be programmed to perform automated demand-side management and to react to changes in energy prices by sending control signals to smart appliances such as boilers or heaters. It is thus beneficial to use unified application layer technology to retrieve the pricing information, report energy usage, provide the human-machine interface, and control various appliances. This is the main motivation for the demand response and load control (DRLC) specification called Smart Energy Profile (SEP) version 2.0 developed by the ZigBee industry alliance. SEP2 is based on HTTP RESTful Web services that carry EXI encoded payload data conforming to a data model defined by an XML Schema definition. The SEP2 data model describes the information exchange occurring in electrical distribution systems. By using efficient Web services, the SEP2 specification provides added value for technology providers because it enables interoperability among heterogeneous devices and systems without the need for additional hardware and software mediators.

CHAPTER 3

XML Data Compaction

The terms *data compaction* and *data compression* are often used interchangeably in the literature for methods that encode a data stream using fewer bits than the original representation. This is achieved by removing redundant information, such as by indexing continuously recurring data segments and representing these segments with a short index in the compressed stream. In general, reducing the size of a data stream is a trade-off between the level of compression, the time it takes to process the stream, and memory usage. A higher level of compression often requires a longer time to process and more memory than the original representation. A subtle difference between compaction and compression is that compaction is usually used to describe lossless algorithms for redundancy reduction that have little or no impact on the processing speed and memory usage, whereas compression is a more general term that includes lossy algorithms that irreversibly remove nonessential information from the data stream. This chapter uses the term data compaction exclusively because it aims to provide background information on possible methods for lossless reduction in the size of XML data with relatively little impact on the processing speed and memory usage.

3.1 Theoretical Background

This section starts with a short overview of basic concepts and generic data compaction methods which are then presented with relation to XML compaction. Many of the discussions in this section are supported, and influenced, by the book by Sayood [54], which gives an excellent introduction to the field of data compression.

Data compaction algorithms consist of two distinct parts: a model of the source data and a code. The model is a mathematical representation of the redundancy in the source data. The code is a mapping between the symbols of a given alphabet, or strings of such symbols, to (shorter) codewords or binary sequences. The alphabet is a finite set of symbols (letters) that are used to represent the input data stream. For example, an English text input stream can be represented over the English alphabet {a-z, A-Z, and punctuation symbols}, whereas a compiled computer program can be represented using the {0,1} alphabet. The process of transforming the input sequence of letters and strings to a sequence of codewords is called encoding. The reverse process of transforming a sequence of codewords into letters and strings over a given alphabet is called decoding.

The model can be based on various forms of redundancy in the representation of the source data with the given alphabet. For example, a model of a continuous tone image (e.g., a digital photograph) can take advantage of the redundancy caused by the similarity in the colors of adjacent pixels. The ASCII encoding of text is another example of redundancy. In ASCII, the digital representation of English text source data maps a seven-bit number to each letter of the English alphabet and some punctuation symbols. This code is simple to process because each symbol uses the same number of bits (i.e. a fixed-length code). However, this is not the optimal representation in terms of compactness because some symbols have a much higher probability of occurrence than others; for example, the letter *e* is approximately 94 times more likely to occur in text than the letter *j* [55]. Using a probabilistic model for the occurrence of each symbol and a variable-length code in which the most commonly occurring letters are encoded with shorter codewords is a simple example of a data compaction approach that has been used since at least the beginning of the 19th century (e.g., Braille code and Morse code).

The more accurately the model describes the redundancy in the source data, the better the level of compression that can be achieved. One extreme case is when the model defines the exact source data i.e. the source data are known in advance. In this case, the source data can be represented with a zero-length code because the data are already known by the communicating parties. The other extreme occurs when the model does not define any redundancy or when the source data contain no redundancy, as for example in the case of random data. In this case, no compaction is possible, and the code length cannot be shorter than the length of the source data. Consequently, the level of compactness that can be achieved depends on the amount of redundancy in the source data and how well this redundancy is described by the model. This is why some compaction algorithms work well on text data but do not perform well in the compaction of images or video, for example.

The specific model and its parameters as well as the code used to encode the source data must be known by the decoder. This information is normally defined in the data compaction standards. Consider the following example illustrating the relationship between the model and the code: a source over the alphabet {0-9} produces a data stream consisting of monotonic sequences of integers, such as the following sample sequence:

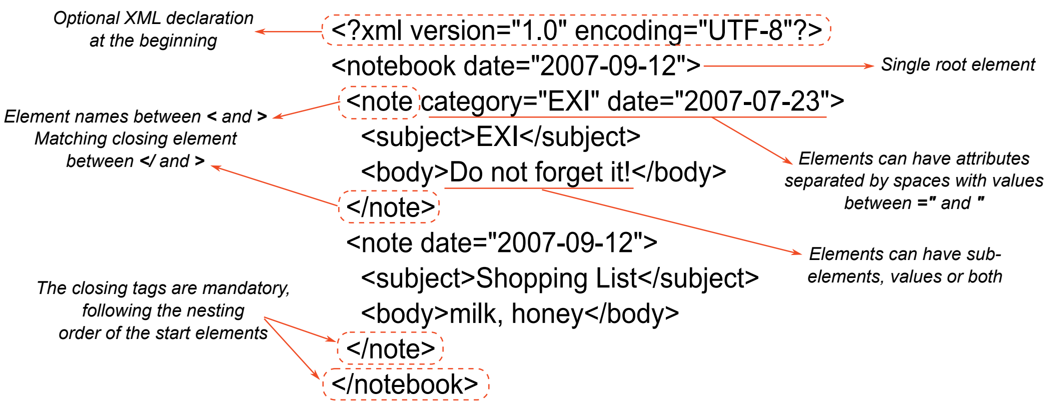
$$0 \quad 1 \quad 2 \quad 5 \quad 5 \quad 6 \quad 7 \quad 10 \quad 11 \quad 11$$

One model that captures the redundancy in this sequence is based on a residual sequence (r_n) in which $r_0 = x_0, \quad r_n = x_n - x_{n-1}$ with r_n and x_n being the nth element of the residual sequence and the original sequence, respectively. In effect, the model transforms the initial source data into a data stream with a lower estimated entropy - in this case, the residual sequence:

$$0 \quad 1 \quad 1 \quad 3 \quad 0 \quad 1 \quad 1 \quad 3 \quad 1 \quad 0$$

A code mapping short codewords to small integers (assuming that small integers have a higher probability of occurrence than large integers) can then be used to encode the residual sequence e.g., using Elias codes [56]. Given the model used to construct the residual sequence and the code used for encoding, the decoder can recover the original integer sequence. Data sources with a more complex structure can benefit from the use of multiple models and codes to encode different parts of the data stream; this approach is more flexible in capturing the data redundancy.

The text-based representation of XML information contains much redundancy e.g., in the repetition of element names in the closing tags and the indentation spaces and tabs between nested elements. Additionally, the structure of a valid XML instance is well defined and known in advance and can also be further constrained by XML Schema. A good model for the XML data source should make use of the manifested redundancy and structural constraints to achieve a high level of compaction while



```
<?xml version="1.0" encoding="UTF-8"?>
<notebook date="2007-09-12">
  <note category="EXI" date="2007-07-23">
    <subject>EXI</subject>
    <body>Do not forget it!</body>
  </note>
  <note date="2007-09-12">
    <subject>Shopping List</subject>
    <body>milk, honey</body>
  </note>
</notebook>
```

Optional XML declaration at the beginning
Single root element
Element names between < and > Matching closing element between </ and >
Elements can have attributes separated by spaces with values between =" and "
Elements can have sub-elements, values or both
The closing tags are mandatory, following the nesting order of the start elements

Figure 3.1: Sample well-formed XML document annotated with the syntactic rules that cause redundancy in the lexical representation.

maintaining processing efficiency. This intuitive statement is supported by the work of Cameron [57], which compares lexical and syntactic source models, and by the empirical measurements performed by the EXI working group in W3C [53].

There are many data source models and codes that have been used or proposed for XML data compaction. This section focuses on the source models and codes used in the Efficient XML Interchange standard because it has been shown to provide consistently higher levels of compaction and processing efficiency than other XML compaction technologies [53]. The goal of the following sections is to present the theoretical background for the key compaction techniques used in EXI and to show how they relate to the results of this thesis. Where relevant, the compaction techniques used in EXI are compared to alternative approaches from the literature.

3.1.1 Formal Grammars

Well-formed XML instances define a context-free language with specific syntax rules as shown in Figure 3.1: for example, only one root element is allowed (i.e. a tree structure), elements must begin with $<$ and end with $>$, every element must have a closing element, and elements can have attributes. These rules, possibly refined by XML Schema definitions, define the structure of XML documents and can be represented with a context-free grammar or, alternatively, a regular tree grammar [58]. This thesis relies on context-free grammars, which can be formally defined as follows:
The tuple $G(N,T,S,P)$ defines a context-free grammar G, where N is a finite set of non-terminal symbols or variables. T is a finite set of terminal symbols, disjoint from N. The set of terminals is the alphabet of the language defined by the grammar G. $S \in N$ is the start variable (a non-terminal symbol), and P is a set of productions (rewriting rules) that define a finite relation from N to $(N \cup T)^*$, where $*$ represents the Kleene star operation. The string $u \in (N \cup T)^*$ is directly derived from the variable X (or X directly yields u) if the production rule $X \Rightarrow u$ is in P. The relationship $X \overset{*}{\Rightarrow} u$ indicates that u is derived from X by recursive application of the rules in P. The grammar productions P specify which words are allowed to occur in the language defined by G. More formally, the language $L(G)$ is the set of words w over the terminal alphabet T that can be derived from the start non-terminal

symbol S, i.e. $L(G) = \{w \in T^* : S \overset{*}{\Rightarrow} w\}$.

Instead of defining the formal grammar over the alphabet of Unicode characters, XML processing methods commonly rely on context-free grammars over the alphabet of XML events. Here, XML events are events that are emitted by streaming XML parsers. XML events describe the structure of XML abstracted from its lexical representation. Consequently, the set of terminal symbols T in XML grammars consists of the XML events describing the structure and content of an XML document: *namespace, start_element, attribute, element_value, end_element, comment* and *processing_instruction*. The parsing of an XML document (or a word in the language consisting of all valid XML instances) defines a derivation tree with the XML events as leaves. The derivation tree constrains the possible events that can occur at each point in the XML source data.

An information source model that can describe the restrictions posed by context-free grammars is given by Cameron [57]. The author presents the syntactic source model, which encodes a word in a context-free language through its derivation tree. The derivation tree has a lower estimated entropy than the lexical representation of the word itself. The tree represents the sequence of derivation steps that are taken during the parsing of the word. Each derivation step is represented over an alphabet of symbols specifying the possible alternative paths that the derivation could take. For example, consider the representation of the derivation step from the variable $X \in N$ to the sentential form u_2, where $u_i \in (N \cup T)^*$ for $i \in [0-4]$, and the following productions are available in the grammar:

$$X \Rightarrow u_0, \quad X \Rightarrow u_1, \quad X \Rightarrow u_2, \quad X \Rightarrow u_3, \quad X \Rightarrow u_4$$

An alternative form to describe the five production rules is the following:

$$X \Rightarrow u_0 | u_1 | u_2 | u_3 | u_4$$

The alphabet over the possible alternative paths of the derivation is then $\{u_0, u_1, u_2, u_3, u_4\}$. If we assign a unique order to the sentential forms $\{u_i\}$, then the alphabet can be further simplified to contain only the indexes of the sentential forms, i.e. $\{0, 1, 2, 3, 4\}$. In this way, the derivation step from X to u_2 is represented by the index 2 in the compacted stream.

In many context-free grammars, the probability of each derivation path ($P(u_i)$ in the example above) is known in advance or can be estimated during encoding. For example, the probability of occurrence of an *attribute* XML event after a *start_element* is much higher than the probability of occurrence of a *comment* or *processing_instruction* event. For this reason, the use of variable-length codes to encode the derivation steps can lead to a higher level of compaction.

Cameron suggests the use of arithmetic coding [59, 60] for the efficient representation of the derivation steps. He further shows that the approach of using a syntactic source model and arithmetic coding provides the best compaction when it is compared to approaches based on lexical source models [57]. The disadvantages of the syntactic source model proposed by Cameron is that developing a context-free grammar that completely describes a given information source can be difficult and time consuming, and the resulting compaction technology is highly specialized.

The use of the syntactic source model to develop specialized XML data compaction systems is very beneficial because it provides a very high level of compaction and also allows for fast processing. The lexical XML parsers build an in-memory representation of the derivation tree for processing and validation, and using the syntactic source model, the tree is directly represented in the compacted stream.

3.1.2 Huffman Coding

There are different ways to construct the variable-length code used to encode the derivation steps in the syntactic source model. The most common approaches are arithmetic coding, as proposed by Cameron, and Huffman coding [61], which is the basis for the code used in EXI. This section provides a brief overview of Huffman coding and describes how it differs from arithmetic coding.

Both the Huffman and arithmetic codes are prefix codes because no codeword is a prefix to another codeword. This ensures that the resulting code is uniquely decodable, which means that for any given sequence of codewords there is one and only one way to decode them. This property is important because it guarantees that the decoder will construct exactly the same derivation tree and thus reconstruct the original XML document.

The Huffman code is optimal for a given alphabet size and probability model for the input symbols; i.e. the average length of the codewords is minimal. Building the code, or creating a mapping between the input symbols and the codewords, is based on two principles: 1) symbols that occur more frequently have shorter codewords, and 2) the two least frequently occurring symbols have the same length. In this way, the code is built iteratively starting from the least significant bits of the longest codewords.

The Huffman code can be extended to use cases in which the statistics of the input alphabet are not known in advance and the probability of occurrence of the input symbols is calculated during processing. This approach is known as adaptive Huffman coding, and several coding methods have been developed based on this approach [62, 63, 64].

The main difference between Huffman coding and arithmetic coding is that arithmetic codes assign codewords to sequences of input symbols of variable length, whereas Huffman codewords are based on fixed-length sequences of input symbols. Consequently, arithmetic coding can be useful when the input alphabet is small and contains highly skewed probabilities, such as when some input symbols are much more likely to occur than others.

3.1.3 Dictionary Techniques

In XML documents, as with other text sources, there are certain patterns or words that occur multiple times in the input data. One approach to model this type of redundancy is to build a list, or dictionary, of commonly occurring patterns and to encode these patterns using their index in the list, or their reference in the dictionary. In general, the use of dictionary techniques is a trade-off between the level of compaction and the memory usage necessary for encoding and decoding. The efficiency of these techniques depends on the number of occurrences of the indexed patterns in the source data.

Static Dictionary

When some prior knowledge of the usage of frequently occurring patterns is available for the source, the dictionary (or some part of it) can be statically populated with these patterns. For example, dynamic typing of values in XML is performed using the following namespaces:

```
"http://www.w3.org/2001/XMLSchema"
"http://www.w3.org/2001/XMLSchema-instance"
```

Because these namespaces, as well as some other XML-specific patterns, are expected to occur frequently in XML documents regardless of the application domain, the use of a static dictionary for their efficient representation is justified.

Adaptive Dictionary

In many cases, the reoccurring patterns are not known in advance and cannot be statically represented in the dictionary. In this case, the dictionary must be dynamically built during the encoding or decoding of the source data. The indexing of patterns in the dictionary can be based on different techniques, such as performing a search for duplicates in a sliding window [65], techniques that incrementally grow the dictionary with patterns that are not yet indexed [66], or heuristic methods that model the probability of reoccurring patterns in different places in the source data.

3.1.4 Data Type Consideration

In addition to the structural constraints modeled by context-free grammars, XML documents can have data type constraints that are either statically set in schema definitions or dynamically specified using the namespaces:

```
"http://www.w3.org/2001/XMLSchema"
"http://www.w3.org/2001/XMLSchema-instance"
```

The data types restrict the value space of specified attributes or elements, allowing a more compact representation than default string encoding. To achieve a high level of compaction, different source models can be used for different data types. This approach should be weighed against the implementation complexity that can arise from dealing with many source models. For example, the XML Schema specification defines 13 data types for integer data alone. The source models should also take into account the probability distribution of the value space of the data types. For example, the value space of the *positiveInteger* data type is the infinite set $\{1,2,3,...\}$, in which small values have a higher probability of occurrence than larger values. The code used to encode the *positiveInteger* data type should be capable of representing arbitrary large numbers but should also assign shorter codes for small positive integers than for large ones. Another example is the year component of the *dateTime* data type. Overall, contemporary dates are more common than past dates, especially dates long in the past. This can be modeled using a residue with respect to a contemporary year constant, as in the Unix time system, in which the time is represented as an offset from 1970.

3.2 Efficient XML Interchange

The EXI standard [67] specifies an XML compaction system that substantially reduces the size of XML source data when stored on disk or transferred over the network (resulting in a 70 % to 80 % smaller representation on average [53]). The use of EXI also leads to faster processing than text XML (6.7 times faster decoding and 2.4 times faster encoding on average, as shown in [68]). These improvements are a result of the use of a composite source model that captures the redundancy in XML well and a system of codes that allows for faster encoding and decoding. This section provides a short introduction to EXI, including its source models and codes, along with a description of the research directions related to EXI compaction that are investigated in this thesis.

An EXI-encoded XML document, or an EXI stream, consists of two parts: a header followed by a body. The header of an EXI stream contains meta-information such as bit patterns to distinguish EXI streams from other data formats, version information, and parameters, or EXI options, used for encoding. The EXI options can be used to adjust the properties of the resulting stream and the EXI processing to meet the requirements of various application domains. For example, the *selfContained* option allows random access to certain parts of the stream in exchange for greater memory usage during processing. Another example is the *compression* option, which enables the use of the DEFLATE compression algorithm [69] on top of the EXI code to further increase the level of compression in exchange for slower processing and increased memory usage. In general, each set of values for the EXI options is a trade-off between memory usage, processing speed, and the level of compression. A description of other options that are significant for the discussions in this thesis is provided below, along with the related data models and codes because many of the options are configuration parameters for the compaction techniques used in EXI.

The composite XML data model used in EXI is a combination of a syntactic source model based on formal grammars that is used to model structural redundancy, dictionary techniques for the efficient representation of reoccurring patterns, and some data-type specific models. The EXI specification defines a variation of the original syntactic source model proposed by Cameron [57]. Instead of using a single context-free grammar to represent the entire input XML document, the EXI specification relies on a set of regular grammars over the alphabet of XML parsing events, augmented with three EXI-specific events (*self_contained*, *start_document*, and *end_document*). Each regular grammar defines the structural constraints for a particular XML element. The processing of nested elements corresponds to the building of a stack of regular grammars; each start element pushes the grammar that describes its content to the grammar stack, and each end element pops a grammar from the grammar stack. The EXI grammars are right regular formal grammars that have exactly one terminal symbol followed by at most one non-terminal symbol on the right-hand side of their productions. That is, all grammar productions in the EXI grammar $G(N,T,S,P)$ are in one of the two forms:

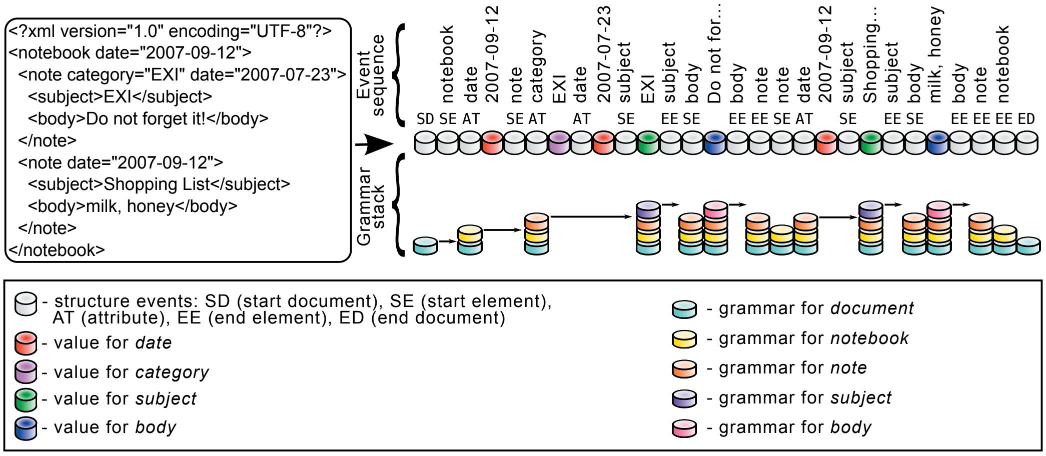
$$X \Rightarrow aY \quad or \quad X \Rightarrow a, \quad where \quad X \in N, \quad Y \in N, \quad a \in T$$

Additionally, the EXI grammars are simple grammars (also known as LL(0) or *s-grammars*) and consequently do not contain two or more grammar productions with the same non-terminal on the left-hand side and the same terminal symbol on the right-hand side. Therefore, the EXI grammars are unambiguous (i.e. every valid sequence of XML events has a unique leftmost derivation) and are parsed in linear time without lookahead using deterministic finite automaton (DFA).

There are two categories of EXI grammars: generic, or built-in, and schema-informed. The generic grammars describe the syntax restrictions posed by the XML language itself and are adaptive. They can describe the content of any XML element and are designed to adapt to the input during processing by adding element-specific grammar productions. The schema-informed grammars are static and do not change during encoding or decoding. The schema-informed grammars describe the constraints defined in a particular XML Schema and are used for processing XML instances that are known to be valid according to that XML Schema. When XML Schema is available, the use of schema-informed grammars in the source model leads to better compaction because the constraints in the schema, and thus the redundancy in the source data, are reflected in the resulting code. Because schema-informed grammars model exactly the constraints in specific XML Schema, the schema or the grammars themselves must be known by both the encoder and decoder. The use of schema-informed grammars is controlled by an EXI header option called *schemaId*. The *schemaId* option specifies whether schema information is used for processing, i.e. whether only generic grammars are used (a schema-less EXI stream) or there are also schema-informed grammars (a schema-mode EXI stream). Moreover, when in schema mode, the *schemaId* option also identifies the schema information used for processing. Additionally, the boolean EXI option *strict* can be used to allow (when *strict* is set to false) or forbid (when *strict* is true) deviations from the schema. Deviations can be useful when implementing schema versioning or handling exceptional cases and unplanned extensions from the schema. When *strict* is set to false, i.e. deviations are allowed, the entropy of the input sequence of XML events is higher, and thus the level of compaction is lower.

As already mentioned, the use of schema mode encoding leads to better compaction, but it requires that the exact XML Schema definitions (or the schema-informed grammars corresponding to them) be shared between the encoder and decoder. In other words, the schema-informed grammars, when available, are part of the source model. In many applications, this requirement is easily met because the schema is known during compile time and it does not change. For other applications that require dynamic schema negotiation (e.g., generic Web services or run-time service composition), the need for pre-shared schema information becomes a problem. The ability to dynamically exchange schema information efficiently is a key research direction in this thesis. The focus of the investigation is to define an efficient yet standard-based way to exchange the schema information and then convert it to schema-informed grammars that can support both strict (i.e. no deviations are allowed) and non-strict processing mode.

The body of an EXI stream is an encoded sequence of EXI events (i.e. XML events and EXI-specific events) that define an XML tree abstracted from the lexical representation. The code for the sequence is based on the syntactic source model in which each EXI event is encoded as a derivation step according to a regular grammar. The regular grammar used for the encoding of each EXI event is determined by a grammar stack that models the nesting of XML elements. Figure 3.2 shows an example XML instance and its corresponding event sequence as well as the use of the grammar stack.



```
<?xml version="1.0" encoding="UTF-8"?>
<notebook date="2007-09-12">
  <note category="EXI" date="2007-07-23">
    <subject>EXI</subject>
    <body>Do not forget it!</body>
  </note>
  <note date="2007-09-12">
    <subject>Shopping List</subject>
    <body>milk, honey</body>
  </note>
</notebook>
```

Figure 3.2: Representation of a sample XML tree through a sequence of EXI events, and the corresponding grammar stack used for EXI processing. The grammar on top of the stack is used for the encoding and decoding of the EXI events. (Extension of a figure originally published in the W3C EXI Primer [70])

Each EXI event (i.e. derivation step) is coded using a simplified Huffman code in which the grammar productions are assigned a prefix code with at most three unsigned integer parts. In this way, the alternative productions for each derivation step are divided into three groups depending on the probability of occurrence of the terminal symbol (i.e. the EXI event) on their right-hand side. The most probable productions have codes with only one part, and the least probable productions have codes with three parts. This differs from the original Huffman code, which requires that the exact probability of each input symbol be known and that the code not be partitioned into a fixed number of parts.

Each part of the EXI prefix code is an unsigned integer that is coded using the least number of bits. The binary length of each part (i.e. the number of bits used to code the unsigned integer) is determined according to the number of productions in the grammar defining the alternative derivation paths for a particular non-terminal symbol with the same number of parts in the prefix code. Because generic grammars are adaptive and allow the addition of productions during processing, the code for the generic grammar productions is also adaptive and varies depending on the context in the EXI stream.

The use of the syntactic source model and simplified Huffman coding reduce the structural redundancy by efficiently representing the EXI event sequence using a stack of regular grammars. However, the qualified names and values for the elements and attributes are not compacted with this scheme. As shown in Figure 3.2, element and attribute names and their values are often repeated in the XML input source. For example, in the sample document shown in Figure 3.2, the elements *<note>*, *<subject>*, and *<body>* are repeated twice, the attribute *date* occurs three times, and the values *"2007-09-12"* and *"EXI"* are also repeated. To model this type of redundancy, the EXI specification defines a set of dictionaries, called string tables or string table partitions, where the reoccurring character strings are indexed. Multiple dictionaries are used to model the correlation between the context in the EXI event sequence and the probability of occurrence of certain strings (e.g., *<subject>* always comes after the *<note>* element in the sample XML document). In this way, different contexts in the source stream are indexed by different dictionaries, which makes the references (indexes) to the strings in the dictionaries very compact. The string tables, or EXI dictionaries, are adaptive and are updated during processing. To control the size of the string tables and thus the memory usage during encoding and decoding, the EXI specification defines the header options *valueMaxLength* (to restrict the maximum length of the strings considered for addition to the value string table partition) and *valuePartitionCapacity* (to restrict the maximum number of entries in the value string table partition). Additionally, EXI statically assigns indexes to certain strings that are known to occur frequently in many XML documents (e.g. the namespace "http://www.w3.org/2001/XMLSchema"). The binary length of the code used to represent dictionary references (indexes) in the EXI stream is the minimum number of bits required to represent all entries in the dictionary. EXI defines two ways to mark whether the string is to be encoded literally in the stream (at the first occurrence) or is to be encoded by its index in the dictionary (at the nth occurrence, for $n > 1$) depending on the probability of repetition of the string. Some string tables are optimized for the frequent use of indexes, and the mark (i.e. the bit sequence encoded in the EXI stream) for the first occurrence is longer than the mark for the nth occurrence; other string tables are optimized for the frequent use of string literals, and the mark for the first occurrence is shorter than the mark for the nth occurrence.

When XML Schema is used during the encoding of EXI streams, the content of elements and attributes can be assigned data types. As already discussed, the data types place constraints on the range of valid values, which enables the use of models and codes that produce a more compact representation than simply encoding the typed value as a character string. For implementation simplicity, the EXI specification defines a small number of core models and codes for data type encoding that are reused to represent all XML Schema data types. An example of a core data type representation is the unsigned integer encoder. This encoder represents arbitrarily large unsigned integer values using a variable-length code and a probability model that assigns a lower probability for large values than for small values. The unsigned integer code is then used in the representation of most other data types, such as signed integers, strings, and decimals. Another example of a specific data type encoder is the EXI representation of the *dateTime* type, which uses an offset value to encode the year and time zone components and takes into account the maximum possible values of the minute, hour, and second time components.

The presentation of the EXI format in this section has described its main compaction techniques solely from a theoretical perspective. For a detailed description of EXI processing mechanisms, including the generation of schema-informed grammars from XML Schema definitions, the reader is advised to refer to the EXI specification itself [67] or to the accompanying EXI Primer document [70].

Chapter 4

Contributions

This chapter summarizes the author's research contributions included in this thesis by describing how the formulated research questions were addressed in the appended papers. The contributions are given in chronological order, reflecting the progress of the research work over the course of the doctoral program.

The literature review presented in Paper A helped to shape the first research question by identifying the gap between solving the overhead problem in Web service technology on one hand and avoiding the use of gateways, which violate the end-to-end networking principle, on the other hand. The design, implementation, and evaluation of a lightweight XML Web service prototype in Paper A answered the first research question, on whether it is feasible to deploy standard Web services on resource-constrained devices, by demonstrating that although the use of XML was possible, it significantly reduced the performance and applicability of the solution. Paper A also led to the formulation of the second research problem: *What are the trade-offs in using Web services for end-to-end interoperability of resource-constrained embedded systems?*

In addressing this question, Paper B and Paper C developed the hypothesis that the use of EXI could increase the efficiency of embedded Web services. In Paper B, a novel low-level EXI processor interface was developed and evaluated. The interface served as part of the testbed used in Paper C to analyze the feasibility of using EXI-based Web services to provide application layer interoperability in industrial applications.

Paper D extended the Web service technology trade-off analysis by investigating the use of UDP/-SOAP and EXI in a concrete application domain: energy management in district heating substations. Different levels of interoperability were discussed, and the need for standards describing the semantics of sensor measurements was highlighted as an important factor in establishing end-to-end application interoperability in heterogeneous systems.

The work described in Paper E built on the semantics requirements defined in Paper D and investigated different energy management standards. Paper E presented an application layer technology analysis and a roadmap for energy management protocols for the Internet of Things.

These studies showed that the use of EXI is beneficial for achieving end-to-end interoperability of Web services in the IoT. However, the possibility of providing advanced functionality, such as Web-based human-machine interfaces or the rapid development of embedded applications, had not been considered for embedded EXI Web services. To this end, the third research question was formulated: *What levels of efficiency and functionality can be achieved using binary coding schemes for XML data* *exchange?*

The initial investigation of this problem was presented in Paper F with a pilot study of the use of EXI RESTful Web services for providing a Web-based human-machine interface for embedded platforms. The third research question was fully addressed in paper G with the development and evaluation of a complete EXI-based Web development toolkit.

4.1 Summary of Appended Papers

Paper A: Integration of Wireless Sensor and Actuator Nodes with IT Infrastructure Using Service-Oriented Architecture
Authors: Rumen Kyusakov, Jens Eliasson, Jerker Delsing, Jan van Deventer, and Jonas Gustafsson
Published in: IEEE Transactions on Industrial Informatics, vol. 9, no. 1, pp. 43-51, Feb. 2013

Summary: The paper presents a survey of the use of SOA in wireless sensor networks and reveals a contradiction in the literature on whether the application of SOA solutions in resource-constrained devices is beneficial or even possible. This question is further investigated by developing and testing a constrained Web service solution for sensor nodes. The proposed solution combines an application-specific XML parser supporting parsing on the fly and a low level TCP/IP stack. The performance measurements suggest that this approach can be used to implement a limited SOA-based interface for wireless sensor platforms; however, the interface substantially decreases the communication speed and response time. Paper A proposes the use of Efficient XML Interchange as a possible approach to overcome the inefficiency of the XML-based Web service implementation.

Contribution: The author prepared the literature review and designed, developed, and tested the constrained Web service solution on a concrete sensor platform.

Paper B: Efficient Structured Data Processing for Web Service Enabled Shop Floor Devices
Authors: Rumen Kyusakov, Jens Eliasson, and Jerker Delsing
Published in: IEEE International Symposium on Industrial Electronics (ISIE), pp. 1716-1721, 27-30 Jun. 2011

Summary: Paper B proposes a design for an EXI processor targeted for highly resource-constrained embedded devices in industrial installations. The goal of the EXI processor design is to increase the efficiency (i.e. processing speed and response time) of Web service implementations that are currently based on XML. Additionally, the paper presents and evaluates a novel low level EXI processor Application Programming Interface (API) for the parsing and serialization of EXI data streams. An open source prototype of an EXI processor featuring the new API is also described.

Contribution: The author developed and tested the hypothesis that a low level EXI processor API could be used to increase the efficiency of EXI-based Web services. As part of the hypothesis testing process, the author designed and implemented an open source prototype of the EXI processor.

Paper C: Efficient XML Interchange in Factory Automation Systems
Authors: Rumen Kyusakov, Henrik Mäkitaavola, Jerker Delsing, and Jens Eliasson
Published in: 37th Annual Conference on IEEE Industrial Electronics Society (IECON), pp. 4478-4483, 7-10 Nov. 2011

Summary: Paper C investigates the application of the EXI format in the XML-based OPC-UA and DPWS communication protocols. As part of this investigation, the paper presents a trade-off analysis of the use of EXI in an industrial environment and proposes three approaches for the integration of legacy systems in EXI-based Web service communication. Additionally, this study includes the development of a prototype demonstrating the possibility of using the EXI format to bridge the gap between Web service technology and application protocols on resource-constrained sensors and actuators.

Contribution: The author developed the methodology and implemented the trade-off analysis of EXI usage in an industrial environment. Additionally, the author designed and took part in the implementation of the EXI-based Web service testbed.

Paper D: Application of Service Oriented Architecture for Sensors and Actuators in District Heating Substations
Authors: Jonas Gustafsson, Rumen Kyusakov, Henrik Mäkitaavola, and Jerker Delsing
Published in: MDPI Sensors, vol. 14, no. 8, pp. 15553-15572, Aug. 2014

Summary: Paper D considers the use of SOA and EXI to enable efficient yet interoperable information exchange in district heating substations. The paper proposes the use of Web services based on a IPv6/UDP/SOAP/EXI communication stack for wireless sensor data acquisition and substation control. The use of loosely coupled sensors and actuators with the capability of reconfiguring and interconnecting them after the initial deployment can enable new types of customer services. This opens up the possibility of the provision of system-wide control, overall optimization of district heating, remote meter reading, fault detection, and improved heating system control by heat suppliers or third party companies. Additionally, Paper D investigates the use of standardized syntax and semantic descriptions of sensor measurements and evaluates the proposed architecture.

Contribution: The author designed, developed, and evaluated the UDP/SOAP/EXI Web service solution and was a main contributor to its integration into a TinyOS-based wireless sensor network prototype.

Paper E: Emerging Energy Management Standards and Technologies - Challenges and Application Prospects
Authors: Rumen Kyusakov, Robert Cragie, Jens Eliasson, Jan van Deventer, and Jerker Delsing
Published in: IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA), pp. 1-8, 17-21 Sep. 2012

Summary: Paper E presents a survey of emerging energy management standards with a focus on enabling application layer technologies for the system-wide interoperability of automated demand response and load management (DRLM) programs. The presented work includes an analysis of the challenges, future trends, security, and application prospects for several energy management standards. A common trend identified in the analysis is the use of Internet-based communication protocols such as IPv6 (through the 6LoWPAN adaptation scheme), XML, and XML Schema for the message format and data model definitions, combined with a possible EXI-based compact representation and RESTful Web services. Additionally, security is identified as a major challenge for devices that have limited resources and no user interface and require cost-effective installation procedures.

Contribution: The author performed a systematic study of the requirements of DRLM programs and how they are addressed in ZigBee Smart Energy Profile (SEP) version 2.0, Open Automated Demand Response (OpenADR) version 2.0, and several building and industrial automation protocols. Based on that study, the author compiled an application layer technology analysis and roadmap for DRLM.

Paper F: Enabling Cloud-connectivity for Mobile Internet of Things Applications
Authors: Pablo Puñal Pereira, Jens Eliasson, Rumen Kyusakov, Jerker Delsing, Asma Raayatinezhad, and Mia Johansson
Published in: IEEE 7th International Symposium on Service-Oriented System Engineering (SOSE), pp. 518-526, 25-28 Mar. 2013

Summary: Paper F proposes a holistic SOA-based network architecture to provide cloud connectivity for mobile low-power IoT devices. The architecture consists of a network connectivity component based on the Bluetooth personal area network profile (PAN), CoAP RESTful application services, and SOA-based cloud integration. Another component of the architecture is a web-based human-machine interface for the configuration, monitoring, and visualization of sensors and actuators based on the XHTML and EXI Web technologies. The network connectivity component and RESTful application services are further evaluated by performing experimental measurements on a small-scale mobile IoT prototype. The results of the experiment show that the proposed architecture can support sample rates of up to several kilohertz while enabling sensor data to be transmitted in near real time using RESTful Web services on resource-constrained mobile platforms.

Contribution: The author analyzed the requirements for a human-machine interface for mobile IoT applications and proposed a Web-based solution using XHTML/EXI to meet these requirements. Additionally, the author suggested an approach for integrating the Web-based human-machine interface into the mobile cloud connectivity architecture.

Paper G: EXIP: A Framework for Embedded Web Development
Authors: Rumen Kyusakov, Pablo Puñal Pereira, Jens Eliasson, and Jerker Delsing
Published in: Accepted for publication in ACM Transactions on the Web (TWEB) on 23 Aug. 2014

Summary: Paper G presents the design and implementation of an embedded Web development framework based on CoAP/EXI/XHTML technologies. The main component of the framework is a lightweight EXI processor that supports efficient run-time processing of XML Schema definitions. Furthermore, the paper proposes a novel design for EXI data binding based on templates and a CoAP/EXI/XHTML Web page engine, for developing Web-based human-machine interfaces for resource-constrained devices. The key results presented in the paper are (I) a theoretical and practical evaluation of the use of binary protocols for embedded Web programming, (II) a novel method for the generation of EXI grammars based on XML Schema definitions, (III) a grammar concatenation algorithm for normalized EXI grammars, and (IV) an algorithm for the efficient representation of possible deviations from XML schema in EXI processors.

Contribution: The author invented, implemented, and evaluated the EXI grammar generation algorithms (II, III and IV) and formulated the overall architecture of the EXIP framework, including the design of the EXI data binding approach and the CoAP/EXI/XHTML Web page engine. Additionally, the author developed the methodology and compiled the evaluation of the use of binary protocols for embedded Web programming (I).

4.2 Related Publications

This section lists publications containing contributions from the author's research work which are related to, but not included in this thesis.

1. P. Nappey, C. El Kaed, AW. Colombo, J. Eliasson, A. Kruglyak, R. Kyusakov, C. Hubner, T. Bangemann, and O. Carlsson, "Migration of a legacy plant lubrication system to SOA," Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE, pp. 7440-7445, 10-13 Nov. 2013

2. J. Eliasson, R. Kyusakov, and P-E. Martinsson, "An internet of things approach for intelligent monitoring of conveyor belt rollers," 10th International Conference on Condition Monitoring and Machinery Failure Prevention Technologies 2013, CM 2013 and MFPT 2013, vol. 2, pp. 1096-1104, 20 Jun. 2013

3. P. Lindgren, R. Kyusakov, J. Eliasson, H. Makitaavola, and P. Pietrzak, "A SOA approach to delay and jitter tolerant distributed real-time Complex Event Processing," Industrial Electronics (ISIE), 2013 IEEE International Symposium on, pp. 1-7, 28-31 May 2013

4. J. Eliasson, J. Delsing, A. Raayatinezhad, and R. Kyusakov, "A SOA-based framework for integration of intelligent rock bolts with Internet of Things," Industrial Technology (ICIT), 2013 IEEE International Conference on, pp 1962-1967, 25-28 Feb. 2013

5. F. Jammes, B. Bony, P. Nappey, A. W. Colombo, J. Delsing, J. Eliasson, R. Kyusakov, S. Karnouskos, P. Stluka, and M. Till, "Technologies for SOA-based distributed large scale process monitoring and control systems," IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society, pp 5799-5804, 25-28 Oct. 2012

6. J. Delsing, J. Eliasson, R. Kyusakov, AW. Colombo, F. Jammes, J. Nessaether, S. Karnouskos, and C. Diedrich, ”A migration approach towards a SOA-based next generation process control and monitoring,” IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society, pp 4472-4477, 7-10 Nov. 2011

CHAPTER 5

Discussion

Before continuing with the detailed description of the thesis work in Part II, it is useful to take a step back and look at the overall results of the presented research. Figure 5.1 depicts the generic steps involved in the research work. As shown in the figure, the starting state of the process is the initial real-world problem that provides the basis for the PhD studies, i.e. the lack of interoperability between different application layer technologies in resource-constrained embedded systems. This also provides the motivation for the research in the thesis, which is represented in the first research question and leads to the development of the main hypothesis, which is that Web services are beneficial for solving the interoperability problem. The hypothesis is then tested using a research methodology that is mostly based on experimental work. The results from the experiments are evaluated against the hypothesis, often raising new research questions. After several such iterations, the accumulated results have sufficient weight to be evaluated against the initial real-world problem. This final evaluation is the focus of this last chapter in Part I of this thesis.

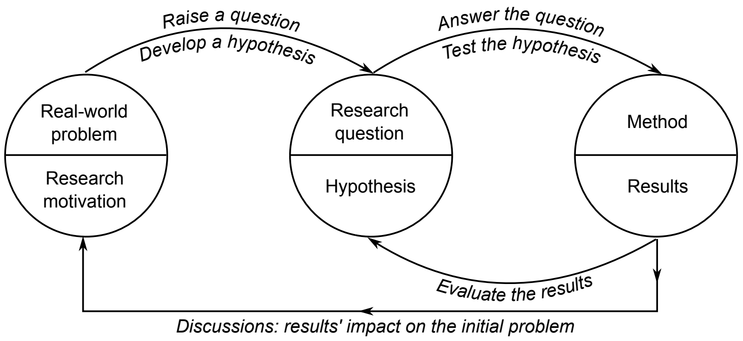


Figure 5.1: Typical workflow for the research work presented in this thesis.

The accumulated results demonstrate that the use of Web service technology is a viable approach for providing end-to-end application layer interoperability for resource-constrained embedded systems. It is shown that the traditional Web service protocol stack (TCP/HTTP/SOAP/XML) incurs high overhead in terms of network bandwidth, volatile and non-volatile memory usage, and processing power. This makes the application of TCP/HTTP/SOAP/XML on embedded devices problematic because it results in high latency, high energy consumption and the need for more expensive hardware and/or a larger form factor. High latency is only acceptable in limited application scenarios in which the monitored or controlled process is very slow (e.g., district heating). Energy consumption is always an issue for battery-powered devices because it results in higher maintenance costs for changing or recharging the batteries. Furthermore, the use of even slightly more expensive hardware in embedded devices results in a substantial increase in the cost of the whole installation when the number of devices is very large (e.g., tens of thousands or more).

The results of this thesis suggest that the use of embedded RESTful Web services based on a UDP/CoAP/EXI protocol stack mitigates most of the overhead problems. The use of UDP/CoAP is suggested based on the studies by Kuladinithi et al. [71] and Colitti et al. [72], which show that CoAP significantly reduces the response times and energy footprint in wireless battery-powered devices when compared to HTTP. EXI substantially decreases the size of the XML messages, which leads to a smaller number of packets sent and received and also reduces the energy footprint of the wireless communications when compared to XML. Nevertheless, EXI introduces new problems when used to develop embedded Web services: integration with XML-based and legacy systems is not directly provided, a pre-negotiated XML Schema is required when schema-mode encoding is enabled, higher processing and memory requirements are necessary than for ad-hoc data formats, and there is a lack of development tools.

The EXI specification provides an unambiguous way to translate between XML and EXI, which allows for transparent and non-intrusive EXI proxies when used in schema-less mode. The use of EXI in schema mode requires knowledge of the exact XML Schema used for encoding by the communicating parties and the proxy. The thesis results show that a set of EXI grammar generation algorithms can be used to enable efficient XML Schema negotiation to support schema-mode EXI encoding, even in scenarios with EXI proxies and dynamically defined XML Schema definitions. Additionally, the thesis presents the design and overall architecture of a complete embedded Web service development framework that addresses the requirement for efficient tool support when developing Web services based on UDP/CoAP/EXI [73].

As part of the thesis results, a low level EXI processor API is presented that enables faster processing speeds than traditional string-based APIs. Additionally, an algorithm for the efficient representation of schema deviations is developed that enables more compact schema-mode EXI grammars that support schema deviations. This algorithm reduces the run-time memory required to process EXI streams in non-strict schema mode. Nevertheless, the memory usage, in terms of programming memory and RAM, and the processing speed can still be significant concerns in highly resource-constrained devices that use EXI. This necessitates further investigation of possible optimization of the memory usage and processing speed, such as the hardware-based EXI parser proposed by Altmann et al. [74].

Compared to highly optimized ad-hoc application protocols, the proposed UDP/CoAP/EXI Web service technology provides better interoperability and comparable message sizes, but due to the EXI processing, it generally requires more memory and time to process. This comparison is very imprecise, but nevertheless, it illustrates the overall strengths and weaknesses of the proposed embedded Web service stack. A more comprehensive comparison would require a set of concrete test messages exchanged using the two approaches on a particular hardware setup. This would also provide only an approximate comparison because the rate at which computing hardware evolves makes such quantitative comparisons obsolete soon after they are released. It is therefore beneficial to estimate the effects of the evolution of computing hardware on the results of this thesis. One way to make an in-

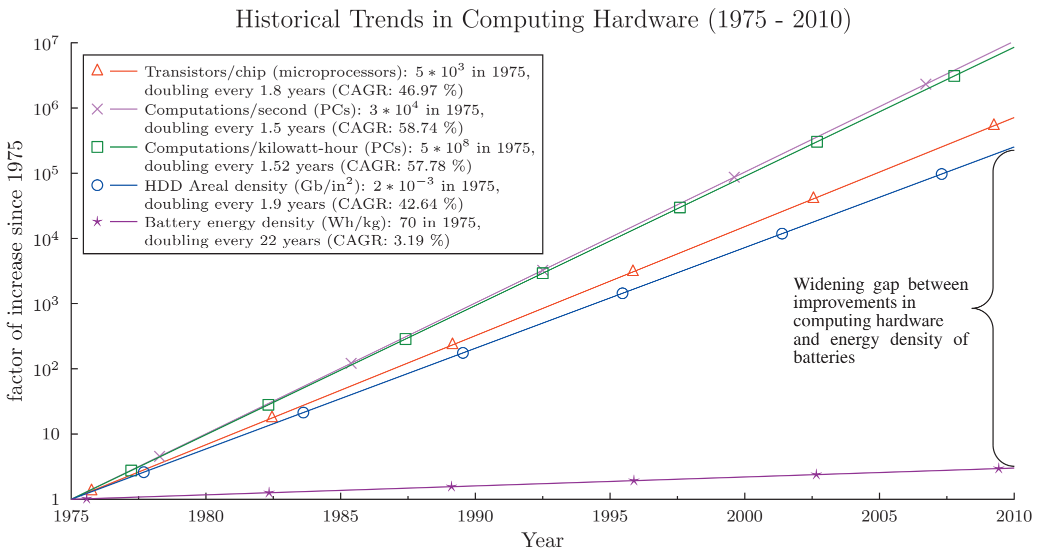


Figure 5.2: Historical trends in computing hardware compared to the energy density of rechargeable batteries. The graph depicts the factor of increase in various hardware parameters since 1975 on a logarithmic scale. The factor of increase is approximated using regression analysis and/or the compound annual growth rate (CAGR). For ease of comparison, the doubling time and the CAGR are calculated for each hardware parameter. *Data sources: Koomey et al. [75] (transistors per chip, computations per second, and computations per kilowatt-hour); Bandic et al. [76], Inamura et al. [77], and Grochowski et al. [78] (HDD areal density); and Zu et al. [79] (battery energy density).*

formed prediction or projection of future developments in computing hardware is to rely on empirical observations such as Moore's law. Figure 5.2 illustrates the historical trends in computing hardware for a period of 35 years (1975 - 2010). As shown in the figure, the number of transistors used in microprocessors doubles approximately every 1.8 years, following Moore's law. Comparable and even faster growth has been observed in the number of transistors (and consequently the storage capacity) in memory chips [80]. Also related to Moore's law is the exponential increase in the maximum number of computations performed per second by personal computers, which has doubled every 1.5 years since 1975. Two computing hardware parameters that have followed a comparable exponential increase but are not directly a result of Moore's law are the hard disk drive (HDD) areal density (given as Gb/in^2) and the energy efficiency of computing (given as the number of computations per kilowatt-hour). The power efficiency of computation in personal computers has doubled every 1.52 years, which has enabled the transition from purely stationary computers to today's proliferation of mobile computing devices. The growth of mobile computing appears to be due to the exponential increase in computational efficiency as opposed to the improvements in battery technology because battery energy density is increasing very slowly relative to Moore's law. The widening gap between improvements in computing hardware and the energy density of batteries seems inevitable unless a breakthrough is made in the area of energy storage [81].

Consequently, one question that this chapter attempts to answer is, *What are the implications of* *the future trends in computing hardware on the thesis results under the assumption that computing hardware and batteries will evolve at a similar pace as they have for the last 35 years?*

An important parameter that has not been discussed yet but is nevertheless important for providing a relevant answer to this question is the cost in terms of energy of communication between devices. One possible measure for this is the amount of data that can be sent from host A to host B using a particular amount of energy (e.g., the measurement units could be Mb/kilowatt-hour). This is a very important parameter for wireless battery-powered devices because it heavily affects their ability to communicate over long periods of time. Unfortunately, information on how this parameter has increased over the years and what the future trends might be for its evolution is not available in the literature. A study that provides some relevant results has been presented by the GreenTouch consortium in their white paper on possible technologies, architectures, components, algorithms, and protocols for reducing energy consumption in communication networks [82]. The study predicts that it will be possible to achieve an improvement of a factor of 1043 in the energy efficiency of mobile networks in 2020 relative to the energy efficiency in 2010. This prediction was developed for LTE mobile networks only and is based on various improvements in the base stations, but it does not consider the energy efficiency of the mobile devices.

Approaching the aforementioned question requires one more assumption, which is that the increase in the energy efficiency of wireless communications is growing at a slower pace than the efficiency of computation, which is doubling approximately every 1.52 years. This would mean that in the (near) future, the total energy consumption of networked wireless devices would continue to be dominated by the energy consumption of the communications. This would result in the communications taking as much as 51 % (as estimated for the current hardware technology [83]) or even more of the total energy available on a resource-constrained wireless device.

Under these assumptions, future wireless battery-powered devices will have much more memory and processing capability, which are much cheaper in terms of energy consumption, and will also be equipped with batteries with slightly better capacity than today's devices from the same class (in terms of the form factor and hardware price). This will extend the impact of the proposed UDP/CoAP/EXI stack to devices that are currently too constrained in terms of memory and processing power and must rely on ad hoc application protocols. Moreover, because communications will continue to be a major source of energy consumption in wireless battery-powered devices, a small message size will continue to be advantageous in application layer protocols in the future. These arguments, together with the ability to provide interoperability with other Web technologies, make the proposed UDP/CoAP/EXI stack a future-proof solution for providing end-to-end interoperability in embedded networks.

5.1 Conclusion

The main outcome of this thesis is the development of methods and tools to enable interoperable EXI-based Web services for resource-constrained embedded devices. It is shown that the UDP/CoAP/EXI protocol stack has the potential to solve the overhead problem of XML-based Web services while conforming to the end-to-end networking principle. This claim is supported by the evaluation of different Web service application protocols deployed on IoT prototypes targeted for industrial and home automation applications. The evaluation suggests that the areas with the greatest dependency on cross-domain heterogeneous system integration (e.g., smart grids) are major beneficiaries of the proposed embedded Web service solution.

The presented study includes an analysis of how the EXI-based Web services can be integrated with existing Web technologies. As part of this analysis, it is shown that EXI-encoded XHTML visualization technology can be readily used to provide Web-based human-machine interaction for resource-constrained devices. Additionally, this thesis proposes a novel approach to solve the problem of efficient dynamic XML schema negotiation, which is needed in many distributed environments in which EXI-encoded messages are exchanged in schema mode. The approach consists of efficient algorithms for the generation of EXI grammars from EXI-encoded XML Schema definitions. Finally, the design and overall architecture of a complete framework for the development of EXI-based embedded Web services is presented and evaluated.

The analysis of future trends in computing hardware indicates that the need for a compact representation of the information exchanged in networks of embedded battery-powered devices will continue to be an important factor for the success of any Web service technology, but the availability and energy efficiency of memory and processing capabilities on the devices will likely improve substantially. These developments would greatly extend the impact of the proposed EXI-based Web services to device classes that are currently only capable of application-specific ad-hoc communication.

Coming back to the research questions of this thesis, the answers to them can be summarized as follows:

1. *Is it feasible to deploy standard Web services on IoT systems without using application layer gateways?*

The results of this thesis show that it is feasible to use Web services without relying on gateways with certain limitations on available memory, processing capabilities, and latency. The concrete limits depend on the specific application and on the use of concrete Web service technology. The thesis shows that the use of some emerging binary protocols for implementing RESTful Web services can substantially reduce these limitations.

2. *What are the trade-offs in using Web services for end-to-end interoperability of resource constrained embedded systems?*

In general, providing Web service support in embedded systems requires more capable hardware platforms and thus larger form factor and/or hardware cost than ad-hoc application layer solutions. Additionally, the end-to-end interoperability requires content negotiation mechanisms and transparent proxies when traditional Web technologies are interfaced with the binary transport schemes (e.g., CoAP) and data formats (e.g., EXI) used for embedded Web service implementation.

3. *What levels of efficiency and functionality can be achieved using binary coding schemes for XML data exchange?*

Depending on the use cases, Efficient XML Interchange can compact Web service messages up to 95 % from their original (i.e. plain XML) size. For small messages the use of schema mode encoding is required to achieve high level of compaction. However, this implies that the schema used during encoding is pre-shared between the encoder and decoder. The results of this thesis demonstrate that efficient run-time schema negotiation is possible for device with at least 100 kB of RAM.

5.2 Future work

Possible extensions of this work include further optimization of the EXI processing to reduce the memory and processing requirements. Another interesting future research direction with implications for the Web integration of embedded devices is the generalization of the method for XML compaction to other context-free languages. A typical application for such generalization would be to support an efficient REST code-on-demand model in which a lightweight scripting code (e.g., binary-encoded JavaScript) could be delivered to, and interpreted by, the clients to extend their functionality.

The scope of this thesis is intentionally limited to syntactic interoperability on the application layer because this is the foundation on which more complex semantic definitions can be established. However, full interoperability across dynamically deployed systems and services, including support for the run-time discovery and composition of services, can only be achieved when there are mechanisms in place allowing the precise description of the functionality provided by, or requested by, a particular service. Moreover, these descriptions must be both machine readable and human readable to enable autonomous service discovery, invocation, and composition on one hand and control, monitoring, and verification of the functionality by humans on the other hand.

Only a limited investigation of how similar functionality can be achieved using domain-specific application standards developed by experts in a particular domain is presented in the thesis. These standards define static mappings between the syntax used for information exchange and the semantics of the exchanged messages, i.e. they specify a domain ontology. Even this approach has severe limitations because the mapping is restricted to a particular application domain. For example, the SEP2 specification defines the semantics of the messages only in demand response and load control applications and cannot provide semantic level interoperability with home automation applications, for example. However, very often the optimization of complex processes requires the cross-domain exchange of information, which then requires the definition of additional mappings between the data models (i.e. the ontologies) developed for each domain. Developing and maintaining such static mappings for multiple application domains requires substantial manual work, which is the main challenge in applying the ontology approach for semantic interoperability. With this in mind, an important future direction for the work presented in this thesis is the investigation of efficient ways of supporting semantic interoperability, even over a limited set of operations, e.g., supporting only dynamic data visualization, storage, and retrieval.

References

[1] M. Weiser, “The computer for the 21st century,” *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[2] O. Vermesan, P. Friess, P. Guillemin, H. Sundmaeker, M. Eisenhauer, K. Moessner, F. Le Gall, and P. Cousin, *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers, 2013, ch. 2. Internet of Things Strategic Research and Innovation Agenda, pp. 7–151.

[3] M. Durvy, J. Abeillé, P. Wetterwald, C. O’Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels, “Making sensor networks ipv6 ready,” in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys ’08. New York, NY, USA: ACM, 2008, pp. 421–422. [Online]. Available: http://doi.acm.org/10.1145/1460412.1460483

[4] Z. Shelby and C. Bormann, *6LoWPAN: The Wireless Embedded Internet*, Wiley, Ed. Wiley, November 2009.

[5] F. Jammes and H. Smit, “Service-oriented paradigms in industrial automation,” *Industrial Informatics, IEEE Transactions on*, vol. 1, no. 1, pp. 62 – 70, 2005.

[6] D. Barisic, M. Krogmann, G. Stromberg, and P. Schramm, “Making Embedded Software Development More Efficient with SOA,” in *Advanced Information Networking and Applications Workshops, 2007, AINAW ’07*, 2007.

[7] C. Groba and S. Clarke, “Web services on embedded systems - a performance study,” in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, march 2010, pp. 726 –731.

[8] V. Trifa, S. Wieland, D. Guinard, and T. M. Bohnert, “Design and implementation of a gateway for web-based interaction and management of embedded devices,” in *Proceedings of the 2nd International Workshop on Sensor Network Engineering (IWSNE 09)*, Marina del Rey, CA, USA, Jun. 2009.

[9] J. H. Saltzer, D. P. Reed, and D. D. Clark, “End-to-end arguments in system design,” *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 4, pp. 277–288, 1984.

[10] L. Snyder, F. Baskett, M. Carroll, M. Coleman, D. Estrin, M. Furst, J. Hennessy, H. Kung, K. Maly, and B. Reid, *Academic Careers for Experimental Computer Scientists and Engineers*. The National Academies Press, 1994, no. 0-309-58568-6, ch. 1.

What is experimental computer science and engineering?, pp. 9–33. [Online]. Available: http://www.nap.edu/openbook.php?record_id=2236

[11] B. Liskov, "Report on workshop on research in experimental computer science," MIT, Tech. Rep., 1992.

[12] M. Papazoglou and W.-J. van den Heuvel, "Service oriented architectures: approaches, technologies and research issues," *The VLDB Journal*, vol. 16, no. 3, pp. 389–415, 2007. [Online]. Available: http://dx.doi.org/10.1007/s00778-007-0044-3

[13] F. Blomstedt, L. L. Ferreira, M. Klisics, C. Chrysoulas, I. M. de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga, "The Arrowhead Approach for SOA Application Development and Documentation," in *Industrial Electronics Society, IECON 2014, 40th Annual Conference of IEEE*, 2014.

[14] N. Fantana, T. Riedel, J. Schlick, S. Ferber, J. Hupp, S. Miles, F. Michahelles, and S. Svensson, *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers, 2013, ch. 3. IoT Applications — Value Creation for Industry, pp. 153–206.

[15] J. Quittek, M. Chandramouli, R. Winter, T. Dietz, and B. Claise, *Requirements for Energy Management*, Internet Engineering Task Force Std., 2013. [Online]. Available: http://tools.ietf.org/html/rfc6988

[16] F. Bouhafs, M. Mackay, and M. Merabti, "Links to the future: Communication requirements and challenges in the smart grid," *Power and Energy Magazine, IEEE*, vol. 10, no. 1, pp. 24 –32, jan.-feb. 2012.

[17] J. Wang and V. Leung, "A survey of technical requirements and consumer application standards for IP-based smart grid AMI network," in *Information Networking (ICOIN), 2011 International Conference on*, jan. 2011, pp. 114 –119.

[18] M. Kovatsch, M. Weiss, and D. Guinard, "Embedding internet technology for home automation," in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, Sept 2010, pp. 1–8.

[19] E. Callaway, P. Gorday, L. Hester, J. A. Gutierrez, M. Naeve, B. Heile, and V. Bahl, "Home networking with ieee 802. 15. 4: a developing standard for low-rate wireless personal area networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 70–77, 2002.

[20] J. Åkerberg, M. Gidlund, and M. Björkman, "Future research challenges in wireless sensor and actuator networks targeting industrial automation," in *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, July 2011, pp. 410–415.

[21] K. Chintalapudi and L. Venkatraman, "On the design of mac protocols for low-latency hard real-time discrete control applications over 802.15.4 hardware," in *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, April 2008, pp. 356–367.

[22] R. Braden, *Requirements for Internet Hosts – Communication Layers*, IETF Std., 1989. [Online]. Available: http://tools.ietf.org/html/rfc1122

[23] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, A. Terzis, A. Dunkels, and D. Culler, "Contikirpl and tinyrpl: Happy together," in *Workshop on Extending the Internet to Low Power and Lossy Networks (IP+ SN)*, 2011.

[24] IEEE Computer Society, *IEEE 802.15.4-2006*, IEEE Standards Association Std., 2006. [Online]. Available: http://standards.ieee.org/findstds/standard/802.15.4-2006.html

[25] Bluetooth protocols and specification. The Bluetooth Special Interest Group. [Online]. Available: https://www.bluetooth.org/Technical/Specifications/adopted.htm

[26] K. Razazian, M. Umari, A. Kamalizad, V. Loginov, and M. Navid, "G3-plc specification for powerline communication: Overview, system simulation and field trial results," in *Power Line Communications and Its Applications (ISPLC), 2010 IEEE International Symposium on*, March 2010, pp. 313–318.

[27] J. Hui and P. Thubert, *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*, IETF Std., 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6282.txt

[28] Z. Shelby, S. Chakrabarti, E. Nordmark, and C. Bormann, *Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*, IETF Std., 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6775.txt

[29] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, *Rpl: Ipv6 routing protocol for low-power and lossy networks*, IETF Std., 2012. [Online]. Available: http://tools.ietf.org/html/rfc6550

[30] J.-P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann, 2010.

[31] Z. Shelby, "Embedded web services," *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 52–57, 2010.

[32] J. E. White, "A high-level framework for network-based resource sharing," in *Proceedings of the June 7-10, 1976, National Computer Conference and Exposition*, ser. AFIPS '76. New York, NY, USA: ACM, 1976, pp. 561–570. [Online]. Available: http://doi.acm.org/10.1145/1499799.1499878

[33] M. Huhns and M. Singh, "Service-oriented computing: key concepts and principles," *Internet Computing, IEEE*, vol. 9, no. 1, pp. 75–81, Jan 2005.

[34] T. Berners-Lee, "Www: past, present, and future," *Computer*, vol. 29, no. 10, pp. 69–77, Oct 1996.

[35] T. Berners-Lee, R. Fielding, and L. Masinter, *Uniform Resource Identifiers (URI): Generic Syntax*, IETF Std., 1998. [Online]. Available: http://tools.ietf.org/html/rfc2396

[36] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, IETF Std., 1999. [Online]. Available: http://tools.ietf.org/html/rfc2616

[37] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, University of California, Irvine, 2000.

[38] R. T. Fielding and R. N. Taylor, “Principled design of the modern web architecture,” *ACM Transactions on Internet Technology*, vol. 2, pp. 115–150, 2002.

[39] C. Pautasso, O. Zimmermann, and F. Leymann, “Restful web services vs. ”big”’ web services: Making the right architectural decision,” in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW ’08. New York, NY, USA: ACM, 2008, pp. 805–814. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367606

[40] *SOAP-over-UDP*, OASIS Std. [Online]. Available: http://specs.xmlsoap.org/ws/2004/09/soap-over-udp/

[41] *WS-I Basic Profile 1.0*, The Web Services-Interoperability Organization Std. [Online]. Available: http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

[42] *Devices Profile for Web Services Version 1.1*, OASIS Std., 2009. [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf

[43] A. Mensch and S. Rouges, “DPWS Core Version 2.1 - User Guide,” Service-Oriented Architecture for Devices, Tech. Rep., 2009.

[44] H. Bohn, A. Bobek, and F. Golatowski, “SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains,” in *International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006*, 2006, p. 43.

[45] Z. Shelby, K. Hartke, and B. C., *Constrained Application Protocol (CoAP)*, IETF Std., 2014. [Online]. Available: http://tools.ietf.org/html/rfc7252

[46] K. Hartke, *Observing Resources in CoAP*, IETF Std., 2014. [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-observe-14

[47] Z. Shelby, *Constrained RESTful Environments (CoRE) Link Format*, IETF Std., 2012. [Online]. Available: http://tools.ietf.org/html/rfc6690

[48] M. Kovatsch, “Towards an end-to-end web experience in the internet of things,” in *W3C Workshop on the Web of Things*, 2014.

[49] *MQTT Version 3.1.1*, OASIS Std., 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[50] P. Saint-Andre, *Extensible Messaging and Presence Protocol (XMPP): Core*, IETF Std., 2011. [Online]. Available: http://tools.ietf.org/html/rfc6120

[51] Y. Shafranovich, *Common Format and MIME Type for Comma-Separated Values (CSV) Files*, IETF Std., 2005. [Online]. Available: http://tools.ietf.org/html/rfc4180

[52] D. Mundy and D. Chadwick, “An xml alternative for performance and security: Asn.1,” *IT Professional*, vol. 6, no. 1, pp. 30–36, Jan 2004.

[53] G. White, J. Kangasharju, D. Brutzman, and S. Williams, “Efficient XML Interchange Measurements Note,” W3C, Tech. Rep., 2007. [Online]. Available: http://www.w3.org/TR/exi-measurements/

[54] K. Sayood, *Introduction to data compression (3rd Edition)*. Morgan Kaufmann, 2005.

[55] D. Salomon, D. Bryant, and G. Motta, *Handbook of Data Compression*. Springer, 2010. [Online]. Available: http://books.google.se/books?id=LHCY4VbiFqAC

[56] P. Elias, “Universal codeword sets and representations of the integers,” *Information Theory, IEEE Transactions on*, vol. 21, no. 2, pp. 194–203, 1975.

[57] R. Cameron, “Source encoding using syntactic information source models,” *Information Theory, IEEE Transactions on*, vol. 34, no. 4, pp. 843–850, Jul 1988.

[58] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, “Tree automata techniques and applications,” Available on: http://www.grappa.univ-lille3.fr/tata, 2007, release October, 12th 2007.

[59] J. Rissanen, “Generalized kraft inequality and arithmetic coding,” *IBM Journal of research and development*, vol. 20, no. 3, pp. 198–203, 1976.

[60] I. H. Witten, R. M. Neal, and J. G. Cleary, “Arithmetic coding for data compression,” *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[61] D. A. Huffman, “A method for the construction of minimum-redundancy codes,” *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[62] R. G. Gallager, “Variations on a theme by huffman,” *Information Theory, IEEE Transactions on*, vol. 24, no. 6, pp. 668–674, 1978.

[63] D. E. Knuth, “Dynamic huffman coding,” *Journal of algorithms*, vol. 6, no. 2, pp. 163–180, 1985.

[64] J. S. Vitter, “Design and analysis of dynamic huffman codes,” *Journal of the ACM (JACM)*, vol. 34, no. 4, pp. 825–845, 1987.

[65] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.

[66] ——, “Compression of individual sequences via variable-rate coding,” *Information Theory, IEEE Transactions on*, vol. 24, no. 5, pp. 530–536, 1978.

[67] J. Schneider, T. Kamiya, D. Peintner, and R. Kyusakov, *Efficient XML Interchange (EXI) Format 1.0*, W3C Std., February 2014. [Online]. Available: http://www.w3.org/TR/exi/

[68] C. Bournez, “Efficient XML Interchange Evaluation,” W3C, Tech. Rep., April 2009. [Online]. Available: http://www.w3.org/TR/exi-evaluation/

[69] P. Deutsch, *DEFLATE Compressed Data Format Specification version 1.3*, Internet Engineering Task Force Std., May 1996. [Online]. Available: http://www.ietf.org/rfc/rfc1951.txt

[70] D. Peintner and S. Pericas-Geertsen, “Efficient XML Interchange (EXI) Primer,” W3C, Tech. Rep., 2009. [Online]. Available: http://www.w3.org/TR/2009/WD-exi-primer-20091208/

[71] K. Kuladinithi, O. Bergmann, T. Pötsch, M. Becker, and C. Görg, “Implementation of coap and its application in transport logistics,” *Proc. IP+ SN, Chicago, IL, USA*, 2011.

[72] W. Colitti, K. Steenhaut, N. De Caro, B. Buta, and V. Dobrota, “Evaluation of constrained application protocol for wireless sensor networks,” in *Local Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*, oct. 2011, pp. 1 –6.

[73] R. Kyusakov, P. Puñal Pereira, J. Eliasson, and J. Delsing, “EXIP: A Framework for Embedded Web Development,” *ACM Transactions on the Web*, vol. 8, no. 4, 2014.

[74] V. Altmann, J. Skodzik, P. Danielis, F. Golatowski, and D. Timmermann, “Real-time capable hardware-based parser for efficient xml interchange,” in *9th IEEE/IET International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP14)*, 2014.

[75] J. G. Koomey, S. Berard, M. Sanchez, and H. Wong, “Implications of historical trends in the electrical efficiency of computing,” *Annals of the History of Computing, IEEE*, vol. 33, no. 3, pp. 46–54, 2011.

[76] Z. Bandic and R. H. Victora, “Advances in magnetic data storage technologies,” *Proceedings of the IEEE*, vol. 96, no. 11, pp. 1749–1753, 2008.

[77] V. I. K. V. R. Inamura, V. J. Toda, and V. T. Morita, “Ultra high density perpendicular magnetic recording technologies,” *Fujitsu Sci. Tech. J*, vol. 42, no. 1, pp. 122–130, 2006.

[78] E. Grochowski and R. D. Halem, “Technological impact of magnetic hard disk drives on storage systems,” *IBM Systems Journal*, vol. 42, no. 2, pp. 338–346, 2003.

[79] C.-X. Zu and H. Li, “Thermodynamic analysis on energy densities of batteries,” *Energy & Environmental Science*, vol. 4, no. 8, pp. 2614–2624, 2011.

[80] H. Sunami, *Advances in Solid State Circuit Technologies.* InTech, 2010, no. 978-953-307-086-5, ch. Dimension Increase in Metal-Oxide-Semiconductor Memories and Transistors, pp. 307–332. [Online]. Available: http://cdn.intechopen.com/pdfs-wm/9855.pdf

[81] J.-M. Tarascon, “Key challenges in future li-battery research,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1923, pp. 3227–3241, 2010.

[82] “GreenTouch Green Meter Research Study: Reducing the Net Energy Consumption in Communications Networks by up to 90 % by 2020,” GreenTouch, Tech. Rep., 2013. [Online]. Available: http://www.greentouch.org/uploads/documents/GreenTouch_Green_Meter_Research_Study_26_June_2013.pdf

[83] M. N. Halgamuge, M. Zukerman, K. Ramamohanarao, and H. L. Vu, “An estimation of sensor energy consumption,” *Progress In Electromagnetics Research B*, 2009.

Part II

Paper A

Integration of Wireless Sensor and Actuator Nodes with IT Infrastructure Using Service-Oriented Architecture

Authors:
Rumen Kyusakov, Jens Eliasson, Jerker Delsing, Jan van Deventer, and Jonas Gustafsson

Reformatted version of paper originally published in:
IEEE Transactions on Industrial Informatics

Integration of Wireless Sensor and Actuator Nodes with IT Infrastructure Using Service-Oriented Architecture

Rumen Kyusakov, Jens Eliasson, Jerker Delsing, Jan van Deventer, and Jonas Gustafsson

Abstract: A large number of potential applications for Wireless Sensor and Actuator Networks (WSAN) have yet to be embraced by industry despite high interest amongst academic researchers. This is due to various factors such as unpredictable costs related to development, deployment and maintenance of WSAN, especially when integration with existing IT infrastructure and legacy systems is needed. Service-Oriented Architecture (SOA) is seen as a promising technique to bridge the gap between sensor nodes and enterprise applications such as factory monitoring, control and tracking systems where sensor data is used. To date, research efforts have focused on middleware software systems located in gateway devices that implement standard service technology, such as Devices Profile for Web Services (DPWS), for interacting with the sensor network. This paper takes a different approach - deploying interoperable Simple Object Access Protocol (SOAP)-based web services directly on the nodes and not using gateways. This strategy provides for easy integration with legacy IT systems and supports heterogeneity at the lowest level. Two-fold analysis of the related overhead, which is the main challenge of this solution, is performed; Quantification of resource consumption as well as techniques to mitigate it are presented, along with latency measurements showing the impact of different parts of the system on system performance. A proof-of-concept application using Mulle - a resource-constrained sensor platform - is also presented.

1 Introduction

The ability of networked, embedded devices to monitor and control various physical parameters of the environment as well as communicate the data over the Internet makes them a foreseeable source of innovation in many fields: from factory automation to use in smart homes and healthcare. While the benefits of integrating these devices with enterprise systems and services are evident from the perspective of business process synergy, many challenges still prevent widespread integration of sensor nodes into Manufacturing Execution Systems (MES), Enterprise Resource Planning (ERP), accounting and distribution systems. More details regarding the opportunities and challenges of applying WSANs in industrial environment are presented in [1] as well as in the work of Andreas Willig on wireless industrial communications [2].

Dealing with the heterogeneity of devices and software systems requires a flexible solution that can lower complexity and decrease development, deployment and system maintenance costs. Service-Oriented Architecture (SOA) has proven successful in leveraging these costs and it is seen as enabling technology for the development of enterprise systems in industrial domain [3]. Moreover, research analysis has shown its applicability for embedded systems development [4]. The prototype systems implemented within the European project SIRENA [5] as well as some commercial products that provide support for Devices Profile for Web Services standard [6] further prove the applicability of the SOA concept in the embedded domain. However, applying SOA to deeply constrained devices such as sensor nodes is still an open research problem due to unacceptable overhead. Some of the

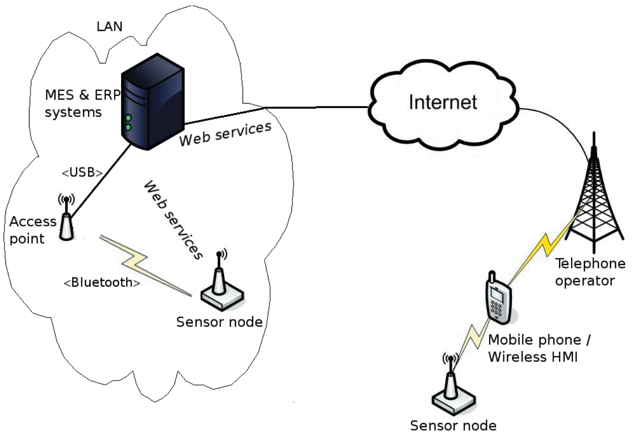


Figure 1: Sensor nodes are integrated with enterprise systems using standard SOAP-based web services

proposed solutions are based on modifications of the SOA protocols that simplify the implementation and lower the resource requirements [7]. However, the majority of research efforts have been directed towards using middleware software running on more capable devices or gateways as suggested by Wolff et al. in [8] or in the work of R. Bosman et al. [9]. This middleware is responsible for exposing the functionality of the whole sensor network as services using standard SOA technology. In this way, communication within the network is still based on specialized, proprietary protocols. This approach has the benefit of leaving the resource-intensive tasks related to standard service implementation to the gateway, but also has some drawbacks such as a single point of failure, an inability to support heterogeneity on the node level [10], etc.

Although the node-level service implementations have already been proposed, there are no studies investigating the applicability of deploying fully interoperable and compliant services, such as those described in WS-I Basic Profile 1.0, directly on the sensor nodes. This is due to the general perception that the use of XML-based services on highly constrained sensor nodes is inapplicable or even impossible, as stated by Leguay et al. [11]. The higher overhead, in terms of power consumption, latency, RAM and CPU usage, related to serialization, transmitting and parsing of verbose XML messages is undisputed, and has in fact been well studied by Groba et al. [12] where empirical data that quantifies the overhead of web services on embedded devices is presented. Especially challenging are the high memory requirements resulting from the need for large buffers used to accommodate the XML documents.

In this paper, we present few techniques for improving efficiency that allow us to deploy standard SOAP web services on resource-constrained sensor nodes. These techniques are implemented in a proof-of-concept application that connects sensor nodes to an enterprise application. The architecture of the experimental setup is shown in Figure 1. Among others, we applied sensor data aggregation for reducing the transmission time and active mode intervals of the nodes, and hence increasing battery life. As this technique is not applicable in general case a real-world scenario which allows for such aggregation is also presented. In [13], Lee et al. used similar approach for industrial monitoring application.

The remainder of this paper is structured as follows - we provide a motivation for our work in Section 2. Section 3 summarizes the related work in the area and presents some of the technologies used. Section 4 goes into details about the problems related to the use of SOA in WSAN. Section 5 presents our sample application together with performance measurements. In Section 6, we give the possible improvements and extensions to our work, and Section 7 concludes the paper.

2 Motivation

As pointed out by Jammes et al. [14], the manufacturing enterprises, pushed by the global competition, are seeking ways to increase their responsiveness to the market demands on a real-time scale. At the same time, the costs for maintaining the process flows evolution or modification are substantial due to the semi-automatic, or even error prone manual, configurations involved. A recent study by Candido et al. proposed an architecture that supports the device and process lifecycle evolution based on SOA and Evolvable Production Systems (EPS) [15]. As part of this architecture, the devices have SOA interfaces that allow high level business applications to interact with them without any intermediary protocol gateways - a concept also suggested in [16]. The support for cross-layer integration between the shop floor and enterprise systems was also a main objective for the SOCRADES project [17]. As an outcome of this project, an architecture for vertical integration based on the SOA approach was proposed, where the ERP and MES systems together with shop floor devices are integrated using web services. Kalogeras et al. presented similar architecture with emphasis on the use of web services, workflows and ontologies [18]. A diagram from the SOCRADES Roadmap shown in Figure 2, represents the concept of applying SOA approach for vertical inter-enterprise integration. As depicted, the resource constrained devices, including wireless sensors and actuators, are exposed to the SOA interface through service gateways or mediators. The work presented in this article is an extension to the aforementioned SOA architecture aiming at deploying the service interface provided by the gateways directly on the wireless nodes. This is made possible due to the advancement in embedded systems hardware but also the application of resource-aware implementation techniques.

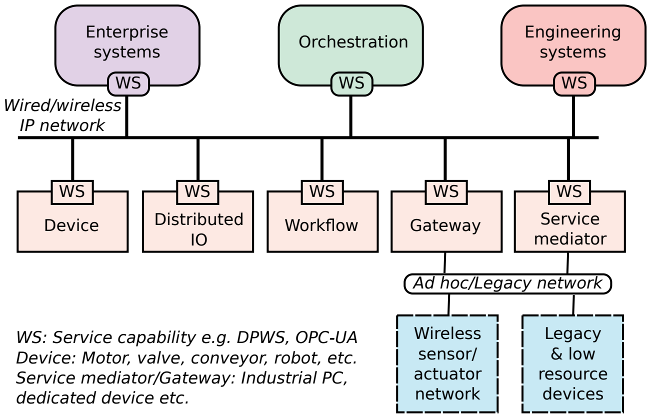

Enterprise systems
WS
Orchestration
WS
Engineering systems
WS
Wired/wireless IP network
WS
Device
WS
Distributed IO
WS
Workflow
WS
Gateway
WS
Service mediator
Ad hoc/Legacy network
WS: Service capability e.g. DPWS, OPC-UA
Device: Motor, valve, conveyor, robot, etc.
Service mediator/Gateway: Industrial PC, dedicated device etc.
Wireless sensor/ actuator network
Legacy & low resource devices

Figure 2: The SOCRADES cross-layer approach

3 Background and related work

Service-Oriented Architecture denotes the usage of well-defined and self-contained function calls between distributed nodes independent of the location and platform of the parties involved. It also implies that interoperable network protocols for communicating service requests and responses are available. Although many challenges still remain, there are different approaches for providing low layer (physical and data-link) integration of wireless networks in an industrial environment [19, 20]. In this work we focus on providing application layer integration with the use of an access point for the data-link layer integration and TCP/IP stack on the network and transport layers.

There are many service technologies that are built upon the SOA approach: CORBA, UPnP, OPC-UA, Jini and different flavors of web service technology (SOAP-based, RESTful [21], etc). The web services conforming to the Web Services Description Language (WSDL) specification are designed to be application- and transport protocol-agnostic, which leads to compatibility issues. For that reason, different web service profiles are specified to leverage the diversity of network protocols used and to adapt the specifications to a particular application domain. Services in enterprise systems mostly conform to WS-I Basic Profile 1.0, while Devices Profile for Web Services is used to define a set of protocols to enable plug-and-play behavior for embedded networked devices. Both profiles rely on SOAP as an application layer protocol for serialization of service requests and responses.

Advantages and disadvantages of the aforementioned service technologies applied to different applications are already being studied by researchers (e.g. [5]); thus, a comparison of them is not included in this paper. The analysis performed in research projects such as SIRENA, SOCRADES and within the research program ITEA gives priority to SOAP-based web services in which the devices are integrated with the IT systems using DPWS. The main argument in support of this architecture is the possibility to apply service orchestration of embedded and system services directly without the need for adapters, as shown in Figure 3. A white paper by Boyd et al. [22] provides further reading on service orchestration and other SOA concepts along with case studies of applying SOA to manufacturing infrastructure.

The use of proprietary or nonstandard SOA implementations requires translation middleware when working with standardized service orchestration, such as Business Process Execution Language (BPEL). As our approach aims at limiting the external dependencies of the SOA implementation for devices the work presented in this paper considers standard SOAP-based web services.

To ensure interoperability, SOAP web services are entirely based on open standards and rely heavily on the usage of XML and XML Schema Definition Language (XSD). Thus, each SOAP message is a XML document that must first be serialized, transmitted, received and then parsed. To avoid these resource-intensive operations being performed on the sensor nodes, researchers are investigating the use of middleware software deployed on gateway devices that first communicate with the nodes in an ad-hoc manner and then translate their functionality as web services to external systems. An example of such a design was proposed by Avilés-López et al. [23]; In their system, the middleware included an advanced registry mechanism. A similar solution that also incorporated a light-weight, ad-hoc service protocol within the sensor network was presented by Leguay et al. [11]. In that work, the translation between internal and external DPWS-compatible services was done on the gateway. The architecture supports one-to-one, but also many-to-one, relations between the services with a highly flexible eventing mechanism built upon hierarchical subscriptions. Another approach more closely related to the work presented in our paper is that by Priyantha et al. at Microsoft Research [24]. Instead of using specialized, ad-hoc services for node-to-node communications, they proposed to use

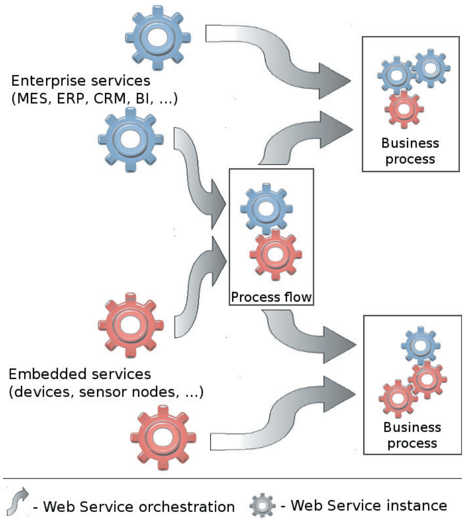


Figure 3: Direct orchestration of sensor node and enterprise web services is made possible due to their compatibility

web services described by WSDL. To keep the overhead low, these services were implemented using HTTP binding and not SOAP. This provides for shorter and easier to parse and serialize messages but also implied constraints on the structure of the data transmitted and impaired the compatibility with enterprise systems. To address these issues, Priyantha et al. proposed an external server called *Controller* that more or less fulfilled the role of the gateway middleware presented in the previous papers. The controller served as a proxy that translated the internal web services to SOAP-based services and provided eventing through the use of WS-Eventing. In this way, the client applications communicated with the sensor nodes indirectly through the controller. Also presented in that paper are different techniques to lower the XML-related impact on performance as well as an analysis of possible application scenarios where this approach will lead to cost savings.

In contrast with the aforementioned approaches, the solution presented in the present paper deploys fully compatible SOAP-based web services directly on a highly constrained sensor platform and hence eliminates the need for additional middleware. In this way, its main contribution is an efficient implementation that combines a lightweight TCP/IP stack - lwIP [25] and a gSOAP [26] web service toolkit. The lwIP provides two different APIs to access the network services: a low-level ”raw” API relies on callbacks, and a higher-level ”sequential” API is easier to work with but also implies higher resource consumption. We used the ”raw” API to minimize the footprint of our solution.

The gSOAP toolkit includes a highly efficient runtime environment to process SOAP messages that uses either a general-purpose XML parser or an application-specific one that can be generated from the service description (WSDL) file. The use of an application-specific parser and serializer provides for lowering the RAM and ROM utilization, as the processing logic for the input and output generators is optimized for the specific usage, and there are no execution paths left unused by the application.

SOAP-based service implementation with a general-purpose XML parser on Tmote Sky was reported by Yazar et al. [27]. Their solution differs from our approach in that it does not provide tool support for developing SOAP-based services but rather is only used to evaluate the performance of their RESTful implementation, as stated in their paper: ”The SOAP-based implementation is used as a reference point in performance evaluation and is not intended for general use.” Also not included in the work of Yazar et al. is a mechanism to dynamically discover the network location of a service, neither by their RESTful nor their SOAP implementation.

A work that is aiming at deploying standard-based embedded web services directly on resource constrained sensor nodes is available from open source project WS4D-uDPWS [28]. Although, the intended outcomes of WS4D-uDPWS and our approach are very similar the implementation techniques differ in many aspects. First, the network layer used in WS4D-uDPWS is uIP TCP/IP stack. It has smaller footprint than lwIP but provides lower throughput. Second, while we built our solution on an existing service implementation (i.e. stripped version of gSOAP), uDPWS provides its own web service runtime which is highly optimized and has smaller RAM and ROM footprint than our runtime. However, WS4D-uDPWS does not provide tool support for generating the application-specific parts of its runtime based on the WSDL service descriptions. Code-generation is provided, but it is based on text files with formatting and naming conventions specific to WS4D-uDPWS. Moreover, while the processing of the SOAP headers is automated, the parsing and serialization of the SOAP body is left to the web service developers.

4 SOA on sensor nodes

Due to the number of nodes in WSN, it is very important to deploy and configure sensor nodes with the least manual work possible. The initialization and configuration parameters can depend on various conditions, most probably originating outside of the sensor network. Looking at factory automation as an example, these conditions are connected to MES systems but also to strategic decisions in ERP systems, historical data from databases, etc. Any changes done in these systems that affect the behavior of the sensor nodes must be propagated down while sensor data, after undergoing filtering and aggregation, must be propagated up. Using SOA all the way down to the smallest devices results in increased compatibility, where auto-configuration and plug-and-play capabilities can be modeled as services. In such way, higher flexibility for tuning or even changing the manufacturing processes is achieved stemming from direct interactions between all system components. However, this flexibility also leads to complex systems integration and difficulties when defining or verifying the required functionality of particular module or the system as a whole. Handling this complexity requires data models that constrain the possible interactions and formats for data exchange. Examples of such data models are OPC-UA information model or Business to Manufacturing Markup Language (B2MML) used to link business systems such as ERP and supply chain management systems with manufacturing systems such as control systems and MES. Similar models used for interfacing sensing devices are described by Sensor Web Enablement Framework of Open Geospatial Consortium.

4.1 Web services

The main drawback to using SOAP-based web services is the need to parse verbose XML documents. However, there are already a number of compression techniques that require a factor of ten less RAM, CPU and bandwidth as compared to text-based XML. The most promising of these is Efficient XML Interchange (EXI) [29], a W3C recommendation as of March 10, 2011. EXI is defined as an alternative mean to represent the XML Information Set [30] that provides one-to-one translation to text-based XML representation. Depending on the document properties and processing options specified, EXI provides between 50% and 99% reduction in size and up to 15 times faster processing [31]. The work presented in this paper shows that even verbose XML can be used as a service message protocol for sensor nodes; future binary XML representations will only extend the applicability of the presented solution. Introducing the EXI encoding to embedded web service implementations however, will require the ability to change the XML parser and serializer with EXI ones. Our first attempt in this direction is the creation of EXIP open source project[1]. Another question arising from the use of binary encoding is how to connect embedded EXI encoded web services with text-based enterprise services. One such techniques that is already available as a commercial product is to introduce a transparent HTTP proxy in between. The role of the proxy is to translate binary EXI encoding to text-based XML and vise versa. More details on the opportunities and challenges of using EXI in industrial environment are presented in [32].

4.2 Tools

The development of web service applications depends upon a runtime system responsible for the network communications, parsing, validation and serialization of service requests and responses. Besides the runtime system, software tools are used to map data structures in the XML to programming language constructs - also known as XML data binding. Based on the characteristics of our target domain, the required properties of the SOA runtime system and supporting tools are as follows:

- written in programming language that is used for sensor and actuator nodes development - currently most widely used are C and its dialect nesC.
- easily portable on different embedded platforms.
- featuring small footprint implementation.
- highly configurable - it should be possible to remove features that are not used or needed.

For C language, there are two web service toolkits, namely, gSOAP and Apache Axis2/C. While gSOAP supports and has been ported on several embedded platforms, Axis2/C is mostly used on Windows and Linux machines. Moreover, gSOAP runtime has a wide range of features that can be selectively included or excluded from it. The version of gSOAP used in our solution has the following components removed: XML DOM parser, HTTPS and SSL support, compression, logging module, all support for attachments including SOAP with Attachments, MIME, DIME or MTOM, HTTP chunked transfer mode.

5 Proof of concept

WS-I Basic Profile 1.0 defines a bare minimum of constraints on the WSDL specification that make different web service implementations compliant. Examples of such constraints are the use of SOAP version 1.1 binding, HTTP 1.0 or HTTP 1.1 as a transport protocol. The applications developed using our solution are compliant with WS-I Basic Profile 1.0 provided that the ”-1” command-line option is used when executing gSOAP *soapcpp2* code generator. The current enterprise systems are mostly conforming to this profile which enables interoperability with our SOA approach for sensor nodes.

[1]Efficient XML Interchange Processor - http://exip.sourceforge.net/

DPWS, in contrast, poses many more requirements aimed at providing plug-and-play capabilities as well as automatic deployment and configuration. It also denotes the usage of SOAP version 1.2 as well as the addressing fields in the SOAP header defined in WS-Addressing specification. Moreover, a set of predefined services must be available on the devices willing to comply with the DPWS standard. As an example, a manifest service called *device* is responsible for hosting and advertising the other services that represent the functionality provided by the device. Another predefined service, with an interface consisting of six operations, is specified in WS-Discovery - a protocol that enables dynamic discovery of available services on the network without the use of centralized registry such as UDDI. All six operations use SOAP-over-UDP to minimize network traffic. Figure 4 shows all of the protocols included in the DPWS specification, with those not covered by our solution illustrated with hatching.

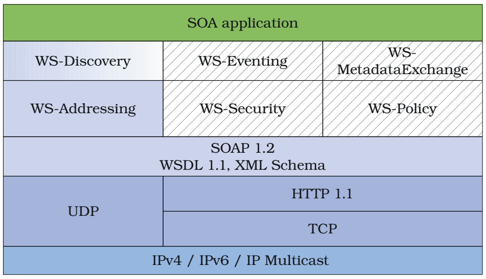


Figure 4: DPWS protocol stack. Parts not covered in this work are illustrated with hatching

When the required settings for using SOAP v1.2 and SOAP-over-UDP are specified as described by the gSOAP documentation, our current solution supports SOAP-over-UDP, SOAP 1.2, WSDL 1.1, WS-Addressing and certain parts of WS-Discovery. Two WS-Discovery operations are included in our implementation: Probe, which is a query multicasted to specific IP multicast address and port, and ProbeMatch, which is the response of the queried nodes to the Probe message. The use of discovery proxies, as defined by the specification is not supported. Nevertheless, this limited implementation is sufficient to locate a service advertised by a WS-Discovery-compliant device.

The security scheme defined by DPWS enables protection of the service executions in three directions: authentication of the parties involved, message integrity protection and confidentiality. While the majority of the target applications will not require confidentiality for sensor data and/or actuator control data, authenticity and integrity are crucial especially for wireless communications. However, the resources available on current sensor platforms are not sufficient for supporting standard based authentication mechanisms based on digital certificates and asymmetric cryptography. For that reason, the presented approach is only appropriate for non-critical applications where the sensor nodes are behind enterprise firewall.

5.1 Architecture

As was already stated, the web service implementation presented in this paper is built upon the gSOAP toolkit. The gSOAP design supports different network layers with BSD-socket API supported out of the box. However, its runtime is written with the perception that the network interface it uses supports sequential execution, which requires the use of threading. Thread-based network APIs provide abstraction of the complex event-driven nature of network communications. The trade-off inherited from this abstraction is a higher resource consumption, which makes it not suitable for highly constrained sensor nodes [25]. So, to use the event-based ”raw” lwIP API, the network layer of gSOAP runtime was rewritten and an additional lwIP wrapper was introduced. This includes splitting of the sequential execution blocks that contain blocking network operations into smaller non-blocking programming sequences connected with callback functions. As an example, consider the following simplified programming fragment that uses threaded network layer:

```
Block_1() {
    blocking_connect();
    /* The TCP connection is established */
    serialize_http_header();
    blocking_send();
    /* The http header is sent */
    serialize_soap();
    blocking_send();
    /* The soap message is sent */
    cleanup();
}
```

The equivalent functionality based on non-blocking lwIP network operations and callbacks is coded as follows:

```
Block_1() {
    store_soap_state();
    lwip_connect(); /* calls Block_2() when connected*/
}
Block_2() {
    serialize_http_header();
    lwip_send(); /* calls Block_3() when the header is sent*/
}
Block_3() {
    serialize_soap();
    lwip_send(); /* calls Block_4() when the soap is sent*/
}
Block_4() {
    cleanup();
}
```

The listings also present the concept of transmission on the fly - when the HTTP header is serialized, it is sent over the network. Then the sending buffer is released and used for storing the SOAP message before its transmission. The same technique is used on the receiving side: when the HTTP header is received it is parsed and then the receiving buffer is released. In this way, the size of the buffers, and hence the RAM usage, can be restricted.

The overall architecture is depicted in Figure 5. The modules responsible for power management, sampling the sensors and aggregating the data are not affected by the service interface; hence, legacy code can be reused. Instead of connecting the input and output of the sensor application to a network API implementing proprietary, specialized protocols, the data are passed to the gSOAP runtime using handlers. The runtime serializes the output data to a SOAP message, and then uses lwIP to send it over the network. The opposite is true for input data: it is first parsed and then forwarded to the sensor

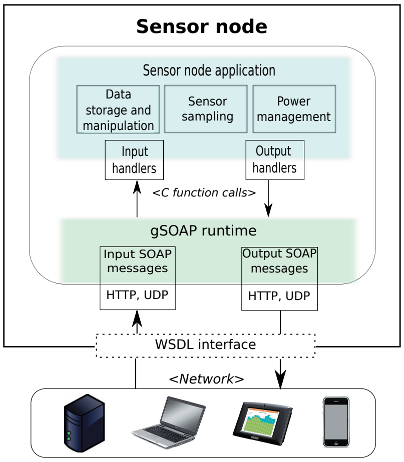


Figure 5: System architecture

application. The interface describing the services provided by and consumed by the nodes is available through the use of standardized Web Service Description Language. This allows for so-called top-down SOA development, where the WSDL interfaces for the nodes are defined first - usually using graphical tools[2]- and then are used to generate the SOAP runtime. At the end the developer connects the provided interface with the sensor application. This is the approach used in the development of our testbed, described in the subsequent subsections.

5.2 Mulle sensor platform

The Mulle sensor platform [33] used in our experimental setup is equipped with a Renesas M16C/62 microcontroller running at 10 MHz with 31kB RAM and 384 kB programming memory. A Mitsumi Bluetooth radio transceiver, operating at 57 kbits/s, was used in our testbed to enable mobility through the use of a mobile phone as an access point. The Mulle sensor platform is also available with an IEEE 802.15.4 radio transceiver, which also can be employed instead of the Bluetooth one, provided that the lwIP stack is configured for using it.

5.3 Proof of concept experiment

To test the applicability and performance of our solution, several services were implemented. The first was a very simple, light service with operations for switching a LED on and off and for checking the status of the LED. Tests were performed under different scenarios with the service being hosted on a sensor node using our solution, on a stationary PC or on both. To check compatibility, two different implementations of the light service were used on the PC. The first was C-based, using a gSOAP port for Linux. The other was Java-based, using JAX-WS API running on a GlassFish server. In both cases the interactions between the sensor node and the PC proceeded without any compatibility problems.

[2]In our use case we used Eclipse WSDL Editor

For the second test, it was decided to replicate a real-world scenario where, despite the overhead, the SOA implementation would still be beneficial to use [34]. In such an application, the system must lack any real-time properties. Also, it should be possible to aggregate the sensor data before its dissemination that should happen at long intervals. The source of inspiration was a district heating project [35] aimed at increasing the efficiency of energy distribution.

District heating scenario

In today's district heating substations, different sensors and actuators are hard-wired together. This limits the possibilities for system optimization as communication barriers limit the information interchange. With wireless sensor platforms integrated in such district heating devices as a circulation-pump, heat meter and temperature sensors, greater opportunities for system optimization are achieved as information can be interchanged without limitations.

With a service-oriented architecture integrated in the end nodes, there is no direct need for a central control unit, as the sensor nodes are powerful enough to control the relatively slow heating process. The slow process makes the use of SOA over WSAN particularly suitable as there is no need for frequent data transmission, which would decrease the expected life-length of the sensor platforms. Thus, the nodes are in sleep mode most of the time with short active intervals for sensor sampling and data aggregation. The transceiver is infrequently turned on only when the highly aggregated data are sent directly to the enterprise systems responsible for heating process management.

```
<hts:GetSummary>
  <Temp>
    <Current>19.3</Current>
    <Average>18.2</Average>
    <Min>17.4</Min>
    <Max>21.0</Max>
  </Temp>
  <Humidity>
    <Average>65</Average>
    <StdDeviation>5.0</StdDeviation>
  </Humidity>
</hts:GetSummary>
```

Figure 6: Segment of the service request initiated by the Mulle sensor node. It contains an aggregation of the sensor data for the period of interest

In our testbed, nodes were equipped with temperature and humidity sensors, and the data sent to the server consisted of multiple metrics, such as current sensor readings as well as the average, minimum, maximum and standard deviation of the temperature and humidity for a given period, as shown in Figure 6. The intervals in which the sensor nodes communicate the data were controlled by the management system. For the implementation of the heating process management system we chose the SOA Swordfish toolkit that supports deployment on a Java EE application server. Also implemented on the server was a Java version of WS-Discovery, which was used to advertise the heating service on the network.

The implementation started with modeling the desired interactions between the sensors and the management system using Web Service Description Language. The abstract WSDL service definitions were then fed into Swordfish framework to generate the serialization and parsing code. The same WSDL interface was used by the gSOAP code generation tools. The code produced was then

SOAP Messages	Parsing time (ms)	Serialization time (ms)
Heating service request 654 bytes	–	14.5
Heating service response 479 bytes	14.5	–
LED check status service request 386 bytes	24	7
LED check status service response 414 bytes	26	16
LED switch service request 415 bytes	25	7.5

Table 1: Time needed by the Mulle sensor platform to process SOAP messages

combined with our modified gSOAP runtime, lwIP and our network layer wrapper, which were deployed on the Mulle sensor platform. To avoid manual configuration of the server address for each sensor node, two operations of the WS-Discovery were also implemented and deployed on the sensor platform to dynamically locate the heating service.

Mobility scenario

The heating management service was also used as a testbed for a mobility scenario where a sensor node is being carried by a person with a Bluetooth-enabled mobile phone. This can be useful for assisting and documenting manual inspections and diagnostics of industrial equipment by technicians for example. The phone provides access to a 3G network that enables connectivity of the sensor node and the Java server on a TCP/IP layer. With this infrastructure setup, the sensor node seamlessly communicates the aggregated sensor data to the enterprise application using web services. However, an important requirement in this scenario is the presence of a secured VPN connection between the mobile phone/wireless HMI and the enterprise network as our solution does not support the security mechanisms defined in the DPWS specification and the connection is established from outside the enterprise firewall.

5.4 Performance measurements

A gSOAP runtime with no network layer or deployed services requires around 5.5 kB of RAM and 123 kB of programmable memory. For each service (client or server) added, an additional 13 kB of ROM is required, on average. During service invocation 3 kB of RAM are allocated and hence need to be available on the system. If only one service is executed at a time, the overall RAM consumption is 8.5 kB independent of the number of services added. However, allowing different service executions to be interleaved requires an additional 3 kB of RAM for each service deployed. The time needed to parse and serialize a particular request or response is highly dependent on its size, structure and the number of namespaces used in the XML document. Table 1 shows the processing time for messages used in the LED and heating service examples.

To evaluate the latency overhead, we used the *GetStatus* operation of the LED web service hosted on a PC running Linux with a Bluetooth v1.2 dongle. A Mulle sensor node, with LED service client implemented using our solution, was also set up within transmission range. All communication were

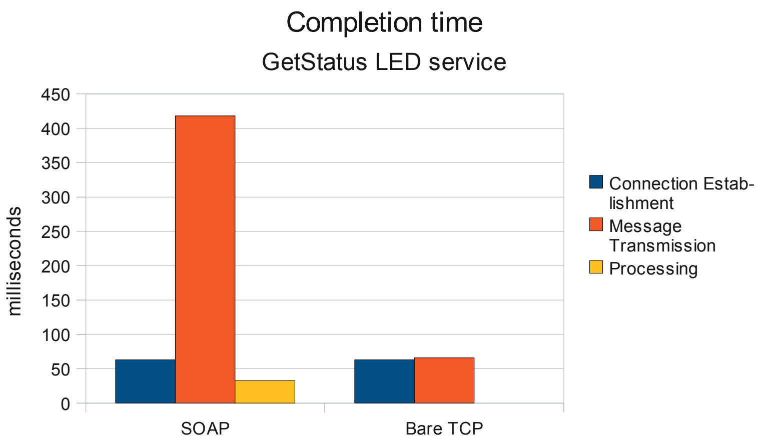


Figure 7: Completion time in milliseconds for service execution

performed using the Bluetooth Personal Area Networking (PAN) profile where the PC was hosting the Network Access Point (NAP) service and the Mulle acting as user (PAN-U). The deployment of the LED service client on the Mulle node allowed it to use sleep mode such that it periodically waked up and sent *GetStatus* SOAP request, then waited for the response, parsed it and went back to sleep mode. Having the node as a LED server would increase the power consumption substantially as it would require the Bluetooth module to be powered on at all times. The current approach allows the Bluetooth module to be duty-cycled.

The same interactions between the PC and the Mulle node were implemented using a bare TCP approach with one-byte payload. In such way the type of operation (*GetStatus* or *Switch*) is encoded using a single bit and another bit is used to indicate the status (*on* or *off*) of the LED. Although it cannot be applied in practice, the ad-hoc one-byte TCP implementation represents the shortest possible encoding of the LED operations over TCP/IP thereby enabling the overhead of our solution to be measured.

Figure 7 shows the completion time for our SOAP-based solution (514 ms) compared to the bare TCP approach (129 ms). The measured time, averaged over 50 transmissions, is given for the three phases of the service execution i.e. TCP connection establishment, SOAP message transmission and XML processing. The results show that the time to parse and serialize SOAP messages by the Mulle sensor node denotes just a small part of the latency related to web service invocation - 33 ms for the *GetStatus* LED service or about 6.5 % of the total service execution time. The larger part is due to the actual transmission. This observation proves that the use of more compact representation of the service messages, even compression and other techniques which affect the processing speed, will improve the real-time properties of the system. Moreover, it takes almost four times longer to complete the SOAP service compared to a one-byte TCP payload ad-hoc representation of the LED service operations.

6 Future work

The work presented in this paper shows that even highly resource-constrained networked sensor nodes can be integrated within an IT infrastructure using standard SOA technology. However, even efficient implementation poses significant performance overhead, which makes the solution only suitable for applications where the sensor data can be heavily aggregated and transmitted over relatively long periods. One such example from energy management domain was presented in the paper, but other applications for home and factory automation networks would also meet this criterion. The level of data aggregation and the length of non-transmission intervals needed depend on many parameters such as power consumption, sleep schedule, real-time requirements, etc. Therefore, a precise analysis showing their exact threshold that would make this solution beneficial is an important topic for future work. This analysis must take into account all parameters, and their interdependence, that play a role in the applicability of the SOAP-based web services. This analysis must also provide a comparison with emerging standards for embedded web services such as those described by Shelby in [36].

Applying the same SOA approach to full-scale sensor networks, where most communications are multihop and the nodes use IEEE 802.15.4 radio, is another area for future exploration. In addition, different ways to lower the related overhead should be investigated. The most important in this respect is the use of binary encoding for the SOAP messages.

7 Conclusion

Integration of high-end systems with deeply embedded wireless sensor nodes is an important area of research that aims to provide new possibilities for control and monitoring applications. The solution presented in this paper enables standard-based and direct application-layer integration between web service-enabled IT systems and resource-constrained sensor nodes. Its main contribution is the efficiency of the provided implementation, which combines light-weight TCP/IP stack implementation and SOAP-based web service implementation. In addition, we included performance measurements on the impact of this method on latency. One important observation was that the overhead related to SOAP message processing is very small compared to message transmission. We also showed an example application that can benefit from the SOA approach, despite the related overhead.

Acknowledgment

The authors would like to thank the European Commission and the partners of IMC-AESOP project for their support. We gratefully acknowledge the valuable comments and suggestions from the anonymous reviewers and associate editor.

References

[1] V. Gungor and G. Hancke, “Industrial wireless sensor networks: Challenges, design principles, and technical approaches,” *Industrial Electronics, IEEE Transactions on*, vol. 56, no. 10, pp. 4258 –4265, oct. 2009.

[2] A. Willig, “Recent and emerging topics in wireless industrial communications: A selection,” *Industrial Informatics, IEEE Transactions on*, vol. 4, no. 2, pp. 102 –124, may 2008.

[3] L. D. Xu, “Enterprise systems: State-of-the-art and future trends,” *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 4, pp. 630 –640, nov. 2011.

[4] S. de Deugd, R. Carroll, K. E. Kelly, B. Millett, and J. Ricker, “SODA: Service Oriented Device Architecture,” *Pervasive Computing, IEEE*, vol. 5, no. 3, pp. 94 –96, july-sept. 2006.

[5] H. Bohn, A. Bobek, and F. Golatowski, “SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains,” in *International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006*, 2006, p. 43.

[6] *Devices Profile for Web Services Version 1.1*, OASIS Std., 2009. [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf

[7] I. K. Samaras, J. V. Gialelis, and G. D. Hassapis, “Integrating wireless sensor networks into enterprise information systems by using web services,” in *SENSORCOMM*, 2009, pp. 580 – 587.

[8] A. Wolff, S. Michaelis, J. Schmutzler, and C. Wietfeld, “Network-centric middleware for service oriented architectures across heterogeneous embedded systems,” in *EDOC Conference Workshop, 2007. EDOC ’07. Eleventh International IEEE*, 15-16 2007, pp. 105 –108.

[9] R. Bosman, J. Lukkien, and R. Verhoeven, “Gateway architectures for service oriented application-level gateways,” *Consumer Electronics, IEEE Transactions on*, vol. 57, no. 2, pp. 453 –461, may 2011.

[10] G. Moritz, E. Zeeb, F. Golatowski, D. Timmermann, and R. Stoll, “Web services to improve interoperability of home healthcare devices,” in *Pervasive Computing Technologies for Healthcare, 2009. PervasiveHealth 2009. 3rd International Conference on*, 2009, pp. 1 – 4.

[11] J. Leguay, M. Lopez-Ramos, K. Jean-Marie, and V. Conan, “An efficient service oriented architecture for heterogeneous and dynamic wireless sensor networks,” in *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*, 2008, pp. 740 –747.

[12] C. Groba and S. Clarke, “Web services on embedded systems - a performance study,” in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, march 2010, pp. 726 –731.

[13] A. Lee and J. Lastra, “Data aggregation at field device level for industrial ambient monitoring using web services,” in *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, july 2011, pp. 491 –496.

[14] F. Jammes and H. Smit, “Service-oriented paradigms in industrial automation,” *Industrial Informatics, IEEE Transactions on*, vol. 1, no. 1, pp. 62 – 70, 2005.

[15] G. Candido, A. Colombo, J. Barata, and F. Jammes, “Service-oriented infrastructure to support the deployment of evolvable production systems,” *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 4, pp. 759 –767, nov. 2011.

[16] A. Ramos, I. Delamer, and J. Lastra, “Embedded service oriented monitoring, diagnostics and control: Towards the asset-aware and self-recovery factory,” in *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, july 2011, pp. 497 –502.

[17] A. Cannata, M. Gerosa, and M. Taisch, “SOCRADES: A framework for developing intelligent systems in manufacturing,” in *Industrial Engineering and Engineering Management, 2008. IEEM 2008. IEEE International Conference on*, 8-11 2008, pp. 1904 –1908.

[18] A. Kalogeras, J. Gialelis, C. Alexakos, M. Georgoudakis, and S. Koubias, “Vertical integration of enterprise industrial systems utilizing web services,” *Industrial Informatics, IEEE Transactions on*, vol. 2, no. 2, pp. 120 – 128, may 2006.

[19] T. Sauter, “The three generations of field-level networks - evolution and compatibility issues,” *Industrial Electronics, IEEE Transactions on*, vol. 57, no. 11, pp. 3585 –3595, nov. 2010.

[20] G. Cena, A. Valenzano, and S. Vitturi, “Hybrid wired/wireless networks for real-time communications,” *Industrial Electronics Magazine, IEEE*, vol. 2, no. 1, pp. 8 –20, 2008.

[21] R. T. Fielding and R. N. Taylor, “Principled design of the modern web architecture,” *ACM Transactions on Internet Technology*, vol. 2, pp. 115–150, 2002.

[22] A. Boyd, D. Noller, P. Peters, D. Salkeld, T. Thomasma, C. Gifford, S. Pike, and A. Smith, “SOA in Manufacturing - Guidebook,” IBM Corporation, MESA International and Capgemini, Tech. Rep., 2008. [Online]. Available: ftp://public.dhe.ibm.com/software/plm/pdif/MESA_SOAinManufacturingGuidebook.pdf

[23] E. Avilés-López and J. A. García-Macías, “TinySOA: a service-oriented architecture for wireless sensor networks,” *Service Oriented Computing and Applications*, vol. SOCA (2009) 3:99-108, pp. 99–108, 2009.

[24] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, “Tiny web services: design and implementation of interoperable and evolvable sensor networks,” in *SenSys ’08: Proceedings of the 6th ACM conference on Embedded network sensor systems*. New York, NY, USA: ACM, 2008, pp. 253–266.

[25] A. Dunkels, “Full TCP/IP for 8-bit architectures,” in *MobiSys ’03: Proceedings of the 1st international conference on Mobile systems, applications and services*. New York, NY, USA: ACM, 2003, pp. 85–98.

[26] R. A. van Engelen and K. A. Gallivany, “The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks,” in *2nd IEEE International Symposium on Cluster Computing and the Grid*, 2002, p. 128.

[27] D. Yazar and A. Dunkels, “Efficient Application Integration in IP-based Sensor Networks,” in *Proceedings of ACM BuildSys 2009, the First ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings*, Berkeley, CA, USA, Nov. 2009, pp. 43–48.

[28] C. Lerche, N. Laum, G. Moritz, E. Zeeb, F. Golatowski, and D. Timmermann, “Implementing powerful web services for highly resource-constrained devices,” in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*, march 2011, pp. 332 –335.

[29] J. Schneider, T. Kamiya, D. Peintner, and R. Kyusakov, *Efficient XML Interchange (EXI) Format 1.0*, W3C Std., February 2014. [Online]. Available: http://www.w3.org/TR/exi/

[30] J. Cowan and R. Tobin. (2004) XML Information Set (Second Edition). W3C. [Online]. Available: http://www.w3.org/TR/xml-infoset/

[31] G. White, J. Kangasharju, D. Brutzman, and S. Williams, “Efficient XML Interchange Measurements Note,” W3C, Tech. Rep., 2007. [Online]. Available: http://www.w3.org/TR/exi-measurements/

[32] R. Kyusakov, H. Mäkitaavola, J. Delsing, and J. Eliasson, “Efficient XML Interchange in factory automation systems,” in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, nov. 2011, pp. 4478 –4483.

[33] J. Johansson, M. Völker, J. Eliasson, Å. Östmark, P. Lindgren, and J. Delsing, “Mulle: A minimal sensor networking device - implementation and manufacturing challenges,” in *Proceedings IMAPS Nordic*, 2004, pp. 265–271.

[34] J. Delsing, J. Gustafsson, and J. van Deventer, “A service oriented architecture to enable a holistic system approach to large system maintenance information,” in *Proceedings CM-MPFT*, 2010.

[35] J. van Deventer, J. Gustafsson, J. Eliasson, J. Delsing, and H. Mäkitaavola, “Independence and interdependence of systems in district heating,” in *Systems Conference, 2010 4th Annual IEEE*, 5-8 2010, pp. 267 –271.

[36] Z. Shelby, “Embedded web services,” *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 52–57, 2010.

PAPER B

Efficient Structured Data Processing for Web Service Enabled Shop Floor Devices

Authors:
Rumen Kyusakov, Jens Eliasson, and Jerker Delsing

Reformatted version of paper originally published in:
IEEE International Symposium on Industrial Electronics (ISIE)

Efficient Structured Data Processing for Web Service Enabled Shop Floor Devices

Rumen Kyusakov, Jens Eliasson, and Jerker Delsing

Abstract: State of the art factory automation systems are now using Service Oriented Architecture (SOA) in order to increase flexibility and lower complexity of process monitoring and control. However, the service technology has not yet penetrated into the lower levels of plant-wide automation processes i.e. throughout shop floor devices such as programmable controllers, embedded sensors and actuators. Different techniques to adapt the web service technology to the specific requirements of embedded systems domain are intensively investigated by researchers and yet real-time properties, limited resources and wireless links are still posing immense challenges. The most promising solutions proposed are based on a newly emerging structured data format - Efficient XML Interchange (EXI). It is designed to compensate for the inefficiency of widely used throughout service implementations XML format. This paper investigates the design of EXI processor targeted at highly resource constrained embedded devices found at the shop floor level. The EXI processor is proposed as an alternative to the XML parsers and serializers currently used in web service implementations. Among the results presented are a novel low level processor interface and measurements quantifying the gained efficiency compared to traditional XML-like interfaces. Reference open source implementation equipped with the new interface is also provided.

1 Introduction and background

Today's global competitive environment provides new opportunities for manufacturing enterprises capable of keeping pace with the volatile markets and rapidly changing business demands. According to IBM 2008 and 2010 Global CEO studies [1, 2], the biggest challenges faced by manufacturing enterprises are the constant demands to change their processes and products and still be able to manage the inherent complexity in all levels of their production environment. In order to provide the IT support needed to cope with this challenges, a new way of designing automation software systems is required.

Service Oriented Architecture is seen as a promising approach to handle the complex interactions in highly heterogeneous and interdependent environment found in the process industry [3]. As a consequence, factory automation providers are now adopting SOA approach in their integrated solutions [4]. However, the currently used SOA implementations are based on web service technology which in turn relies on XML for communicating the service request and response messages. Using structured data formats like XML has the benefit of providing great flexibility to application developers and enhance interoperability but also brings high overhead in terms of memory, CPU, latency and power [5]. This makes the application of SOA on resource constrained devices problematic.

In order to take advantage of the SOA approach but avoid the unacceptable performance impact on the embedded devices, middleware software systems acting as a mediator between SOA capable enterprise systems and low-level networked embedded devices are employed. These systems integrate the shop floor devices such as supervisory control and data acquisition/distributed control systems

(SCADA/DCS) with Manufacturing Execution Systems (MES), Enterprise Resource Planning (ERP), Enterprise Asset Management (EAM) systems etc. In such scenario, the expected gain from SOA deployment is limited to information access, system decomposition and application integration on enterprise system level but not on device level [6].

Different techniques to bring the XML-based service technology to highly resource constrained devices has been proposed. These include usage of compact tags and avoid SOAP binding for service messages [7] or the use of application specific XML parsers and serializers, transmission on the fly and data aggregation. Nevertheless, all these techniques come short to meet the real-time requirements and resource constraints for industrial machinery applications. Complementary to the aforementioned techniques, higher efficiency of SOA implementations can be achieved by using a binary structured data format instead of text-based XML. In order to preserve interoperability and flexibility and meet the resource constraints, it is required that the format is completely compatible with XML, provides higher processing efficiency and compactness and has smaller footprint implementation. The Efficient XML Interchange format [8] was developed by W3C Working Group to fulfill the above mentioned requirements and bring the benefits of XML to the resource constrained devices. On the other hand, the current EXI implementations are only supporting XML-like interfaces. This is necessary so that the legacy systems working with text-based XML will be able to use EXI without any modifications. However, we claim that for shop floor devices, where there are no legacy XML applications deployed, this is unnecessary and only decrease the EXI efficiency. The rest of this paper is devoted to analyzing the requirements for efficient EXI processor Application Programming Interface (API) for deeply embedded devices and propose an alternative to the currently used interfaces. The gained efficiency is also presented with concrete measurements.

The remainder of this paper is structured as follows - Section 2 presents the related work in the area. Section 3 discuss different implementation scenarios for EXI web services for shop floor devices. Section 4 analyzes the EXI API requirements from embedded systems perspective and describes our low level interface. Section 5 summarizes the gained efficiency and shows performance measurements for our proposed solution. In Section 6, we give the possible improvements and extensions to our work, and Section 7 concludes the paper.

2 Related work

Unifying the data exchange between different applications and systems under an open and standardized format could lead to huge savings on software development and maintenance especially for heterogeneous distributed systems. So far, XML has proven successful in wide range of applications due to its flexibility and is also established as a structured data carrier for most of the SOA implementations. This is the reason for having so many attempts to provide binary representation capable of bringing XML to embedded systems domain. The comparison between different binary structured data formats has been studied by both - researchers [9] and W3C binary XML working groups [10, 11]. The results from these analyses are giving preference to EXI over other formats as it provides higher compression ratio and faster processing that supports schema and schema-less encoding and decoding. The measurements are done on variety of test data sets covering different applications. A study which is specifically devoted to the use of binary XML for embedded web services is presented by Moritz et al. [12]. It shows the compactness of Devices Profile for Web Services (DPWS) [13] messages when different XML encodings are applied. Once again the most compact representation is achieved using EXI.

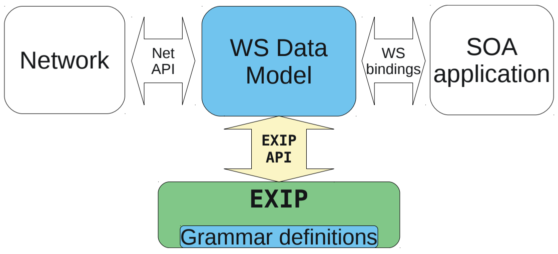


Figure 1: Using EXI as a message carrier for web service implementations. The *WS Data Model* and *Grammar definitions* components can be automatically generated from the WSDL service definitions.

The aforementioned studies are concentrated on proving the applicability of Efficient XML Interchange to embedded systems domain and showing its superiority over other binary structured data formats. However, a detailed investigation on the optimal EXI processor interface has not been carried out yet. Similar studies for text-based XML, however, exist [14] and show the effects of different API on application programming and system maintainability. The analysis presented in this paper shows that for EXI, the design of the Application Programming Interface can have a significance impact on performance as well.

Figure 1 depicts the software components included in an EXI enabled web service implementation. As depicted, the network component provides an EXI encoded input/output using network API - which in many cases is TCP/IP Berkeley Software Distribution (BSD) socket API. Then the web service data model interacts with the EXI processor through its interface for encoding an EXI stream from data objects or decoding an EXI stream to data objects. In such way, the SOA application works on the data objects using the operations defined with Web Service Description Language (WSDL).

3 Efficient XML Interchange for web services

The bijection relationship between XML and EXI representations of the XML Information Set [15] provides for seamless integration of EXI with web service technology. In scenarios such as factory automation systems, where existing XML-based web services deployed on enterprise level systems must co-operate with EXI web services running on shop floor devices, the interoperability is from great importance. When implementing such a system, the following requirements must be taken into account:

1. The introduction of EXI services must be as transparent as possible. This means that modifications on existing systems should be avoided.
2. The complexity of the systems must not increase.
3. The overhead of deploying EXI services on networked embedded devices must be kept low. This implies that performance characteristics of the devices are preserved without upgrading their hardware.

Three implementation strategies can be identified. The first one is to use a gateway between embedded EXI services and enterprise services running on MES, ERP and EAM systems. The messages coming from one side of the gateway will be re-written and send to the other side in an appropriate representation - either binary or text-based. In this way, the modifications on the enterprise level are avoided. This approach however, introduces additional software and possibly hardware component to the system that will need additional maintenance efforts especially when new versions of the web service protocols are deployed. Moreover, it accounts for increased latency and must be redundant in order to avoid single point of failure.

The second strategy is to deploy XML parser and serializer on the shop floor devices in addition to the EXI ones. In this way, the device-to-device services will be executed over EXI and interaction between enterprise systems and embedded devices will be carried out using XML. This approach again does not require any modifications on the existing systems, but put additional overhead to resource constrained devices.

The third strategy is to plug additional EXI processor in the enterprise level service implementations. This modification can easily be implemented if the web service stack supports adding new transport protocols as plugins. Example of such stack is JAX-WS [16]. The main benefits of this approach are the optimal performance and low resource requirements for the embedded devices. The use of XML compliant EXI interface for the enterprise services is required in order to keep the modifications to a minimum. However, for embedded services the use of the same XML-like interface is unnecessary and only leads to degrading the device performance.

The next section discuss the requirements for efficient EXI interface for shop floor devices and gives overview of our approach.

4 XML infoset processor interfaces

There are two programming models for working with XML Information Sets: streaming and in-memory model. The later one stores the entire document tree and the complete infoset state in memory. This provides for easy to use and intuitive interface, but is not appropriate when working with large documents or memory constrained devices. Example of such interface is DOM (Document Object Model) [17]. On the other hand, in the streaming model, the XML infoset is processed in one direction from the start to the end of the document without preserving any state in memory. Instead, the streaming processor generates events when passing over information set objects in the document. This provides for small memory and processing footprint. As an example, Simple API for XML (SAX) [18] and Streaming API for XML (StAX) [19] are both streaming interfaces. The main difference between the two is the way the streaming events are delivered to the application. In the case of SAX, they are pushed to registered callback event-handling functions hence ”push parsing”. StAX however, uses the opposite approach - the application controls when the events will be fired i.e ”pulls” them from the parser.

As of the time of writing this paper there are only two EXI parser implementations. One is provided with commercial license - *Efficient XML* [20] and the other is open source - *EXIficient* [21]. As our attempt to receive evaluation copy of *Efficient XML* was not successful, our implementation is only evaluated against *EXIficient* processor which uses SAX and DOM interfaces.

4.1 Use cases

The requirements for our EXI API are derived from the target domain for our EXI implementation - resource constrained embedded devices. In such scenario the following features are of great importance:

1. Ability to turn on and off features depending on the needs of particular application i.e. application specific EXI processor
2. Support for web service code generation based on WSDL definitions as described by Kätibisch et al. [22]
3. Efficiency

Moreover, the observations of real-world data exchanged between shop floor devices, lead us to the following conclusions:

1. The amount of text data is very low as opposed to business domain transactions. The most prevalent is numeric data - integers and floating point numbers. However, raw binary data is also used in some occasions. As an example, besides the low level sensor data, the process monitoring can include CCD cameras for image capturing and subsequent processing for detecting defects in products or malfunctioning equipment.
2. Frequent exchange of small messages with strict timing constraints

4.2 Design for efficiency

The use of SAX or DOM for EXI processing has several advantages. First, any application working with XML data can start using binary EXI streams with little or no modifications. This means that the XML web service stacks in use today can be reused rather than replaced by new ones. Second, since both APIs are widely used, no new knowledge is required from programmers. Third, DOM is W3C standard and SAX is industry standard which leads to compatible and easier to maintain source code. Nevertheless, both SAX and DOM are designed to work on text-only data. When using them on top of an EXI processor, two extra transformation are needed - from native types to string data and then back from string data to native types. As an example, an EXI encoded unsigned integer [23] must be extracted as native type (32 bits or higher), casted to character string so that it conforms to the XML SAX or DOM API and then when the application starts working with the data it again must be casted from character string back to native unsigned integer representation. This is also the case for other data types. Another example is the raw binary data that require additional Base64 or HEX encoding and decoding. These transformations are unlikely to affect enterprise applications running on powerful servers with low real-time requirements. So in this scenario the benefits of using standard XML APIs outweigh the small performance cost. For programmable controllers, embedded sensors and actuators, however, this performance cost is too high, which requires new interface design meeting the requirements highlighted in our use cases - Subsection 4.1.

As our target platforms are memory and CPU constrained, the streaming programming model was selected for working with EXI encoded XML infoset objects. Both, pushing and pulling event processing styles have been investigated for applying to our interface. We have not found any differences between the two from performance perspective. However, they differ from usability point of view. Based on our personal impression from the experiments we did, the pushing streaming style was applied to EXI parsing while more control over streaming events is given for EXI serialization (pull streaming style).

The listings below show excerpts from our low level EXI interface. Full access to the source code is available from Efficient XML Interchange Processor (EXIP) project web page [24].

EXIP Serializer API Encoding EXI streams

```
struct EXISerializer
{
  // For handling the meta-data (document structure)
  errorCode (*startDocumentSer)(EXIStream* strm);
  errorCode (*endDocumentSer)(EXIStream* strm);
  errorCode (*startElementSer)(EXIStream* strm,QName qname);
  errorCode (*endElementSer)(EXIStream* strm);
  errorCode (*attributeSer)(EXIStream* strm, QName qname);

  // For handling the data
  errorCode (*intDataSer)(EXIStream* strm, int32_t int_val);
  errorCode (*booleanDataSer)(EXIStream* strm,
       unsigned char bool_val);
  errorCode (*stringDataSer)(EXIStream* strm,
      const StringType str_val);
  errorCode (*floatDataSer)(EXIStream* strm,
      double float_val);
  errorCode (*binaryDataSer)(EXIStream* strm,
      const char* binary_val, size_t nbytes);
  errorCode (*dateTimeDataSer)(EXIStream* strm,
      struct tm dt_val, uint16_t presenceMask);
  errorCode (*decimalDataSer)(EXIStream* strm,
      decimal dec_val);

  // EXI specific
  errorCode (*exiHeaderSer)(EXIStream* strm,
      EXIheader* header);

  // EXIP specific
  errorCode (*initStream)(EXIStream* strm, char* buf,
      size_t bufSize, IOStream* ioStrm,
      struct EXIOptions* opts, ExipSchema* schema);
  errorCode (*closeEXIStream)(EXIStream* strm);
};
```

These functions are used to write to an EXI stream using native data types directly. The stream is constructed sequentially with immediate error reporting. As the pulling style is used, multiple streams can be created simultaneously in the same programming thread.

EXIP Parser API Decoding EXI streams

```
struct ContentHandler
{
  // For handling the meta-data (document structure)
  char (*startDocument)(void* app_data);
  char (*endDocument)(void* app_data);
  char (*startElement)(QName qname, void* app_data);
  char (*endElement)(void* app_data);
  char (*attribute)(QName qname, void* app_data);
  // For handling the data
  char (*intData)(int32_t int_val, void* app_data);
  char (*booleanData)(unsigned char bool_val,
       void* app_data);
  char (*stringData)(const StringType str_val,
       void* app_data);
  char (*floatData)(double float_val, void* app_data);
  char (*binaryData)(const char* binary_val,
       size_t nbytes, void* app_data);
  char (*dateTimeData)(struct tm dt_val,
       uint16_t presenceMask, void* app_data);
  char (*decimalData)(decimal dec_val, void* app_data);

  // For error handling
  char (*warning)(const char code,
       const char* msg, void* app_data);
  char (*error)(const char code,
       const char* msg, void* app_data);
  char (*fatalError)(const char code,
       const char* msg, void* app_data);

  // EXI specific
  char (*exiHeader)(const EXIheader* header,
        void* app_data);
};
```

When decoding/parsing an EXI stream the events are pushed to the application. This requires the event-handler functions with signatures shown in the listing above be registered with the EXI processor. Error reporting is also delivered as an event. This API is very similar to SAX with the main difference being the use of native data types instead of character strings.

5 Performance measurements

EXIP interface evaluation

In this section we quantify the overhead of using text-based API, such as SAX or DOM, for an EXI processor as compared to our low-level interface on a resource constrained device. That is, we measure how much time it takes for the conversion of native data types to string data and then back to native data. The measurements are highly dependent on the hardware platform as well as the converting algorithms employed, the parameters passed to the compiler and the compiler itself. The testing platform for the experiments was Mulle sensor node [25]. It is equipped with a Renesas M16C/65 microcontroller running at 10 MHz with 47 kB RAM and 512 kB ROM. The Mulle sensor platform has Bluetooth or IEEE 802.15.4 radio transceiver. Numeric data conversions were implemented by using the standard functions provided with GNU C Library (glibc) - *sprintf()*, *atoi()*, *atol()* and *atof()*. As the only exception from that rule is when converting floating point numbers to character strings. In this case, the *sprintf()* function performance was very slow so we used our own implementation that executes more than five times faster. Binary data were character represented with Base64 encoding

Table 1: Conversion time for different data types on Mulle sensor platform

Native data type	Conversion time [ms]
Boolean	0.028
Signed 16 bit int	1.3
Unsigned 32 bit int	2.2
Floating point	11
Binary (1000 bytes of random data)	58

which creates 33% larger data block as compared to 50% larger when HEX encoding is applied. For the Base64 encoding/decoding routines, Bob Trower's open source implementation was employed [26]. The source code was built with GCC 4.4.1 cross-platform compiler for Renesas M16C microcontroller with no optimization parameters.

The results in Table 1 shows that for a Mulle sensor node to process an EXI encoded service request containing two signed 16 bits integers, a boolean and a floating point number it will take 13.628 ms longer if XML-like interface was used for the EXI parser running on Mulle. As another example, if a CCD camera is attached to the sensor node and a 10 kB JPEG image is sent from the sensor node to a processing unit using EXI web services, it will take 580 ms longer and will require additional memory when text-based EXI API is used. This is because the raw image received from the camera first needs to be Base64 encoded to fit into the SAX or DOM API and then in order for the EXI processor to encode the data in an EXI stream the Base64 encoded image must be decoded back to binary form.

EXIP implementation

Our EXI implementation is written in C and currently it only supports the default EXI encoding and decoding options. Hence, it cannot use schema to optimize the message size and the processing speed. Nevertheless, we performed some preliminary measurements to compare its performance to application specific XML parser available in gSOAP [27] that uses schema to predict the structure of the XML documents. We used a gSOAP port for our Mulle sensor platform and few simple web service messages. These XML messages were then converted to EXI representation using the default encoding options. Although, the size of the EXI messages is about 2 times smaller, the EXIP parser processing takes longer than the gSOAP parsing. This comes to show the importance of schema for optimizing performance, but also a number of areas for improvement in our processing algorithms employed in the implementation. Nevertheless, if we take into account the latency of a IEEE 802.15.4 radio-link induced by the larger messages our suboptimal EXI implementation is still providing better overall performance than text based XML and schema-enabled gSOAP parser.

As an example, for a 6LoWPAN (IEEE 802.15.4 radio) link, if we take an average TCP throughput of 15.2 kbits/s [28] and message size between 418 bytes and 2089 bytes (test set of DPWS messages [12]), the latency for a single hop transmission accounts for between 220 ms and 1099 ms respectively. While, when the same messages are binary encoded using default EXI options their size is between 234 bytes and 1118 bytes which leads to transmission time of 123 ms and 588 ms over 6LoWPAN TCP connection. When we took the smallest DPWS message the gain from EXI encoding is 97 ms which is much higher than the difference in processing between our EXIP parser and gSOAP which is around 40 ms for sample 415 bytes message. If we assume the presence of channel interference

Table 2: Processing time for EXI encoded SOAP messages

Message	Size [bytes]	Parsing time [ms]	Serialization time [ms]
SOAP request (EXI schema-less)	451	182	252
SOAP response (EXI schema-less)	651	280	425

[29], then the TCP throughput is even lower and hence the benefits of our EXIP solution higher.

A more accurate statement of the current EXIP stack processing speed is shown in Table 2. The measurements are acquired by setting up a test environment using two Mulle nodes with IEEE 802.15.4 radios, that run on TinyOS and use Berkeley Low-power IP stack (BLIP) as a 6LoWPAN network layer implementation. During the experiment, the first node requested a temperature readings from the second node. The request and response messages were formated according to the SOAP-over-UDP [30] standard and encoded and decoded in a binary EXI form in schema-less mode rather that plain XML.

6 Future work

Having efficient EXI processor API is not enough to guarantee the performance of web service executions required by programmable controllers, embedded sensors and actuators. Different parameters of the Efficient XML Interchange format can be used to further increase the compactness of the messages or even the processing efficiency. The most important are the use of schema-enabled processing and compression. Currently our EXI implementation only supports the default parameters, but providing support for all options as defined in the specification is in progress. Important area for improvement is the optimization of processing algorithms in our implementation.

A feature that can be exploited to increase efficiency is the use of Datatype Representation Maps. The application of this technique is described by Peintner et al. [31] where it is shown that in some scenarios it significantly increases the compactness of the messages.

7 Conclusion

The introduction of Service Oriented Architecture in factory automation systems brings new opportunities but also new challenges. The web service standards adopted in business applications cannot meet the real-time requirements and resource constraints of the shop floor devices. The Efficient XML Interchange structured data format is seen as promising alternative to the text-based XML that could bring the efficiency needed by embedded systems. However, the early EXI implementations are not designed for deployment on highly resource constrained devices. In this paper we presented our EXI processor targeted at programmable controllers, embedded sensors and actuators. The measurements presented shows the performance gain achieved by using our low level processor interface on a small sensor node as compared to standard SAX or DOM solutions.

Acknowledgment

The authors would like to thank the European Commission and the partners of the EU FP7 project IMC-AESOP (www.imc-aesop.eu) for their support.

References

[1] (2008) The enterprise of the future. IBM Global Business Service. [Online]. Available: http://www.ibm.com/ibm/ideasfromibm/us/ceo/20080505/

[2] (2010) Capitalizing on complexity. IBM Global Business Service. [Online]. Available: http://www-935.ibm.com/services/us/ceo/ceostudy2010/index.html

[3] R. Credle, V. Akibola, V. Karna, D. Pannerselvam, R. Pillai, and S. Prasad, *Discovering the Business Value Patterns of Chemical and Petroleum Integrated Information Framework*, I. Redbooks®, Ed. IBM® Redbooks® publication, 2009, no. 0738433136. [Online]. Available: http://www.redbooks.ibm.com/abstracts/sg247735.html?Open

[4] S. Castro, "SAP Manufacturing Service Oriented Architecture (SOA) Current and Future Strategy," SAP Labs, LLC, Tech. Rep., 2009. [Online]. Available: http://www.sdn.sap.com/irj/scn/index?rid=/library/uuid/b0a3b05a-105f-2c10-8abe-a79220d8eb40

[5] C. Groba and S. Clarke, "Web services on embedded systems - a performance study," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, march 2010, pp. 726 –731.

[6] A. Duca, N. Freeman, and S. Drews, "SOA What? Demystifying SOA for the Process Industry," Honeywell International Inc, Tech. Rep., 2008. [Online]. Available: www.instrumentation.co.za/papers/C9206.pdf

[7] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services: design and implementation of interoperable and evolvable sensor networks," in *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*. New York, NY, USA: ACM, 2008, pp. 253–266.

[8] J. Schneider, T. Kamiya, D. Peintner, and R. Kyusakov, *Efficient XML Interchange (EXI) Format 1.0*, W3C Std., February 2014. [Online]. Available: http://www.w3.org/TR/exi/

[9] S. Sakr, "XML compression techniques: A survey and comparison," *J. Comput. Syst. Sci.*, vol. 75, pp. 303–322, August 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1530897.1531090

[10] (2005) XML Binary Characterization Working Group. W3C. [Online]. Available: http://www.w3.org/XML/Binary/

[11] G. White, J. Kangasharju, D. Brutzman, and S. Williams, "Efficient XML Interchange Measurements Note," W3C, Tech. Rep., 2007. [Online]. Available: http://www.w3.org/TR/exi-measurements/

[12] G. Moritz, D. Timmermann, R. Stoll, and F. Golatowski, "Encoding and Compression for the Devices Profile for Web Services," in *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, 2010, pp. 514 –519.

[13] *Devices Profile for Web Services Version 1.1*, OASIS Std., 2009. [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf

[14] F. Simeoni, D. Lievens, R. Conn, and P. Mangh, "Language bindings to XML," *Internet Computing, IEEE*, vol. 7, no. 1, pp. 19 – 27, 2003.

[15] J. Cowan and R. Tobin. (2004) XML Information Set (Second Edition). W3C. [Online]. Available: http://www.w3.org/TR/xml-infoset/

[16] J. Kotamraju, *The Java API for XML-Based Web Services (JAX-WS) 2.2*, Sun Microsystems, Inc. Std. [Online]. Available: http://jcp.org/en/jsr/detail?id=224

[17] (2010) Document Object Model (DOM). W3C. [Online]. Available: http://www.w3.org/DOM/

[18] (2010) Simple API for XML (SAX). [Online]. Available: http://www.saxproject.org/

[19] (2010) Streaming API for XML (StAX). Java Community Process. [Online]. Available: http://jcp.org/aboutJava/communityprocess/final/jsr173/index.html

[20] Efficient XML. AgileDelta Inc. [Online]. Available: http://www.agiledelta.com/

[21] D. Peintner. (2013) EXIficient. [Online]. Available: http://exificient.sourceforge.net/

[22] S. Käisch, D. Peintner, J. Heuer, and H. Kosch, "XML-based Web service generation for microcontroller-based sensor actor networks," in *Factory Communication Systems (WFCS), 2010 8th IEEE International Workshop on*, May 2010, pp. 181 –184.

[23] (2010) EXI Unsigned Integer. [Online]. Available: http://www.w3.org/TR/2011/REC-exi-20110310/#encodingUnsignedInteger

[24] R. Kyusakov. (2014) Efficient XML Interchange Processor. LTU. [Online]. Available: http://exip.sourceforge.net/

[25] J. Johansson, M. Völker, J. Eliasson, Å. Östmark, P. Lindgren, and J. Delsing, "Mulle: A minimal sensor networking device - implementation and manufacturing challenges," in *Proceedings IMAPS Nordic*, 2004, pp. 265–271.

[26] B. Trower. (2010) Base64 implementation. [Online]. Available: http://base64.sourceforge.net/

[27] R. A. van Engelen and K. A. Gallivany, "The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks," in *2nd IEEE International Symposium on Cluster Computing and the Grid*, 2002, p. 128.

[28] W. Guo, "Performance Analysis of IP over IEEE 802.15.4 Radio using 6LoWPAN," Washington University in St. Louis, Tech. Rep., 2008. [Online]. Available: http://www1.cse.wustl.edu/~jain/cse567-08/ftp/7lowpan.pdf

[29] J. Delsing, J. Eliasson, and V. Leijon, “Latency and packet loss of an interferred 802.15.4 channel in an industrial environment,” in *Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on*, 2010, pp. 33 –38.

[30] *SOAP-over-UDP*, OASIS Std. [Online]. Available: http://specs.xmlsoap.org/ws/2004/09/soap-over-udp/

[31] D. Peintner, H. Kosch, and J. Heuer, “Efficient XML Interchange for rich internet applications,” in *Multimedia and Expo, 2009. ICME 2009. IEEE International Conference on*, 282009-july3 2009, pp. 149–152.

Paper C

Efficient XML Interchange in Factory Automation Systems

Authors:
Rumen Kyusakov, Henrik Mäkitaavola, Jerker Delsing, and Jens Eliasson

Reformatted version of paper originally published in:
37th Annual Conference on IEEE Industrial Electronics Society (IECON)

Efficient XML Interchange in Factory Automation Systems

Rumen Kyusakov, Henrik Mäkitaavola, Jerker Delsing, and Jens Eliasson

Abstract: The advent of Service-Oriented Architecture (SOA) in the automation domain has made possible the cross-layer vertical integration of devices, manufacturing systems and business processes. However, the use of standard web service technologies is not always possible in an industrial environment with high real-time requirements and limited hardware resources due to the overhead connected to XML processing. The work presented in this paper analyses the opportunities, advantages and challenges when applying the newly emerged Efficient XML Interchange (EXI) standard for XML encoding to the factory automation systems. The two major SOA-based automation middleware architectures, namely OPC Unified Architecture (OPC UA) and Devices Profile for Web Services (DPWS), were investigated. Furthermore, we present an EXI-based approach for extending the reach of the service technology covering deployments on resource constrained embedded devices.

1 Introduction and related work

The transition from static, single-purpose manufacturing systems to multi-functional, highly flexible and configurable industrial processes requires support for complex interactions and data exchange among numerous components within the automation system. This induces the extensive application of object-oriented technology, distributed computing and service oriented approach in the development of today's middleware automation solutions. Many successful protocols and concepts used in the domain of enterprise systems and the Internet infrastructure are directly applied to the automation systems while others must be adapted to the real-time, robustness and safety critical requirements found in the industrial environment. Example of this is the utilization of the SOA approach and web service protocol stack in particular. Unification of the communication infrastructure i.e. the use of Industrial Ethernet, IP on the network layer, TCP/UDP as a transport and SOA on the application layer, is seen as a way to provide better interoperability and hence lower system complexity, deployment and maintenance costs. A motivation, broader overview and technology survey on the application of Service-Oriented Architecture in industrial automation are given by Jammes et al. [1].

Today's service technologies are often making use of XML for structuring the content and metadata of the service messages. As an example, web service protocol called Simple Object Access Protocol (SOAP) [2] can be used as a means for defining the formatting of service requests and responses and specifying the interchange patterns. Each SOAP message is represented as a XML document with a predefined structure (XML schema). RESTful web services [3] are another SOA approach that does not rely on SOAP. Instead, the HTTP protocol capabilities are used to carry the service messages directly in the HTTP body. Although the data in the RESTful HTTP payload can be represented in different ways, in practice XML, JSON or similar encodings are applied. The verbose structure of XML and the high resource requirements for its processing is, in many cases, the bottleneck when applying the web service technology to the embedded systems often found on the plant floor. More detailed discussion and quantification of the performance overhead of using web services on deeply constrained networked embedded devices is provided in [4].

Different binary structured data representations, which are compatible with XML, have been suggested as an alternative to the text-based XML format. In [5], Sakr presents a survey on that with comparison of the most widely used compression techniques for XML. The binary XML working group of the World Wide Web Consortium (W3C) has also performed measurements and comparison between different structured data formats available in [6]. According to these studies the Efficient XML Interchange (EXI) standard [7] provides highest compactness ratio and fastest processing than other representations of the XML Information Set [8]. Based on that and a set of other evaluation criteria such us interoperability, compatibility, efficiency and compact implementation among others, the W3C standardization body released the EXI specification as a recommendation on 10th of March 2011. While there are some general discussions on applying EXI (see EXI Best Practices [9] and EXI Impacts [10] for overview), an analysis on the direct impact, applicability and possible benefits of using EXI in factory automation systems has not been performed yet. A study by Moritz et al. [11] on compression techniques for DPWS [12] is related to the work we present here as it considers the use of EXI for representing the messages defined in DPWS specification. However, its focus is rather narrow as it only investigates the applicability of EXI for DPWS from the perspective of compactness of the messages. The investigation we outline here is broader including processing efficiency along with compactness. In addition, we consider more aspects of the EXI application in the automation middleware systems - not only DPWS but also OPC UA [13] which is the latest most widely adopted industrial specification for factory automation. Integration of legacy systems is also covered and is essentially extension to the SOA for devices system presented in [14] while taking the EXI into account.

An overview of the web service technology and a state-of-the-art survey on its application for embedded devices is presented by Shelby in [15]. Therefore, our focus is on presenting the web service technology exclusively in connection to its usage in DPWS and OPC UA.

2 Background

This section provides short overview of the major concepts and technologies which are central for our discussions presented in the subsequent parts of the paper.

2.1 OPC UA

OPC Unified Architecture is an evolutionary upgrade to the classic OPC (Object Linking and Embedding (OLE) for Process Control) suite for data exchange in industrial automation systems. It provides a complete protocol stack for plant floor, operation and control networks by specifying the communication infrastructure including security mechanisms, transport protocols and exchange patterns as well as data representation and information models as presented in [16]. The first OPC specifications (classic OPC) are based on the proprietary Component Object Model (COM) and Distributed COM (DCOM) technologies which are now deprecated in favor of the Microsoft .NET Framework. Although the classical OPC was successful and widely used, its major drawback was the platform dependency stemming from the use of COM/DCOM. In order to address this issue and enhance the classic OPC with SOA support, the OPC foundation released the XML Data Access (XML-DA) specification which is entirely based on web service technology. It uses SOAP for communication,

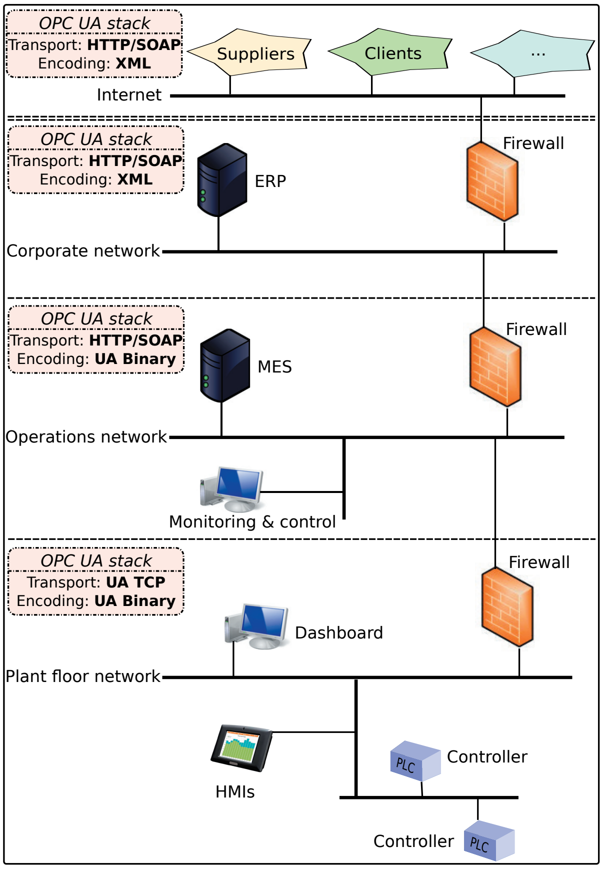


Figure 1: OPC UA transport protocols and encodings used in different layers in an industrial enterprise: plant floor, operational, corporate and inter-enterprise

Web Service Description Language (WSDL) for defining the service messages and XML Schema for specifying the data type hierarchy in the OPC information models. However, the performance of this SOA-based solution is not adequate for use in time-critical systems mainly due to the XML processing overhead. As a result, the OPC UA framework, released after XML-DA, provides an additional transport protocol and encoding for the predefined service messages described in [17]. A proprietary transport protocol called UA TCP is included and it operates directly on top of TCP. UA TCP is designed to substitute the resource expensive HTTP/SOAP in an environment with real-time requirements and hardware constrains. In such an environment, the text-based XML representation of the service payloads does not provide the required performance for parsing, serialization and compactness as well. Therefore, OPC UA defines additional encoding scheme called UA Binary that is proprietary binary representation of the OPC data models. Figure 1 depicts the relation of these encoding schemes (XML and UA Binary) and the two transport schemes (HTTP/SOAP and UA TCP) with respect to their intended usage throughout the industrial enterprise. As shown in the figure, the service calls on the plant floor are executed over UA TCP with UA Binary as encoding scheme. The target platforms on this level are controllers, wired/wireless HMIs and process visualization and reporting consoles (dashboards). For the interactions on the operations layer, the standard HTTP/SOAP transport is better suited for interconnecting the variety of higher level applications such as Manufacturing Execution Systems (MES) and monitoring and control panels. The timing in this layer is important in many cases - delivering of alarms and other critical events for example, so fast and efficient encoding of the messages is preferable. On the other hand, the fully interoperable web service representation based on HTTP/SOAP with XML is the best choice when connecting business systems such as Enterprise Resource Planning (ERP), Enterprise Asset Management (EAM), Accounting etc.

2.2 DPWS

As opposed to OPC UA, DPWS is an open and freely available OASIS specification. Moreover, there are several open source tools available (see [18, 19] for example) for developing DPWS applications in C/C++ and Java. The core building block in the DPWS standard is the SOAP-based web service technology where a subset of the WS-* family of protocols are employed. The usage of these protocols is further constrained or augmented in order to fit into the requirements posed by the target domain for DPWS - smart networked embedded devices. This formally specified set of constrains and usage restrictions is known as web service profile. As shown in Figure 2, the profile relies on WSDL 1.1, SOAP 1.2, SOAP-over-UDP and several WS-* protocols.

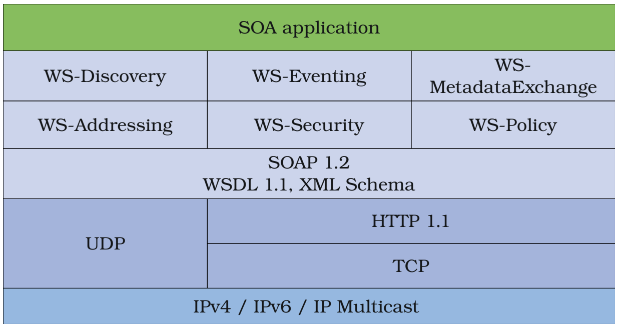


Figure 2: DPWS protocol stack

The problems that DPWS is designed to address are connected to enabling vendor independent and secure SOA-based communication with plug-and-play capabilities, automatic configuration, remote management and deployment. DPWS, however, is not specially targeting automation system applications and thus cannot provide a rich domain specific information model as the one available in OPC UA [20]. The predefined services in DPWS for example are limited to describing the properties and functionality of a generic device i.e. device model, name and manufacturer properties and generic services such as service discovery, listing of hosted services and resources. Nevertheless, the application of DPWS in industrial environment has been studied by researchers and it is shown to be feasible and beneficial for certain use-cases with low real-time requirements. Several demonstrators and case studies were implemented within the SIRENA project [21] for example. The lack of automation-specific information model in DPWS can be overcome by using external standardized domain data models. For example, Business to Manufacturing Markup Language (B2MML) can be used to link business systems such as ERP and supply chain management systems with manufacturing systems such as control systems and MES. The Senor Web Enablement framework [22] of the Open Geospatial Consortium (OGC) is another data model applicable in industrial environment for unification of the sensor data representation and semantics.

2.3 Efficient XML Interchange

EXI is a binary representation of the XML Information Set that is designed for compactness and high performance parsing and serialization. The theoretical foundation of EXI format is derived from information and formal language theories such that each XML Info Set document can be constrained by a set of formal grammars in Greibach normal form [23]. Moreover, each terminal symbol in these grammars i.e. a XML element, attribute, element content, closing element etc., is represented with variable length codes such that the most likely to occur symbols are encoded in fewer bits. Each EXI document consists of a header and a body. The header defines different encoding/decoding parameters that affect the compactness, processing efficiency and memory usage when processing the document. For example, the document size can be further reduced by using the EXI compression option indicating the application of DEFLATE algorithm for data compression [24]. Another important parameter is the use of a XML schema that sets constraints on the structure and content of the document. The schema information must be available before the EXI encoding/decoding, and it should be either statically set or communicated in advance using schema languages such as Document Type Definition (DTD), W3C XML Schema or RELAX NG. Schema-enabled EXI processing is much faster and results in smaller EXI documents compared to schema-less processing. For more detailed discussions on the EXI format and a primer see [25].

The main advantages of the EXI format can be summarized as follows:

- Highly compact representation comparable to a hand-optimized application specific encodings.
- Capability to preserve all features found in a XML document - even comments and processing instructions. This property guarantees that proven technologies, such as web services, XHTML, SVG and many others that are based on XML, are amendable to EXI representation.
- Efficient processing algorithms available which are also generic enough to work on documents with or without schema information or in a mixed mode where only part of the document is structured according to the schema. In addition, these algorithms can be coded in a small footprint implementation.

3 Analysis results

This section presents the results of our analysis on the application of Efficient XML Interchange format as an encoding technique for OPC UA and DPWS. The evaluation of the opportunities, benefits and challenges is covering three different aspects: compactness of the service messages, processing efficiency and legacy systems integration. We based our study on the OPC UA performance measurements presented in [26], the W3C EXI evaluation studies [6, 27] as well as the data for size reduction of DPWS messages with EXI encoding [11].

3.1 Compactness

In this work, the compactness of the EXI encoding is given as a percentage of the EXI document size compared to the text-based XML representation. The compactness is highly dependent on the document characteristics (size and content density) as well as EXI encoding options (usage of compression, schema information etc.). As shown in [6], the EXI compactness span from 1% for the large documents with compression and schema encoding enabled till up to 95% for schema-less encoding of very small documents. Therefore, in order to assess the compactness of the EXI format for OPC UA and DPWS, it is important to take into account the characteristics of the service messages and the targeted EXI options. In OPC UA and DPWS the service messages are relatively small, ranging from 225 bytes to more than 10 kB in XML representation. Schema information is available, yet schema-less encoding can be used for the service payload in OPC UA so both schema and schema-less metrics are important. As indicated in [11], the DEFLATE compression has little effect on the small messages used in DPWS and it also impacts the processing speed. For that reason, the use of DEFLATE compression option in EXI is not considered in our analysis.

The average compactness for the EXI encoded DPWS messages given in [11] is 20.6% for the schema encoding and 59.6% in schema-less mode. We performed similar measurements for the OPC UA services [17]. The description and formating of the service messages are taken from the OPC UA service descriptions[1] freely available for non-members. The average compactness achieved for auto-generated service messages including the SOAP envelope is given in Table 1.

Table 1: Average compactness of the OPC UA service messages

Encoding type	Average compactness in %	Average size [bytes]
XML (text)	100.0	2547
EXI (schema-less)	32.5	523
EXI (schema)	3.2	80

As shown in Table 1, the compactness achieved for OPC UA is much better than the one for DPWS mainly due to the highly structured messages used in OPC UA. It can be concluded that Efficient XML Interchange encoding for DPWS and OPC UA provides huge improvements in messages size compared to text-based XML.

3.2 Processing efficiency

Based on the measurements presented in [26], the use of HTTP/SOAP transport in OPC UA decreases the service execution speed by 50% on average compared to UA TCP. However, the negative impact on the performance is much higher if a text-based XML encoding is enabled instead of UA Binary. The use of XML accounts for 50% increase in latency for small single variable messages and up to 1800% for large messages with 1000 variables. According to the authors, the test setup for this measurements consisted of two PCs running .NET UA Stacks with a 100 MBit network link in between. The performance degradation is caused by the XML processing as well as the network latency as a function of the throughput and the size of the messages. If the use of EXI for OPC UA service calls provides a performance gain in the same rank as the one observed from the use of UA Binary, then

[1]Services.wsdl - http://opcfoundation.org/UA/2008/02/Services.wsdl

enabling EXI encoding in OPC UA can serve as a single data representation providing flexibility, efficiency and interoperability at the same time. The use of a single data format will lower the complexity and the size of the OPC UA implementations as there is no need to support multiple parsers and serializers for XML and UA Binary. On the other hand, the smaller program footprint of the OPC UA stack could enable the use of SOA-based communications on highly resource constrained controllers and sensor nodes - a concept further explored in the next section.

Measurements comparable to the above ones can be found in [6], where the EXI processing efficiency for a similar setup is given. The tests were made on a PC and a server machine connected over 100 MBit network link using Java EXI parsers and serializers. In both encoding and decoding cases, the performance gain compared to plain XML is in similar magnitude as the one seen for the UA Binary encoding above: the approximate average value in the 95% confidence interval being between [250% - 350%] for all test cases with schema-less processing and between [430% - 520%] in the schema-enabled mode. It is also expected that the performance gain for DPWS will be close to the one for OPC UA due to the similarities in size and structure of the messages.

3.3 Legacy systems integration

The introduction of EXI encoding for DPWS and OPC UA will provide substantial improvements in compactness and processing efficiency and hence lowering the hardware requirements for the legacy systems integration approach presented in [14]. In this way, the proposed SOA gateways that expose the legacy systems functionality as services can be less powerful and put closer to the systems they wrap. Moreover, the real-time properties of the network will be easier to satisfy as the compact EXI messages are less traffic intensive.

However, the new EXI based SOA protocols for factory automation must ensure the backward compatibility not only with legacy systems using proprietary and vendor dependent protocols but also XML capable devices. We identified the following methods for EXI integration of SOA ready XML-based enterprise and middleware systems running DPWS or OPC UA:

1. Deploy software updates or plugins providing support for EXI processing - characterized by optimal performance, but raising safety issues and relatively long non-working maintenance window required
2. Introducing EXI network proxies that translate EXI to XML and vice versa - transparent and non-intrusive method requiring additional networked hardware components
3. Modification in the application protocols used such that EXI is included as an option in the encoding negotiations

4 EXI on wireless sensor and actuator nodes

The use of flexible and standard-based service messages exchanged throughout all levels of the industrial enterprise is certainly providing many benefits as already discussed in the previous section. In such an environment, the seamless and cost-effective integration of small ubiquitous sensor and actuator devices spread across the shop floor is further providing new opportunities for process monitoring and control by supplying timely and accurate measurements for the executed workflows. A promising and already established standard for low-cost, low-power and limited bandwidth wireless communications is IEEE 802.15.4 [28]. By means of this standard, battery powered embedded devices called wireless sensor and actuator nodes can be easily deployed to provide close to the source measurements and events information. This information can be filtered, aggregated and enriched on different levels: on the sensor nodes as well as on intermediate and higher level systems. The network layer integration of such devices is largely addressed by the 6LoWPAN specification [29] for IPv6 compatible communications over IEEE 802.15.4 based networks. However, the application layer protocols are mostly proprietary and specially optimized for use over 6LoWPAN as the frame size of IEEE 802.15.4 is only 127 bytes and fragmentation is required when the application layer payload is bigger than between 76 and 116 bytes (depending on the addressing and security options). The approach presented here is based on the use of the same application protocols already used on the plant floor. The motivation for this approach is the simpler and cost-effective configuration and deployment as the need for middleware gateways is diminished. The use of efficient encoding and processing of the application data is also required and we suggest the usage of standard EXI encoding executed on the sensor nodes. In order to verify the proposed extension of the SOA communications down to the wireless sensor nodes, we performed a number of performance tests. For our experimental setup, we assume an OPC UA environment similar to the one depicted in Figure 3.

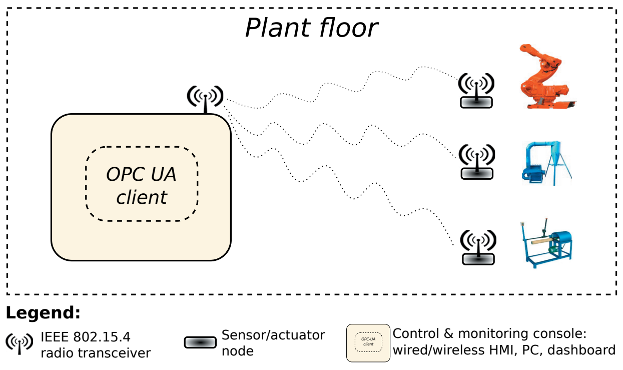

Plant floor
OPC UA client
Legend:
IEEE 802.15.4 radio transceiver
Sensor/actuator node
Control & monitoring console: wired/wireless HMI, PC, dashboard

Figure 3: Example OPC UA setup: integration of wireless sensor nodes through the use of UA TCP as a transport with EXI encoded payload

In our simplified testing configuration we used a PC and two Mulle v6.2 sensor platforms [30]. The Mulle nodes were equipped with a Renesas M16C/65 microcontroller running at 10 MHz with 47 kB RAM and 512 kB programming memory using 2.4 GHz IEEE 802.15.4 radio transceiver for wireless communications. The software running on the nodes included TinyOS environment and Berkeley Low-power IP stack (BLIP) as a 6LoWPAN network layer implementation.

According to the OPC UA specification, a session establishment is needed before the data exchange between an OPC UA client and an OPC UA server is possible. The session establishment is accomplished through a handshake service messages which are not considered in our setup in order to reduce the implementation efforts required. That way, we assume the presence of already available and active session between one of the sensor nodes and the PC. The studied exchange pattern is a reading of a value from a OPC UA server instance - in our case it is one of the sensor nodes, by a OPC UA client instance i.e. the PC. The second node is only used to deliver the IEEE 802.15.4 packets to and from the PC i.e. it provides access to the IEEE 802.15.4 wireless medium. This is done by

Table 2: Size of the OPC UA *Read* service request and response messages for plain XML, EXI schema-less and EXI schema-enabled

	Message size [bytes]	
Encoding type	Request	Response
XML (text)	912	712
EXI (schema-less)	362	251
EXI (schema)	56	31

installing the *IP BaseStation* TinyOS utility application to that node such that it transmits the received packets from the serial port over the radio and sends the received packets over the radio to the serial port connected to the PC. The service messages are standard *Read* service request and response for a single variable value. Table 2 shows the size of the request and response messages with different encodings applied. In our experimental setup, we used the EXIP open source library[2] to parse and serialize the EXI encoded OPC UA *Read* service messages. We used the EXIP low-level API [31] to map the message parameters encoded in an EXI binary stream to a C structure. As the current version of the EXIP library does not provide support for schema-enabled EXI processing, the messages were encoded without schema information using the default EXI header options.

In our experiment, the PC initiated a read request over TCP consisting of UA TCP protocol header (28 bytes) followed by the EXI encoded message (362 bytes). The receiving sensor node parsed the request, then serialized the response with the value of the requested variable and sent it back to the PC. The response again included a 28 bytes UA TCP header and an EXI payload of 251 bytes. We measured the time for the whole service call that includes: TCP connection establishment, serialization of the service request by the PC, transmission of the request, parsing of the request by the sensor node, serialization of the response by the sensor node, transmission of the response back to the PC and parsing of the response by the PC. The average time for complete service execution in this setup was measured 1.2 seconds. While the PC parsing and serialization time for the EXI encoded messages can be neglected (approximately 0.5 ms) these operations for the sensor node are quite time consuming as they accounted for around 534 ms. However, these measurements can be seen as an upper limits as the EXIP library does not employ the most optimal processing algorithms. We envision significant improvements in the EXI message processing when more efficient representation of the EXI grammars is employed. Furthermore, the utilization of the available schema information for the service messages will substantially improve not only the processing, but also the transmission time as the size of the messages is several times smaller (see Table 2). The use of more compact messages reduces the number of 6LoWPAN packages required and hence the transmission time.

Along with the improvements in service execution time, memory consumption must also be addressed. The EXIP library and the simplified service engine used in our scenario occupy almost 57 kB of programming memory. The Random Access Memory (RAM) consumed during EXI parsing and serialization is around 8 kB. Both programming memory and RAM footprints are expected to decrease when schema information is employed in the EXI processing.

[2]Efficient XML Interchange Processor - http://exip.sourceforge.net/

5 Conclusion and future work

Based on the presented analysis, the application of Efficient XML Interchange in factory automation systems could bring significant improvements in resource utilization (network throughput, computational power and memory footprint) when compared to plain XML encoding used in DPWS and OPC UA. Similar improvements could be achieved with other binary encodings (UA Binary, Fast Infoset, BSON, esXML etc.) so more in-depth studies are needed to determine the benefits of EXI while taking into account non-functional properties such as interoperability and backward compatibility.

The concept of extending the SOA approach to embedded devices through the use of EXI was also discussed and proof of concept measurements were presented. The first results in this direction showed the feasibility of this approach while pointing out the need for further investigation of efficient processing algorithms for EXI. Finally, considering the available room for improvements, we can conclude that the Efficient XML Interchange encoding is a viable solution for extending the applicability of the SOA approach across small wireless sensor and actuator devices.

Acknowledgment

The authors would like to thank the European Commission and the partners of the EU FP7 project IMC-AESOP (www.imc-aesop.eu) for their support.

References

[1] F. Jammes and H. Smit, "Service-oriented paradigms in industrial automation," *Industrial Informatics, IEEE Transactions on*, vol. 1, no. 1, pp. 62 – 70, 2005.

[2] *Simple Object Access Protocol (SOAP) 1.1*, W3C Std. [Online]. Available: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[3] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology*, vol. 2, pp. 115–150, 2002.

[4] C. Groba and S. Clarke, "Web services on embedded systems - a performance study," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, march 2010, pp. 726 –731.

[5] S. Sakr, "XML compression techniques: A survey and comparison," *J. Comput. Syst. Sci.*, vol. 75, pp. 303–322, August 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1530897.1531090

[6] G. White, J. Kangasharju, D. Brutzman, and S. Williams, "Efficient XML Interchange Measurements Note," W3C, Tech. Rep., 2007. [Online]. Available: http://www.w3.org/TR/exi-measurements/

[7] J. Schneider, T. Kamiya, D. Peintner, and R. Kyusakov, *Efficient XML Interchange (EXI) Format 1.0*, W3C Std., February 2014. [Online]. Available: http://www.w3.org/TR/exi/

[8] J. Cowan and R. Tobin. (2004) XML Information Set (Second Edition). W3C. [Online]. Available: http://www.w3.org/TR/xml-infoset/

[9] M. Cokus and D. Vogelheim, "Efficient XML Interchange (EXI) Best Practices," W3C, Tech. Rep., 2007. [Online]. Available: http://www.w3.org/TR/2007/WD-exi-best-practices-20071219/

[10] J. Kangasharju, "Efficient XML Interchange (EXI) Impacts," W3C, Tech. Rep., 2008. [Online]. Available: http://www.w3.org/TR/2008/WD-exi-impacts-20080903/

[11] G. Moritz, D. Timmermann, R. Stoll, and F. Golatowski, "Encoding and Compression for the Devices Profile for Web Services," in *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, 2010, pp. 514 –519.

[12] *Devices Profile for Web Services Version 1.1*, OASIS Std., 2009. [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf

[13] *OPC Unified Architecture*, OPC Foundation Std., 2009. [Online]. Available: www.opcfoundation.org/ua/

[14] S. Feldhorst, S. Libert, M. ten Hompel, and H. Krumm, "Integration of a Legacy Automation System into a SOA for Devices," in *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*, 2009.

[15] Z. Shelby, "Embedded web services," *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 52–57, 2010.

[16] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer, 2009, ch. Introduction, pp. 1–16.

[17] ——, *OPC Unified Architecture*. Springer, 2009, ch. Services, pp. 125–189.

[18] (2011, April) Service-oriented architecture for devices. [Online]. Available: http://forge.soa4d.org/

[19] (2011, April) Web services for devices (ws4d). University of Rostock, University of Dortmund and MATERNA. [Online]. Available: http://ws4d.e-technik.uni-rostock.de/

[20] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer, 2009, ch. Standard Information Models, pp. 107–122.

[21] H. Bohn, A. Bobek, and F. Golatowski, "SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains," in *International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006*, 2006, p. 43.

[22] Sensor Web Enablement. Open Geospatial Consortium. [Online]. Available: http://www.opengeospatial.org/projects/groups/sensorweb

[23] S. A. Greibach, "A new normal-form theorem for context-free phrase structure grammars," *J. ACM*, vol. 12, pp. 42–52, January 1965. [Online]. Available: http://doi.acm.org/10.1145/321250.321254

[24] P. Deutsch, *DEFLATE Compressed Data Format Specification version 1.3*, Internet Engineering Task Force Std., May 1996. [Online]. Available: http://www.ietf.org/rfc/rfc1951.txt

[25] D. Peintner and S. Pericas-Geertsen, “Efficient XML Interchange (EXI) Primer,” W3C, Tech. Rep., 2009. [Online]. Available: http://www.w3.org/TR/2009/WD-exi-primer-20091208/

[26] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer, 2009, ch. Performance, pp. 305–309.

[27] C. Bournez, “Efficient XML Interchange Evaluation,” W3C, Tech. Rep., April 2009. [Online]. Available: http://www.w3.org/TR/exi-evaluation/

[28] IEEE Computer Society, *IEEE 802.15.4-2006*, IEEE Standards Association Std., 2006. [Online]. Available: http://standards.ieee.org/findstds/standard/802.15.4-2006.html

[29] *IPv6 over Low power WPAN (6LoWPAN)*, IETF Std., August 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4919.txt

[30] J. Johansson, M. Völker, J. Eliasson, Å. Östmark, P. Lindgren, and J. Delsing, “Mulle: A minimal sensor networking device - implementation and manufacturing challenges,” in *Proceedings IMAPS Nordic*, 2004, pp. 265–271.

[31] R. Kyusakov, J. Eliasson, and J. Delsing, “Efficient structured data processing for web service enabled shop floor devices,” in *Industrial Electronics (ISIE), 2011 IEEE International Symposium on*, june 2011, pp. 1716 –1721.

PAPER D

Application of Service Oriented Architecture for Sensors and Actuators in District Heating Substations

Authors:
Jonas Gustafsson, Rumen Kyusakov, Henrik Mäkitaavola, and Jerker Delsing

Reformatted version of paper originally published in:
MDPI Sensors

Application of Service Oriented Architecture for Sensors and Actuators in District Heating Substations

Jonas Gustafsson, Rumen Kyusakov, Henrik Mäkitaavola, and Jerker Delsing

Abstract: Hardwired sensor installations using proprietary protocols found in today's district heating substations limit the potential usability of the sensors in and around the substations. If sensor resources can be shared and re-used in a variety of applications, the cost of sensors and installation can be reduced, and their functionality and operability can be increased. In this paper, we present a new concept of district heating substation control and monitoring, where a service oriented architecture (SOA) is deployed in a wireless sensor network (WSN), which is integrated with the substation. IP-networking is exclusively used from sensor to server; hence, no middleware is needed for Internet integration. Further, by enabling thousands of sensors with SOA capabilities, a System of Systems approach can be applied. The results of this paper show that it is possible to utilize SOA solutions with heavily resource-constrained embedded devices in contexts where the real-time constrains are limited, such as in a district heating substation.

1 Introduction

To apply a System of Systems approach, where sub-systems of different nature will interact, system-wide communication must be enabled. A system-wide communication architecture enables more efficient resource usage where measurements and services can be reused in a "wrap-and-reuse" fashion instead of the expensive "rip-and-replace" method used today. The introduction of a system-wide communication does not come easily and a number of limitations have to be overcome. In this paper, we evaluate the energy and timing cost of utilizing a high level SOA architecture on resource-constrained wireless devices - a research problem with important implications in wireless sensor/actuator network research [1]. The time and energy cost induced by the SOA-on device approach should be weighed against the cost of using a multitude of specialized solutions, requiring advanced middleware that often requires frequent "re-calibration" or updates.

As an example application, district heating suppliers can introduce new customer services and high-level control to improve district heating (DH) system efficiency. As district heating is a widespread heating method in Europe, and growing in, e.g., China, large financial benefits are possible through improved monitoring and control.

Combining sensors, meters and actuators distributed throughout a DH system with heat plant control and enterprise systems will enable completely new possibilities for system wide control and optimization. Customer services such as remote meter reading, fault detection and heat-system control can be significantly improved and provided directly by heat suppliers or third party companies.

Within a district heating system (and other energy distribution networks such as the electricity grid) there are often many independent data collection systems for billing purposes and automatic meter reading (AMR), e.g., power line communication and proprietary radio solutions. These are often non-flexible and expensive methods that requires much support from different maintenancestaff. By the introduction of SOA technology a more generic approach is achievable, where meters can integrate directly with higher level system without the use concentrators or adapter solutions.

The SOA solution will also enable the possibility to integrate the DH meter reading system with other building monitoring & control and home automation, e.g., ventilation, alarms and lightning, creating a System of Systems architecture. A larger scale system of systems view of our approach is in the context of optimization of heat production, where SOA enabled devices at end-customers can be orchestrated in real-time to reduce peak-production at the heat plants.

We now present a short introduction to the key components of our proposed approach.

1.1 Service Oriented Architecture

Service oriented architecture (SOA) is seen as a promising technique to bridge the gap between various industrial devices and enterprise applications [2]. Closer integration between ubiquitous embedded systems such as wireless sensor nodes and high-end systems could lead to higher flexibility in process optimization and evolution. The application of this concept has been a research objective of the European ITEA projects Sirena [3], Soda [4] and EU framework project Socrades [2] which had great support from the large industrial actors such as ABB, SAP, Schneider Electric and Siemens. As part of the outcome of these projects, a high-level architecture built on top of the web service technology was proposed. A key element in this architecture was the use of the same communication protocols on all levels of the industrial enterprise - business, manufacturing execution systems (MES) and the shop floor. As the deployment of the web service technology on resource-constrained devices leads to unacceptable overhead in terms of RAM, CPU and network usage the use of intermediate components such as gateways and mediator services was suggested. However, the introduction of additional hardware and software modules to the system increases its complexity and maintenance costs. As opposed to this approach, the solution presented in this paper deploys the web service interface directly on the resource-constrained wireless sensor nodes. To lower the resulting overhead, a newly emerging encoding technique for the XML service messages was employed.

1.2 Wireless Sensor Networks

To enable easy installation and avoid “wirenests”, a wireless approach is preferable. In this paper, we are advocating a wireless solution based upon open well-established standards. As there are a number of sensing and actuating devices to be interconnected, a Wireless Sensor and Actuator Network (WSAN) approach is appropriate.

A WSAN consists of a number of wireless sensor platforms (nodes) distributed over an area to monitor surrounding conditions, e.g., temperature or flow, or to control a physical property such as pump speed. The sensor platforms can communicate wirelessly, generally using radio technology. The number of connected nodes (sensor platforms) can vary from just a few up to several thousands [5].

During the last few years, the technology of 6LoWPAN [6, 7] has enabled the next generation Internet protocol IPv6 to be used on top of IEEE 802.15.4 low-power radio modules. This has enabled the use of standard Internet-suite communication protocols (TCP/IP family) and cheap, resource-constrained devices [8]. With the ability to use TCP/IP at the sensor level, the need for gateways with advanced middleware solution is obviated. It also enables the use of standardized SOA-technologies directly on the sensor platforms. However, most existing SOA-suites were developed to function on powerful desktop or server computers connected with high-bandwidth networks [9], which is a major challenge when deploying SOA on lightweight embedded devices.

In this paper, the focus is on sensor level performance, and not network performance. Network performance and simulation of complete network are very interesting and important aspects, but we did not find them suitable to include in this paper, and hence left it out for future work. In addition, security aspects are an important factor when it comes to large scale deployment, this has also been left out of paper, since we consider it to be a too complex question to fit inside the boundaries of this paper.

1.3 District Heating

District heating is a technology in which a central heat plant (boiler) heats a distribution medium (usually water) to be distributed in an often city-wide underground pipe system (distribution network). This system can achieve a higher fuel efficiency than the use of independent boilers in each house, and it also reduces installation cost and complexity in connected properties because no local boiler has to be installed [10].

The distribution medium temperature (T_{ps}) is controlled by the heat supplier; it is normally increased in response to decreased outdoor temperatures to compensate for the increased heat demand and lowered at lower heat loads to limit heat overproduction and thermal losses.

In DH-connected buildings, the heat carried by the distribution medium is transferred to house internal housing systems. In most modern DH-systems the distribution medium is separated from the internal heating systems (space heating and tap water heating) in a so-called district heating substation (DHS). The heat is transferred through a setup of heat exchangers to heat both the tap-water and space heating systems. A simplified DHS installation can be seen in Figure 1. In the substation, there are also a number of sensor and electro-mechanical actuators that control the heat transfer rate to maintain a stable indoor temperature. For billing purposes, a heat meter is also found in the substation to measure how much energy the customer uses.

These devices have traditionally been statically hardwired in a predefined setup using proprietary protocols (analog or digital). In the last few years, wireless indoor and outdoor temperature sensors have become more common; however, they are still using proprietary communication methods. The other devices are still hardwired to a control unit that interconnects the peripheral devices and hosts the control algorithms.

In [11], a WSN-enabled DHS control system was created using the 6LoWPAN/IPv6 technologies, where the control calculations were performed on the wireless sensor platforms. In this paper, we are evaluating the possibility of adding a service oriented architecture to the system presented in [11] in order to improve its flexibility and interoperability.

1.4 Paper Structure

The rest of the paper is structured as follows: Section 2 provides overview of the related work and how the current approach relates to the state-of-the-art research in the area. In section 3, we first explain the field of application, which is district heating and what challenges and opportunities we see. Further we go through the technical specification of the wireless sensor network platform and network architecture used in the experiments. Section 4 provides detailed description of the protocol suite and service oriented architecture used in the setup. In section 5 we give an overview of the

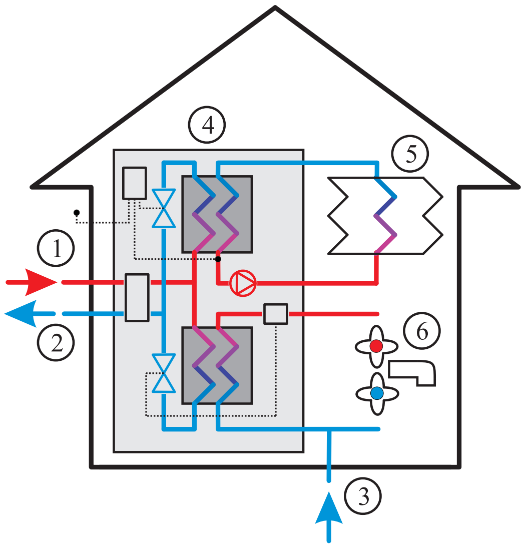


Figure 1: 1, Incoming district heating. 2, Returning district heating. 3, Incoming cold tap water. 4, District heating substation. 5, Radiator system. 6, Water tap.

data standards that are of interest for the sensor data acquisition field and present an example of how we structure the data. In section 6 an overview of the experimental setup hardware is presented. In section 7 the experimental results are presented and discussed. Section 8 concludes the paper and analyses the applicability of the approach.

2 Related Work

The idea of using Service Oriented Architecture for systems level integration and systems evolution has already been investigated and proved beneficial for improving the resource utilization and extensibility. A study on application of SOA for enabling evolvable production systems has been presented by Candido *et al.* [12]. This work proposed a high level architecture for customizable production infrastructure where system functionality is discovered and composed dynamically based on existing services and semantic meta-data in the form of ontologies. A similar approach is presented for the domain of maritime surveillance by Parlanti *et al.* in [13]. They introduced a SOA-based platform that is targeted at the integration and interoperability of heterogeneous systems owned by different stakeholders. In this article we applied similar approach for the domain of district heating and extended the aforementioned architectures with concrete measurements and application of device level web services.

A work that is more closely related to the discussions we present here is given in [14]. The authors of this work were proposing an approach for integration of wireless sensor nodes in a service oriented facility management architecture. They suggested the use of SOA for integration of various systems such as heating, ventilation and air conditioning (HVAC), security, electricity systems *etc.* As opposed to our approach, they were using gateway devices for accessing the WSNs and the sensor nodes were not equipped with web service interfaces. Moreover, their work did not provide validation of their architecture through simulations or prototype experiments. Another difference is that, while their architecture inherited many principles from the Sensor Web Enablement (SWE) framework [15], their approach was not compliant with the SWE specifications.

The Sensor Web Enablement (SWE) framework is a major building block in the Service Oriented Sensor Web architecture proposed in [16]. However, the authors of this architecture implement and evaluate the SWE specification on a gateway and the communication with the sensor devices is still based on ad-hoc protocols.

3 Functionality and Architecture of Future DH Substations

In this paper, we have chosen to use district heating as an example application area because we know that in the DH-industry there is a need for open, system-wide solutions that can evolve over time.

In the competition for heating customers, it is of the utmost importance to maintain a high quality of service (QoS) at a competitive price. Historically, district heating has been known to be a reliable and convenient heating method that requires very little or no maintenance at the customer level. However, the ability of both customers and heat suppliers to obtain useful information on how and where the energy is used within the building has been limited (which is not a unique problem for district heated houses). The space heating control system(s) has also been difficult to calibrate for maximum performance. Systems for high-level monitoring and control that includes DHS functionality have been non-existent until recently. By introducing wireless sensors, which are available for a multitude of purposes, the functionality and performance of the substation (and indirectly the complete DH system) can be improved. The use of loosely coupled sensors and actuators with the possibility to reconfigure and interconnect them can enable new types of services.

Over the last 5–10 years, several new inventions have emerged that would increase the functionality and QoS for both heat suppliers and customers; see [17, 18, 19, 20] for a selection of promising recent research results. Each of these results can be implemented today using specialized hardware installations, making it expensive for the customer to utilize them all. For that reason we suggest the utilization of open, vendor-independent standards for inter-systems communication. Specialized solutions using proprietary protocols and hardware are by default incompatible with products from other manufacturers. This often leads to an expensive vendor lock-in for both energy companies and property owners.

Using the well-established open communication suite of TCP/IP at the sensor level enables a reliable communication architecture that will support high-level SOA integration. However, as previously discussed, most SOA technologies today are too complex for low-power, highly resource-constrained embedded devices. Using emerging encoding technologies like EXI (see Section 4 for further details), it becomes possible to apply open and SOA compatible solutions applicable for WSN installations.

3.1 Wireless Sensor Network Architecture

The sensor platform used in the underlying experiments to this paper is the so-called Mulle v6.2 platform, modified to support 2.4 GHz radio frequency. The Mulle platform originated from the EISLAB division at Luleå University of Technology, but it is now commercially available through Eistec AB [21]. The Mulle is designed with low-power consumption in mind, but it is still powerful enough to support, e.g., a TCP/IP stack. Both TinyOS [22] and Contiki [23] lightweight operating systems are ported to the Mulle; in this paper, we have used TinyOS exclusively.

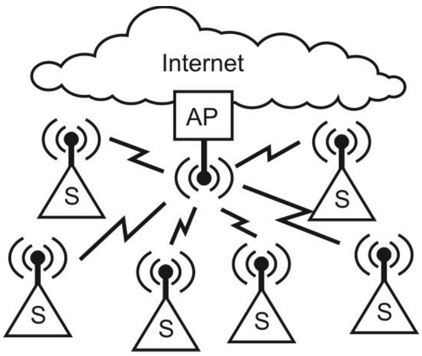


Figure 2: Sensor network. Sensors and actuators are connected to wireless sensor platforms (S). The sensor platforms can advertise their services within the sensor network and on-line as they utilize the Internet Protocol (IPv6). All on-line traffic is routed through the access point (AP).

The wireless sensor platforms connect to the Internet through an Internet access point (AP) as depicted in Figure 2. The AP also acts as an edge-router for the 6LoWPAN network and supports the following interfaces: Ethernet, 3G/GPRS, IEEE 802.15.4 and Bluetooth communication. The operating system running the Access Point is μCLinux [24], running on a Blackfin [25] micro-controller from Analog Devices.

3.2 Network Technologies

The network connection between the sensor platform and the access point relies on the PHY and MAC layer specified in IEEE 802.15.4 [26]; see also Figure 3. The standard supports multiple radio frequency band profiles, in the experiments conducted for this article we used the 2.4 GHz profile with a bit rate set to 250 kb/s.

On the network layer, we used the next generation Internet Protocol (IPv6) [27], which is already supported by most routers on the Internet today. This makes the WSAN in the DH substation globally accessible for systems connected to the Internet.

The IPv6 header has fixed length of 40 bytes (320 bits) with a MTU (Maximum Transfer Unit) of 1280 bytes. However, the frame size of IEEE 802.15.4 is only 127 bytes, with 76–116 bytes of payload available, depending on the addressing and security options [7]. This requires fragmentation of the IPv6 packets that is handled by the 6LoWPAN adaptation layer [6], found in between the Network- and Link-Layers in the protocol stack, see Figure 3. The 6LoWPAN also handles the header compression of the IPv6 packages to make the transmission over the IEEE 802.15.4 link faster and more resource efficient.

The transport protocol of choice in the experimental setup is UDP. Due to the limited bandwidth and computational power of the resource-constrained sensor platforms, TCP can be experienced as slow, with long round-trip time (RTT) [28].

At the application level, we are using the simple object access protocol (SOAP) to encapsulate messages to be sent and received over the network as further explained in Section 4. A possible

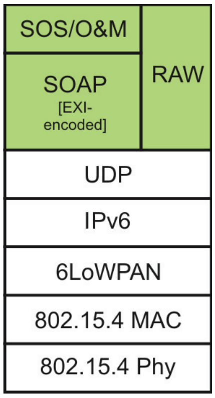


Figure 3: The complete network protocol stack used in the setup.

alternative or complementary approach to SOAP would be to use CoAP (Constrained Application Protocol) [29], which is specifically developed to work well with resource constrained systems. However, as parts of the underlying work of this paper were conducted before the CoAP standard was established, we choose not to introduce CoAP any further in this particular paper, but consider it for future work. An overview of CoAP features and comparison with SOAP for embedded systems is presented in [30].

4 Web Services

Although there are a number of different open and proprietary solutions conforming to the SOA paradigm, web service technology is the most widely adopted by the industry for at least two reasons. First, it is not encumbered by any patents and licenses but is built upon the same standards and protocols already used and proven in the world wide web. Second, not only are the web service specifications and standards free to the public, but there are also many open source software tools and development kits for building SOA applications in different programming languages. A service message carrier protocol called Simple Object Access Protocol (SOAP) is often used as a means for defining the formatting of service requests and responses and specifying the interchange patterns. Each SOAP message is serialized as an XML document with a predefined structure and can be sent using different network protocols. Although SOAP is independent of the transport protocol used, in practice, the SOAP documents are embedded in the HTTP body or sent directly as a UDP payload. Recent efforts aimed at applying the SOA approach to embedded systems have resulted in the Devices Profile for Web Services (DPWS) [31] specification, where the binding of SOAP to UDP was formally defined. The SOAP-over-UDP [32] supports unicast one-way, multicast one-way, unicast request, unicast response, multicast request and unicast response message patterns through the use of WS-Addressing specifications. The web service implementation used in our experimental setup was based on SOAP-over-UDP on top of an IPv6 network layer, and the tests included the unicast request and unicast response message exchange patterns.

Moritz *et al*. investigated the use of SOAP on top of CoAP by defining formal binding for the two protocols [33]. They showed that the new binding provides certain benefits such as lower delays and

Table 1: Size of the request and response messages for plain XML, EXI schema-less and EXI schema-enabled encoding.

	Message Size [bytes]	
Encoding Type	**Request**	**Response**
XML (text)	761	1100
EXI (schema-less)	451	651
EXI (schema)	169	258

overhead compared to HTTP binding and higher reliabilities than simply using UDP. However, the SOAP-over-CoAP binding [34] has not progressed further and was not adopted as a standard.

The use of web services for inter-system communications fosters interoperability among heterogeneous and complex business and industrial processes. The benefits achieved result from the service's vendor independence and ability to interconnect different operating systems, programming languages and hardware platforms. However, the main building block in the web services specifications, namely, Extensible Markup Language (XML), is too verbose and resource consuming to be used for battery powered resource-constrained embedded devices. The size of the XML documents used to carry the service messages is the main limiting factor preventing the application of the SOA approach in the embedded domain, especially over wireless links. Nevertheless, recent advancements in binary encodings for XML have achieved performance that is very close to that of highly optimized data formats. The use of these binary encodings enables XML processing for severely constrained wireless sensor nodes. Evaluations of several high-performance XML encoding formats have been made available by XML Binary Characterization Working Group (Efficient XML Interchange Measurements—http://www.w3.org/TR/exi-measurements/). In addition, surveys comparing different binary representations for use in web service messages are investigated in [35, 9]. The highest efficiency and processing speed is achieved using the Efficient XML Interchange (EXI) format [36] developed by the W3C Binary XML Working Group. It is based on information theory and formal languages and uses knowledge of the structure of the XML documents to achieve compact representations for both document meta-data and payload. Each EXI document consists of a header and a body. The header defines different encoding/decoding parameters that affect the compactness, processing efficiency and memory usage when processing the document. An important parameter is the use of an XML schema that sets constraints on the structure and content of the document. The schema information must be available before the EXI encoding/decoding, and it should be either statically set or communicated in advance using schema languages such as W3C XML Schema. Schema-enabled EXI processing is much faster and results in smaller EXI documents compared to schema-less processing. Table 1 shows the sizes of the request and response messages used in our experiment encoded using plain XML, schema-less EXI and schema-enabled EXI.

In our experimental setup, we used the EXIP open source implementation (Efficient XML Interchange Processor—http://exip.sourceforge.net/) to parse and serialize EXI messages, as it was specially designed for resource-constrained embedded systems [37]. Our prototype implements schema-less processing but schema-driven encoding and decoding are also possible.

Due to the pre-conditions while doing the experiments for this paper, we did not have the opportunity to run schema-enabled EXI transmissions. However, a compression rate exceeding 90% compared to uncompressed XML when using strict mode schema-informed EXI serialization is reported by Altmann *et al.* [38], these results are comparable to what we would expect in our set-up.

5 Data Models

We find it important to follow available standards as much as possible. This also applies to the semantics of the XML structure describing the service messages. The Open Geospatial Consortium (OGC) is working on standards for enabling sensor communication over the web. The Sensor Web Enablement (SWE) [15] framework includes a number of standards under development to support developers making sensors and actuators discoverable, accessible and usable via the web. We identified Observation and Measurement (O&M) [39], Sensor Observation Service (SOS) [40] and SensorML [41] as the most essential for our application domain.

- **O&M** specifies an abstract data model to be used for physical observations and measurements
- **SensorML** defines standard syntactical constructs to specify the functionality and available services on a particular sensor platform.
- **SOS** provides an interface for managing sensor platforms and retrieving sensor data

```
<?xml version="1.0" encoding="UTF-8"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing"
xmlns:hts="http://www.ltu.se/HeatingNodes/"
xmlns:sos="http://www.opengis.net/sos/1.0">
  <s:Header>
    <a:To>soap.udp://valveNode:5555</a:To>
    <a:MessageID>urn:ltu:hts:1</a:MessageID>
    <a:ReplyTo>
<a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
    </a:ReplyTo>
  </s:Header><s:Body>
    <hts:reqMeasurements>
<sos:GetObservation>
  <sos:observedProperty>urn:ogc:def:phenomenon:OGC:temp</sos:observedProperty>
  <sos:responseFormat>text/xml;subtype="om/1.0.0"</sos:responseFormat>
  <sos:resultModel>om:Observation</sos:resultModel>
</sos:GetObservation>
    </hts:reqMeasurements>
  </s:Body>
</s:Envelope>
```

Figure 4: SOAP service request message in XML text format. EXI-encoded total size: 451 bytes.

The service messages included in our proof of concept experiment followed the formatting rules defined in O&M and SOS standards. However, it does not include the messages defined by SensorML. A complete description of the sensor nodes, including detectors, physical structure, input/outputs could also be added to the system using SensorML as a future work.

Figure 4 shows the complete request message used in the experimental setup. The responding platform parses the message and encodes a response message using the O&M standard, which can be seen in Figure 5.

6 Experimental Setup

To test the hypothesis of using a SOA in a low-power sensor network, a simple experimental setup was created. Two sensor platforms were programmed to be used in the experiment. One platform was

```
<?xml version="1.0" encoding="UTF-8"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing"
xmlns:hts="http://www.ltu.se/HeatingNodes/"
xmlns:om="http://www.opengis.net/om/1.0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <s:Header>
    <a:To>http://www.w3.org/2005/08/addressing/anonymous</a:To>
    <a:MessageID>urn:ltu:hts:21</a:MessageID>
    <a:RelatesTo>urn:ltu:hts:1</a:RelatesTo>
  </s:Header><s:Body>
    <hts:respMeasurements>
<om:Observation gml:id="Temp">
  <gml:description>Outdoor Air Temperature</gml:description>
  <om:samplingTime><gml:TimeInstant gml:id="ot1t">
<gml:timePosition>2011-01-11T16:22:25.00</gml:timePosition>
  </gml:TimeInstant></om:samplingTime>
  <om:procedure xlink:href="http://dh.eislab.com/procedure/outdoortemp.xml"/>
  <om:observedProperty xlink:href="urn:ogc:def:phenomenon:OGC:temp"/>
  <om:result xsi:type="gml:MeasureType" uom="urn:ogc:def:uom:OGC:Cel">22.3</om:result>
</om:Observation>
    </hts:respMeasurements>
  </s:Body>
</s:Envelope>
```

Figure 5: SOAP service response message in XML text format. EXI-encoded total size: 651 bytes.

programmed to request temperature information from the other platform using standard Web services. We refer to the requesting platform as the *Requester* and the responding platform as the *Responder*.

The Web service interface of the sensor platforms is a simplified SOAP engine that uses the Berkeley Low-power IP Stack (BLIP) of TinyOS to sent and receive UDP packets with SOAP messages as a payload (see Figures 4 and 5 for example messages). Instead of using plain XML representation the SOAP messages were encoded in EXI to reduce their size. The EXI encoded UDP payload is parsed by the EXIP library to extract the SOAP fields. The values of these fields are then passed to a protocol state machine that reacts on the content of the SOAP message - store the value, send a response, send an error when problems *etc*.

The current usage of the sensor platforms was measured using the simple measurement setup depicted in Figure 6. A small shunt-resistance ($\mathrm{R_s}$) of 1Ω was connected in series with the sensor platform, and the voltage ($\mathrm{V_s}$) generated across $\mathrm{R_s}$ was amplified 100 times and fed into a data acquisition card. The data acquisition frequency was set to 50 kHz to record all sensor-platform activity. From the recorded data, the current usage can easily be calculated using Ohm's law in combination with the amplification factor A; see Equation (1).

$$I_s = \frac{V_o}{A \cdot R_s} \tag{1}$$

The OGC-compliant Sensor Web Enablement (SWE) service requests and responses were compared against a minimalistic custom data transfer (RAW). In our setup RAW represents the minimal bytes sequence needed for carrying a set of structured data. The RAW message corresponds to a predefined C structure and is loaded without tokenizing the bit stream. The RAW format we chose has severe limitations that makes it impractical for real-world usage but we find it as an useful base for comparison as it represents the minimum possible size of a message. Figures 7 and 8 show the RAW-message constructs. In order to be usable in practice, the RAW format must be extended to correctly handle differences in endianness between hosts. Also adding the possibility to transmit different data

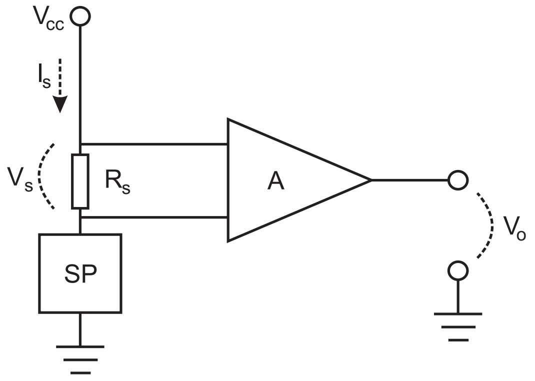


Figure 6: Current measurement setup. SP indicates Sensor Platform and A indicates amplifier. V_o is measured at 50 kHz by an A/D data acquisition card.

types and varying number of fields and support for meta-data fields is needed for application beyond a specific use case.

```
struct sensor_msg_header {
   unsigned char msg_type;
   unsigned int msg_id;
   unsigned char measurement_type;
};
```

Figure 7: RAW request message. Transmission size: 4 bytes.

```
struct sensor_msg {
   struct sensor_msg_header header;
   char descr[30];
   char unit[10];
   float value;
   unsigned long time_stamp;
};
```

Figure 8: RAW response message. Transmission size: 52 bytes.

To get an idea of time and energy required for serializing/sending and receiving/parsing EXI-encoded XML-messages the current usage was measured throughout the operation. Because the RAW transmissions do not require serialization like the EXI-encoding, the total transmission time will be significantly reduced for RAW-message operation. The smaller size of the RAW messages (maximum of 52 bytes) also obviates the need for message fragmentation by the 6LoWPAN layer. Thus, only one IEEE 802.15.4 packet is needed to transmit the required RAW data. In Section 7, the measurement results are analyzed further.

7 Results

The measurement results were divided into 4 independent processes to identify how much they contribute to the total time and energy usage for one interaction:

- **Serialization, SR.** Encoding and serializing the message using the EXI-standard.
- **Sending, SE.** Sending the EXI-encoded message using IPv6/6LoWPAN.
- **Receiving, RE.** Receiving the message.
- **Parsing, PA.** Parsing the received EXI-encoded message.

Depending on whether the current-measurement is performed on the Requester or the Responder, the order of the processes mentioned above will vary.

For the Requester, we also measure the total time from the initiation of serialization to the parsing of the received result also see t_{Return} in Figure 9a; we call this the *Return*-time. At the Responder, we measure the time from the reception starts until the transmission of the response message finishes, which we call the *Response*-time; see also $t_{Response}$ in Figure 9b.

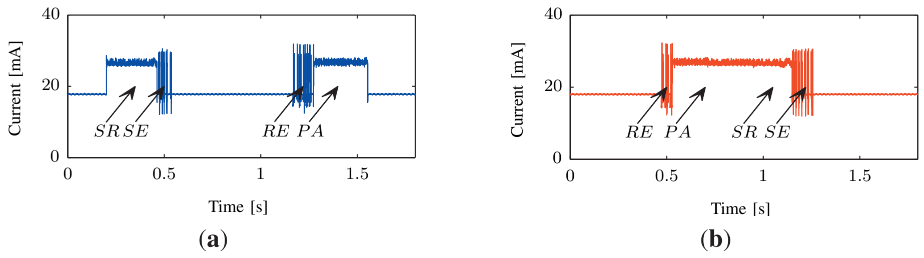


Figure 9: Current usage over time, at requester and responder, during an EXI-transaction. SR—Serialize, SE—Send. RE—Receive, PA—Parse. The sensor platform is not put into sleep mode in between activities; thus, current usage is ∼18 mA while idle. (**a**) Requester; (**b**) Responder.

In Table 2, the timing results for serializing, sending, receiving and parsing messages at both the Requester and the Receiver are presented, these results are also visualized in Figure 10. These measurement results reveal that it is the parsing and serialization of the EXI-messages that requires the most time and energy in the communication process.

Table 2: Average processing times in communication process. All results are in milliseconds.

	Requester [ms]					**Responder [ms]**				
Encoding	**Serialize**	**Send**	**Receive**	**Parse**	**Return**	**Receive**	**Parse**	**Serialize**	**Send**	**Response**
EXI (schema-less)	252	82	130	280	1350	82	182	424	130	818
RAW	2 *	5	9	*	48	5	*	2 *	9	16

* No serialization or parsing is required in RAW mode; however, the creation of the message structure will require a short period of time.

Compared to sending a minimal RAW payload over IPv6/6LoWPAN, the EXI-encoded SOA solution requires more energy and time to complete the operation of requesting the temperature. The RAW method return time is less than 0.5% of the EXI-encoded return time. However, the longer return time of the EXI solution should be weighted against the cost of having several individual specialized systems requiring specialized maintenance and support.

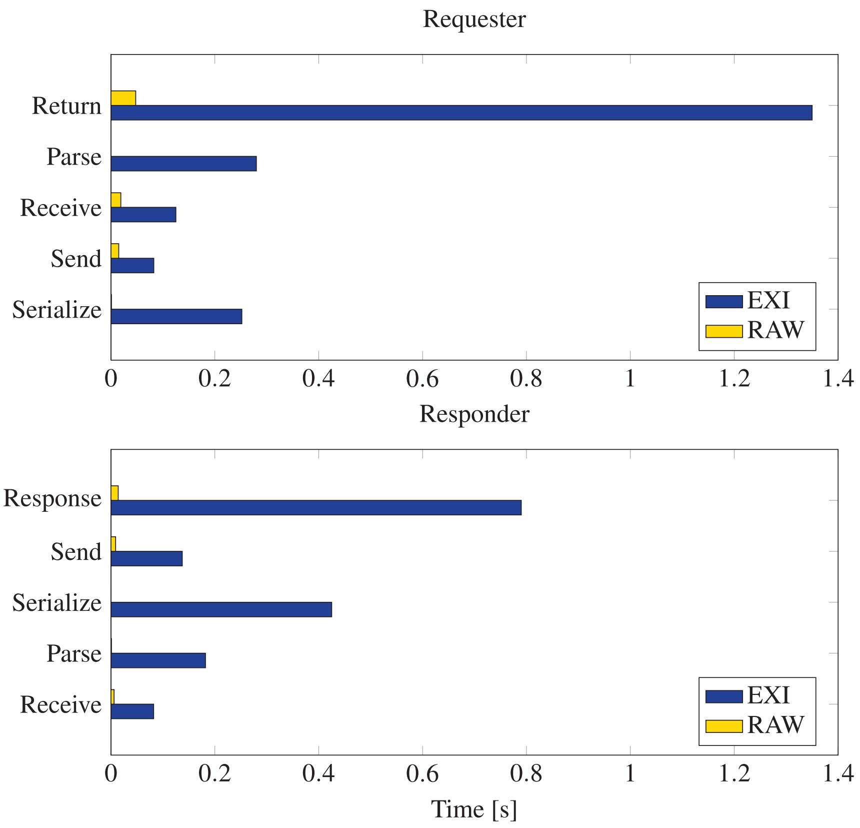


Figure 10: Visual presentation of Table 2, average processing times in communication process.

7.1 Time and Energy Constrains

With a return time of ∼1.3 s, the SOA approach used in this paper might not be suitable for use in real-time processes with significant time constraints. However, in our target application (DH-substations), a return time of 1.3 s does not have any direct implications for measurement and space heating control because the real-time requirements are low. For hot-water control in buildings without hot water circulation (HWC), a shorter return time might be advantageous to avoid temperature oscillation in tap water. Today, hot-water control in smaller properties without HWC is usually controlled by self-acting mechanical solutions; as this is not within the primary application area, we consider it to be a marginal issue.

Using SOA on battery-powered sensor platforms will affect the life expectancy of the sensors by increasing the computational and transmission time. To evaluate the effect of increased current consumption, an extrapolating calculation was performed.

The outdoor and indoor temperature sensors connected to a DH substation are typical devices where a battery is the only reasonable power source. Because these temperatures do not change rapidly, a sampling frequency of e.g., four measurements per hour is mostly enough for measure-

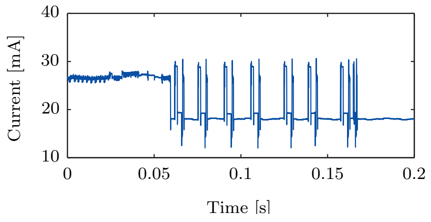


Figure 11: The figure shows the current consumed during transmission of an IPv6 packet. The 6LoWPAN fragmentation of the IPv6 packet is clearly visible. The IPv6 packet is divided into 8 6LoWPAN frames.

ment and control purposes. Because the control system in a district heating substation is working continuously, there is no need to ask specifically for a certain temperature each time it is required. A subscription method where the sensors involved automatically send their temperature readings to subscribing receivers is instead considered. Sending the temperature using an EXI-encoded O&M message to one subscribing receiver four times per hour requires the WSN platform to wake up from deep sleep (idle mode), read the temperature, serialize and send the message.

The energy Q_s used by a sensor platform for a specific time t can be calculated by Equation (2). The serialization of the temperature measurement takes ~425 ms at a reasonably constant current consumption, I_s of ~27 mA (see Figure 9) at a V_{cc} of 4.2 V. This results in an energy usage of ~48 mWs for one response message.

Figure 11 shows the current usage of a transmission in detail; observe the 6LoWPAN fragmentation (eight spikes) of the IPv6 package. The energy usage for the transmission is also calculated using Equation (2). Sending the serialized payload of 651 bytes will use approximately use 10 mWs. Thus, one complete temperature serialization and transmission uses approximately 48 + 10 = 58 mWs. Sending four messages per hour results in 238 mWs per hour which equals ≃65 µW in average power usage needed for serialization and transmission. In addition, 12 µW are consumed while the sensor platform is in deep sleep between transmissions, and there is also a necessary hardware-dependent energy for sensor reading, which we estimate to 8 µW on average in this example. The total average power consumption P can thus be approximated as 85 µW (65 + 12 + 8).

$$Q_s = V_{cc} \int_0^t I_s(t)dt \tag{2}$$

We can now estimate how much energy is needed to keep a this sensor platform alive for a specific time. Because heat meters in district heating substations need to be re-calibrated at 5 or 10 year intervals (Due to country specific rules and regulations), it would be convenient to have the battery powered devices running for at least the same amount of time between battery changes. The energy needed to keep a sensor platform alive for 5 years (43800 h) will require at least ~3.7 Wh; see Equation (3). However in practice, we realize that a somewhat bigger battery is required since there will be energy losses within the battery. Approximately twice the theoretical capacity should in practice be enough. Commonly found batteries in commercial heat meters are often around 5 Ah at 3.6 V (17 Wh) with a physical dimension of around 6 × 3 cm. Adding roughly 2 Ah to this battery, either by adding another battery or replacing it with a bigger one would in most cases not require any change to the physical dimensions of the heat meter.

$$Q_{tot} = t_{life} \cdot P = t_{life} \cdot (P_{tx} + P_{stop} + P_{temp.read}) \quad (3)$$

7.2 Memory Usage

In addition to the time and energy constrains, wireless sensor nodes have limited amounts of programming and random access memory (RAM). For example, the Mulle node used in our experimental setup features 512 kB of ROM and 47 kB RAM. The EXIP library and the service engine handling the SOAP messages occupy almost 80 kB of programming memory. The RAM used is allocated during the EXI message processing and is freed immediately afterwards to be made available for the IP stack routines. Although the RAM consumed by our web service solution can be safely shared and reused by other modules running on the sensor node, its size is relatively large, between 7.1 and 9.1 kB for serialization and between 7.7 and 9.8 kB for parsing of the EXI messages.

The measured ROM and RAM consumption can be seen as an upper limit for this particular case, as we have not implemented any optimizations on the code, and we have not used any optimization parameters for the compiler. In a simple test to support this statement, we changed the dimension of several 32 bit integer variables in the EXIP library to 16 bit integers, assuring that the correctness of the execution would be preserved. This change alone resulted in between 0.5 kB and 0.8 kB less dynamic memory used during service request/response execution.

7.3 Comparison with Related Work

It is desirable to compare the timing, energy and memory measurements presented in this section with similar approaches in the literature in order to provide a sensible context for evaluation. Unfortunately, there are not many studies using standard Web services on sensor nodes and providing concrete measurements of the system properties. One such study [42] reports response time overhead for Web services of about 2.5 times, *i.e.*, the SOAP-based Web service interactions take about 2.5 times longer to complete compare to conventional plain text representation for a Sun SPOT sensor platform. This measure is much lower compared to our experimental results mainly due to the added overhead for parsing and serialization of the EXI encoding which is not used in the aforementioned study. Furthermore, the time difference between conventional representation and Web services messages becomes larger when only a single sensor reading is included in the message as it is the case in our test setup. This is also shown in the evaluation section of the study on using DPWS in sensor networks [43] where and overhead of about 4 times is reported with a single sensor reading using text encoded SOAP messages.

This comparison highlights the EXI processing as a main source of latency in the proposed architecture. While there are many optimizations that are possible in the EXI processing part of our approach, the application domain of district heating is relatively slow process that does not require real-time communication with the sensors and the currently presented response times are sufficient for its operation.

8 Discussion and Conclusion

In this paper, we have evaluated the possibilities of using a service oriented architecture on top of wireless sensor networks in district Heating substations. Our long-term goal in this research is to integrate sensor data from embedded devices in district heating substations with enterprise level systems through the use of service oriented architectures. This will improve the potential sensor data utilization and system flexibility.

The results in this paper demonstrate that it is possible to use SOA on small embedded devices, such as wireless sensor platforms. With schema-enabled EXI parser/serializer in place, the size of transmitted packages will become even smaller, enabling even more constrained and cheaper sensor platforms to comply with the required hardware specifications. With a more optimized software for parsing and serializing EXI-messages, we expect that the return-time can be reduced with at least 50%. When schema-enabled EXI is operational, further reduction in parsing/serializing will be even more efficient, and send/receive times will also be shortened because the message size will be reduced. Still, the processing and transmission time will be longer, and hence the energy requirement will be larger than those in highly specialized solutions, such as the RAW approach used in this paper. However, it is important to put the increased time and energy usage in relation to the cost of keeping several specialized solutions operational and compatible.

The approach of loosely coupled sensors, with the ability to interconnect in a non-predefined way, is controversial in the district heating industry, which is heavily based on strong tradition. However, to offer new services to the customers without expensive additional hardware solutions, it is essential that open and vendor independent solutions are implemented. This was the main approach presented in this article towards a SOA based System of Systems architecture.

To make the presented work available for commercial use, security and data validation issues must be considered. The introduction of security models will affect the expected life-length of battery power sensors as energy is needed to encrypt/validate the data, however, we considered the security issue to be an issue for future work.

The application of the web service enabled wireless sensor nodes presented in this paper is not in any way limited to the area of district heating. Other domains that would benefit from this technology are process automation, facilities management and maritime surveillance to name a few.

Acknowledgment

The authors are grateful for all the support from the IMC-AESOP and Arrowhead projects.

References

[1] F. Xia, “Qos challenges and opportunities in wireless sensor/actuator networks,” *Sensors*, vol. 8, no. 2, pp. 1099–1110, 2008.

[2] S. Karnouskos, D. Guinard, D. Savio, P. Spiess, O. Baecker, V. Trifa, and L. M. S. de Souza, “Towards the real-time enterprise: Service-based integration of heterogeneous soa-ready industrial devices with enterprise applications,” in *Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM).*, Moscow, Russia, Jun. 2009.

[3] H. Bohn, A. Bobek, and F. Golatowski, “Sirena-service infrastructure for real-time embedded networked devices: A service oriented framework for different domains,” in *Networking, International Conference on Systems and International Conference on Mobile Communications and*

Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on. IEEE, 2006, pp. 43–43.

[4] R. Harrison, F. Jammes, H. Smit, and T. Kirkham, "Revolutionizing sensor-based automation in manufacturing," *ERCIM News*, vol. 2009, no. 76, 2009. [Online]. Available: http://ercim-news.ercim.eu/en76/special/revolutionising-sensor-based-automation-in-manufacturing

[5] C. S. Raghavendra, K. M. Sivalingam, and T. F. Znati, Eds., *Wireless sensor networks*. Springer, 2004.

[6] *IPv6 over Low power WPAN (6LoWPAN)*, IETF Std., August 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4919.txt

[7] Z. Shelby and C. Bormann, *6LoWPAN: The Wireless Embedded Internet*, November 2009.

[8] J.-P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP - The Next Internet*. Elsevier, 2010.

[9] Z. Shelby, "Embedded web services," *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 52 –57, 2010.

[10] S. Fredriksen and S. Werner, *Fjärrvärme, Teori, teknik och funktion*. Lund, Sweden: Studentlitteratur, 1993.

[11] J. Gustafsson, H. Mäkitaavola, J. Delsing, and J. van Deventer, "Integration of an ip based low-power sensor network in district heating substations," in *The 12th International Symposium on District Heating and Cooling*, 2010.

[12] G. Candido, A. Colombo, J. Barata, and F. Jammes, "Service-oriented infrastructure to support the deployment of evolvable production systems," *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 4, pp. 759–767, nov. 2011.

[13] D. Parlanti, F. Paganelli, and D. Giuli, "A service-oriented approach for network-centric data integration and its application to maritime surveillance," *Systems Journal, IEEE*, vol. 5, no. 2, pp. 164–175, june 2011.

[14] A. Malatras, A. Asgari, and T. Bauge, "Web enabled wireless sensor networks for facilities management," *Systems Journal, IEEE*, vol. 2, no. 4, pp. 500–512, dec. 2008.

[15] *Sensor Web Enablement (SWE)*, Open Geospatial Consortium (OGC) Std., March 2011. [Online]. Available: http://www.opengeospatial.org/ogc/markets-technologies/swe

[16] X. Chu and R. Buyya, "Service oriented sensor web," in *Sensor networks and configuration*. Springer, 2007, pp. 51–74.

[17] J. Gustafsson, J. Delsing, and J. van Deventer, "Improved district heating substation efficiency with a new control strategy," *Applied Energy*, vol. 87, no. 6, pp. 1996–2004, 2010.

[18] F. Wernstedt, P. Davidsson, and C. Johansson, "Demand side management in district heating systems," in *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM, 2007, pp. 1–7.

[19] P. Lauenburg and J. Wollerstrand, “Adaptive control of radiator systems for a lowest possible return temperature,” in *The 12th International Symposium on District Heating and Cooling*, 2010, pp. 206–214.

[20] K. Yliniemi, J. Delsing, and J. van Deventer, “Experimental verification of a method for estimating energy for domestic hot water production in a 2-stage district heating substation,” *Energy and Buildings*, vol. 41, no. 2, pp. 169–174, 2009.

[21] “EISTEC AB,” March 2010. [Online]. Available: http://www.eistec.se/

[22] “Tinyos,” March 2010. [Online]. Available: http://www.tinyos.net

[23] “Contiki,” March 2014. [Online]. Available: http://www.contiki-os.org/

[24] “uCLinux,” March 2011. [Online]. Available: http://www.uclinux.org/

[25] Analog Devices, “Blackfin processors,” January 2009. [Online]. Available: http://www.analog.com/en/embedded-processing-dsp/blackfin/content/index.html

[26] IEEE Computer Society, *IEEE 802.15.4-2006*, IEEE Standards Association Std., March 2011. [Online]. Available: http://standards.ieee.org

[27] *Internet Protocol, Version 6 (IPv6) Specification*, IETF Std. RFC 2460, December 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2460.txt

[28] W. Guo, “Performance Analysis of IP over IEEE 802.15.4 Radio using 6LoWPAN,” Washington University in St. Louis, Tech. Rep., 2008. [Online]. Available: http://www1.cse.wustl.edu/~jain/cse567-08/ftp/7lowpan.pdf

[29] Z. Shelby, K. Hartke, and B. C., *Constrained Application Protocol (CoAP)*, IETF Std., Dec 2013. [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-coap-18

[30] B. C. Villaverde, D. Pesch, R. De Paz Alberola, S. Fedor, and M. Boubekeur, “Constrained application protocol for low power embedded networks: a survey,” in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. IEEE, 2012, pp. 702–707.

[31] *DPWS, Devices Profile for Web Services Version 1.1*, OASIS Std., February 2011. [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf

[32] *SOAP-over-UDP*, OASIS Std., February 2011. [Online]. Available: http://specs.xmlsoap.org/ws/2004/09/soap-over-udp/

[33] G. Moritz, F. Golatowski, and D. Timmermann, “A Lightweight SOAP over CoAP Transport Binding for Resource Constraint Networks,” in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, Oct 2011, pp. 861–866.

[34] G. Moritz. (2011) Soap-over-coap binding. [Online]. Available: http://tools.ietf.org/html/draft-moritz-core-soap-over-coap-01

[35] G. Moritz, D. Timmermann, R. Stoll, and F. Golatowski, “Encoding and compression for the devices profile for web services,” in *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, 2010, pp. 514 –519.

[36] *Efficient XML Interchange (EXI) Format 1.0*, W3C Std. [Online]. Available: http://www.w3.org/TR/exi/

[37] R. Kyusakov, J. Eliasson, and J. Delsing, “Efficient structured data processing for web service enabled shop floor devices,” in *20th IEEE International Symposium on Industrial Electronics, 2011*, 2011.

[38] V. Altmann, J. Skodzik, F. Golatowski, and D. Timmermann, “Investigation of the use of embedded web services in smart metering applications,” in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, Oct 2012, pp. 6172–6177.

[39] *Observations and Measurements*, Open Geospatial Consortium (OGC) Std., February 2011. [Online]. Available: http://www.opengeospatial.org/standards/om

[40] *Sensor Observation Service*, Open Geospatial Consortium (OGC) Std., March 2011. [Online]. Available: http://www.opengeospatial.org/standards/sos

[41] *Sensor Model Language*, Open Geospatial Consortium (OGC) Std., March 2011. [Online]. Available: http://www.opengeospatial.org/standards/sensorml

[42] C. Groba and S. Clarke, “Web services on embedded systems - a performance study,” in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, march 2010, pp. 726 –731.

[43] R. Kyusakov, J. Eliasson, J. Delsing, J. van Deventer, and J. Gustafsson, “Integration of Wireless Sensor and Actuator Nodes With IT Infrastructure Using Service-Oriented Architecture,” *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 43–51, Feb 2013.

PAPER E

Emerging Energy Management Standards and Technologies - Challenges and Application Prospects

Authors:
Rumen Kyusakov, Robert Cragie, Jens Eliasson, Jan van Deventer, and Jerker Delsing

Reformatted version of paper originally published in:
17th Conference on Emerging Technologies Factory Automation (ETFA)

Emerging Energy Management Standards and Technologies - Challenges and Application Prospects

Rumen Kyusakov, Robert Cragie, Jens Eliasson, Jan van Deventer, and Jerker Delsing

Abstract: The continuously rising costs and the environmental impact of energy generation, transmission and consumption are a major concern for governments, industry and society alike. Among research in renewable energy sources as well as in energy efficiency of buildings, electrical appliances, vehicles etc., a considerable amount of attention has been devoted to effective energy management. In this paper, we present a survey on emerging energy management standards with focus on enabling application layer Information and Communications Technologies (ICT) that are a central part of these standards. The presented work includes an analysis on the challenges, future trends, security and application prospects of energy management standards. As part of the survey, the emerging Open Automated Demand Response (OpenADR) version 2.0 and Smart Energy Profile (SEP) version 2.0 were identified as the most promising and complete solutions. The presented survey provides an important insight on the future developments in the area of energy management protocols and highlights a number of key ICT solutions and challenges.

1 Introduction

Enabling automatic exchange of information between different interdependent systems can optimize wide range of industrial processes. The access to dynamic pricing information and automatic load control can for example be used to employ methods that manage the electricity consumption or delivery in a way that lowers the costs and environmental impact of the consumed electrical energy [1]. The use of such two-way communication infrastructure for electrical power grids is called Smart Grid. The goal of the Smart Grid is to enhance reliability of electricity distribution, reduce peak demand, shift usage to off-peak hours, lower total energy consumption and carbon dioxide footprint.

The deployment of Smart Grid infrastructure encompasses the coordination of the processes in a large number of systems: electricity producers (power plants and Distributed Energy Resources (DER)), electricity transmission and distribution, residential/commercial buildings and other energy consumers [2]. These systems have interdependencies with other systems that can be used to further increase the process optimization. For example, connecting district heating and cooling management systems with Smart Grid infrastructure can be used to reduce the peak energy demands especially when combined heat and power (CHP) plants are used [3].

Even small improvements in reducing the peak demands and energy footprint in residential and commercial buildings have substantial impact on the energy consumption. For example, the buildings sector in the U.S. accounted for almost 40 % of all primary energy consumption in 2008 - that is 43 % more than the transportation sector and 24 % more than the industrial sector [4].

There are a number of problems in today's energy management systems. First, there is a discrepancy between the customer load and the value of energy at a given point in time. Unlike other consumer markets, the energy market is very inflexible in a sense that the demand is not sensitive to supply constraints and prices which is caused by poor communication. It is not unusual that the energy demand is at its highest during peak prices periods. Second, the increase in the number of DER, including generation (e.g. wind and solar power, industrial co-generation) as well as storage (e.g. electrical vehicles), makes the coordination of supply and demand to maintain the reliability of the distribution network challenging.

It is clear that a communication infrastructure, that connects all the heterogeneous systems and facilitates an automatic coordination, monitoring and control based on predefined energy management policies, is needed in order to address the problems of the energy market. However, the currently available communication standards for energy management does not encompass all the aspects and requirements of such communication infrastructure. Instead, they are defined within a narrow scope e.g. transmission (IEC 61970), distribution (MultiSpeak, IEC 61968), substation automation (IEC 61850), Advanced Metering Infrastructure (AMI) (M-Bus, IEC 62056, Universal Metering Interface (UMI)) or supervisory control and data acquisition (SCADA) for power system automation (IEC 60870-5, IEC 60870-6, DNP3/IEEE 1815-2010).

While the interoperability between these different systems and the large number of devices is crucial, the communication technologies in use are often incompatible, based on proprietary solutions, unable to support heterogeneous PHY/MAC layers (e.g. SEP v1.0) or impossible to scale to all use cases. One parameter connected to that is the efficiency of communication protocols i.e. low bandwidth, CPU and memory requirements, which are needed to support the growing number of devices that are taking part in the energy management systems.

1.1 Internet of Things in energy management

The concept of distributed energy management requires a number of systems that traditionally have been isolated to cooperate and exchange information. The sources and scope of the information flows are very heterogeneous and include measurements and sensor data from various sparsely distributed processes. In gathering the data and controlling these processes, the Internet of Things (IoT) technology can be a viable and cost-effective solution [5]. A key enabling technology for low-cost IoT energy management deployments is the use of wireless communication for sensor and control data. In this case, the overall cost of deployment is much lower as compared to wired installations.

The use of IoT technology also imposes some additional constraints, such as low bandwidth communication, limited radio coverage, resource-constrained devices etc. This might prohibit, or limit, the use of standardized security protocols, full-fledged communication stacks, and require new, more efficient data communication solutions.

1.2 Problem formulation

The main objective of this work is to investigate the state of the art and the newly emerged application layer technologies used for energy management and the Smart Grid in particular. Broader discussions on the available demand-side management techniques along with motivation, benefits, opportunities and challenges of their application are presented by Strbac [6]. The analysis presented in his study highlights the lack of ICT infrastructure as a major challenge for demand-side management.

Nevertheless, there are no systematic studies on the requirements and possibilities of using available and emerging ICT in energy management applications. The analysis in [7] investigates the communication requirements and challenges from very high level and abstract perspective without connecting the results of the study to concrete protocols and technologies. In contrast, the work presented in [8] is limited to ANSI C12.22 and Session Initiation Protocol (SIP) and does not consider the latest developments in data representation formats and application layer protocols.

For that reason, the work presented here is dedicated to ICT challenges and enabling technologies in some of the recent energy management standards. The considered research questions include:

- What are the ICT and security challenges when applying the emerging energy management standards?
- How can recent developments in efficient data formats and application protocols be used to mitigate these challenges?
- What is the level of support for IoT devices by the investigated standards and technologies?

2 Requirements

This paper considers a set of technical requirements used as a standpoint during the analysis of the energy management standards. The requirements are focused on the ICT and do not include general functionality that the standards should cover as presented in [7]. A mapping between an exhaustive list of key energy management functions, similar to the one presented in [9], and the concrete standards is very important research topic. However, this calls for defining representative set of use cases, market assessments etc. and it is out of the scope of this work.

The set of requirements that we consider builds on the work presented in [8] and includes the following:

- Interoperability - it is vital that the ICT technologies support end-to-end communication services in a wide range of application domains for the energy management such as Smart Grid, district heating/cooling, ventilation, building and industrial automation. The interoperability requires the use of standard-based and open communication technologies that support heterogeneous communication links.
- Efficiency - low overhead in terms of network bandwidth, CPU, RAM and programming memory is essential for the IoT devices.
- Scalability - the technologies must scale to large networks - both in terms of geographical distribution and number of nodes.
- Security
- Efficient installation and network management

3 Communication technologies

Building the whole Smart Grid communication infrastructure from scratch would require huge R&D and development efforts. Therefore, it is almost universally agreed that the Internet, including network protocols and related infrastructure, should be used as communication backbone for the energy management systems.

Our primary interest is on high level standards and technologies that define data serialization techniques and exchange patterns rather than frame and packet definitions on PHY/MAC, network and transport layers. Nevertheless, this section describes various low-level communication technologies that are important for the energy management systems. The support for these networking technologies by the application layer energy management standards is a key criterion for comparison and analysis used later on in the paper.

Another motive for the broader investigation in this section is security - the application layer energy management standards depend entirely on the lower layers network protocols for securing the communication. Traditionally, building automation systems and sensor networks have been closed networks running on specific wiring and therefore have never really required many of the security controls that are taken for granted in the Internet. However, with the advent of wireless technology and widespread access to the Internet, it has become increasingly important to design security into the communication protocols from the start.

3.1 Physical and data link layers

Wired technologies such as Ethernet and powerline provide high robustness and bandwidth with relatively low latency. Offices, residential buildings and industrial sites often have Ethernet installations. Powerline communication is also gaining acceptance in the networking community, especially since 6LoWPAN was selected in the new G3-PLC standard to transmit IP data over a standard powerline installation.

Being primarily of academic interest, wireless solutions have rapidly gained acceptance. The IEEE 802.11 family of standards (WLAN) and low power radio communications based on IEEE 802.15.4 such as WirelessHART, ISA100.11a and ZigBee are now widely adopted for industrial and consumer use. WiFi, with bandwidths of up to several hundreds Mbit/s, can support virtually any energy management service with low real-time requirements. On the other hand, the bandwidth of IEEE 802.15.4-based radio solutions, as well as some other low-power radio technologies such as Z-Wave, is constrained to no more than 250 kbit/s. Battery powered sensors and actuators are usually equipped with these low power radios and employ duty cycling and sleeping schedules to limit the periods in which the radio is turned on, hence reducing the power consumption. The combination of low bandwidth and battery power limitations requires networking, security and application layer protocols that are very light-weight in terms of message sizes and computational resources.

Many wireless MAC/PHYs specify a network access mechanism to ensure that only trusted devices join the network. The Extensible Authentication Protocol is often used to carry handshaking, which can check credentials of joining devices and provide a key which can be subsequently used to provide link-layer security. An example of this is Wireless Protected Access (WPA) in the 802.11 specification and Protocol for Authentication and Network Access (PANA) [10].

A higher level of security at the link layer is also common. For example, 802.15.4-2006 specifies the use of AES-CCM, which is a NIST-approved mode of operation [11] using the AES-128 block cipher [12]. Similarly, 802.11 specifies the use of AES-CCM amongst other protocols. AES-CCM uses a symmetric key and provides confidentiality, integrity and data origin authentication. When used with a unique frame counter, it also provides replay protection. Many of the chipsets which support these wireless protocols have the AES-CCM and AES-128 security engines built into the hardware, thus simplifying the implementation of a secure link layer protocol.

Network layer

IP protocol is the *de facto* network layer solution on the Internet. It has gained wide acceptance in industrial applications and for highly constrained networked devices where the interoperability with external systems is essential. The adoption of the latest version of the protocol (IPv6) has also progressed as it is crucial for supporting the huge number of IoT devices. Adaptation layer technologies such as IETF 6LoWPAN and lightweight routing protocols such as IETF RPL (both adopted in ZigBee IP) allow low-power devices with limited processing capabilities to take part in the Internet of Things.

IP-based networks can use IPsec, which is a suite of IETF specifications that provides confidentiality, integrity and data origin authentication to IP datagrams through security associations. The security association is established using a handshake based on one of a number of cipher suites, which passes credentials either one way or both ways. The credentials are checked and then used to establish one or more keys, which form the basis of the security association.

Transport layer

TCP and UDP are established transport layer protocols used in wide range of applications. Transport Layer Security (TLS) for TCP connections and Datagram Transport Layer Security (DTLS) for UDP serve as additional layers on top of the transport layer and provide confidentiality and integrity checking based on a secure session. The secure session is established in a similar way to the secure association for IPsec, i.e. using a handshake based on one of a number of cipher suites.

Presentation layer and cryptography

Two of the functions of the presentation layer are machine independent data representation and encryption/decryption. A key requirement for the data representation is interoperability of the exchanged information. As a result, distributed information systems often use XML, JSON or similar serialization of the data. The use of open serialization formats is a central part of the web service application layer technology where the messages are often stateless and exchanged over HTTP protocol with URL used for resource addressing. The main drawback of the web service protocols is the size of the messages which makes them a challenging approach for battery powered embedded devices. Nevertheless, recent developments in embedded web service protocols has shown promising results in this area [13].

Encryption and decryption functions use cryptography to provide confidentiality of the data. Public and symmetric key cryptography are also used as a basis for the authentication and key establishment protocols described earlier. Public key cryptography relies on difficult to solve mathematical relationships such as integer factorization and the discrete logarithm problem. However, these are often based on very large prime numbers, which require a considerable amount of processing power and produce large key sizes. Elliptic Curve Cryptography is a public key cryptography which requires much smaller key sizes and much less processing power and is therefore more appropriate for the often constrained nature of the IoT devices.

Symmetric key cryptography generally requires much smaller key sizes and processing power than public key cryptography. The main issue is distributing the same confidential information to all parties involved in the communication.

4 Data models

For complete interoperability between different energy management standards and systems supporting demand-side management, the common representation of the exchanged information - both syntactical and semantical, is of utmost importance. Various data serialization formats are in use today such as ASN.1, data tables (ANSI C12.19) and different proprietary solutions. Another well-established and gaining wider acceptance technology for defining the representation of the communicated data is XML and XML Schema definition language. One important benefit of this approach is the possibility to structure the data with human-readable meta-data in the form of hierarchies of elements and attributes. The use of informative names of the elements and a structure that represents the properties of the real-world objects of interest makes the development work, connected to wiring different application interfaces, much easier.

An important aspect of the data modeling is semantic definitions, i.e. a common understanding of the meaning of exchanged information. The semantic definitions are often represented with a natural language, UML models and ontology databases intended for system developers. While the human readable descriptions are important, they often contain ambiguities and are impossible to process by machines. One approach to address these problems is the use of ontologies that augment the syntactical description of the data with references to well-known ontology databases. As the amount of data generated by all devices and systems part of the Smart Grid infrastructure will be tremendous, the use of well-defined ontologies for the data representation should also be considered. This could bring more flexibility to the system operations and allow for automated reasoning when processing the information.

5 Energy management standards

A key requirement for successful application of demand-side energy management is interoperability of the communication infrastructure. Interoperability depends upon common standards [14], which define how data is exchanged, what data formats are used and how different systems react to certain events according to a common energy management policy that assures reliability of the energy management system.

This section presents the investigated standards and technologies. The focus is on the aspects that were defined in the problem formulation section: the use of ICT, application scope and support for constrained networked devices (IoT).

5.1 IEEE Smart Grid interoperability guide

This standard contains guidelines and best practices for interoperability of Smart Grid infrastructure [15]. It only serves as a high level architectural view of the system components and is not sufficient *per se* for deriving concrete implementations nor ensuring interoperability. However, it provides engineering principles and design evaluation criteria for the other concrete standards included in this study.

5.2 OpenADR version 2.0

Open Automated Demand Response (OpenADR) standard version 2.0 is an evolution and extension to the first version of the specification developed by Demand Response Research Center at Lawrence Berkeley National Laboratory. OpenADR 2.0 is supported by OpenADR industry alliance and developed as part of the OASIS Energy Interoperation 1.0 standard published in February 2012 [16].

The core of the OpenADR 2.0 standard is a set of data models and exchange patterns that define standard demand-response (DR) signals and the interfaces between energy markets (dynamic and transaction pricing information), Independent System Operators (ISO), utilities, Distributed Energy Resources (DER) and energy consumers (industrial/residential buildings). The data models (Energy Market Information Exchange (EMIX) and WS-Calendar) are defined using XML Schema definition language and hence all the DR information is communicated as XML messages. OpenADR 2.0 does not specify the communication medium and technologies used below the application layer. While the standard is designed to be agnostic regarding the actual transport scheme and communication protocols, the exchange patterns (interfaces) are based on Service Oriented Architecture (SOA) and are defined using Web Service Description Language (WSDL).

OpenADR 2.0 specifies different levels of conformance that together with the OpenADR certification program guarantees interoperability between different OpenADR 2.0 products. In order to claim full compliance however, the implementations should be compatible with WS-I Basic Profile, which requires the use of SOAP web services. Deploying SOAP web services on resource-constrained devices is challenging [17] and therefore the OpenADR scope does not cover IoT applications. While mainly targeted at Smart Grid deployments, OpenADR 2.0 can be applied to other distribution networks such as natural gas and water.

5.3 SEP version 2.0

Smart Energy Profile (SEP) version 2.0 [18] is an application layer specification for devices that are part of Demand Response and Load Management (DRLC) programs. It is supported by the Consortium for SEP 2 Interoperability (CSEP) and has been identified by the National Institute of Standards and Technology (NIST) as a primary candidate specification for energy information and control on the consumer side.

The specification includes smart metering, pricing and DRLC applications for devices in residential and light commercial buildings operating on a Home Area Network (HAN), sometimes called Premises Area Network. In this way, the scope of SEP2 differs from that of OpenADR: SEP2 is focused on HAN devices while OpenADR is designed for high level communications between the AMI Systems, utilities' communication networks and ISOs.

SEP2 builds on top of the IP protocol and is agnostic of the underlying PHY/MAC layers. Therefore, it supports Ethernet, WiFi, powerline communications and different low-power radio technologies. For the data exchange, SEP2 relies on RESTful web services that are defined as a set of CRUD operations (create, read, update, and delete) on a remote resource identified by a URL. The implementation of RESTful web services can use different application protocol bindings with HTTP being the most widely adopted where the CRUD operations are mapped to the HTTP methods GET, PUT, POST, and DELETE. As with OpenADR, the data model in SEP2 is defined using XML Schema definition language and maps directly to the IEC 61968 standard. As SEP2 is targeted at IoT deployments, the need for efficient representation of the XML messages is important, therefore the SEP 2 specification has identified Efficient XML Interchange (EXI) [19] as the most promising solution for

XML compression.

5.4 Building and industrial automation protocols

The success of demand side energy management depends on the active participation of energy consumers in controlling their loads in response to changes in energy prices. For the building sector, this requires a high level of automation of load control in which human interactions are restricted to choosing comfort and efficiency levels from an intuitive graphical user interface. In the case of industrial sites, the dynamic energy price can be included as a variable in the process control algorithms. Therefore, there is a need for close integration between the protocols delivering demand/response signals and the systems interpreting and reacting on these signals. The following paragraphs summarize the ICT used in some widely adopted automation protocols:

BACnet can use a number of PHY/MAC layers including Ethernet, RS-232, LonTalk etc. and work is being done to support ZigBee Building Automation profile as well. Some of the latest developments of BACnet are directed towards more open communication technologies such as BACnet over TCP/IP networks (BACnet/IP) and even defining web service profile (BACNet/WS).

LonWorks consists of a communication protocol (with associated transport channels) known as LonTalk (standards ISO/IEC 14908-1 to -4) and a proprietary application layer protocol. There are some initiatives to support tunneling of LonTalk over IP as well as providing web service interface to LonWorks.

ZigBee Home/Building Automation are two application profiles built on the ZigBee stack, which is in turn based on the IEEE 802.15.4 low-power wireless standard. They do not support IP and standard web services without protocol translation on a gateway device.

oBIX is an OASIS specification that aims at providing an open, XML- and web service-based exchange of information for building control systems.

IEC 61499 used with a widely adopted compliance profile (CPFD) requires the use of IP, TCP/UDP and a proprietary data encoding based on modified ASN.1

MTConnect is based on web service technology with HTTP RESTful interface and data serialization based on XML and XML Schema.

OPC UA defines several profiles that run on top of TCP/IP and use either XML or proprietary encoding format, as well as either HTTP-SOAP web services or proprietary transport scheme.

6 Technology analysis

Successful deployment of Smart Grid infrastructure requires interoperability of the communication systems that are part of it - starting from high-end enterprise management systems running on powerful servers or even cloud infrastructure to embedded sensors and actuators. Interoperability, on the other hand, depends on common standards, certification and conformance test programs. Looking at the scope of SEP2 and OpenADR2 for example (Figure 1), the end-to-end interoperability must

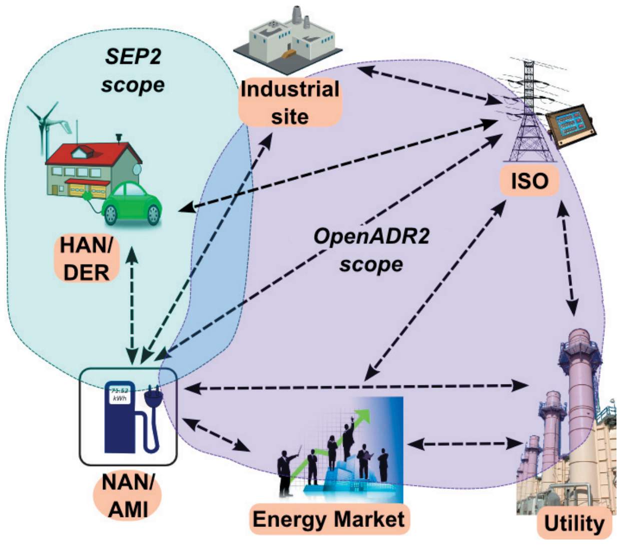


Figure 1: Application scope of OpenADR2 and SEP2

include a gateway device that translates the OpenADR2 demand-response signals to SEP2 load control and pricing messages. This requires certification and conformance test programs for end-to-end interoperability that go beyond the scope of a single standard. As such, the certification activities of the OpenADR alliance and CSEP must be aligned in the future to meet this challenge.

NIST's Smart Grid Interoperability Panel (SGIP) highlighted a number of gaps and inconsistencies along with action plans for the available communication, data representation, smart metering and energy management standards. SGIP also identified the need for open ICT that are proven to scale well over huge networks. Example of such proven technologies are the communication protocols that underpin the Internet - TCP/IP protocol stack, XML on the presentation layer and web services on the application layer. The trend towards the use of Internet protocols can be seen in the latest energy management standards (SEP2 and OpenADR2) and to some extend in the building and industrial automation sector. However, the use of open communication standards, XML and web services (SOAP- or RESTFul-based) in particular, brings new challenges that would make the transition from proprietary ICT problematic. The first group of challenges is related to the resource requirements of such solutions and the second one is related to security.

6.1 ICT resource requirements

The switch from proprietary solutions to a standard web service technology will often require more powerful hardware and hence increase the per unit cost of the final products. For example, a smart meter equipped to support SOAP-based OpenADR2 signals or SEP2 RESTful web services needs much more RAM, computing power and battery power compared to a meter that supports EN 13757-3 (the application layer protocol of M-Bus). Even more challenging is the use of web services over low-power and low-bandwidth radio links that are often used for remote meter readings [20]. In many

Table 1: SEP2 EXI/XML sample messages

Encoding type	Avg. comp. [%]	Avg. size [bytes]
XML (text)	100.0	462
EXI (schema)	7.89	40

cases, protocol gateways will be required to translate between low-level specialized protocols and SOA-based energy management communications. This will inevitably complicate the installations and make the maintenance of the system troublesome.

Nevertheless, some newly emerged ICT can be used to increase the efficiency of the web service implementations. Application protocol, called Constrained Application Protocol (CoAP), is currently under development by IETF and is specially designed for IoT deployments. It is a binary REST protocol that resembles HTTP but provides some features for embedded networked devices that go beyond the capabilities of HTTP e.g. multicast and event-driven asynchronous message passing. The advantages of CoAP over HTTP for IoT, including quantitative measurements of the gained efficiency, are presented by Colitti et al. [21].

Another important technology for embedded web services is efficient representation of the XML. A primary candidate for that is Efficient XML Interchange (EXI) specification - W3C recommendation since 2011. The studies performed by W3C EXI working group showed compression of up to 100 times as well as multifold processing speed increase compared to XML (depending on the encoding options and the document structure) [22, 23]. By using EXI, the energy management protocols will ensure interoperability with enterprise management systems while keeping the overhead of IoT deployments to a minimum.

For quantifying the EXI compression for a concrete protocol, a sample of 9 SEP2 messages typical for a smart thermostat were selected and converted to an EXI representation using the latest SEP2 XML schema definition.

As shown in Table 1, on average the EXI messages are only 7.89 % of their XML counterparts with an average size of 40 bytes. By using EXI serialization, the SEP2 messages are very likely to fit into a single 6LoWPAN packet which is an important requirement for battery powered sensor nodes.

Another IoT resource requirement is the programming memory of the EXI implementations. The programming memory is proportional to the complexity and size of the XML schema as its definitions are stored in the form of programming language constructs. We performed tests with two embedded EXI implementations available as open source: EXIP[1] and WS4D-uEXI[2]. The measurements of the programming memory size were done with a sample EXI decoder for SEP2 messages on a 32 bit Linux PC. The programming memory for the EXIP sample application was measured 2702 KB while for WS4D-uEXI the size was 392 KB. These results reveal the difference in the way the XML schema definitions were stored: EXIP uses regular grammars in the form of C structures while WS4D-uEXI uses finite state machines implemented with hierarchy of *switch* statements. In both cases there are methods that can be used to lower the size of the programming memory such as storing only the XML schema definitions that are used by the device, representing multiple read-only definitions with a single structure etc.

Introducing new protocols and data serialization formats will bring a new challenge to the Smart

[1]EXIP - http://exip.sourceforge.net/
[2]WS4D-uEXI - http://code.google.com/p/ws4d-uexi/

Grid implementations - namely the migration of legacy deployments in utilities and ISO management systems. On the other hand, the investments in this migration could be partly covered by the increased reliability and lower maintenance cost of the future Smart Grid.

6.2 Security

Keeping the Smart Grid secure has to do with many different aspects that cover the communication security but also the practical application of security policies by all the actors in the system [24]. In this section, we examine some of the key security challenges for the emerging energy management standards.

Ease of installation vs security

One of the biggest challenges facing devices in Smart Grid infrastructure, sensor networks and appliances is the need for proven security often with a very limited or non-existent user interface. In the past, default passwords or codes have been used (e.g. the ”0000” pin used in Bluetooth pairing) but these have severe limitations.

Most computing devices today have a high resolution graphical display and a keyboard or touchscreen and therefore entering a credential such as a username and password is easy. Moreover, it is often the user of the computing device who is being authenticated and subsequently authorized for, e.g. a banking transaction, and not the device itself.

Devices running in a HAN or Smart Grid network may need themselves to be authenticated and subsequently authorized. An example is an electric vehicle; it may be eligible for cheap electricity but it is important to know that the device which is plugged in and drawing a large current is in fact an electric vehicle and not for example a large battery which can then be used to power a home instead of taking power from the grid.

Public key cryptography can be used to install a device certificate in the device, which is signed in a Public Key Infrastructure (PKI). The device certificate identifier is given out-of-band to the utility which then sends it to the metering device. When the vehicle is plugged in, it sets up a secure session based on its device certificate and the metering device can then be sure the device is genuine. No username or password is needed in this case. This technique can be applied to all HAN devices that need validating in themselves, as opposed to user validation.

Other techniques for devices without any user interface exist. Near-field communications (NFC) is a rapidly-growing wireless technology which can transfer information wirelessly over a short distance. This makes it difficult to eavesdrop and therefore it can be used to configure devices securely with a master unit by simply holding them in close proximity. Similar approaches for registering security credentials are for example RFID tags, barcodes or QR codes.

Prominent security threats

The Smart Grid provides a large and varied attack space for attempted security breaches. Threats may come from many sources. Given the wide distribution of devices into homes, the most likely attacker will be the consumers themselves, trying to obtain cheaper bills. Smart Grid devices form part of critical infrastructure, therefore they will be a likely target for cyber terrorists who wish to cause major disruption by e.g. taking out electricity supplies. Attacks include eavesdropping whereby an attacker can understand confidential information, man-in-the-middle attacks whereby an attacker can manipulate data or masquerade as a legitimate device and injection attacks whereby an attacker can inject spurious data into the network to cause disruption.

The security protocols already mentioned should be used at different layers to prevent eavesdropping, man-in-the-middle and injection attacks. Firewalls and air gaps must exist in the networks to ensure that security domains are distinct and any traffic passing between the domains is very carefully policed. Access control must be in place to ensure only privileged users and devices have authorized access to data.

7 Discussion

A common trend among energy management protocols is the use of proven ICT for ensuring scalable and secure communication. By use of compression schemes such as 6LoWPAN, IPv6 is slowly getting wider acceptance on the network layer. The use of XML and XML Schema definition language for representing the data (e.g. smart meter readings, pricing information, load control) is another widely adopted approach. In order to achieve the resource efficiency required for IoT deployments, current efforts are focused on the use of compression schemes for the verbose XML messages. One promising solution for efficient representation and processing of XML data is the EXI binary format.

While having potential benefits, semantic descriptions based on machine processable ontology databases are not part of the energy management standards on the presentation layer. This might change in the future as the need for more automated and flexible energy management communication infrastructure becomes evident.

Web services is another technology that is underlying the emerging SEP2 and OpenADR2 standards. One promising technology for lowering the resource requirements on the application layer is the CoAP protocol that can be used to implement RESTful as well as SOAP-based web services.

The security aspects in energy management domain are very important side of all standardization efforts. The major challenge in this respect is the need for high level of security for devices with limited resources that have no user interface and are required to support cost-effective installation procedures. One way to lower the resource requirements for these devices is to use lighter cryptographic algorithms such as Elliptic Curve public-key cryptography and hardware security engines built into the chipsets. On the other hand, near-field communications and barcodes can be used to configure security credentials and simplify installations.

Acknowledgment

This work is partly funded by the European Commission and EU FP7 project IMC-AESOP. We thankfully acknowledge the constructive feedback of the anonymous reviewers.

References

[1] M. Albadi and E. El-Saadany, “A summary of demand response in electricity markets,” *Electric Power Systems Research*, vol. 78, no. 11, pp. 1989 – 1996, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378779608001272

[2] H. Farhangi, "The path of the smart grid," *Power and Energy Magazine, IEEE*, vol. 8, no. 1, pp. 18 –28, january-february 2010.

[3] R. Aringhieri and F. Malucelli, "Optimal operations management and network planning of a district heating system with a combined heat and power plant," *Annals of Operations Research*, vol. 120, pp. 173–199, 2003, 10.1023/A:1023334615090. [Online]. Available: http://dx.doi.org/10.1023/A:1023334615090

[4] "Buildings Energy Data Book," 2010, u.S. Department of Energy. [Online]. Available: http://buildingsdatabook.eren.doe.gov/docs/DataBooks/2010_BEDB.pdf

[5] J. Liu, X. Li, X. Chen, Y. Zhen, and L. Zeng, "Applications of internet of things on smart grid in china," in *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, feb. 2011, pp. 13 –17.

[6] G. Strbac, "Demand side management: Benefits and challenges," *Energy Policy*, vol. 36, no. 12, pp. 4419–4426, 2008.

[7] F. Bouhafs, M. Mackay, and M. Merabti, "Links to the future: Communication requirements and challenges in the smart grid," *Power and Energy Magazine, IEEE*, vol. 10, no. 1, pp. 24 –32, jan.-feb. 2012.

[8] J. Wang and V. Leung, "A survey of technical requirements and consumer application standards for IP-based smart grid AMI network," in *Information Networking (ICOIN), 2011 International Conference on*, jan. 2011, pp. 114 –119.

[9] J. Quittek, R. Winter, T. Dietz, B. Claise, and M. Chandramouli. (2012, July) Requirements for energy management. IETF. [Online]. Available: http://tools.ietf.org/html/draft-ietf-eman-requirements-07

[10] D. Forsberg, Y. Ohba, B. Patil, H. Tschofenig, and A. Yegin, *Protocol for Carrying Authentication for Network Access (PANA)*, IETF Std., 2008.

[11] M. Dworkin, "NIST SP 800-38C: Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality," NIST, Tech. Rep., 2007. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf

[12] *NIST FIPS-197: Advanced Encryption Standard*, NIST Std., 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[13] Z. Shelby, "Embedded web services," *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 52–57, 2010.

[14] R. DeBlasio and C. Tom, "Standards for the smart grid," in *Energy 2030 Conference, 2008. ENERGY 2008. IEEE*, nov. 2008, pp. 1 –7.

[15] *IEEE Guide for Smart Grid Interoperability of Energy Technology and Information Technology Operation with the Electric Power System (EPS), End-Use Applications, and Loads*, IEEE Std., 10 2011.

[16] *Energy Interoperation Version 1.0*, OASIS Std., Feb 2012. [Online]. Available: http://docs.oasis-open.org/energyinterop/ei/v1.0/

[17] R. Kyusakov, J. Eliasson, J. Delsing, J. van Deventer, and J. Gustafsson, "Integration of Wireless Sensor and Actuator Nodes With IT Infrastructure Using Service-Oriented Architecture," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 43–51, Feb 2013.

[18] A. Petrick and S. V. Ausdall. (2013, April) Smart Energy Profile 2.0. ZigBee Alliance, HomePlug Powerline Alliance. [Online]. Available: http://www.zigbee.org/Standards/ZigBeeSmartEnergy/Version20Documents.aspx

[19] J. Schneider, T. Kamiya, D. Peintner, and R. Kyusakov, *Efficient XML Interchange (EXI) Format 1.0*, W3C Std., February 2014. [Online]. Available: http://www.w3.org/TR/exi/

[20] C. Groba and S. Clarke, "Web services on embedded systems - a performance study," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, march 2010, pp. 726 –731.

[21] W. Colitti, K. Steenhaut, N. De Caro, B. Buta, and V. Dobrota, "Evaluation of constrained application protocol for wireless sensor networks," in *Local Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*, oct. 2011, pp. 1 –6.

[22] G. White, J. Kangasharju, D. Brutzman, and S. Williams, "Efficient XML Interchange Measurements Note," W3C, Tech. Rep., 2007. [Online]. Available: http://www.w3.org/TR/exi-measurements/

[23] C. Bournez, "Efficient XML Interchange Evaluation," W3C, Tech. Rep., April 2009. [Online]. Available: http://www.w3.org/TR/exi-evaluation/

[24] H. Khurana, M. Hadley, N. Lu, and D. Frincke, "Smart-grid security issues," *Security Privacy, IEEE*, vol. 8, no. 1, pp. 81 –85, jan.-feb. 2010.

Paper F

Enabling Cloud-connectivity for Mobile Internet of Things Applications

Authors:
Pablo Puñal Pereira, Jens Eliasson, Rumen Kyusakov, Jerker Delsing, Asma Raayatinezhad, and Mia Johansson

Reformatted version of paper originally published in:
Seventh International Symposium on Service-Oriented System Engineering

Enabling Cloud-connectivity for Mobile Internet of Things Applications

Pablo Puñal Pereira, Jens Eliasson, Rumen Kyusakov, Jerker Delsing, Asma Raayatinezhad, and Mia Johansson

Abstract: The number of small embedded devices connected to the Internet is increasing. This growth is mostly due to the large number of Internet of Things (IoT) deployments, with applications such as: industrial monitoring, home automation, and others. One common aspect with the majority of application areas is the lack of mobility. Most IoT devices are stationary and often use IEEE 802.15.4/6LoWPAN solutions. When a high level of mobility is required, the use of IEEE 802.15.4 is not possible without adding additional hardware for the user to carry.

In this article, a holistic network architecture consisting of heterogeneous devices is presented. The architecture is composed of Embedded Internet Systems (EIS) and uses standard communication protocols. One important feature is the use of the Service-oriented architecture (SOA) paradigm. The use of SOA, by utilization of the CoAP protocol and standard services, enables the proposed architecture to exchange sensor- and actuator data with an Internet-based cloud as well as a user's local cloud consisting of sensor IoT devices, smart phones and laptops. Another component of the architecture is a web-based human-machine interface for configuration, monitoring and visualization of sensor and actuator data using emerging web technologies for structured data processing.

Results from experiments and real-world tests show that the proposed architecture can support sample rates of up to several kHz while enabling sensor data to be transmitted to SOA services in real time. This proves that the use of SOA, and RESTful web services in particular, is feasible on resource-constrained platforms while supporting true mobility.

1 Introduction

In the last few years, mobile devices have become more and more pervasive, and by the growth of the technology, their processing speed has increased resulting in the ability to develop more advanced mobile applications. Billions of smart devices will soon be connected to the Internet to form the Internet of Things (IoT) [1]. In the last years, the number of devices using technologies with support for IP (Internet Protocol) has increased.

With the increase of the number of connected embedded devices it becomes particularly important to enable intuitive and simplistic human-machine interface that allows the users to interact with these devices. This interface must allow the deployment of various IoT services for enhancing the quality of life, safety and productivity of mobile users in a number of application areas. Moreover, the users should not be required to learn new interaction models for every device or vendor, hence the interface should be based on standards that are not proprietary and are widely adopted.

One application area requiring mobile and robust sensor solution is the security field, specifically the personal security field. Combining and analyzing data from sensors in wearable electronics, cameras or sound collectors with secure communication is essential to technical solutions where the individual's security is in focus. An example of such a solution is a prototype developed by the

Säktek cluster [2] of security and technology, where a combination of mobile sensor data is analyzed to create alarms that are transferred to operators using a secure mobile platform for further actions. A sophisticated network of devices ensures a highly mobile and robust solution which is essential for maintaining the individual's freedom while being monitored for specific purposes.

For mobile sensor applications, the following properties and requirements are important: devices must be of small size to not interfere with the user's daily life, communicate with standard protocols to allow maximum interpperability so that standard consumer devices can be used, and be low power so that the battery lifetime is extended. Example applications include: health care and elderly care [3], security (policemen, firemen etc.), industrial application, as shown by Karnouskos et al. [4], vehicle testing [5] and winter road conditioning [6]. Typical requirements for these types of applications ranges from a few data transmissions per hour (posture, position, temperature etc) to continuues data transmission with sample rates of several kHz (vibration, audio, CAN-bus, etc). Some of these applications could transmit sensitive information, therefore, the security is a really important requirement. In this paper, we discuss the security, but it is only part of the future work.

However, an often seen aspect is that proprietary communication protocols are used in many mobile applications. The use of non-standard communication protocols severely limits interoperability and makes it difficult to create new systems and services using existing hardware and software. A better solution would be to enhance the Internet of Things paradigm by having Internet-connected devices communicating using the Service-oriented architecture (SOA) approach. This would enable creation of Systems of systems and formation of a personal mobile sensor cloud where sensor- and actuator nodes can cooperate directly with human users and standard consumer devices, such as smart phones and tablets, as well as the global Internet.

The paper is outlined as follows: Section 2 presents related work followed by enabling technologies - Section 3, used for the proposed architecture. Sections 4 and 5 present the architecture for mobile sensor clouds and the experimental setup, respectively. The results are presented in Section 6. Section 7 outlines future work and the paper is concluded in Section 8.

2 Related Work

This section focuses on analyzing the state of the art of different technologies and research areas used in this paper.

Internet of Things

The Internet of Things (IoT) is a concept of communication between people and smart objects, e.g. mobile phones and sensors, and communication among Internet connected devices. A number of applications rely on enabling the objects in everyday living environment to communicate with each other in order to exchange the information they have collected from their surroundings. One of the valuable applications is in transportation to use the traffic information for traffic control purposes. Healthcare is another example where IoT technology can be applied as well as disaster alerting or recovery in work environments such as mines [7].

Mobile Cloud

Mobile phones have become a crucial part of people lives because of their computation and communication capabilities. While they suffer from deficiencies in performance (e.g. battery life, storage, and bandwidth) and in security (e.g. reliability and privacy) [8], they are programmable and come with a variety of embedded sensors, such as microphone, camera, GPS, digital compass, gyroscope, and accelerometer. This collection of sensors enables applications in wide range of domains, such as healthcare, social networking, transportation, environmental monitoring, business, safety [9]. The data obtained from the sensors can be sent over the Internet to *clouds* where the data storage and data processing are performed. This concept can also be used to provide more computational capabilities for resource constrained IoT devices. An example of that is the *thin server architecture* proposed by Kovatsch et al. [10] where even low-level firmware functionality is executing in the cloud. The *mobile cloud computing* helps overcome the obstacles and deficiencies of the mobile phones, and brings a broad range of new services and facilities [8].

Bluetooth PANs

Bluetooth is a specification for the use of low-power radio communications to connect wirelessly wide range of mobile electronic devices such as mobile phones, PCs, laptops, media players, tablets, ect. The distance range is not high, around 10 meters, but nevertheless this is sufficient to create Personal Area Networks (PANs). The relatively short range helps to increase the security as it is very difficult to eavesdrop the network.

Bluetooth is a mature and reliable technology with large number of development tools and extensive studies and information in the literature about Bluetooth and the PAN Profile [11].

Wearable Electronics

Novel techniques on miniaturization in the semiconductor's world and the recent investigations on new materials with electromagnetic properties have opened up new ways to understand the electronic design.

The wearable electronics with a low power design could be a powerful tool to measure data on a human body with a low level of body-intrusion. Already now, the current technology allows to create a pseudo-complex wired(less) sensor network.

One of the most important things in wearable electronics is to avoid extensive use of wires. This is the main motivation of studies in wearable antennas [12] that aim to increase the performance with some extra features like more flexibility, waterproofness etc.

When it comes to sending data to the users there are different approaches that can be used - if the application requires visualization of some data it is possible to use a display directly on the clothes [13] or use special glasses (like Google Glass). The screens on the clothes can get more technical failures, but are more comfortable and less intrusive than the glasses. Another recent studies are focused on haptic displays using dielectric elastomers to recreate surfaces directly on fingers [14].

The wearable sensors are an important tool to get data from the human body with the least level of intrusion. As an example, it is possible to take measurements of the level of oxygen in the blood, electrocardiograms, breath rate, temperature, blood pressure, skin humidity, level of stress etc.

3 Enabling Technologies

In this section, a number of enabling technologies are presented together with their characteristics.

3.1 Wireless communication

A mobile (sensor) network, or Personal Area Network (PAN), is usually based on general-purpose technologies and protocols, and has higher processing capabilities than an average IoT or WSN device. PAN devices are more often used on, or in the vicinity of, human users.

Today, two of the most widely used radio technologies for personal networks found in mobile devices are WiFi and Bluetooth. WiFi is becoming more and more low power, but still cannot compare with Bluetooth. Especially the relatively new standard of Bluetooth 4.0 with the extension of Bluetooth Low Energy (BLE). Bluetooth Low Energy uses only a fraction of the power compared to Bluetooth 2.0. Even though IEEE 802.15.4 with 6LoWPAN is widely used for IoT applications, the technology is better suited for stationary monitoring as todays mobile phones are not supporting it. Therefore, the only way of using IEEE 802.15.4 with IPv6 in PAN is by using an additional gateway. This approach is feasible in some applications, but should be avoided when minimum size and low cost is important. In some simple cases, each sensor node can feature its own GPRS modem for easy Internet-connectivity, but this is a very costly solution. By using Bluetooth, no separate gateway is needed, and the user is not forced to have several SIM cards.

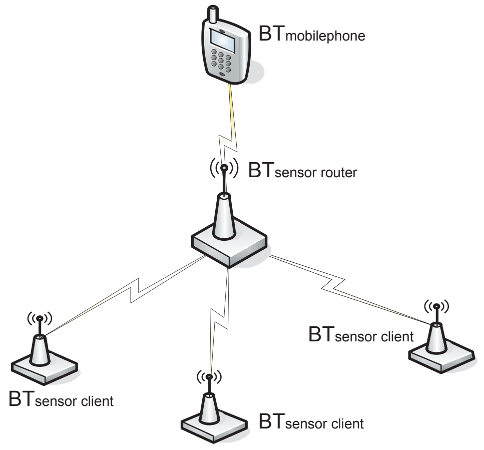


Figure 1: Bluetooth sensor network

Bluetooth-equipped networked sensor nodes can achieve good interoperability with consumer devices, have lower power consumption than WiFi, and have a lower cost. Bluetooth is also by far the most widespread technology supported by existing consumer devices, which further makes it an interesting technology to use for Personal Area (Sensor) Networks. A sensor network composed of Bluetooth-equipped EIS devices used in the context of sensor networks is called a Bluetooth Sensor Network (BSN) as shown in Figure 1. Regarding the communication security, Lindell et al. [15]

showed that Bluetooth 2.1 can provide a good level of security. By using Bluetooth and the standard PAN profile, virtually any Bluetooth-enabled mobile phone can act as a gateway to the Internet.

3.2 CoAP (Constrained Application Protocol)

The IETF Constrained Application Protocol is an application-layer protocol designed to provide web services working with constrained nodes - devices using microcontrollers with small amounts of ROM and RAM, running 6LoWPAN network stacks with high packet error rates etc. The protocol is designed for low-power networking allowing the nodes to switch to sleep mode to extend their battery life.

CoAP provides a request/response interaction model between application end-points, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs, RESTful interaction, extensible header options, ect. CoAP easily interfaces with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments. CoAP runs over UDP unlike HTTP.

Some features of CoAP are:

- Two types of request messages: Confirmable Message (CON) - the message is retransmitted (maximum four times) with an exponential timeout waiting for an Acknowledged Message (ACK) or the correct response from the server. The second type is the Non-Confirmable Message (NON) - the message is sent without any kind of response.
- The URI format allows the use of standard and specialized service endpoints. One such example is the resource discovery defined in RFC 5785 that uses the */.well-known/core* path and the CoRE Link Format.
- CoAP also allows to send very big messages with a stop-and-wait mechanism called ”blockwise transfers” (splitting messages).

3.3 Embedded Web Technologies

The user interactions in our platform are based on the web architecture. Key technologies to enable embedded RESTful web services, web linking and data representation are CoAP and Efficient XML Interchange (EXI) protocol for binary representation of XML structured information. The EXI format has W3C recommendation status [16] and is designed to increase the compactness and processing efficiency of XML data while keeping the compatibility with the XML Infoset.

By using EXI, XHTML visualization of the sensor data can be efficiently transmitted and processed and hence allowing the use of standard web technologies to interact with the sensor nodes through the user's mobile phone. The envisioned support in the mobile browsers of the IETF CoRE technologies such as CoAP, Observe, Blockwise transfers, CoRE Link Format; will enable the use of asynchronous RESTful client-server applications hosted on the sensor nodes.

4 Mobile Cloud IoT Platform

The architecture of the proposed IoT platform consists of the following components:

4.1 Network Connectivity for Mobile Devices

The devices part of the network architecture are Bluetooth-based sensor nodes and a standard mobile phone acting as an access point to the Internet. In our experiments we used the Mulle sensor platform shown in Figure 2 developed by Eistec AB [17]. One Mulle is used as an IP-router that is started by initiating a Bluetooth connection towards the mobile phone's access point service. When the router Mulle has established a Bluetooth PAN connection and acquired an IP-address using a DHCP client, it starts its own access point service. Other devices (Mulles, PDAs etc) can now connect to the router Mulle's PAN-NAP profile, using the PAN-U on the clients. When a client Mulle has an established connection, DHCP is used on both router and sensor Mulles in order to distribute IP addresses. The router Mulle also features a NAT (Network Address Translation) service, allowing up to seven sensors (clients) to share one Internet-connection at the same time. As a result, a sensor network consisting of one mobile phone, one router Mulle and up to seven client Mulles can be formed. Mulles either use the NTP protocol or CoAP-based time synchronization for security reasons and to correctly timestamp sensor data.

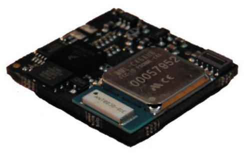


Figure 2: Mulle v3.2

The Mulle platform has low power consumption and its large number of I/Os make it a suitable component in mobile personal sensor networks. The Bluetooth version is capable of communicating with virtually any Bluetooth-enabled devices, e.g. computers, PDAs and mobile phones, using only standard Bluetooth protocols and profiles. The inherent support for TCP/IP, by the lwIP stack [18], enables the Mulle to transmit sensor data directly to the Internet without proprietary gateways or middleware services.

4.2 Application Services

The REST engine of our platform is based on a modified version of *libcoap* implementation first presented in [19]. The CoAP implementation covers Draft IEFT Core CoAP 11 [20] therefore the REST Engine provides functions to initialize and configure a RESTful Web service resources according to the layered model presented in Figure 3.

The REST engine offers three different types of RESTful resources:

- Resource: A basic REST resource should have an URI-path, allowed methods, and a string for the Web Linking information. For every resource, the application must provide a resource handler function, which receives the request and generates the corresponding response. Both messages are accessed through the REST Engine API. This resources could be masked/hidden depending on the client or even have different options.

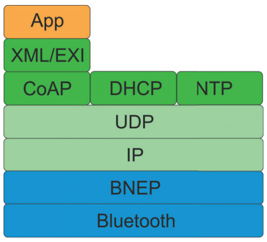


Figure 3: Mulle communication stack

- Event Resource: This type requires a second handler function to be implemented by the application developer. A client-defined event triggers this handler, and it generates a PUT message with the content that would like to change.
- Timed Resource: Additional to the signature of the basic resource, it is possible to define a time interval or an specific time/date when the REST Engine periodically calls a second handler function similar to the one for event.

4.3 Cloud Integration

Integration of server applications, mobile phones and sensors is based on the SOA approach. A server has the ability to communicate with mobile devices or furthermore connect to each sensor directly. Each sensor node, a Mulle v3.1, is connected to a mobile device and communicates with the server via the Internet provided by the mobile device (as shown in Figure 4). The server is composed of several services which can be integrated in mobile phones or can be distributed on cloud infrastructure. Additional services, such as data filtering, alarm management and others, can be implemented on the sensor nodes as well. Some services that are important for the proposed mobile platform are listed below:

1. Configuration service - it is used by all the devices to get a configuration parameters such as the IP and port addresses of all other services.
2. Time service - it is used to synchronize the devices.
3. Proxy service - facilitates the communication with the sensors and other mobile devices.
4. Filter service - the data is analyzed and filtered.
5. Alarm service - based on certain rules it warns the user for possible critical conditions through a SMS, email or other mechanisms.
6. Historian service - collects the data and stores it.

The server has the ability to communicate with the devices in different types of formats such as text, binary, XML, and EXI. The advantage of the text format is its convenience when reading, but it is hard to parse. Binary is efficient to communicate, but requires external tools to make it readable by users. XML is understandable and very well structured, but the size of its messages is big and it is much worse to parse compared to binary formats. EXI has smaller size and it is very efficient to process and easy to transfer.

The proposed SOA architecture can be deployed on three different network levels, or a combination of the three levels. Each level provides its own set of requirements, restrictions and performance.

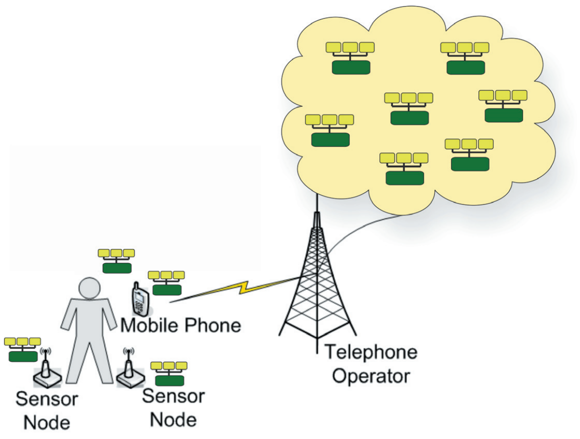


Figure 4: Mobile SOA-enabled IoT architecture

The levels are:

Node

The first level is the local mobile network of sensor- and actuator platforms. Deployed services on this level allows for maximum performance since no data need to be transmitted to the Internet and back just for two nodes to communicate. Instead services can be invoked with a minimum of overhead. Deploying services on node/network level will also allow the mobile cloud to function where no Internet connectivity exists, such as in very rural areas or in mines for example.

Network

The second level is to rely on a SOA-enabled gateway. This approach has the advantages that a mobile phone or a typical sensor network gateway [21] has drastically more processing power, memory and bandwidth. Disadvantages are that not any mobile phone can be used since the phone must support TCP/IP and SOA.

Internet

The third level is the global Internet. By deploying services on web servers or even data centers, all limitations of processing power are mitigated. However, more data must be sent and received by the sensor- and actuator nodes since much of the control logic has been moved out of the local mobile cloud. Integration of Internet services and sensor node is currently an important research field, see for example [22].

Since a user might want to be able to configure his or her network in an optimal way, a combination of the three levels might be desirable. By dynamically utilize services at different levels, an optimum on performance and power consumption can be achieved. For example, several sensors can cooperate using level 1 in order to detect an anomaly. When a deviation from the normal is detected, one or several services can be utilized on the gateway level, thereby allow services with relatively high memory and processing power to be utilized. Finally, when alarms are to be generated, or other systems and services could be invoked, the third level could be used to offload node and gateway devices.

4.4 Human Machine Interaction

Another component of the proposed IoT platform is a standard-based human machine interface that supports visual input and output to small embedded nodes.

The advancements of micro electronics in the last decades have led to lowering the cost and size of computing hardware to levels that allow the integration of embedded devices into everyday objects and activities a.k.a. ubiquitous computing. The first developments in this area can be traced back to the work of Mark Weiser in the late 80s and early 90s [23] who stressed the importance of intuitive interface between the humans and the computing devices. The web technologies have since enabled the fusion of digital and real worlds by providing a simple yet intuitive platform to interact with cloud services and virtual reality platforms. While the development of more advanced techniques to interact with IoT devices are available and will play important role in the future [24], our focus is on enabling the web technologies on IoT devices. By using the touch displays available in the current mobile phones, the mobile users can interact with the sensor and actuator devices in a intuitive and standard-based manner.

4.5 Communication Security

Our focus in this work is on analyzing the different security levels that could be implemented in our system while taking into account the energy consumption and performance of different solutions. Currently, we identified a set of possible technologies on different layers of the proposed design (see Table 1). As usual, a higher level of security requires more CPU cycles, and hence increased cost and latency, and a bigger packet size, which directly involves a greater expenditure of energy, and could be critical on mobile devices as this shortens the battery life of the devices.

Layer	Possible Security
Application	CoAP encryption
UDP	None
IP	IPsec
Bluetooth 2.0	SAFER+

Table 1: Security Layer Model

The level of security depends on two distinct types of solutions because there are two kinds of connections. In a Point-to-Point (P2P) connection the communication is among nodes of the same PAN and here it is possible to use the CoAP encryption and SAFER+ for Bluetooth. This connection is used also to connect all the nodes of a PAN to the router. In a Point-to-Internet (P2I) connection the communication is between the router of the PAN and the server and in this case it is possible to use the same security modes but there is another possibility i.e. IPsec. The security on the CoAP layer has 3 modes (Security section of IEFT CoAP 11 [20]):

- PreSharedKey: DTLS is enabled and there is a list of pre-shared keys and each key includes a list of which nodes it can be used to communicate. At the extreme there may be one key for each node that a particular CoAP node needs to communicate with (1:1 node/key ratio).
- DTLS is enabled and the device has a raw public key certificate that is validated using an out-of-band mechanism. The device also has an identity calculated from the public key and a list of identities of the nodes it can communicate with.
- Certificate: DTLS is enabled and the device has an asymmetric key pair with an X.509 certificate that binds it to its Authority Name and is signed by some common trust root. The device also has a list of root trust anchors that can be used for validating a certificate.

5 Experimental Setup

The constrained REST engine implementation used in this work is based on the IETF CoAP [20] draft. The UDP layer on Mulle sensor nodes is callback-based (RAW API [25]). This does not allow the use of sockets and the specific socket functions. During the programming design step, we focused on maximum performance of the code to get the maximum rate of packets per second as well as low-power features.

The Mulle software has a full set of control functions to manage the parameters of the UDP connections (with optimized power-consumption). In order to manage all the incoming and outgoing messages, the libcoap queuing mechanism for CoAP packets was extended to store the data of the connection including IP addresses, ports, number of retransmissions, etc.

The incentive for this design of the Mulle nodes software is to enable the sensor to communicate its data to a server. In this scenario the software has to be able to manage all the connections (low number if possible) and create new packets (take data from the sensors, analyze the data and compress several samples in a new packet). To do that, the sensor nodes have to use events (interrupts), timers and callbacks. Specifically for our use cases, any sensor is able to dispatch 6 CoAP packets (Non Confirmable Messages) every 250 ms and check the incoming queue for new messages.

As with all constrained embedded devices, the memory consumed is one of the biggest problem especially on devices with network connectivity where the connection state needs to be stored. In CoAP the maximum packet size is defined as 1.4 kB and all the incoming messages need to be saved in memory together with the confirmable messages. This can be a huge problem in some applications requiring the traffic to be routed through a busy proxy node. However, this is not a problem on the final sensor nodes because they must send data primarily with Non Confirmable Messages (NON). To prevent overflows during the packet saving step, the system should have a limit on the number of packets that can be saved at the same time on the Mulle's packet queues, defining two types of limits: an outgoing limit for Confirmable Messages (COM) and an incoming limit for the received messages. One of the powerful features of CoAP is the packet-retransmission of Confirmable Packets (see section 3.2), but this could create problems on the receiver because it might store the same packet more than one time. To prevent multiple allocations in memory of the same data the system has to check (only) the previous packet ID and save it only if this ID is not already in memory (it is very important do this step exactly when an incoming message is received, not when the content of the packet is analyzed).

In this system, one node can receive requests from a server or from another node. As the response generation could take a long time, it is required that separate responses are used to prevent unnecessary request retransmissions or even a request timeout. This method is based on sending an empty ACK when a packet is received and then, when the answer is ready, sending a CON message.

The communication between the server and client applications is facilitated by the use of services, as time services, log, security, etc. According to the idea of dynamic services, the clients are able to send a request to the server about a list of available services and use any of them at run-time.

The advantages of this design are the possibility to do updates (the system allows to add new services to the server without modifications on the clients) and the intuitive mapping of the service endpoints to the server resources by URLs.

6 Results

To evaluate our design, we created a test to measure the memory footprint and the quality of our design between a server and a Mulle sensor node. For this test we used a PC with Ubuntu 12.04 as a server running our version of CoAP server and another tools like echo programs and network analyzers. We used an Android 2.3.7 mobile phone to create the Bluetooth PAN which was connected to the Internet by a wireless LAN. It is important to note that in this process we did not use any additional software on the phone. The software of the Mulle nodes has been compiled with the m32c-elf-gcc (GCC) version 4.4.3. For the performance tests on the transmission and reception we measured the latency for 10.000 messages (samples) for different payloads between 0 and 256 bytes.

6.1 Memory Footprint

	Total Memory Usage(kB)
System Base	17.3
Reserved Memory for Packets	6.1
CoAP core	4.9
CoAP dispatcher	0.9
CoAP packet emission	0.1
CoAP packet reception	0.2

Table 2: Memory Usage by Functionalities

The memory footprint is mostly of interest on the Mulle devices in our mobile cloud design, because the other devices has much more resources that are needed. Table 2 shows a detailed memory footprint of our implementation of CoAP for Mulle sensor nodes.

The system requires 29.5 kB of RAM memory, the reserved memory for new incoming/outgoing packets is 6.1 kB, but it could be bigger if the applications requires it.

6.2 Transmission Performance

The nodes (Mulles) have been designed mainly to send data to the server or other systems, thereby the sending rate is a very important feature. Figure 5 shows the result of sending multiple messages with different size of the payload (10.000 packets per payload size), as expected the latency is increasing with the payload size.

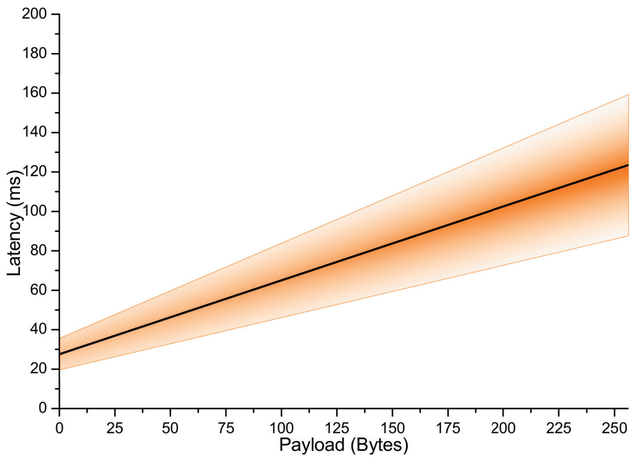


Figure 5: Test of Transmission

6.3 Reception Performance

Another interesting thing is the incoming packet rate. In this case the type of the incoming CoAP packet has to be ignored to disregard the time of package management. Figure 6 represents the time for CoAP packet reception (including the time for critical package testing) versus the payload of these packets. If the latency of the incoming and outgoing packets is compared, one can observe that the reception is much faster than transmission. This is a good feature that prevents packet loss during the communication between two identical systems. This asserts that the communication should be stable and should not create blockages.

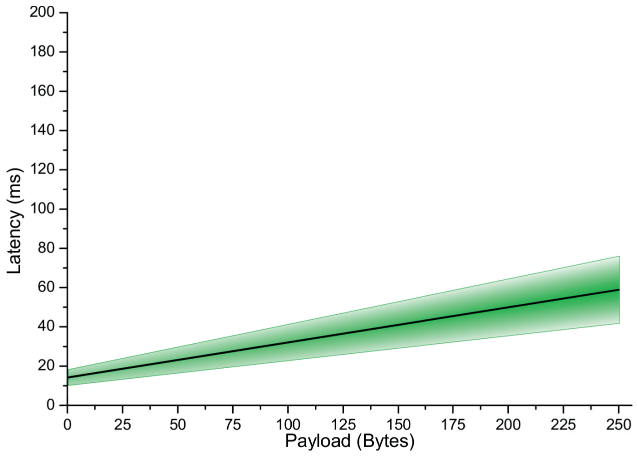
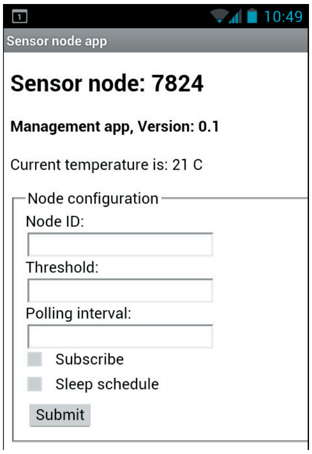


Figure 6: Test of Reception


10:49
Sensor node app

Sensor node: 7824

Management app, Version: 0.1

Current temperature is: 21 C

Node configuration

Node ID:

Threshold:

Polling interval:

☐ Subscribe

☐ Sleep schedule

Submit

Figure 7: Example XHTML that can be used to visualize and control sensor nodes

6.4 Configuration and Data Visualization

In order to validate our approach of visualizing and interacting with sensor and actuator devices by using standard web technologies we performed some preliminary tests. As discussed earlier our main enabling technology in this area is EXI that allows us to compress and process more efficiently XHTML input and output on embedded nodes with limited memory and computing capabilities. By using standard web technologies the human interactions follow a familiar and intuitive pattern already used in the world wide web and thus the learning curve for the users is not steep. Moreover, the use of embedded XHTML/EXI servers on the nodes eliminates the need of special software installed on the users' mobile phones - it is sufficient to have an Internet browser.

A typical use of our approach is to visualize sensor data and control the embedded nodes behavior. A sample XHTML that supports this functionality is given in Figure 7 rendered by smart phone browser.

The size of this XHTML when represented in text form is 1137 bytes. By using EXI representation the same document takes only 390 bytes - that is 34.3 % of the original size. Moreover, the processing efficiency is also improved and hence the requirements on RAM and CPU are less severe. While the suggested XHTML/EXI embedded server component has not been implemented in the present work, the achieved compactness suggests that in many scenarios the use of EXI-based web technologies will be beneficial. Future implementations can also use an optimized XML schema, instead of the generic XHTML schema used in this example. An optimized schema can better describe the concrete XHTML document to achieve even higher efficiency in terms of compactness and processing as compared to generic one.

7 Future Work

For some applications, such as process monitoring, health care, and vehicle testing, a reliable stream of sensor data is important. This is especially true for high-frequency data such as vibration data, audio signals, CAN-bus data, etc. In order to fully support these types of applications, the system should be enhanced with support for Quality of Service (QoS).

Another important feature is dynamic configuration and re-configuration of sensors. Using event-based communication with the support of an advanced rule-based scripting-language would enable users to create filters and rules for events that the sensors should detect.

Another important issue is security. It is a very important feature for data streams sent over public networks such as the Internet. In the proposed architecture security has two different levels; the first level is the local wireless network (using Bluetooth) and the second one is the Internet. Currently, the Mulle platform does not support security on Bluetooth, IP, or UDP layers except for simple PIN-code pairing, but as shown in Table 1, the possible security features of these layers are enough to get a good level of security on the complete system. Implementing security mechanism is however outside of the scope of this paper.

8 Conclusion

In this paper, a new architecture for mobile cloud sensor applications based on the Internet of Things approach is presented. The architecture consists of Bluetooth-based low-power sensor nodes communicating using standardized protocols and profiles. The use of a user's mobile phone and the mobile telephone access network allows true mobility. Furthermore, the use of CoAP and the Service-oriented architecture (SOA) paradigm allows a multitude of services and features to be accessible from the user's local sensor cloud as well as on the global Internet. In this architecture, services executing on sensor nodes, the user's mobile phone or pad, or even on the Internet, can collaborate in order to exchange data and perform distributed processing.

Performed experiments and tests show that the proposed solution is a viable solution for standards-based high performance sensor and actuator platforms. The proposed architecture can be used in applications that require a high level of mobility with requirements on small size, low power consumption yet the need to be able to communicate with standard consumer products and services. Two application areas; personal safety and security and vehicle testing, which both require a very high level of mobility is presented as use-cases, and the results are mapped toward these cases.

Acknowledgment

The authors would like to thank for their support the European Commission, and the partners of the EU FP7 projects IMC-AESOP (www.imc-aesop.eu) and I2Mine (http://www.i2mine.eu) for the fruitful discussions. We would also like to express our gratitude towards the CASTT project at Luleå University of Technology, and the Säktek organization.

References

[1] J.-P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP: The Next Internet.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010.

[2] "Säktek," http://www.saktek.eu/, Oct 2012.

[3] D. Hoang and L. Chen, "Mobile cloud for assistive healthcare (mocash)," in *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*, dec. 2010, pp. 325 –332.

[4] S. Karnouskos, A. W. Colombo, T. Bangemann, K. Manninen, R. Camp, M. Tilly, P. Stluka, F. Jammes, J. Delsing, and J. Eliasson, "A SOA-based architecture for empowering future collaborative cloud-basedindustrial automation," in *38th Annual Conference of the IEEE Industrial Electronics Society(IECON 2012), Montréal, Canada.*, 2012. [Online]. Available: http://diktio.dyndns.org/files/2012_IECON_architecture.pdf

[5] T. ElBatt, C. Saraydar, M. Ames, and T. Talty, "Potential for intra-vehicle wireless automotive sensor networks," in *Sarnoff Symposium, 2006 IEEE*, march 2006, pp. 1 –4.

[6] S. M. Casselgren, J. and J. Leblanc, "Model-based winter road classification," *International Journal of Vehicle Systems Modelling and Testing*, vol. 7, pp. 268–284, 2012.

[7] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2010.05.010

[8] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*. [Online]. Available: http://dx.doi.org/10.1002/wcm.1203

[9] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," *Comm. Mag.*, vol. 48, no. 9, pp. 140–150, Sep. 2010. [Online]. Available: http://dx.doi.org/10.1109/MCOM.2010.5560598

[10] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things," in *Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012), Palermo, Italy*, 2012.

[11] R. A. Rashid and R. Yusoff, "Bluetooth performance analysis in personal area network (pan)," 14 September 2006.

[12] S. Kim, M. M. Tentzeris, and S. Nikolaou, "Wearable biomonitoring monopole antennas using inkjet printed electromagnetic band gap structures," 2011.

[13] C. Zysset, N. Münzenrieder, T. Kinkeldei, and G. Tröster, "Woven active-matrix display," *IEEE TRANSACTIONS ON ELECTRON DEVICES*, vol. 59, no. 3, pp. 721 – 728, March 2012.

[14] I. Koo, K. Jung, J. Koo, and H. R. Choi, "Wearable fingertip tactile display," 18 October 2006.

[15] A. Y. Lindell, "Comparison-Based Key Exchange and the Security of the Numeric Comparison Mode in Bluetooth v2.1," in *Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, ser. CT-RSA '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 66–83.

[16] *Efficient XML Interchange (EXI) Format 1.0*, W3C Std., March 2011. [Online]. Available: http://www.w3.org/TR/2011/REC-exi-20110310/

[17] "Eistec AB," http://www.eistec.se/, Oct 2012.

[18] A. Dunkels, T. Voigt, and J. Alonso, "Making TCP/IP Viable for Wireless Sensor Networks," in *Proceedings of the First European Workshop on Wireless Sensor Networks (EWSN 2004), work-in-progress session*, Berlin, Germany, Jan. 2004. [Online]. Available: http://www.sics.se/~adam/ewsn2004.pdf

[19] K. Kuladinithi, O. Bergmann, T. Pötsch, M. Becker, and C. Görg, "Implementation of coap and its application in transport logistics," *Proc. IP+ SN, Chicago, IL, USA*, 2011.

[20] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, *draft-ietf-core-coap-11*, CoRE Working Group Internet-Draft, 16 July 2012.

[21] W. Wang, Y.-X. Zou, G. Shi, and Y. Zhu, "A web service based gateway architecture for wireless sensor networks," in *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*, vol. 02, feb. 2009, pp. 1160 –1163.

[22] F. Delicato, P. Pires, L. Pinnez, L. Fernando, and L. da Costa, "A flexible web service based architecture for wireless sensor networks," in *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, may 2003, pp. 730 – 735.

[23] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[24] M. Kranz, P. Holleis, and A. Schmidt, "Embedded interaction: Interacting with the internet of things," *Internet Computing, IEEE*, vol. 14, no. 2, pp. 46 –53, march-april 2010.

[25] A. Dunkels, "Full TCP/IP for 8 Bit Architectures," in *Proceedings of the First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys 2003)*, San Francisco, May 2003. [Online]. Available: http://dunkels.com/adam/mobisys2003.pdf

Paper G

EXIP: A Framework for Embedded Web Development

Authors:
Rumen Kyusakov, Pablo Puñal Pereira, Jens Eliasson, and Jerker Delsing

Reformatted version of paper originally published in:
ACM Transactions on the Web

EXIP: A Framework for Embedded Web Development

Rumen Kyusakov, Pablo Puñal Pereira, Jens Eliasson, and Jerker Delsing

Abstract: Developing and deploying Web applications on networked embedded devices is often seen as a way to reduce the development cost and time to market for new target platforms. However, the size of the messages and the processing requirements of today's Web protocols, such as HTTP and XML, are challenging for the most resource-constrained class of devices that could also benefit from Web connectivity.
New Web protocols using binary representations have been proposed for addressing this issue. Constrained Application Protocol (CoAP) reduces the bandwidth and processing requirements compared to HTTP while preserving the core concepts of the Web architecture. Similarly, Efficient XML Interchange (EXI) format has been standardized for reducing the size and processing time for XML structured information. Nevertheless, the adoption of these technologies is lagging behind due to lack of support from web browsers and current Web development toolkits.
Motivated by these problems, this article presents the design and implementation techniques for the EXIP framework for embedded Web development. The framework consists of a highly efficient EXI processor, a tool for EXI data binding based on templates, and a CoAP/EXI/XHTML Web page engine. A prototype implementation of the EXI processor is herein presented and evaluated. It can be applied to Web browsers or thin server platforms using XHTML and Web services for supporting human-machine interactions in the Internet of Things.
This article contains four major results: (1) theoretical and practical evaluation of the use of binary protocols for embedded Web programming; (2) a novel method for generation of EXI grammars based on XML Schema definitions; (3) an algorithm for grammar concatenation that produces normalized EXI grammars directly, and hence reduces the number of iterations during grammar generation; (4) an algorithm for efficient representation of possible deviations from the XML schema.

1 Introduction

Web technologies are rapidly expanding to networked embedded devices with studies showing that in 2013 there were more Web-connected gadgets than people in the U.S.[1] This process is expected to accelerate due to the increased IPv6 adoption rate and the availability of small-sized, cheap, off-the-shelf hardware that is powerful enough to execute full-featured network stacks. Already now, the number of TCP/IP connected sensor and actuator devices using low-power wireless technologies or even power-line communication is huge. The application areas cover home automation [1], energy management, and industrial process monitoring and control [2].

With the increase in the number of devices, the requirements on their interfaces are also higher. Consumers are demanding "smart" gadgets that are easy and intuitive to deploy, configure, interact with, and integrate with other devices and systems. An example from the home automation domain is a smart thermostat that can communicate with the user's smart phone to display the current temperature in the house along with energy costs as well as control settings. It is becoming more common

[1]According to data from research firm NPD Group

to equip the traditionally simple sensor and actuator devices with additional diagnostics, logging, and security capabilities. This phenomenon leads to developing more complex embedded applications, which are often required to support Web connectivity for human-machine interfacing. As the code base increases, so are the product cost and time-to-market for new devices. The development and support for different hardware platforms becomes especially challenging, and thus the need for a common development platform based on established and globally adopted standards. The Web development has proved successful in leveraging a set of global standards for unification of the development for front-end tools and applications over a large number of desktop and mobile platforms. In addition ICT research, as argued by Borriello et al. [3], suggests that embedded computing will also benefit from Web development platforms.

The trade-off between Web and native applications has been a turning point for development strategies in the mobile market. As discussed by Charland and Leroux [4], Web applications are cheaper to build, deploy, and maintain, but are often lagging behind in performance and user experience when compared to the native apps. This gap is narrowing, thanks to HTML5 and new Web toolkits such as Argos [5] which provides direct access to devices' capabilities from JavaScript code. While the app stores made the management of native applications much easier and user-friendly, their main drawback remains - supporting different platforms often requires substantial rebuild of the code base that needs to be kept up-to-date with new versions of the different operating systems. As Charland et al. conclude, one size does not fit all, and there are use cases when it is better to use one or the other approach. While there are a number of differences between the smart phone and the embedded systems segments, it is possible to draw some similarities and list a number of applications where building Web applications is more beneficial even for resource-constrained hosts when compared to developing proprietary solutions. The simplified use case presented in Section 6, which demonstrates a human-machine interface with a sensor platform, provides an example of such application. In this scenario, the user interface is implemented as dynamic Web application based on CoAP/EXI/XHTML and using the EXIP framework.

The approach of using standard binary protocols for enabling Web connectivity for constrained hosts differs from the most common methods described in the literature. The state-of-the-art solutions to the problems of embedded Web development (e.g., memory, network, and processing constraints) can be classified into two groups. The methods in the first group rely on powerful gateway devices that translate the standard Web protocols to some lightweight messaging framework, and vice versa. An example of this approach is the work by Trifa et al. [6], which describes a gateway architecture for providing Web connectivity to highly resource-constrained nodes. The methods in the second group focus on implementing efficient and stripped-down version of the standard text-based Web protocols. High-impact research results based on this method are the techniques for implementing an efficient HTTP server for embedded devices presented by Kang et al. [7] and Duquennoy et al. [8], as well as the small-footprint XML Web service implementation by Van Engelen [9].

Using text-based protocols that rely on simple character encoding such as ASCII, was important requirement in the early days of distributed computing systems. During that time, the ability to debug the interactions between the systems with one's bare hands was crucial to the acceleration of the adoption of the protocols. Nowadays, practically all text editors and development tools support UTF-8 character encoding. The tools also parse the XML documents before printing them to the screen to support syntax highlighting. Proper tool support opens up new possibilities for efficient representation of the information on the wire. The new binary encoding schemes are transparent for the user - if, in any case, the XML documents are parsed before printing them, then it is better to use faster, binary encoding which is easier to process than text-based representation. However, implementing highly optimized binary coding schemes is much more challenging than processing text-based streams. Even more challenging, is the use of such binary processors on resource-constrained embedded devices where the memory footprint and CPU usage are crucial. As an example, a common way to compress the size of an XML document is by indexing frequently used tags and value items. Instead of encoding each occurrence in the stream, the repeated information items are represented by their index. Using more extensive indexing increases the compression, but also makes the memory footprint required to store the indexed information larger. Providing efficient methods to build and store the indexes is just one example of optimization that is needed for running binary encoding schemes on embedded hosts.

In this work, we present design and implementation strategies for running an Efficient XML Interchange processor on embedded devices for enabling Web connectivity through RESTful interface that is based on Constrained Application Protocol. The RESTful interface can be used for human-machine interactions with Internet of Things hosts as well as for implementing embedded distributed systems based on the Service Oriented Architecture, as discussed by Shelby [10].

Unlike XML, the EXI specification mandates the use of schema-specific parsing [11] when the EXI document is encoded with schema knowledge i.e. using schema mode. In order to address all possible use cases, the presented EXI processor supports both the schema and schema-less modes of operation. This is achieved by using dynamic state machine abstraction that can evolve through addition of new states and state transitions. The main benefit of using static state machines, as in the EIGEN [12] and libEXI [13] libraries, is the small footprint and hence the ability to implement highly optimized, dedicated EXI processors. In order to efficiently support a static mode of operation - in other words, strict schema processing with no deviations, the EXIP library needs to be configured to strip the code responsible for evolving the state machines. This can be done easily during compile time due to EXIP modular architecture.

One important component of EXI implementations supporting schema-enabled processing is the automatic generation of the state machines based on XML schema language definitions. These definitions are used to construct a set of formal grammars that describe a particular XML language which is then recognized by the generated state machines. EXIP includes an optimized and lightweight grammar generation utility that can be executed efficiently at run time. This allows it to support dynamic XML schema negotiations even on embedded hosts. The main contributions of this article are the grammar generation algorithms that are the core of the high performance of this utility. To the best of our knowledge, all other EXI implementations use an external library for processing the XML Schema definitions that are used for the grammar extraction. A commonly used external XML Schema library is Apache Xerces. However, its usage for embedded Web development is limited to static compile-time generation of the EXI state machines.

A prominent research work that is based on the approach of compile-time generation of the state machines is presented by Käbisch et al. [14]. The authors show that the use of EXI for embedded Web service development brings substantial benefits in hardware utilization (network, CPU, RAM and programming memory). Moreover, their work includes the design of a Web service code generator based on Simple Object Access Protocol (SOAP) and the HTTP/EXI/SOAP protocol stack. Promising future research work, as stated by the authors of that study, is to add support for CoAP RESTful Web service interface to the proposed generator. As such, the EXIP framework described herein is extending and further specifying the suggested CoAP RESTful Web service generator.

EXI is not the only possible data format that can meet the requirements of the embedded Web programming, but it has been shown to provide the highest efficiency compared to rival binary XML solutions [15]. Lightweight text formats such as JSON and Comma-separated values (CSV) or binary encoding schemes (ASN.1, BSON, Protocol Buffers, Thrift etc.) are also capable of representing very efficiently structured information. However, the lack of formally defined mapping between these technologies and the XML Information Set [16] makes them unable to guarantee interoperability with existing Web technologies and protocols such as XHTML, Scalable Vector Graphics (SVG), Extensible Messaging and Presence Protocol (XMPP), and RSS feeds to name a few.

The initial goal of the EXIP library was only to provide efficient implementation of an EXI processor for embedded systems. Since the initial version of the prototype EXI processor, the EXIP library was used in a number of research projects and prototypes as in [17] and [18]. Based on the recurring need of higher processing efficiency and Web integration, the scope of the EXIP project has now extended, and new processing algorithms are employed. In addition to the grammar generation algorithms that are part of the EXI processor prototype implementation, this work defines the overall architecture of the EXIP Web development toolkit. The architecture consists of three main modules: the EXI processor library, EXI data binding, and the CoAP/EXI/XHTML Web page engine. Their functionality, required properties, and overall design in the context of embedded Web development are discussed in Section 2. Detailed descriptions of each of these modules and the associated research questions that are investigated are presented in Sections 3, 5, and 6, respectively.

2 Background

Optimizing the hardware utilization by the Web protocols is a key requirement for their application on embedded platforms. Very often the connected devices have limited memory (both RAM and programming memory), and use low-cost CPUs. If the device is battery powered, the communication overhead is a main contributor to the power consumption that needs to be carefully modeled in order to guarantee the intended up-time periods. Simulation tools such as PowerTOSSIM [19] can be employed to highlight areas of the protocol implementations that are mostly responsible for draining the battery. Among the use of radio duty cycling and CPU sleep modes, reducing the number of packets sent and received is another way of cutting the power consumption, especially in wireless applications.

W3C performed an extensive evaluation of the EXI format [20, 15] that shows substantial improvements in compactness compared to text encoding as well as other XML binary formats. Additionally, EXI has superior processing performance compared to plain XML. Both the compactness and processing efficiency depend heavily on the structure of the encoded documents and the options used for processing. For example, the use of XML schema information during encoding and decoding can cut the size of small documents more than 50 %, as the element and attribute qualified names are encoded as indexes instead of strings. This allows for substantial reduction of the number of packets required for communication of structured information over the network, and thereby minimizes the power consumption. Existing Web technologies that are formally described using XML schema language such as XHTML, for example, can then be efficiently represented for use in embedded applications.

Compaction and processing improvements of CoAP compared to HTTP are also significant, as reported by Kuladinithi et al. [21]. Moreover, the asynchronous design of the CoAP protocol makes it much more suitable for event-driven interactions. Publish/subscribe protocols are often preferred in embedded systems, as they provide better hardware and network utilization compared to polling schemes that are used by HTTP, for example.

Figure 1 provides an overview of the state of the art of embedded Web development along with a high level architectural view of the use of binary protocols for network communication and infor-

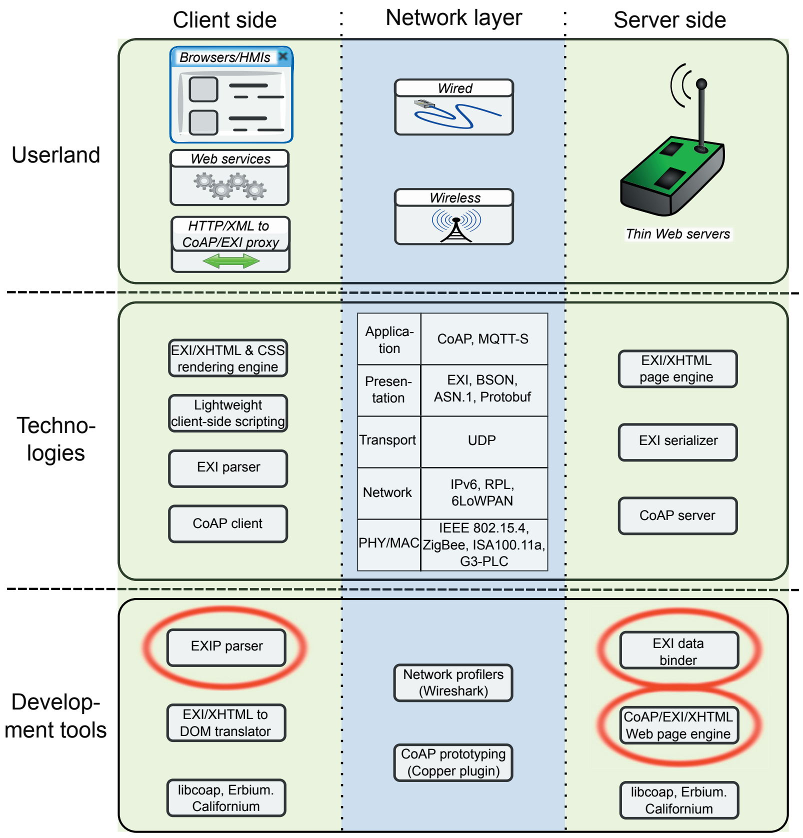

Client side
Network layer
Server side
Userland
Browsers/HMIs
Web services
HTTP/XML to CoAP/EXI proxy
Wired
Wireless
Thin Web servers
Techno-logies
EXI/XHTML & CSS rendering engine
Lightweight client-side scripting
EXI parser
CoAP client
Applica-tion
CoAP, MQTT-S
Presen-tation
EXI, BSON, ASN.1, Protobuf
Transport
UDP
Network
IPv6, RPL, 6LoWPAN
PHY/MAC
IEEE 802.15.4, ZigBee, ISA100.11a, G3-PLC
EXI/XHTML page engine
EXI serializer
CoAP server
Develop-ment tools
EXIP parser
EXI/XHTML to DOM translator
libcoap, Erbium. Californium
Network profilers (Wireshark)
CoAP prototyping (Copper plugin)
EXI data binder
CoAP/EXI/XHTML Web page engine
libcoap, Erbium. Californium

Figure 1: Overview of the tools and technologies for embedded Web development that are based on standard binary protocols. The tools that are the main focus of this work are marked with red ellipses.

mation exchange. The suggested components of the architecture are grouped depending on their role - client-side, networking layer, and server-side execution; and their application domain - user tools and applications, technologies/protocols/specifications, and development tools. The goal of this categorization is to show how the work presented in this article relates to the current technologies and applications, and to further motivate the need for this research.

As shown in Figure 1, client-side user applications of the embedded Web include browsers, graphical Web clients (HMI devices), embedded Web services, and proxy devices translating the binary Web protocols to their text-based counterparts. The technologies to implement these client-side user applications are CoAP client, EXI parser, lightweight client-side scripting engine, and EXI/XHTML/CSS rendering engine. The concrete development tools that can be used for implementing these technologies are the EXIP parser library, which is the primary objective of this work, EXI/XHTML to DOM translator, and CoAP libraries such as libcoap [21], Erbium [22] and Californium [23].

Similarly, the networking layer shows different wired and wireless network stacks and protocols grouped according to the OSI model along with developing tools used for debugging.

The server-side is represented by resource-constrained embedded devices that are conforming to the *thin server architecture* suggested by Kovatsch et al. [24]. The server technologies include CoAP server, EXI serializer, and EXI/XHTML page engine. The proposed development server-side tools are the EXI data binder and CoAP/EXI/XHTML Web page engine that are described in detail in Sections 5 and 6 of this work. Other server-side tools for embedded Web development are again, the CoAP libraries libcoap, Erbium, and Californium.

2.1 EXI

EXI data format significantly reduces the size of XML when stored on disk or transferred over the network and also speeds up the parsing and serialization. According to [15] the compression level varies between 1 % of the original size for large and sparse documents with compression and schema options enabled to 95 % for schema-less encoding of very small and dense documents. Nevertheless, EXI format has few drawbacks that are inherited from XML and must be taken into account in the discussions that follow in this article. XML notation and semantics are perceived as complex both for humans to understand but also for machines to process which stems from the design goal of the format to be flexible and easily extendable for application in variety of use cases. This flexibility creates a lot of special cases and exceptions that must be specifically handled with if-then-else statements during serialization and parsing. While EXI is very efficient in removing the redundancy in the XML syntax, it does not simplify the processing - it merely speeds it up. Besides, EXI adds another level of flexibility by introducing encoding options that can be used to influence the level of compression, processing speed and RAM usage during parsing and serialization. Providing support for all possible EXI options requires large and complex code base that can hardly fit into the programming memory of a highly resource constrained embedded device. Therefore, the application of EXI on such platforms often requires defining a profile of the EXI specification which restricts the supported EXI options to particular values and predefines the XML Schema. Different EXI profiles and how they are supported by EXIP are further discussed in Section 3.

Selecting the values for the EXI options is often a trade-off between memory usage, processing speed and level of compression (for example when setting the values of valuePartitionCapacity, valueMaxLength and compression options). Furthermore, as these parameters heavily depend on the structure of the documents and even on the schema design (as shown by [20, 15]) it is difficult to predict the level of efficiency when applying EXI on a particular set of XML documents without performing an extensive empirical study.

EXI theoretical foundations

The goal of this section is to provide the necessary background information for supporting the discussions on the EXI processor architecture and algorithms for embedded processing that follow without going into details of the inner workings of the EXI specifications. For in-depth overview of the EXI format, the reader is advised to refer to the W3C specification [25] and white paper [26].

An EXI stream is a sequence of events that describe the content of the XML document. These events are analogous to the streaming XML events and denote the start of an element or attribute, value items, closing tags and so on. For achieving higher compactness, the events are represented by a simplified Huffman coding [27] scheme. The occurrence of each event in the EXI stream is controlled/described by a set of formal grammars. The EXI specification very broadly identifies the formal grammars used as being in restricted Greibach normal form [28]. Support for the theoretical fitness of the discussed grammar generation algorithms is given in the next paragraph. It provides more concrete classification of the EXI grammars.

Unlike Greibach grammars, the EXI grammars have at most one non-terminal symbol on the right-hand side of the grammar productions. Therefore, all EXI grammar rules are in one of the following two forms: 1) $Z \rightarrow aY$ or 2) $Z \rightarrow a$,where Z and Y are intermediate (non-terminal) symbols and a is a terminal symbol. As all grammar rules are in one of these two forms, the EXI grammars are also *regular* and in particular *right linear grammars* as they require exactly one terminal on the right-hand side and at most one non-terminal which is at the end of the grammar rule. The regular grammars are strict subset of the context-free grammars according to the Chomsky hierarchy, and as every context-free grammar can be represented in Greibach normal form [28], they are also a subset of the Greibach grammars.

Identifying the EXI grammars as regular grammars provides much more insight into their properties. For example, context-free grammars define very broad class of languages and are equivalent to pushdown automaton (PDA), while regular grammars are equivalent to nondeterministic finite automaton (NFA). Moreover, the EXI grammars are *simple* a.k.a. *s-grammars* [29] as each pair $Z \rightarrow a...$ appears only once in each EXI grammar. Based on this constraint, the EXI grammars are also *unambiguous* and support linear parsing time by deterministic finite automaton (DFA).

The process of converting a set of XML Schema definitions to EXI grammars includes four steps:

1. Create a set of proto-grammars that describe the content model according to the schema. The EXI proto-grammars are strictly context-free grammars that are neither regular nor in Greibach normal form as they allow unit productions: $Z \rightarrow Y$ where both Z and Y are intermediate (non-terminal) symbols.
2. Normalize the proto-grammars to EXI grammars. The normalization includes simplification of the proto-grammars by removal of the unit productions. This creates regular grammar that can be *ambiguous*, in other words, lacking unique leftmost derivation tree for every input. In this case a second simplification is performed in which the *ambiguous* regular grammars are transformed to unambiguous *s-grammars*.
3. Assign event codes to grammar productions
4. Extend the EXI grammar with additional productions that describe the possible deviations from the XML Schema

Section 3.3 describes an extension to the algorithm for creating proto-grammars from schema definitions [step (1)] that guarantees that the resulting grammars are regular *s-grammars*. This allows for avoiding the normalization of the proto-grammars as a separate second-step process.

Section 3.4 describes a modified version of the algorithm for augmenting the EXI grammars for handling schema deviations [step (4)]. The new version of the algorithm allows the removal of redundant grammar productions that are otherwise required by the approach described in the EXI specification.

Related work for XML grammars

The formal grammars used in the EXI specification express the constraints defined in the XML Information Set [16] and are not specific to EXI format itself. As such, the formal models and theoretical results developed for XML are also valid for EXI. There are two main theoretical models for studying the properties of XML languages and XML schema languages. The first model treats XML instances as strings and schema languages as formal languages that define particular sets of strings representing the possible XML instances that are valid according to a certain schema. This model is based on context-free (word) grammars and their more restricted forms such as parenthesis and balanced grammars as presented by Berstel and Boasson [30].

In the second model, the XML instances are treated as trees and the schema languages as formal languages defining sets of trees representing the valid instances according to a certain schema [31, 32]. The nested structure of the XML forms ordered unranked trees i.e. trees with nodes allowed to have any number of ordered child nodes. The theoretical foundation of this model are regular tree grammars which can be seen as a generalization of regular word grammars. The tree model is appropriate when studying the expressive power of different XML schema languages as shown by Murata et al. [33]. In this work, Murata et al. present a formal classification and comparison between DTD, W3C XML Schema, and RELAX NG based on the regular tree grammar theory.

Context-free word languages and regular tree languages are closely related. For example, it is proven that the set of derivation trees for a language defined by a context-free word grammar forms a regular tree language [34]. In addition, Brüggemann-Klein et al. show that tree grammars, and even more generally hedge grammars, are effecively identical to balanced grammars and that balanced languages are identical to regular tree languages, modulo encoding [35]. These results demonstrate that the two models are equally expressive and can be used interchangeably when studying or characterizing languages based on XML Information Set.

The discussions in this paper are following the first model, because the EXI specification defines the XML content with a set of regular word grammars as already presented in Section 2.1. For that reason, all grammars in this work are assumed to be *word* grammars even if not explicitly stated.

Instead of defining the terminal alphabet in terms of ASCII or UTF-8 characters, which is commonly used in word grammars, the EXI grammars use XML events (start element, attribute definition, end element etc.) as terminal symbols. This provides high level description of the XML content model without affecting the theoretical results developed for regular grammars. As XML Information Set defines context-free language parsed by pushdown automaton, a single regular grammar (a single DFA) is, in general, unable to represent (parse) the content of a whole XML document. Using a single regular grammar (or a single DFA) for describing (parsing) the whole content of an XML is possible when certain restrictions on the document structure are met by the XML/EXI instances. For example, this approach is used for efficient processing of SOAP Web services that are *ordered* XML documents with predefined schema [36]. A less restrictive form of schema-specific XML parsing that uses an extended version of PDA is presented by Chiu and Lu [11]. Unlike these approaches, the EXI specification defines the parsing and serialization of XML Information Set documents based on a stack of regular grammars. Each regular grammar in the stack describes the content of particular XML element. The stack of grammars is used to model the nesting of elements (e.g. parsing a nested element equals adding its regular grammar on the stack) similarly to the role of the stack in the PDA.

For illustrating how the grammar stack is used during processing in EXI it is convenient to represent the XML Information Set in terms of extended context-free grammars (ECFG) which describe exactly the context-free languages and are the basis for DTD schema language [37]. In an extended context-free grammar each right-hand side of a production consists of a regular expression which is in turn equivalent to regular grammar or finite automaton. Consider the example XML instance and its corresponding ECFG shown in Table 1. The set of regular grammars, used during processing of EXI documents, corresponds to the set of regular expressions in ECFG which describe the content of all possible elements. At every step the EXI processor uses the regular grammars on top of the grammar stack to process the content of the current element. Starting of a nested element involves pushing its

Table 1: Extended context-free grammar for a sample XML instance where element <notebook> can have zero or more <note> elements with optional <subject> and mandatory <body>. The following operators are used in the regular expressions in ECFG: **.** - denotes concatenation, ***** - Kleene star operator (zero or more occurrences), **?** - zero or 1 occurrence and **[ ]** - matches a single character from the specified set within the brackets. The non-terminal symbols are in uppercase letters.

Sample XML	Corresponding ECFG
<notebook>	
<note>	NOTEBOOK → <notebook>.(NOTE)*.</notebook>
<subject>Sample</subject>	NOTE → <note>.(SUBJECT)?.BODY.</note>
<body>XML Instance</body>	SUBJECT → <subject>.[UTF-8 characters]*.</subject>
</note>	BODY → <body>.[UTF-8 characters]*.</body>
</notebook>	

grammar to the stack and closing an element pops its grammar from the stack. In this way, parsing the XML document shown in Table 1 involves: (1) parse the content of <notebook> element according to the regular grammar for that element which is initially the only grammar in the stack; (2) the start of the nested <note> element requires pushing its regular grammar on the stack and parsing its content according to that grammar; (3) on start of the nested <subject> element its grammar is pushed to the stack and used for parsing; (4) When all the content of <subject> element is parsed and there are no more nested elements at this level pop its grammar from the stack and continue processing according to the <note> grammar that is currently on the top of the stack; (...) the same procedure is repeated for the rest of the elements in this example.

Unlike DTD which defines a *local* language, the language defined by the set of regular grammars in EXI is a *single-type* language that corresponds to the expressive power of W3C XML Schema [33]. This essentially means that two or more elements sharing the same name but having different types are evaluated using different regular grammars that match their type. This differs from DTD where the name of an element uniquely identifies its content model (or, equivalently, the regular expression or the regular grammar of its content).

2.2 CoAP

The Constrained Application Protocol [38] is specially designed for use with resource-constrained hosts over low-bandwidth network links. CoAP functionality resembles the HTTP request/response interaction model, and is based on the Representational State Transfer (REST) architecture of the Web [39]. CoAP also supports well established concepts of the Web such as URIs and Internet media types. This allows for transparent translation between CoAP and HTTP traffic while enabling Web interactions with embedded systems.

CoAP fulfills the requirements of the embedded domain such as providing support for asynchronous message exchange, multicast capabilities, lightweight discovery mechanism, very low overhead, and implementation simplicity. This is possible by using UDP as a transport protocol with optional reliable unicast support and Datagram Transport Layer Security (DTLS) instead of TCP and TLS. The use of UDP enables the implementation of CoAP lightweight publish-subscribe mechanism [40] supporting dynamic content exchange between embedded servers and Web clients. The built-in asynchronous exchange of events encoded with EXI provides features similar to the AJAX frame-

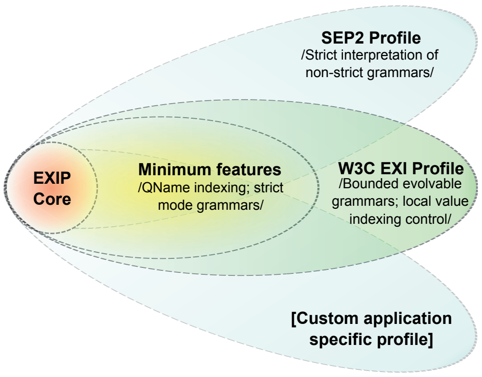

SEP2 Profile
/Strict interpretation of non-strict grammars/
EXIP Core
Minimum features
/QName indexing; strict mode grammars/
W3C EXI Profile
/Bounded evolvable grammars; local value indexing control/
[Custom application specific profile]

Figure 2: EXIP modular architecture and application profiles

work, but with much lower cost in terms of network bandwidth and hardware requirements for the hosts.

Application areas that would greatly benefit from an open and standard way to connect embedded hosts to the Web include various Internet of Things and machine-to-machine (M2M) applications such as home automation and energy management.

3 EXI Processor Design and Implementation

Deploying EXI-based RESTful Web services on resource-constrained hosts requires a modular implementation of the EXI processor library that can support different compile-time configurations depending on the application scenario. For example, some target platforms can make use of hash tables for fast lookups in the string tables, while others have too little RAM for that. In other cases, certain EXI options (e.g., compression, random access, etc.) are not allowed, and hence the code for processing them can be pruned from the library.

In this section, we present the modular design of the EXIP library [41] that enables compile-time profiling of the code base. As shown in Figure 2, by using fine-grained components that have low interdependencies, it is possible to define different profiles of the library that support a variety of use cases. Such profiles can be application-specific (e.g., full-featured, most-restricted, etc.), or defined as part of different communication standards - EXI Profile for limiting usage of dynamic memory [42], Vehicle to grid communication interface (ISO 15118), or other energy management standards [43] such as Smart Energy Profile 2.0 [44], and OpenADR, for example.

The encapsulation of the components' source code is done with the standard mechanisms available in the C programming language - splitting the code into different header and source files, and hiding the implementation in static functions, strictly avoiding the use of global variables and, where needed, using conditional C preprocessor macros. This enables the implementation of a simple and easy-to-maintain *Makefile* build system which can track the dependencies between the components. With this build system in place the developers can cherry-pick only the components that are needed during compile time which allows using the EXI Processor library for different application profiles or contexts.

3.1 Problem Formulation

The first step in supporting the requirements of the EXI-based embedded Web programming is to provide efficient Application Programming Interface (API) to encode and decode EXI streams. Already established XML APIs such SAX, DOM, and StAX are widely used in Java processors, but are shown to provide less than optimal efficiency for resource-constrained devices [45]. Other requirements of the EXI processor implementation include a small footprint and an easy-to-use code base that executes quickly, and consumes as little RAM as possible while being portable across a wide range of embedded platforms. Although the main goal of the EXIP library directly follows from these requirements, detailed description and evaluation of the degree to which these requirements are met is out of the scope of this paper. The reason for excluding these discussions is the low research value of the implementation technicalities that are involved in writing efficient and portable C code, a subject which is better presented by the EXIP library developers' documentation[2]. Instead, this section is focused solely on the grammar generation functionality that is an essential part of a number of use cases connected to dynamic/runtime exchange of schema information. The need for such runtime negotiation of the document structure is evident in supporting versioning of the schema documents and implementing generic Web services such as information logging and archiving, data visualization of uncategorized information, dynamic Web service composition, and peer-to-peer services. A concrete example where the XML Schema documents are processed during runtime to generate EXI grammars is the specification draft for using EXI over Extensible Messaging and Presence Protocol (XMPP) [46].

The dynamic processing of XML schema information can also be employed in cases where no schema information is available to describe a particular set of XML documents. In such cases the XML schema can be inferred from the set of available XML examples, and used to enable more compact EXI encoding. Both the schema inference and the generation of EXI grammars can be done at runtime. Example approaches for schema inference include learning of deterministic regular expressions [47], as well as learning chain regular expressions, in the case of the Trang open source software library [48].

3.2 Efficient EXI Grammar Generation

The standard way of generating EXI grammars from XML Schema is to rely on a generic XML Schema parser/validator such as the Apache Xerces library. The role of the XML Schema parser is to load the schema definitions into appropriate structures in the memory. These structures are then converted to EXI grammars based on the algorithms specified in the EXI specification. The EXIP library takes a different approach by including a dedicated EXI grammar generator without external dependencies on schema parsers, which uses a modified version of the algorithms described in the EXI specification.

Many embedded targets use EXI because XML processing is too heavy to support. In such cases, the dynamic generation of the EXI grammars cannot be achieved in a standard way, as it requires processing text-based XML schema definitions. One possible solution is to use proprietary encoding for the EXI grammars, which is against the principles of the Web, and will still require some loading code that expands the programming memory footprint.

The dedicated EXI grammar generator solves this problem by using two simple ideas. First, the

[2]Available at http://exip.sourceforge.net/

XML Schema document is itself an XML document that can be represented in binary using EXI, thus reducing its size and improving the loading time. Second, once represented in EXI, the XML Schema document can be parsed by the EXI parser itself without the need of an external library for that; in other words, the EXI decoder code is reused to extract the XML schema definitions.

3.3 EXI Grammar Concatenation and Normalization

The EXI specification defines an algorithm for building a set of context-free grammars that directly correspond to the definitions in the W3C XML Schema specification. These grammars are called proto-grammars as they are intermediate representation which is only used during EXI grammar generation. The process of building proto-grammars is roughly as follow:

1. a set of simple proto-grammars are defined that describe the content model for each atomic XML schema definition (attributes, simple types, element terms, wildcard terms)
2. the proto-grammars for composite schema definitions are built by using the proto-grammars of their sub-components. For example, the <*sequence*> compositor equals to concatenation of the proto-grammars of its child elements and <*choice*> compositor equals to the union of the proto-grammars of its children.

The next step in the process of building EXI grammars is to normalize the proto-grammars such that all unit productions ($Z \rightarrow Y$ where both Z and Y are intermediate symbols) are removed and there are no ambiguities in the grammars. This essentially converts the proto-grammars to EXI grammars that are then used for processing EXI documents conforming to a schema.

The review of the algorithm for creating EXI proto-grammars from XML Schema definitions in section *8.5.4.1 EXI Proto-Grammars* of the EXI specification leads to the conclusion that the only way for creating proto-grammars that contain unit productions, and hence are not regular, is as an output of the grammar concatenation operator (see *8.5.4.1.1 Grammar Concatenation Operator* of the specification). However, all atomic grammars used as an input to the concatenation operator are regular and from the closure property of the regular languages under concatenation [49] we know that the resulting output grammar can also be presented in a regular form.

This section defines an extended grammar concatenation operator that produces regular EXI grammars, thereby removing the need for additional normalization of the grammars by removal of unit productions. The extended operator depends on the following recursive definition:

DEFINITION: Weak equality of grammar productions *The grammar production $A : Z_1 \rightarrow a_1Y_1$ and the grammar production $B : Z_2 \rightarrow a_2Y_2$ are* ***weakly equivalent*** *if:*

1. *$a_1 \equiv a_2$ and $Y_1 \equiv Y_2$*
   OR
2. *$a_1 \equiv a_2$. Let the sets of productions in the EXI grammar that have Y_1 and Y_2 as a left-hand side be denoted as $\{Y_1\}$ and $\{Y_2\}$ respectively. The two sets have the same cardinality, and each production $P \in \{Y_1\}$ is* ***weakly equivalent*** *to a production in $\{Y_2\}$.*

The grammar concatenation operator defined below is very similar to the one in the EXI specification in the sense that it creates a new grammar given two input grammars. The new grammar accepts any set of symbols accepted by the left operand followed by any set of symbols accepted by the right operand of the concatenation operator. The main difference is that the operator defined here produces regular EXI grammars, given its operators are also regular grammars.

DEFINITION: Extended grammar concatenation operator *The EXI Grammars $L(N_l, T, S_l, P_l)$ and $R(N_r, T, S_r, P_r)$ are given, where N_l and N_r are finite sets of non-terminals, T is the set of terminal symbols representing the EXI events, $S_l \in N_l$ and $S_r \in N_r$ are both designated initial symbols, and P_l and P_r are the sets of grammar productions in L and R respectively. All grammar productions in P_l and P_r are in one of the following two forms: $Z \to aY$ where $a \in T$ and $a \neq EE$ or $Z \to EE$ where $EE \in T$ is the terminating* end element *EXI event.*

*The result of applying the grammar concatenation operator to L and R, $L \bigoplus R$, is a new grammar $C(N_l \cup N_r, T, S_l, P_c)$ where the set of productions P_c is defined as follows: each production $l \in P_l$, where $l \neq Z \to EE$ for every $Z \in N_l$, is part of P_c; each production $r \in P_r$, where $r \neq S_r \to aY$ for every $a \in T$, and $Y \in N_r$ is part of P_c. For each production $e_l \in P_l$, where $e_l \equiv Z \to EE$ for every $Z \in N_l$, the following set of productions is also part of P_c: the set $\{Z \to aY\}$ where a production s_r of the form $S_r \to aY$ exists in P_r, and s_r is not **weakly equivalent** to any production in P_l that has Z as a left-hand side non-terminal symbol. There are no other productions in P_c besides those defined with these rules.*

When the extended concatenation operator is used for XML Schema $\langle$*sequence*$\rangle$ definitions, the resulting regular grammar might contain productions with duplicate terminal symbols i.e. the result can be an ambiguous regular grammar. In this case the algorithm in section *8.5.4.2.2 Eliminating Duplicate Terminal Symbols* of the EXI specification should be further applied to the resulting concatenated EXI grammar. It is worth noting that these cases are extremely rare and can only occur when optional element particles are allowed to repeat more than once. Example content model that contains duplicate terminal symbols and leads to the creation of ambiguous regular grammar is the following:

```
<sequence maxOccurs="2">
  <element name="a" maxOccurs="3"/>
  <element name="b" minOccurs="0"/>
</sequence>
```

3.4 Efficient Representation of Schema Deviations

The EXI specification defines an algorithm that augments the EXI Grammars with additional grammar productions which are used to handle possible deviations from the XML schema. Such deviations are often used to add extensions to a particular protocol or handle cases that require additional information in the XML documents. Furthermore, certain XML events that are not explicitly declared in the schema may also occur in the instance documents without making them invalid (e.g. comments, processing-instructions, type casts using *type* attribute from `http://www.w3.org/2001/XMLSchema-instance` namespace).

One constraint that must be followed when adding productions to the normalized EXI grammars is that addition of productions allowing attribute deviations must only occur before the element content - otherwise the grammars describe a document which is not well formed. The algorithm as described in the EXI specification (see *8.5.4.4.1 Adding Productions when Strict is False* [25]) depends on a set of redundant productions in the normalized EXI grammars in order to fulfill this requirement. The redundant productions are a copy of the productions describing the possible states for starting the content of an XML element that has wildcard attributes or a mixed-content model. An example of such redundant productions is the EXI grammar describing element fragments (see *8.5.3 Schema-informed Element Fragment Grammar* [25]).

The algorithm described in this section augments the EXI grammars for accepting schema deviations without having a dependency on redundant productions in the input EXI grammar. The algorithm is presented by highlighting only the modifications and differences with comparison to the algorithm in the EXI specification. An example of applying the modified algorithm is given in Appendix A.

The algorithm depends on the definition of a *content* non-terminal symbol, and an index called *content index* for each input EXI grammar. The assignment of *content index* and *content* to a non-terminal symbol is identical to the process defined in the EXI specification, and a prose description of it is given below:

DEFINITION: content non-terminal symbol *The **content** non-terminal symbol is the symbol that indicates that all attributes (AT terminal symbols) are already encoded. The **content** non-terminal symbol represents all the states for starting the encoding of the content of a particular XML element.*

DEFINITION: content index *Assign index numbers to all non-terminal symbols such that the designated initial symbol of the EXI grammar has index 0 and all other indexes are larger than 0. The index of the **content** non-terminal symbol, in other words, the **content index**, is then the smallest index that is larger than the indexes of all non-terminal symbols that are used as a left-hand side in grammar productions with AT terminals.*

DEFINITION: Grammar augmentation for schema deviations *Create a copy of all grammar productions that have the **content** non-terminal on the left-hand side if and only if there are AT productions that have the **content** non-terminal symbol on their right-hand side or the **content index** is 0. The copy of the **content** non-terminal symbol - **content2** if available, is inserted just before the **content** i.e. it has index of (**content index - 1**). In the case when the **content index** is 0, that would mean that the **content2** is now the entry non-terminal symbol of the grammar. After the copying, there should be no productions with **content2** non-terminal on the left-hand side that have **content2** on their right-hand side - instead they should have only **content**. All AT productions that have a **content** non-terminal symbol on their right-hand side are changed to point towards **content2** instead.*

*Apply the procedure in **8.5.4.4.1 Adding Productions when Strict is False** [25] while applying the following modifications to the algorithm:*

- *The designated initial symbol of the EXI grammar is changed to **content2** when **content index** is 0.*
- *Change each occurrence of **content** with **content2** and vice versa, that is, each occurrence of **content2** with **content**.*
- *If there is no **content2** non-terminal, then do not perform the procedure for it and assume the **content2 index** is smaller than the **content index**, but larger than the indexes of all non-terminals that are used in AT productions.*

4 Performance Evaluation

The goal of this section is to evaluate the performance of the dedicated EXI grammar generator implemented as part of the EXIP library. The grammar generator accepts EXI encoded XML Schema definitions as an input, and uses the extended grammar concatenation operator and the algorithm for efficient representation of schema deviations. The measurements in this section are indicative and aim to compare the execution time and memory usage of grammar generation on real-world data. As the core contribution of this work is in the grammar generation utility, this section does not evaluated the overall EXI processing performance. Measurements of the EXI parsing speed are included only to the extent needed to put the grammar generation evaluation in context.

Description of the test setup A set of 5 XML schema documents were used for decoding 15 instances (XML examples that conform to the schema; 3 instances per each schema document) by 3 different EXI processors. Decoding in this experiment refers to converting a binary EXI file to its text-based XML representation. The EXI processors are EXIficient v0.9.1 Java [50], OpenEXI v1.0238.0 Java [51], and EXIP v0.5.3 C [52]. At the time of writing this article - June 2014, there is one more open source EXI parser - WS4D-uEXI[3]. WS4D-uEXI is written in C and is designed for constrained embedded devices. It is not included in this comparison as it uses EXIficient library for building the EXI grammars at compile time and therefore does not support runtime grammar generation [53]. Moreover, WS4D-uEXI implements a subset of the EXI specification and its current version (SVN r2) is unable to decode some of the EXI instances in this evaluation due to missing features.

The evaluation uses the following XML schema documents: netconf.xsd[4], SenML.xsd[5], sep.xsd[6], OPC-UA-Types.xsd[7], and XMLSchema.xsd[8]. All of them were accessed from the local hard-drive, including the imported XML schema files, so there were no dependencies on the network performance.

The tests were executed on a desktop PC (Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz, 4GB RAM @ 1067 MHz) running 32-bit Linux Ubuntu 13.10. The version of the Java Virtual Machine (JVM) used for running EXIficient and OpenEXI is Java HotSpot(TM) Server VM 1.7, and the C compiler used for EXIP is GCC 4.8.1.

Two distinct measurements of the execution time were performed for each EXI processor: (1) the time it takes for loading an XML Schema and converting it to EXI grammars, and (2) the time it takes to generate the EXI grammars as well as decode a sample XML instance. The time was measured using $System.nanoTime()$ in Java and $clock\_gettime()$ in C, in other words, we measured wall-clock time which can vary depending on the external load of the system. In order to get comparable results, the tests were executed ensuring similar conditions on the system load, and taking the mean value of 300 measurements. Moreover, the mean value is calculated for two distinct runs of the test framework - one with optimizations and one without applying optimizations. In the unoptimized case the Java processors run on a "cold" JVM i.e. the code is executed for the first time on the VM and hence the classes for grammar generation and instance decoding are loaded at runtime. Also the "cold" JVM has smaller chance for applying run-time optimizations such as Just-In-Time (JIT) compilation. Conversely, the optimized case uses "warmed-up" JVM where the tests are run 5 times on the JVM before the measurement are taken. The EXIP processor is compiled with $-O0$ flag for unoptimized case and with $-O3$ for the optimized run.[9]

[3]http://code.google.com/p/ws4d-uexi/

[4]Network Configuration Protocol: https://www.iana.org/assignments/xml-registry/schema/netconf.xsd

[5]Sensor Markup Language: http://tools.ietf.org/html/draft-jennings-senml-10

[6]SEP2: http://www.zigbee.org/Standards/ZigBeeSmartEnergy/SmartEnergyProfile2.aspx

[7]OPC-UA: http://opcfoundation.org/UA/2008/02/Types.xsd

[8]Schema for XML Schema: http://www.w3.org/2001/XMLSchema

[9]The automated test framework for configuring and executing the evaluation is available open source at http://github.com/kjussakov/exip-eval

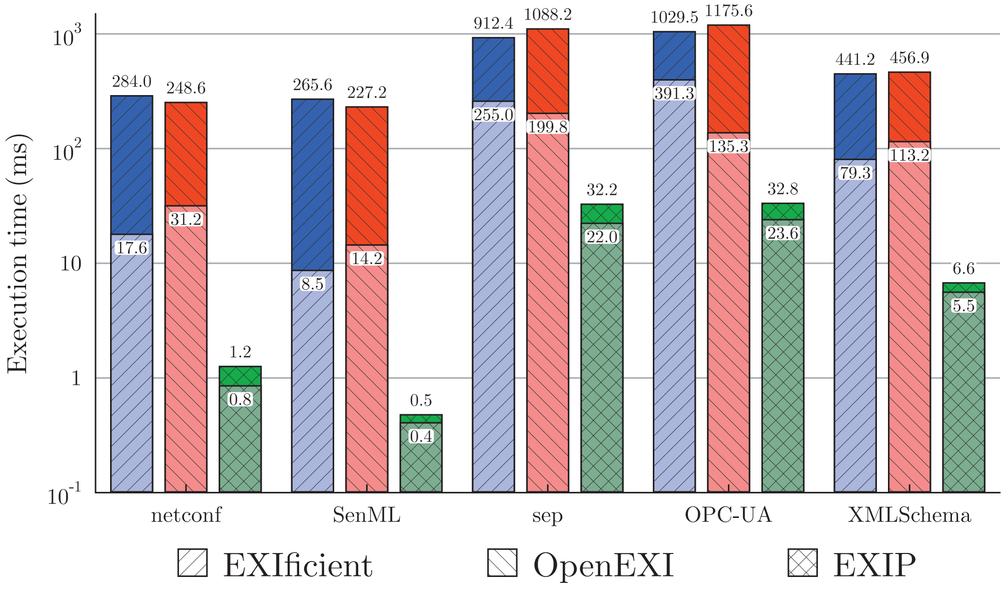


Figure 3: Grammar generation execution times for each XML Schema test case. The averaged times per XML schema are given on the logarithmic *Y* axis for each of the tested EXI processors - EXIficient (leftmost column, forward slash hatching), OpenEXI (middle column, backslash hatching) and EXIP (rightmost column, grid hatching). Each bar in the chart represents the execution times when explicit optimizations are applied (lighter colored part of the bar) and when no optimizations are applied.

Figure 3 and Figure 4 show the averaged execution times per each XML schema test case with enabled and disabled optimizations. In Figure 3 the times are for grammar generation only while Figure 4 shows the execution times for both grammar generation and instance decoding. In both charts, the execution times on the *Y* axis are represented in logarithmic scale for enhancing the visual representation.

On average, among all test cases, the execution times for grammar generation and instance decoding are given in Table 2.

Table 2: Averaged execution times (*ms*) for all XML Schema test cases

EXI Processor	**Optimized**		**Unoptimized**	
	Grammar	**Grammar+Instance**	**Grammar**	**Grammar+Instance**
EXIficient	150.3	168.7	586.5	651.4
OpenEXI	98.8	106.9	639.3	676.2
EXIP	10.5	11.3	14.7	15.8

As shown in the table, EXIP generates the grammars about 9 times faster than OpenEXI and 14 times faster than EXIficient when compile time optimizations for the C code and run-time JVM optimizations for the Java code are enabled. This cannot be attributed solely to the performance differ-

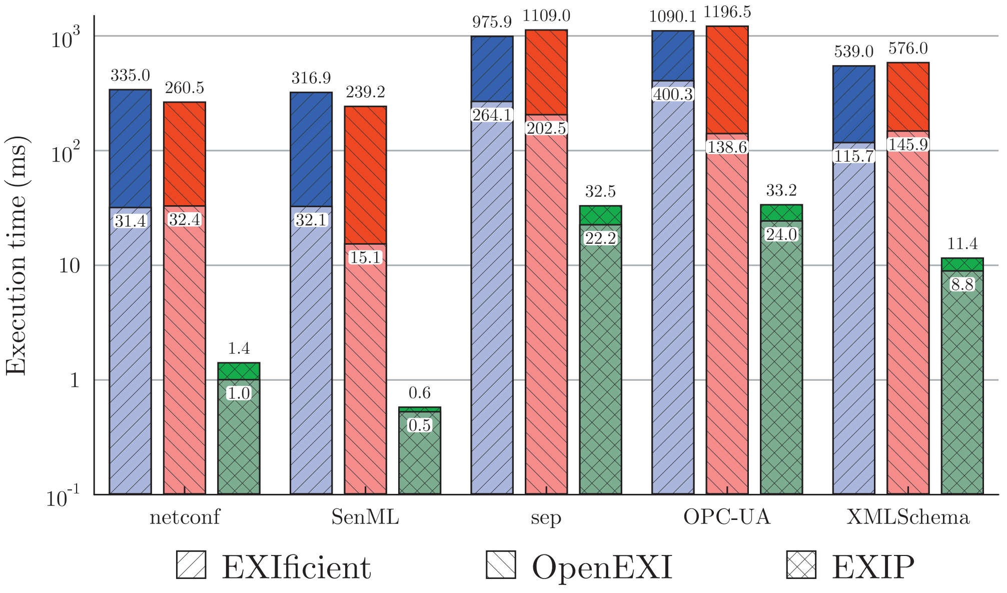


Figure 4: Grammar generation and instance decoding execution times for each XML Schema test case. The averaged times per XML schema are given on the logarithmic *Y* axis for each of the tested EXI processors - EXIficient (leftmost column, forward slash hatching), OpenEXI (middle column, backslash hatching) and EXIP (rightmost column, grid hatching). Each bar in the chart represents the execution times when explicit optimizations are applied (lighter colored part of the bar) and when no optimizations are applied.

ence in native code versus Java byte code execution where on average Java programs are somewhere between 50 % faster to 4 times slower than their C counterparts[10].

The superior performance of EXIP grammar generation is mainly due to the use of EXI-specific XML Schema parser that accepts EXI encoded XML Schema definitions as opposed to the use of general purpose XML Schema parser. By using the extended grammar concatenation operator (see Section 3.3), EXIP has to perform one less iteration over the set of all grammar rules which has noticeable benefits mainly in large XML Schemas such as SEP2 (sep.xsd). The grammar augmentation algorithm presented in Section 3.4 has no effect on processing efficiency, but slightly improves memory usage. Code optimizations, in terms of avoiding unnecessary loops and selecting appropriate searching and sorting algorithms (for example the use of a hash table for mapping element definitions to their globally defined types instead of iteration), have impact on the performance as well but are harder to quantify.

[10]Source: http://benchmarksgame.alioth.debian.org/u32/java.php

Table 3: Size of a SenML instance for different encoding modes and memory usage for EXIP and the light-weight XML parser library MiniXML on a Raspberry Pi system. The rows are ordered by document size.

Encoding mode	Size bytes	RAM/heap usage (kB)			
		EXIP		MiniXML	
		Encoding	Decoding	Encoding	Decoding
Plain XML	387	-	-	1.36	1.55
EXI Schema-less byte aligned	248	7.95	8.26	-	-
EXI Schema-less no value index	237	6.93	6.79	-	-
EXI Schema-less default options	200	7.90	8.26	-	-
EXI Schema mode no value index	137	1.93	2.27	-	-
EXI Schema mode default options	100	2.87	2.21	-	-
EXI Schema mode strict	98	2.89	2.23	-	-

4.1 Memory usage

This section provides some insight into the memory consumption of EXIP, and EXI in general, as memory is often a bottleneck in embedded system applications. Section 2.1 already discussed that the dynamic memory usage for EXI processing can be controlled by some of the parameters defined in the EXI header. This is done by adjusting the extent of the content indexing used to detect and reduce redundancy in the data which also affects the compactness and processing speed. However, the mechanisms provided in the EXI specification cannot guarantee bounded run-time memory usage when deviations from the XML schema are present. For that purpose, an extension to these mechanisms are developed in a complementary specification called EXI Profile for limiting usage of dynamic memory [42]. A subset of this profile is supported by EXIP but its impact on the memory consumption is not evaluated in this section as the tests presented here are restricted to a schema valid instance of the SenML standard. Table 3 shows the size and memory usage during encoding and decoding for a sample instance document borrowed from the SenML specification[11].

The size and memory consumption are given for different encoding options. The platform used for testing is Raspberry Pi embedded computer with ARM-based system on chip including 700 MHz processor with 512 MB of RAM. The memory usage presented in Table 3 shows only the amount of dynamic memory (heap) usage for statically compiled EXI grammars and is measured using DHAT (dynamic heap analysis tool) that is part of the code profiling library Valgrind.

An interesting observation is that although the document is relatively small, turning off the indexing of repeating values (i.e. setting *valuePartitionCapacity* parameter to 0) substantially inflates the size of the resulting EXI representation. This is due to the high redundancy in the attribute values which has profound affect even in schema mode encoding. This simple example shows the high variation of compression and dynamic memory usage depending on the content of the documents and the encoding options in use.

The compile-time allocated RAM used by the EXIP library (calculated as the sum of .rodata, .data and .bss sections in the Executable and Linking Format (ELF)) is 23 kB (of which 8 kB EXI grammar definitions used for schema mode cases) while the light-weight XML parser MiniXML v2.8 requires only 3 kB. EXIP SenML parser uses 79 kB programming memory while MiniXML uses only

[11] Available at: http://tools.ietf.org/html/draft-jennings-senml-10#section-7

16 kB. Additionally, as shown in Table 3, MiniXML is more efficient in the use of dynamic memory compared to EXIP. These results indicate that EXI processing, and EXIP library in particular, require more RAM compared to highly optimized XML processing. The main reason for this is the use of content indexing and grammar information during EXI processing. Further optimizations of the RAM usage in EXIP are possible both for the size of the content index as well as the in-memory grammar representation. It should also be noted that schema-based EXI processing implicitly performs partial schema validation while MiniXML is a non-validating parser.

Enabling run-time EXI grammar generation from the SenML schema additionally requires 57 kB of dynamic memory and 37 kB of programming memory. These memory requirements show that the run-time grammar generation module fits easily in embedded devices such as Raspberry Pi but is too heavy for the most constrained platforms. As an example, the popular Stellaris LM4F120H5QR 32-bit ARM Cortex-M4F microcontroller (80 MHz CPU frequency, 256 KB flash and 32 KB SRAM) does not have enough RAM for supporting run-time EXI grammar generation. Nevertheless, by using static grammars the EXIP library is capable of running on such platforms with averaged total RAM usage of about 20 kB[12] and 60kB of programming memory for the SenML sample instance.

5 EXI data binding

The information contained in an XML/EXI document is often loaded into the memory for further processing and mapped to a hierarchy of data structures or objects that are maintained by the applications. For example, a status report by a device can include various hierarchical information such as network status (which in turn contains parameters like RTT, signal strength, connected peers etc.) or resource utilization (storage space, battery level etc.) that is mapped to a corresponding hierarchy of programming objects. The process of generating an XML/EXI document from a hierarchy of objects and vice versa is known as *XML/EXI data binding*. The process of building objects from an XML/EXI input document is called *unmarshalling* and the reverse - the generation of XML/EXI output document from objects, is called *marshalling*. The *unmarshalling* is implemented as a software module that connects to the parser API, and generates memory structures that correspond to the structure and content of the XML document. The *marshalling* is implemented as a module that transforms a set of objects in the memory to a sequence of calls to the serialization API.

The *XML/EXI data binding* code can be complex to write and maintain manually. For that reason, it is often automatically generated. There are two main approaches when generating the code and keeping it in sync with the XML/EXI documents - direct, and indirect mapping. In direct mapping, the source code is generated based on XML schema definitions or vice versa - the XML schema can be built based on the existing source code definitions. When no schema information is available or needed, the XML tree can be directly mapped to a memory representation, as in the case of the Document Object Model (DOM). The data binding frameworks that are based on direct mapping of the XML Information Set and the memory representation, are widely adopted in desktop and enterprise applications - examples include DOM, JAXB, XMLBeans, and others [54]. Their main advantage is that it is very easy to build and maintain the XML-to-source code mapping. An example of a pure XML direct mapping framework for embedded systems development is the gSOAP toolkit [55]. A similar approach, but applied to EXI and targeted at highly resource-constrained embedded devices is the automatic EXI Processor generation reported by Käbisch et al. [56].

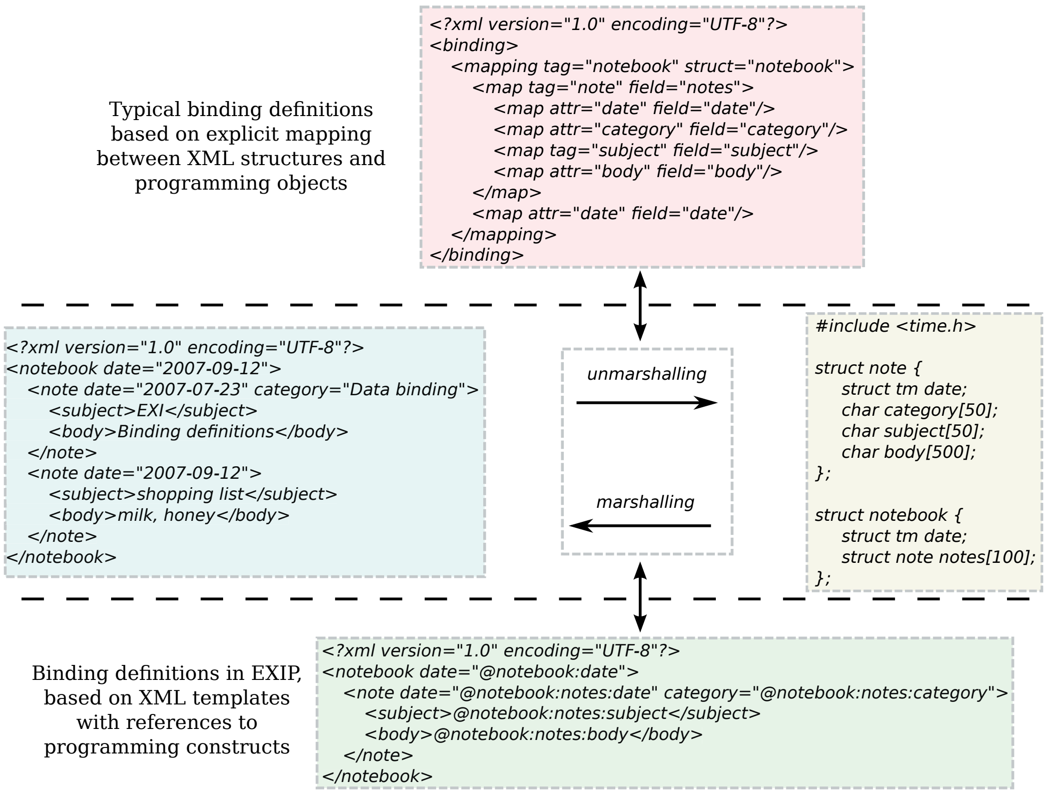
[12] The RAM usage in schema mode is 20 kB (1 kB stack size + 2.5 kB heap + 16.5 kB .data and .bss) while the RAM usage in schema-less mode is 19 kB (1 kB stack size + 7.5 kB heap + 10.5 kB .data and .bss)

The indirect mapping is a more flexible approach that allows discarding the unnecessary XML structures or reusing existing objects in the memory by defining a layer of indirection between the XML Information Set and the memory representation. Example libraries in this category include Castor and JiBX [57] - both only available in Java, and targeted at server/desktop applications. A comparison between the two approaches i.e. direct and indirect mapping, along with performance measurements, are presented by Sosnoski in *IBM developerWorks* article on data binding tools for Java/XML [58].

The EXI binding presented in this section falls into the category of indirect mapping, and it is targeted at embedded systems development. Its design is based on the following requirements:

- The mapping rules should have intuitive syntax and semantics.
- The binding definitions should be independent from the programming language in use - the same binding definition should work for programs written in C, Java, Python, and so on.
- The EXI binding should be efficient to use on embedded platforms.
- The mapping layer should allow for loading the binding definitions and building the objects in memory dynamically at run-time.

To optimally fulfill these requirements, we propose template-based binding definitions that are


Typical binding definitions based on explicit mapping between XML structures and programming objects
unmarshalling
marshalling
Binding definitions in EXIP, based on XML templates with references to programming constructs

```
<?xml version="1.0" encoding="UTF-8"?>
<binding>
   <mapping tag="notebook" struct="notebook">
      <map tag="note" field="notes">
         <map attr="date" field="date"/>
         <map attr="category" field="category"/>
         <map tag="subject" field="subject"/>
         <map attr="body" field="body"/>
      </map>
      <map attr="date" field="date"/>
   </mapping>
</binding>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<notebook date="2007-09-12">
   <note date="2007-07-23" category="Data binding">
      <subject>EXI</subject>
      <body>Binding definitions</body>
   </note>
   <note date="2007-09-12">
      <subject>shopping list</subject>
      <body>milk, honey</body>
   </note>
</notebook>
```

```
#include <time.h>

struct note {
     struct tm date;
     char category[50];
     char subject[50];
     char body[500];
};

struct notebook {
     struct tm date;
     struct note notes[100];
};
```

```
<?xml version="1.0" encoding="UTF-8"?>
<notebook date="@notebook:date">
   <note date="@notebook:notes:date" category="@notebook:notes:category">
      <subject>@notebook:notes:subject</subject>
      <body>@notebook:notes:body</body>
   </note>
</notebook>
```

Figure 5: Comparison between typical binding definitions and the EXIP templates

written in XML and converted to EXI before being used for code generation or loading at runtime. The binding templates are very similar to other frameworks for dynamic content delivery based on templates such as JavaServer Pages (JSP) technology. An in-depth overview of template-based code generation is presented in [59] where the authors describe the theoretical foundations of template systems and include comparison with other code generation techniques. The proposed EXI template framework is a heterogeneous code generator that follows the model-view-controller design pattern as suggested by Arnoldus et al. [59].

Figure 5 shows a comparison of this approach to what is a commonly used method for defining such binding definitions. As depicted, the mapping between dynamic EXI content and programming constructs is done using a special character @ and semicolon notation. As such, the definitions are intuitive to define as well as simple to process by the loading code. As with other such approaches based on templates, these special characters must be escaped when used in a static content. As an example, the value for a static attribute *email* within an EXI binding definition should be defined as *example@@com* to escape the special character that indicates the beginning of dynamic content mapping.

6 CoAP/EXI/XHTML Web page engine

This section presents a prototype implementation of a dynamic Web interface for an embedded sensor platform based on CoAP/EXI/XHTML technologies. The implementation is developed using the EXIP framework, and consists of an experimental Java browser running on a laptop PC that connects to a wireless sensor device (Mulle version 3.2 [60]) over Bluetooth. The laptop user can navigate to the device Web interface using mDNS/DNS-SD or CoAP built-in discovery capabilities - multicast service discovery [38], or CoRE Resource Directory [61]. In our simplified test setup, the network address of the sensor device is predefined so the discovery process was not implemented.

The EXI encoded XHTML page is dynamically generated on the sensor platform on a CoAP GET request, and it contains an *iframe* tag with a link to an external observable CoAP resource:

```
...
<p>Current temperature is:</p>
<iframe src="coap://192.168.150.10:5683/temp"/>
...
```

The observable CoAP resource is linked to a temperature sensor on the wireless device, and is updated whenever there is a change in the measured air temperature.

As shown in Figure 6, upon user request, the Java browser[13] sends a CoAP GET request to the sensor device using the *Californium* library. The wireless device receives the request and generates a CoAP response using the *libcoap* library. The CoAP version 13 payload is a dynamically generated EXI/XHTML Web page using the EXIP framework. Once the packet is transmitted back to the Java browser, the EXI document is decoded by the *OpenEXI* library, and the *iframe* link is resolved by initiating an additional CoAP request to fetch the temperature. By using CoAP Observe [40], the Java client subscribes to changes in the temperature without requiring additional periodic polling requests. The temperature is represented in plain text, and visualized on the browser window each time a CoAP notification is received.

[13]Based on Flying Saucer XHTML/CSS 2.1 renderer: https://code.google.com/p/flying-saucer/

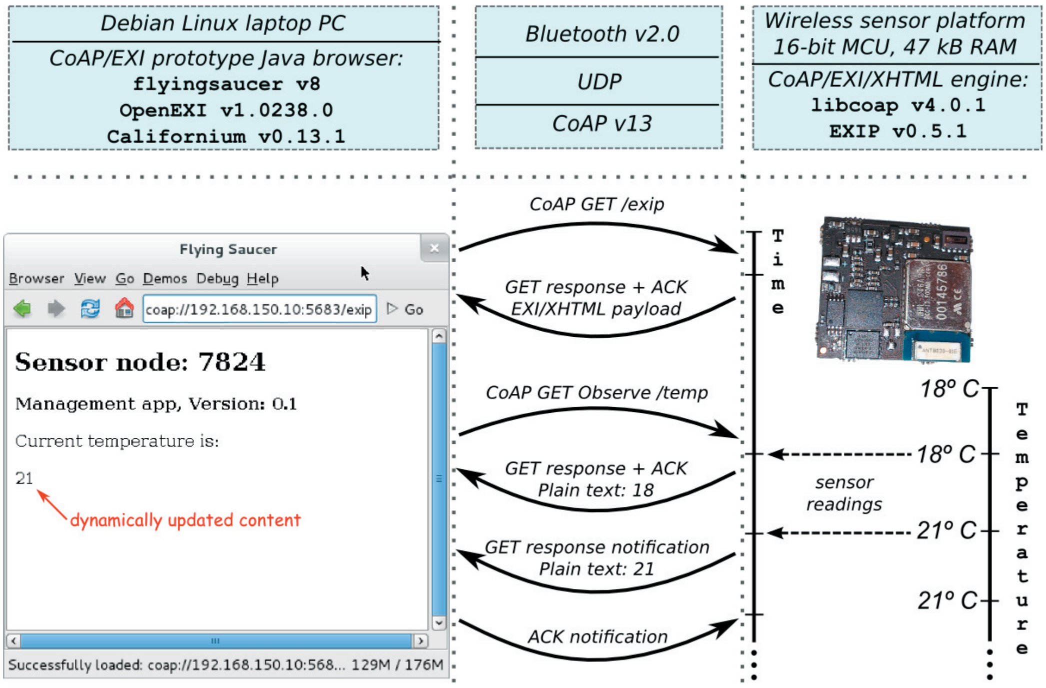


Figure 6: CoAP/EXI/XHTML dynamic Web interface demonstration.

This prototype demonstrates how the newly emerging binary Web protocols can be employed to enable a dynamic Web interface for highly resource-constrained embedded devices. The Web interface can be used in a wide range of mobile applications, as suggested by Puñal Pereira et al. [62]. The approach of using the *iframe* tag with CoAP Observe enables very lightweight event-based content delivery that is suitable for low-power radio communications such as IEEE 802.15.4 (6LoWPAN, ZigBee), Z-Wave, or Bluetooth low energy. Example application domains for the EXIP framework include, but are not limited to: industrial process monitoring and control, eHealth and elderly care, wearable electronics, home automation, and energy management.

Data visualization technologies based on XML encoding such as SVG[14] and X3D[15] can be readily included in the CoAP/EXI/XHTML engine to efficiently represent graphical indicators (e.g., battery level, signal strength) and visualize measurements and configuration parameters. An evaluation of EXI encoding for SVG in rich media applications for embedded systems presented by Peintner et al. [63] shows that EXI significantly increases the efficiency of the SVG format. Also shown in this work is an approach using the EXI header option *datatypeRepresentationMap* to further optimize the compression of graphics formats for embedded web applications.

The presented CoAP/EXI/XHTML Java browser always tries to subscribe to the *iframe* CoAP links - if the resource is not observable, the subscription is not established. When the resource is

[14]Scalable Vector Graphics (SVG): http://www.w3.org/TR/SVG11/
[15]X3D Specification for 3D Graphics: www.web3d.org/x3d

observable but should be treated statically for display in the browser (for example representing a snapshot of dynamic data), the embedded server should reject the subscription request by the browser. This approach can be too limited in certain scenarios, in which case different ways to indicate whether the browser should subscribe to changes on the *iframe* resource can be employed - adding an extra boolean argument *observe* to the *iframe* tag as an XHTML schema deviation, or requesting the resource description in CoRE Link Format before sending the subscription request. Similarly, in more complex scenarios the use of plain text encoding for the *iframe* resource might be too limiting. In such cases a structured format such as EXI can be used instead of plain text. The definition of the data format (including parameters and schema if available) for particular *iframe* can be defined as XHTML schema deviation or read from the CoRE Link Format as suggested for the observe use case.

Implementation details The information provided hereafter gives more insight into the actual implementation, and is useful for reproducing the test setup. The Mulle sensor platform has a 16-bit Renesas M16C/62P microcontroller and Mitsumi Bluetooth 2.0 wireless module. The application runs on bare metal, in other words, without an OS, on top of a port of *lwIP* TCP/IP stack and a *libcoap* v4.0.1 library. The EXI/XHTML generation is done in schema-less mode using *EXIP* v0.5.1.

The laptop PC is running Debian Linux, and is equipped with a USB Bluetooth 2.0 adapter. Debian packages *bluez-compat v4.99-2*, *bridge-utils v1.5-6*, and *isc-dhcp-server* were installed and configured on the system to enable TCP/IP communication over Bluetooth.

The size of the EXI/XHTML Web page is 239 bytes, and is generated directly in binary (EXI) form without transition to plain XML. If converted to text XHTML, the size is 427 bytes. The temperature notifications are in plain text, and account for 14 bytes of CoAP packet size (UDP payload) in total, assuming 2 bytes for the plain text temperature value.

7 Conclusions

The newly emerging transport and data representation protocols based on binary encoding - CoAP and EXI - provide an efficient way to connect embedded systems to the Web across scenarios as diverse as mobile computing, home automation, and smart grid. As the translations between CoAP $\Leftrightarrow$ HTTP and EXI $\Leftrightarrow$ XML are well defined, the integration of these binary protocols to the existing Web infrastructure is standardized and conforms to the well-established programming interfaces. For example, EXI processors often provide the same API as XML processors, and CoAP/HTTP proxies are simple to deploy and are transparent for the Web users.

The work presented in this paper shows that the use of CoAP/EXI stack and the EXIP Web development toolkit enables reuse of the existing pool of Web technologies and developers' skills, even on very resource-constrained embedded platforms. The development process and especially the integration with existing systems is much faster and easier to maintain as compared to the use of handcrafted communication protocols.

Moreover, the presented EXI processor design, and the EXI grammar generation algorithms in particular, provide superior processing performance compared to the methods described in the EXI specification with order-of-magnitude speed-up in some of the test cases. This could enable exchange patterns supporting dynamic XML schema negotiations even for embedded hosts. The use cases for such an approach include support for schema versioning, generic Web services, and runtime service composition.

Finally, the presented prototype of dynamic Web interface for sensor platforms demonstrates the possibility to use event-based Web content delivery with a very low overhead in terms of network bandwidth and processing power. The development of the Web interface or Web service exchange can be automated by using the template-based EXI data binding. As the data binding creates indirect mapping between the EXI document and the programming constructs, the memory structures and programming objects can be reused when generating or decoding the EXI streams.

Possible extensions of this work include in-depth memory consumption evaluation and trade-off analysis as well as developing a formal specification of the EXIP data binding, and implementing prototypes in C and Java to evaluate the proposed approach against existing XML data binding frameworks. Providing support for light-weight client-side scripting as part of the CoAP/EXI/XHTML embedded Web programming is also an interesting and important topic for future investigation. It is also worth analyzing the application of CoAP/EXI, and the EXIP framework in particular, for mobile platforms and even for desktop applications that are not resource-constrained. Lowering the network traffic and CPU cycles for Web content delivery on mobile phones and tablet PCs could potentially increase the battery life for these devices, lower the networking cost for both operators and users, and even lead to energy savings if applied on a global scale.

APPENDIX A: Grammar Augmentation Algorithm Example

This appendix gives an example of how the augmentation procedure is applied to the wildcard XML schema type *anyType* which is the base type definition for all other XML schema types. A minimal (without redundant productions) EXI grammar that describes the content model according to the process of creating proto-grammars is:

```
anyType-0:
            AT(*)    anyType-0
            SE(*)    anyType-1
            EE
            CH       anyType-1

anyType-1:
            SE(*)    anyType-1
            EE
            CH       anyType-1
```

There is no need to copy the content grammar productions (the ones with anyType-1 on the left-hand side) because there are no AT productions that points to it and the *content index* is 1.

Then apply the procedure in *8.5.4.4.1 Adding Productions when Strict is False* [25]:
As there is EE production already do not add additional one. Adding the AT(xsi:type) and AT(xsi:nil) productions produces:

```
anyType-0:
            AT(*)            anyType-0
            SE(*)            anyType-1
            EE
            CH               anyType-1
```

```
             AT(xsi:type)     anyType-0
             AT(xsi:nil)      anyType-0

anyType-1:
             SE(*)            anyType-1
             EE
             CH               anyType-1
```

"For each non-terminal $Element_{i,j}$, such that $0 \leq j \leq content$ ..."
becomes:
"For each non-terminal $Element_{i,j}$, such that $0 \leq j \leq content2$ ..."
Because there is no $content2$ we apply that rule only to anyType-0:

```
anyType-0:
             AT(*)                    anyType-0
             SE(*)                    anyType-1
             EE
             CH                       anyType-1
             AT(xsi:type)             anyType-0
             AT(xsi:nil)              anyType-0
             AT(*)                    anyType-0
             AT(*)[untyped value]     anyType-0

anyType-1:
             SE(*)                    anyType-1
             EE
             CH                       anyType-1
```

After adding the NS and SC productions:

```
anyType-0:
             AT(*)                    anyType-0
             SE(*)                    anyType-1
             EE
             CH                       anyType-1
             AT(xsi:type)             anyType-0
             AT(xsi:nil)              anyType-0
             AT(*)                    anyType-0
             AT(*)[untyped value]     anyType-0
             NS                       anyType-0
             SC Fragment

anyType-1:
             SE(*)                    anyType-1
             EE
             CH                       anyType-1
```

"Add the following productions to each non-terminal $Element_{i,j}$, such that $0 \leq j \leq content$."
becomes:
"Add the following productions to each non-terminal $Element_{i,j}$, such that $0 \leq j \leq content2$."
The result of applying this rule is:

```
anyType −0:
            AT(∗)                  anyType−0
            SE(∗)                  anyType−1
            EE
            CH                     anyType−1
            AT(xsi:type)           anyType−0
            AT(xsi:nil)            anyType−0
            AT(∗)                  anyType−0
            AT(∗)[untyped value]   anyType−0
            NS                     anyType−0
            SC Fragment
            SE(∗)                  anyType−1
            CH[untyped value]      anyType−1

anyType −1:
            SE(∗)                  anyType−1
            EE
            CH                     anyType−1
```

”Add the following productions to $Element_{i,content2}$ and to each non-terminal $Element_{i,j}$, such that $content < j < n$, where n is the number of non-terminals in $Element_i$.”
becomes:
”Add the following productions to $Element_{i,content}$ and to each non-terminal $Element_{i,j}$, such that $content2 < j < n$, where n is the number of non-terminals in $Element_i$.”
The final grammar is:

```
anyType −0:
            AT(∗)                  anyType−0
            SE(∗)                  anyType−1
            EE
            CH                     anyType−1
            AT(xsi:type)           anyType−0
            AT(xsi:nil)            anyType−0
            AT(∗)                  anyType−0
            AT(∗)[untyped value]   anyType−0
            NS                     anyType−0
            SC Fragment
            SE(∗)                  anyType−1
            CH[untyped value]      anyType−1

anyType −1:
            SE(∗)                  anyType−1
            EE
            CH                     anyType−1
            SE(∗)                  anyType−1
            CH[untyped value]      anyType−1
```

Acknowledgment

The authors would like to thank Takuki Kamiya, Yusuke Doi, Daniel Peintner, and the other members of the W3C EXI working group for the fruitful discussions and valuable feedback. We are very grateful to the contributors and the users of the EXIP project for their feedback, code review, and source code contributions. We acknowledge the valuable discussions and support by the partners of the EU projects IMC-AESOP and Arrowhead.

References

[1] M. Kovatsch, M. Weiss, and D. Guinard, "Embedding internet technology for home automation," in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, 2010, pp. 1–8.

[2] G. Bumiller, L. Lampe, and H. Hrasnica, "Power line communication networks for large-scale control and automation systems," *Communications Magazine, IEEE*, vol. 48, no. 4, pp. 106–113, 2010.

[3] G. Borriello and R. Want, "Embedded computation meets the world wide web," *Commun. ACM*, vol. 43, no. 5, pp. 59–66, May 2000. [Online]. Available: http://doi.acm.org/10.1145/332833.332839

[4] A. Charland and B. Leroux, "Mobile application development: web vs. native," *Commun. ACM*, vol. 54, no. 5, pp. 49–53, May 2011. [Online]. Available: http://doi.acm.org/10.1145/1941487.1941504

[5] R. Gossweiler, C. McDonough, J. Lin, and R. Want, "Argos: Building a web-centric application platform on top of android," *Pervasive Computing, IEEE*, vol. 10, no. 4, pp. 10–14, april 2011.

[6] V. Trifa, S. Wieland, D. Guinard, and T. M. Bohnert, "Design and implementation of a gateway for web-based interaction and management of embedded devices," in *Proceedings of the 2nd International Workshop on Sensor Network Engineering (IWSNE 09)*, Marina del Rey, CA, USA, Jun. 2009.

[7] Q. Kang, H. He, and H. Wang, "Study on embedded web server and realization," in *Pervasive Computing and Applications, 2006 1st International Symposium on*, aug. 2006, pp. 675–678.

[8] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle, "Smews: Smart and mobile embedded web server," in *Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09. International Conference on*, march 2009, pp. 571–576.

[9] R. Van Engelen, "Code generation techniques for developing light-weight xml web services for embedded devices," in *Proceedings of the 2004 ACM symposium on Applied computing*. ACM, 2004, pp. 854–861.

[10] Z. Shelby, "Embedded web services," *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 52–57, 2010.

[11] K. Chiu and W. Lu, “A compiler-based approach to schema-specific xml parsing,” in *First International Workshop on High Performance XML Processing (May 2004)*, 2004.

[12] Y. Doi, Y. Sato, M. Ishiyama, Y. Ohba, and K. Teramoto, “Xml-less exi with code generation for integration of embedded devices in web based systems,” in *Internet of Things (IoT), 2012 3rd International Conference on the*, oct. 2012, pp. 76 –83.

[13] A. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, “Web services for the internet of things through coap and exi,” in *Communications Workshops (ICC), 2011 IEEE International Conference on*, june 2011, pp. 1–6.

[14] S. Käbisch, D. Peintner, J. Heuer, and H. Kosch, “Optimized XML-based Web service generation for service communication in restricted embedded environments,” in *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, 2011, pp. 1–8.

[15] G. White, J. Kangasharju, D. Brutzman, and S. Williams, “Efficient XML Interchange Measurements Note,” W3C, Tech. Rep., 2007. [Online]. Available: http://www.w3.org/TR/exi-measurements/

[16] J. Cowan and R. Tobin. (2004) XML Information Set (Second Edition). W3C. [Online]. Available: http://www.w3.org/TR/xml-infoset/

[17] R. Kyusakov, H. Mäkitaavola, J. Delsing, and J. Eliasson, “Efficient XML Interchange in factory automation systems,” in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, nov. 2011, pp. 4478 –4483.

[18] D. Caputo, L. Mainetti, L. Patrono, and A. Vilei, “Implementation of the EXI Schema on Wireless Sensor Nodes Using Contiki,” in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, 2012, pp. 770–774.

[19] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh, “Simulating the power consumption of large-scale sensor network applications,” in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, ser. SenSys ’04. New York, NY, USA: ACM, 2004, pp. 188–200. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031518

[20] C. Bournez, “Efficient XML Interchange Evaluation,” W3C, Tech. Rep., April 2009. [Online]. Available: http://www.w3.org/TR/exi-evaluation/

[21] K. Kuladinithi, O. Bergmann, T. Pötsch, M. Becker, and C. Görg, “Implementation of coap and its application in transport logistics,” *Proc. IP+ SN, Chicago, IL, USA*, 2011.

[22] M. Kovatsch, S. Duquennoy, and A. Dunkels, “A low-power coap for contiki,” in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*. IEEE, 2011, pp. 855–860.

[23] M. Kovatsch, M. Lanter, and Z. Shelby, “Californium: Scalable cloud services for the internet of things,” in *Proceedings of the 4th International Conference on the Internet of Things (IoT 2014)*, 2014.

[24] M. Kovatsch, S. Mayer, and B. Ostermaier, “Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things,” in *Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012), Palermo, Italy*, 2012.

[25] J. Schneider, T. Kamiya, D. Peintner, and R. Kyusakov, *Efficient XML Interchange (EXI) Format 1.0*, W3C Std., February 2014. [Online]. Available: http://www.w3.org/TR/exi/

[26] D. Peintner and S. Pericas-Geertsen, “Efficient XML Interchange (EXI) Primer,” W3C, Tech. Rep., 2009. [Online]. Available: http://www.w3.org/TR/2009/WD-exi-primer-20091208/

[27] D. A. Huffman, “A method for the construction of minimum-redundancy codes,” *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[28] S. A. Greibach, “A new normal-form theorem for context-free phrase structure grammars,” *J. ACM*, vol. 12, pp. 42–52, January 1965. [Online]. Available: http://doi.acm.org/10.1145/321250.321254

[29] A. J. Korenjak and J. Hopcroft, “Simple deterministic languages,” in *Switching and Automata Theory, 1966., IEEE Conference Record of Seventh Annual Symposium on*, Oct 1966, pp. 36–46.

[30] J. Berstel and L. Boasson, “Xml grammars,” in *Mathematical Foundations of Computer Science 2000*. Springer, 2000, pp. 182–191.

[31] B. Chidlovskii, “Using regular tree automata as xml schemas,” in *Advances in Digital Libraries, 2000. Proceedings. IEEE*, 2000, pp. 89–98.

[32] F. Neven, “Automata theory for xml researchers,” *ACM Sigmod Record*, vol. 31, no. 3, pp. 39–46, 2002.

[33] M. Murata, D. Lee, M. Mani, and K. Kawaguchi, “Taxonomy of xml schema languages using formal language theory,” *ACM Transactions on Internet Technology (TOIT)*, vol. 5, no. 4, pp. 660–704, 2005.

[34] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, “Tree automata techniques and applications,” Available on: http://www.grappa.univ-lille3.fr/tata, 2007, release October, 12th 2007.

[35] A. Brüggemann-Klein and D. Wood, “Balanced context-free grammars, hedge grammars and pushdown caterpillar automata.” in *Extreme Markup Languages®*, 2004.

[36] R. A. Van Engelen, “Constructing finite state automata for high performance web services,” in *IEEE International Conference on Web Services*, 2004.

[37] D. Wood, “Standard generalized markup language: Mathematical and philosophical issues,” in *Computer Science Today*. Springer, 1995, pp. 344–365.

[38] Z. Shelby, K. Hartke, and B. C., *Constrained Application Protocol (CoAP)*, IETF Std., Dec. 2013. [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-coap-18

[39] R. T. Fielding and R. N. Taylor, “Principled design of the modern web architecture,” *ACM Transactions on Internet Technology*, vol. 2, pp. 115–150, 2002.

[40] K. Hartke, *Observing Resources in CoAP*, IETF Std., February 2013. [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-observe-08

[41] R. Kyusakov, “EXIP User Guide,” EISLAB, Tech. Rep., 2013. [Online]. Available: http://exip.sourceforge.net/exip-user-guide.pdf

[42] Y. Fablet and D. Peintner, *Efficient XML Interchange (EXI) Profile for limiting usage of dynamic memory*, W3C Std., May 2014. [Online]. Available: http://www.w3.org/TR/exi-profile/

[43] R. Kyusakov, J. Eliasson, J. van Deventer, J. Delsing, and R. Cragie, “Emerging energy management standards and technologies - challenges and application prospects,” in *Emerging Technologies Factory Automation (ETFA), 2012 IEEE 17th Conference on*, 2012, pp. 1–8.

[44] A. Petrick and S. V. Ausdall. (2013, April) Smart Energy Profile 2.0. ZigBee Alliance, HomePlug Powerline Alliance. [Online]. Available: http://www.zigbee.org/Standards/ZigBeeSmartEnergy/Version20Documents.aspx

[45] R. Kyusakov, J. Eliasson, and J. Delsing, “Efficient structured data processing for web service enabled shop floor devices,” in *Industrial Electronics (ISIE), 2011 IEEE International Symposium on*, june 2011, pp. 1716 –1721.

[46] P. Waher and Y. Doi, *XEP-0322: Efficient XML Interchange (EXI) Format*, XMPP Standards Foundation Std., Rev. 0.3, July 2013. [Online]. Available: http://xmpp.org/extensions/xep-0322.html

[47] G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren, “Learning deterministic regular expressions for the inference of schemas from xml data,” *ACM Trans. Web*, vol. 4, no. 4, pp. 14:1–14:32, Sep. 2010. [Online]. Available: http://doi.acm.org/10.1145/1841909.1841911

[48] J. Clark, “Trang - multi-format schema converter based on relax ng,” 2013.

[49] J. E. Hopcroft and J. D. Ullman, *Formal Languages and Their Relation to Automata*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1969.

[50] D. Peintner. (2013) EXIficient. [Online]. Available: http://exificient.sourceforge.net/

[51] T. Kamiya. (2013) OpenEXI. [Online]. Available: http://openexi.sourceforge.net/

[52] R. Kyusakov. (2014) Efficient XML Interchange Processor. LTU. [Online]. Available: http://exip.sourceforge.net/

[53] G. Moritz, F. Golatowski, C. Lerche, and D. Timmermann, “Beyond 6lowpan: Web services in wireless sensor networks,” *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 4, pp. 1795–1805, Nov 2013.

[54] F. Simeoni, D. Lievens, R. Conn, and P. Mangh, “Language bindings to XML,” *Internet Computing, IEEE*, vol. 7, no. 1, pp. 19 – 27, 2003.

[55] R. A. van Engelen and K. A. Gallivany, “The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks,” in *2nd IEEE International Symposium on Cluster Computing and the Grid*, 2002, p. 128.

[56] S. Käbisch, D. Peintner, J. Heuer, and H. Kosch, “Efficient and Flexible XML-Based Data-Exchange in Microcontroller-Based Sensor Actor Networks,” in *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, 20-23 2010, pp. 508 –513.

[57] D. Sosnoski. (2014) JiBX: Binding XML to Java Code. [Online]. Available: http://jibx.sourceforge.net/

[58] ——, “Xml and java technologies: Data binding,” *developerWorks—XML or Java Technology. IBM*, 2003.

[59] J. Arnoldus, M. van den Brand, A. Serebrenik, and J. Brunekreef, *Code Generation with Templates*, ser. Atlantis Studies in Computing. Atlantis Press, 2012. [Online]. Available: http://books.google.se/books?id=UvC0MJHSqjkC

[60] J. Eliasson, P. Lindgren, and J. Delsing, “A bluetooth-based sensor node for low-power ad hoc networks,” *Journal of Computers*, vol. 3, pp. 1–10, 2008.

[61] Z. Shelby, S. Krco, and C. Bormann, *CoRE Resource Directory*, IETF Std., Sep. 2013. [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-resource-directory-00

[62] P. Puñal Pereira, J. Eliasson, R. Kyusakov, J. Delsing, A. Raayatinezhad, and M. Johansson, “Enabling cloud connectivity for mobile internet of things applications,” *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, vol. 0, pp. 518–526, 2013.

[63] D. Peintner, H. Kosch, and J. Heuer, “Efficient XML Interchange for rich internet applications,” in *Multimedia and Expo, 2009. ICME 2009. IEEE International Conference on*, 282009-july3 2009, pp. 149–152.