Beyond 6LoWPAN: Web Services in Wireless Sensor Networks

Guido Moritz*, Frank Golatowski, Christian Lerche, and Dirk Timmermann

***Abstract*—To date, Wireless Sensor Networks (WSN) require their own specific methodologies, tools, and technologies. With the rise of 6LoWPAN protocols (i.e., IPv6 over Low Power Wireless Personal Area Networks), WSNs can use IP to share a common network layer with other networks and the Internet for highly interoperable device communication. But additional efforts are necessary to develop new or adapt existing application layer protocols for IP-enabled WSNs. Hence, this paper investigates W3C SOAP Web Services (WS) in the context of WSNs. In particular, it is shown that the Devices Profile for Web Services (DPWS) can be used as an application layer protocol in WSNs.**

This paper presents an improved DPWS architecture that takes into account the requirements of WSNs. For instance, in order to reduce overhead, messages are scaled down by an optimized data encoding based on the emerging W3C Efficient XML Interchange (EXI) format. Moreover, a novel SOAP-binding on top of the IETF Constrained Application Protocol (CoAP) is introduced for an efficient transport of SOAP messages. The paper evaluates the proposed approaches by dedicated implementations of DPWS, EXI, and the SOAP-over-COAP binding. The results claim that the proposed approaches satisfy the requirements of mote class devices with only tens of KB RAM and ROM.

***Index Terms*—6LoWPAN, constrained application protocol (CoAP), device centric, devices profile for web services, efficient XML interchange (EXI), IPv6, sensor networks, service-oriented architecture, service-oriented architectures (SOA), wireless.**

I. Introduction

CURRENT Wireless Sensor Networks (WSN) generally necessitate their own proprietary methodologies, tools, and technologies. Accordingly, their protocols are designed for isolated applications in dedicated scenarios [1]. However, with the appearance of the "Internet of Things" vision, a new way of thinking has appeared. That is to say, it is tried to (re-)use Internet technologies and the well-known layered architecture within WSNs for a seamless integration into existing IT infrastructures. One of the main driving forces in the area of WSNs is the 6LoWPAN (i.e., IPv6 over Low power Wireless Personal Area Networks) protocol that allows the efficient use of IP while taking the tight resources of the devices into account [2], [3].

The objective to find suitable application layer protocols for 6LoWPANs can be achieved by (re-)using and optimizing well-established Internet technologies. With respect to the Internet, two main architectural styles are currently prevalent: REpresentational State Transfer (REST) and Service-Oriented Architectures (SOA). However, how to make the right architectural decision is out of scope of this paper. For the sake of completeness though, a short discussion is given in the related work section. Nonetheless, this paper investigates the application of general Web services by means of SOAP Web services (WS) in WSNs. Decades of usage in the World Wide Web have proven, that SOAP Web services are platform and programming language independent, scalable and secure even in heterogeneous environments. However, for WSNs and thus target platforms with only tens of kB RAM and ROM, current Web service protocols and technologies are not feasible and need adaptations and enhancements as described in this paper.

Initially, in this paper the Devices Profile for Web Services (DPWS), which is currently an OASIS standard [4], is chosen as a suitable subset of Web services protocols for machine-to-machine (M2M) communication. DPWS has an architectural concept that is similar to the Web Service Architecture (WSA), but it is better suited for M2M scenarios. The main difference is, that service discovery can be done via multicast communication instead of querying a central service registry such as UDDI. The service usage on devices is similar to the service usage in WSA, whereby DPWS devices can be directly integrated into WSA-based enterprise systems [5]–[8]. Additionally, DPWS also supports dynamic device and service description, eventing, and security. Research has also shown that DPWS can provide real-time capabilities [9].

While neither DPWS nor SOAP have specifically been developed for devices with highly constrained resources, this paper demonstrates that they can be improved in order to fulfill the requirements of WSNs. Therefore, uDPWS is presented, which is a DPWS-stack running on wireless sensor nodes with only 8 kB RAM and 48 kB ROM. Furthermore, this paper proposes necessary adaptions and enhancements for the use of SOAP Web services in WSNs.

1) An improved DPWS architecture suitable for WSNs.
2) An optimized data encoding based on the W3C Efficient XML Interchange (EXI) format.
3) A new SOAP-over-CoAP binding that is based on the IETF Constrained Application Protocol (CoAP).

The challenge is to (re-)use existing protocols and assemble standards, which have never been intended to be combined. The DPWS specifications have not been developed for WSNs, but

Manuscript received August 09, 2011; accepted April 02, 2012. Date of publication May 16, 2012; date of current version October 14, 2013. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org. Paper no. TII-11-409. *Asterisk indicates corresponding author.*

*G. Moritz is with the Institute of Applied Microelectronics and Computer Engineering, University of Rostock, 18055 Rostock, Germany (e-mail: guido.moritz@uni-rostock.de).

F. Golatowski, C. Lerche, and D. Timmerman are with the Institute of Applied Microelectronics and Computer Engineering, University of Rostock, 18055 Rostock, Germany (e-mail: Frank.Golatowski@uni-rostock.de; christian.lerche@uni-rostock.de; dirk.timmermann@uni-rostock.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2012.2198660

offer degrees of freedom to meet the resource constraints. While the EXI format has been proven to avoid verbosity of traditional XML, it has neither been evaluated in detail for SOAP deployments in WSNs nor be designed to be implemented for sensor node class devices. Both, the evaluation and an implementation of EXI are part of the contribution of this paper. In contrast to EXI, CoAP has been developed for WSNs. CoAP is assumed as holistic application layer protocol, based on RESTful design principals and part of the IETF standards family. But CoAP has not been supposed as SOAP transport protocol as proposed in this paper.

As a proof of concept, implementations of the proposed adaptions and enhancements are presented and thoroughly analyzed. It is shown that these implementations fulfill the platform requirements of WSN class devices.

In this paper, DPWS is used in a simplified home automation application scenario with a DPWS-enabled air conditioner. This scenario is the reference example of all DPWS stacks of the Web Services for Devices initiative (WS4D).[1] The existing DPWS implementations are furthermore used for compatibility tests. Strictly speaking, the reference with the air conditioner demonstrates all main features of DPWS. This includes, besides the functionalities of discovery executing simple requests (e.g., setting a new temperature) or exchange request-response messages (e.g., inquiring the current temperature). Beyond that, the air conditioner implements eventing (i.e., publish/subscribe patterns) to get notified if the temperature changes.

II. Related Work

Research on WSNs commonly focuses on highly optimized solutions for applications with a huge number of nodes distributed in an unknown environment with potentially dozens of communication hops between the source and the sink [1], [10]. Most of the current work on such scenarios focuses on link layer efficiency [11]–[13] or link layer coexistence [14]. In contrast, this paper focuses on application domains like smart grid, factory automation, healthcare, as well as home and building automation. These application domains have very different requirements compared to most of the current research for WSNs. In such domains, protocols need to be optimized for the interaction of devices with services in the Internet [15], [16] rather than to be highly optimized for the use in large multihop networks. Hence, the challenge here is to apply Internet technologies despite the limited resources like computing power, memory, energy, and communication bandwidth.

For classical Web services applications, two prevailing architectural styles exist. First, the Representational State Transfer (REST) which is a result of the Roy Fielding's dissertation [17]. Fielding describes the current World Wide Web communication and its design principles. In particular, the web is mainly browsers requesting resources of servers (i.e., websites) and human-to-machine (H2M) interaction. Second, Service-oriented Architectures (SOA) are characterized by complex functional blocks that are encapsulated and provide more abstract services, whereas services or service implementations can be reused across different applications. SOAs are often used to model and realize complex business flows. A good discussion on REST and SOA is provided by Pautasso [18]. Pautasso argues that comparing SOA and REST is like comparing apples and oranges: REST is an architectural style for the web, whereas SOAs can be used to realize a middleware based on interoperable standards.

The best known RESTful protocol is HTTP, which uses a certain number of standardized methods (e.g., GET, PUT, POST, DELETE) to deal with resources. For SOAs, SOAP is commonly used in conjunction with the WS-* specifications. The large number of WS-* specifications and their extensions facilitates a high flexibility, but also implies a steep learning curve compared to HTTP.

It should be mentioned though that SOAP-based protocols not necessarily lead to a SOA (as well as HTTP does not necessarily lead to a RESTful architecture). For example, the working group of the W3C for Web Services Resource Access[2] specifies SOAP-based and resource-oriented Web services. Similarly, the use of DPWS, which is used as a WS-subset in this paper, does not force the user to build a SOA. Concluding, although DPWS was originally intended to build service-oriented device networks, it can be used for both service-oriented and resource-oriented.

Essentially, SOA and REST are two different architectural styles having different advantages and disadvantages. Thus, the architectural decision mostly depends on the specific application. Accordingly, this paper is beyond the discussion of SOA versus REST. Instead, the paper concentrates on necessary adaptations for existing protocols to meet the resource constraints of WSNs.

The feasibility of using RESTful Web services in an IP-based sensor network (which comprehends multihop communication and low-power) as well as an evaluation of performance and power consumption was published in [19]. The aim of this work was to allow the direct interaction between Web service-based IT systems and IP-based sensor networks. The results claim that Web service requests can be completed below 1 s and with low-power dissipation. In [20], an architecture was presented that integrates real-world embedded devices into the Web by means of a device mashup application. In contrast to [19], the proposed architecture lacks support for IP connectivity for the focused class of devices. Instead, Smart Gateways were used to provide the functionality of the devices through a RESTful API to the Web. Another REST-based architecture was described in [21]. Even though the sensor nodes in this architecture run an embedded IP stack, a gateway is still required to apply a transparent compression and decompression of the data based on JavaScript Object Notation (JSON) [22]. Thus, compressed JSON is used for proxy-to-device communication, while standard JSON is used for proxy-to-Web communication. However, it is presumed that JSON compression can be omitted in favor of an architecture without gateways.

HTTP, the most widespread protocol for RESTful deployments, only defines methods to access resources and requires extensions to fit into device-centric M2M applications. Hence, the

[1]http://www.ws4d.org

[2]http://www.w3.org/2008/08/ws/charter.html

great advantage of using SOAP-based Web services is the seamless integration into common SOAP-based enterprise applications. Furthermore, it allows the application designer to make use of a huge repertoire of WS-* specifications for different use cases. This includes, e.g., WS-Discovery to discover local Web services, WS-Eventing for Publish/Subscribe patterns, WSDL to describe a service and WS-MetadataExchange to retrieve additional service metadata such as the device manufacturer.

Helander [23] analyzed the use of *XML Web services* on highly resource constrained devices. The author argues that *XML Web services* are desirable for the usage in embedded systems. Despite the use of *XML Web services* on devices, the authors assume that the widely used SOAP protocol is not supported by the devices themselves. For this purpose, an additional controller gateway shall be used, whereas DPWS is not considered as a promising approach [23].

Summing up, previous studies analyzed RESTful approaches or gateway architectures to apply Web services for device communication [24]. However, only very few efforts have been made to implement and evaluate SOAP-based Web services directly on highly resource constrained devices.

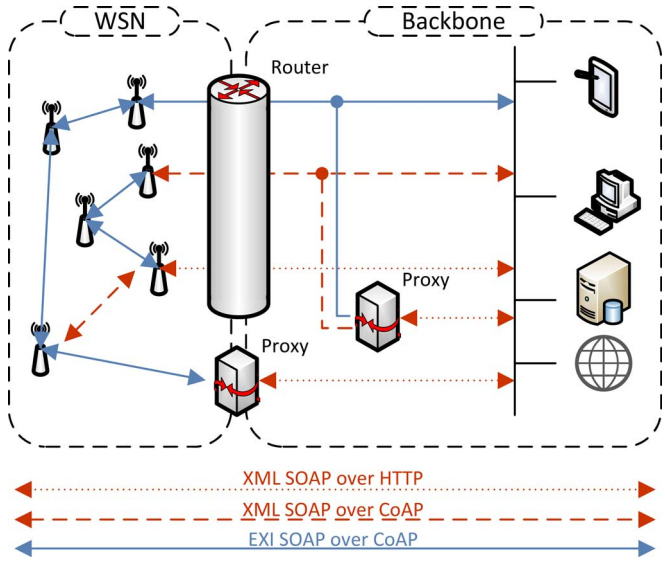

Fig. 1. Messaging scenarios.

III. Network Topology and Architecture

To integrate non-IP-based WSNs into IT infrastructure (e.g., the Internet), heavyweight application layer gateways are required. These gateways must be located at the edge of the WSN and the IT infrastructure. On the WSN side, the gateway may implement proprietary interfaces to collect information or invoke actions. On the IT infrastructure side, the gateway models the capabilities of the WSN, e.g., as services or resources by using IP-based protocols and interfaces. Thus, the gateway is responsible to map the functionalities and mechanisms of the WSN protocols to the external protocols and *vice versa*. In most cases, a change of the WSN application entails a change in the gateway as well. This leads to very inflexible deployments and lowers reusability of implementations over different applications and domains [25].

In IP enabled WSNs (i.e., 6LoWPANs), the network layer part of the gateway can be replaced by a standard IP router. Only a translation of the application data is further necessary. Another benefit is that the application layer gateway not necessarily needs to be located at the edge of the WSN but rather can be located for example on a backend server in the IT infrastructure. But this does not prevent the gateway to be changed when the WSN application changes. Only if the application layer gateway could translate the data seamlessly, without considering semantics and the current state, the gateway could be independent of the application. In this case the gateway becomes a proxy. A proxy does not necessarily take into account the data semantics and the state (a gateway may do). If two data representations can be mapped stateless from one into another, than a proxy can map these two different data representations without considering the semantics. This reduces the complexity significantly and allows a higher flexibility in designing new applications.

For example compliant utf-8 XML, Fast Infoset and EXI are all capable of representing XML-Infoset [26] based documents like SOAP envelopes. But re-encoding one format into another one by a proxy is stateless, can be performed transparent and requires no understanding of the specific document meaning.

Depending on the routing strategy of the underlying network topology, messages may flow explicitly (i.e., both endpoints are aware of the intermediary proxy) or implicitly (i.e., none or not all endpoints are aware of the intermediary proxy) through the proxy. Thereby, proxies are often used to realize caching functionalities and supplement the deployments to unburden the infrastructure load. Advantages of routers and proxies instead of gateways are the high degree of flexibility instead of the limited degrees of freedom of gateways. It should be mentioned though, that intermediary devices like proxies are not necessarily required like gateways are. Endpoints in the backbone network and the WSN may also have a direct end-to-end connection.

In [27], the feasibility of XML compressors for deployments using simple SOAP messaging is discussed, whereby EXI results in the best compression rates. Combined with the CoAP binding described in this paper, compressed SOAP transported over CoAP is shrinking heavyweight WS down to the resource requirements of WSNs. Sensor nodes inside the WSN are likely to use only the highly optimized mechanisms and data representations like compressed XML SOAP-over-CoAP. For external communication, also this compressed XML SOAP-over-CoAP can be used. If sensor nodes are capable of parsing standard XML, SOAP envelopes may be encoded in standard XML, but still carried over CoAP. But also the usage of standard XML encoded SOAP-over-HTTP messaging or compressed XML over HTTP are still possible, because the routers only translate on addressing layer. Fig. 1 depicts a few possible messaging scenarios. Because both EXI and CoAP are designed to map seamless and transparent to compliant XML and HTTP, dedicated proxies may be used to complement the messaging and re-encode external XML SOAP-over-HTTP to internal EXI SOAP-over-CoAP, as depicted in Fig. 1. By using such proxies, existing external implementations are not required to be changed, but still fit into the architecture.

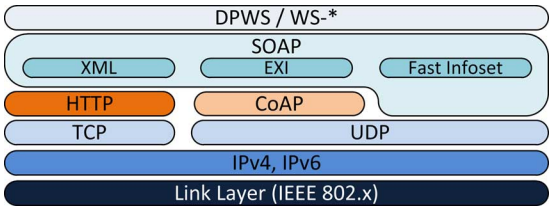

Fig. 2. WS-* stack with CoAP binding.

IV. SOAP-Over-CoAP

Most deployments of SOAP are using the existing SOAP-over-HTTP [28] binding for the transport of SOAP envelopes. The SOAP-over-HTTP binding does not use all HTTP functionalities extensively, but uses HTTP for transport and, e.g., for bypassing firewalls. But SOAP is not exclusively bound to HTTP. For instance, OASIS has defined the SOAP-over-UDP [29] binding. The intention of SOAP-over-UDP is to send SOAP envelopes over IP multicast, which is required for device discovery in the absence of central repositories like UDDI.

Both the HTTP and the UDP binding are not appropriate for use in highly resource constrained networks like 6LoWPANs. On the one hand, TCP suffers from expensive handshakes for establishing and closing connections. On the other hand, UDP datagram messaging is lightweight at the expense of reliability. While CoAP is geared to HTTP, it is based on lightweight UDP but provides at the same time reliable communication. Therefore, this section describes a new and efficient SOAP-over-CoAP binding. Fig. 2 shows the protocol stack with the three feasible SOAP bindings. CoAP defines a detailed mapping procedure of CoAP to HTTP in dedicated proxies. This feature of a CoAP/HTTP proxy can also be used to map the SOAP-over-CoAP to the SOAP-over-HTTP binding for example at the edge router of the 6LoWPAN.

A. Introducing the Constrained Application Protocol (CoAP)

The IETF Constrained RESTful Environments (CoRE) working group[3] is developing a protocol suite around CoAP (i.e., the Constrained Application Protocol) [30]. CoAP is intended to be used in 6loWPAN networks as application layer protocol. In contrast to the approach of (re-)using existing protocols as described in this paper, CoAP is a completely new protocol. Although not identical, CoAP is geared to HTTP. In contrast to HTTP though, CoAP is based on UDP. Due to the unreliable nature of UDP, CoAP separates between the request/response layer (cf. HTTP) and the messaging layer, whereas the former defines well-defined methods (i.e., GET, PUT, POST, DELETE) to be applied on abstract data objects (i.e., resources) on servers. Similar to HTTP, the result of the performed action is indicated by corresponding response codes (e.g., 2.xx, 3.xx, 4.xx). Hence, the messaging layer of CoAP realizes transport reliability on the application layer without the need for heavyweight handshakes. Furthermore, the messaging layer in combination with UDP provides duplicate detection mechanisms and multicast support for CoAP. In addition to the use of reliable UDP instead of TCP, the advantage of CoAP is a more compact message format, because the CoAP header is encoded binary and no string values are used as in HTTP.

While CoAP is currently developed at the IETF, a draft of the SOAP-over-CoAP binding specification has been published in the corresponding IETF CoRE working group to assist in shaping CoAP. If the new proposed SOAP binding gains also industrial interest, finalizing the specification might be performed either in the IETF or in the OASIS WS-DD working group.

B. Usage of CoAP for a SOAP Binding

Most of the required protocol mechanisms are already embedded in DPWS and SOAP by means of the appropriate use of corresponding WS-* specifications. For example, WS-Addressing can be used to assign unique identifiers to messages and to define communication endpoints in the SOAP envelope. CoAP provides several mechanisms which are different in their approach but similar in functionality. Hence, the proposed SOAP-over-CoAP binding makes beneficial usage of CoAP as application layer protocol. For example, successful and failure responses are indicated by the corresponding CoAP response codes (e.g., 2.xx, 4.xx, 5.xx). Nevertheless, similar to the existing SOAP-over-HTTP binding, the CoAP binding is not intended to fully exploit the features of CoAP. Among those features, which are not required for this binding, are certain discovery and eventing (cf. publish/subscribe) features of CoAP because they are already covered by, e.g., WS-Discovery and WS-Eventing.

C. Messaging Patterns

DPWS deployments support in general two different message exchange patterns (MEP). The first pattern is the one-way MEP that consists of only one SOAP message for a request without a SOAP envelope in the response. Thus, when a one-way MEP is sent using the datagram-oriented SOAP-over-UDP binding, no feedback is returned (neither success nor error). By contrast, HTTP is connection-oriented due to TCP. HTTP is also based on a request/response pattern which is independent of the DPWS MEP. Thus, for the HTTP binding, the HTTP response header is returned to the sender even in case of the one-way MEP. Hence, the SOAP-over-HTTP binding may indicate success or failures due to the HTTP response codes. The CoAP binding provides similar mechanisms like HTTP and thus also the possibility for feedback to the origin sender of the request in absence of a corresponding SOAP response. Therefore, in a one-way MEP, the request carries a SOAP envelope and the response consists only of the CoAP response header.

The second pattern, the two way MEP contains a SOAP envelope in both request and response messages. While the UDP binding includes the SOAP envelopes directly in the datagram, the HTTP binding supplements the HTTP request and response headers with the corresponding SOAP envelopes. Because CoAP is also based on the request/response pattern, it can operate similar to HTTP and embed the corresponding SOAP envelopes in the message body. Hence, in a two-way MEP, the CoAP response header is additionally supplemented by a SOAP response envelope.

In addition to the request/response layer, the CoAP messaging layer differentiates between confirmable (CON) and

[3]http://tools.ietf.org/wg/core/

non-confirmable (NON) messages. CON messages must be acknowledged, while NON only consist of a single message without acknowledgment. To achieve the required reliable message transport, the SOAP-over-CoAP binding should use the CON feature of CoAP. For unreliable messaging, the existing SOAP-over-UDP binding might be sufficient.

CoAP uses message ids to relate CON messages and acknowledgments as well as for the duplicate detection. The IDs are unique for each message exchange. In order to match requests and responses, the *CoAP Token Option* is used. Consider that *WS-Addressing* defines the *MessageID* property to identify and match SOAP requests/responses in time and space and thus provides the same mechanism as the *CoAP Token Option*. The *CoAP Token Option* is much more compact by providing equal functionality like the *WS-Addressing:MessageID*. In case of using the CoAP binding, the *WS-Addressing:MessageID* property can be left empty and must only contain a value when the *CoAP Token Option* is not present or not sufficient. The *WS-Addressing:MessageID* property has a typical size of 45 bytes and cannot be compressed significantly because the value of this property is unique for each request/response. Hence, the intention of this adaptation is to reduce message size.

D. Serialization

To reduce overhead of the XML data representation, the emerging EXI [31] encoding is a qualified candidate as presented by Moritz *et al.* in [27]. Both EXI and standard XML are based on XML-Infoset [26] and thereby interoperable. Hence, the SOAP-over-CoAP binding only requires the capability to represent XML-Infoset documents. This includes utf-8 XML to attain interoperability with existing SOAP implementations and compressed XML (such as EXI). The message serialization is indicated by the media-type identifier in the CoAP header.

E. Endpoint Identification

WS-Addressing defines an *anonymous* URI that can appear in the *address* property of an endpoint reference. In absence, the implied value of the *reply endpoint* property is an endpoint reference with an *address* property carrying the *anonymous* value. If the *reply endpoint* property of a SOAP request transmitted over CoAP has an *address* property with this value, the UDP source address and port are considered to be the address to which reply messages should be sent. If the *address* in the *reply endpoint* property is different from that value, the response is intended to be sent to an endpoint different from the origin requester.

Implementations of the CoAP binding must be aware of that difference. Hence, it might be required to send the acknowledgment of a CON message to the origin requester and to use an additional message exchange to send the CoAP and SOAP responses to the third party endpoint indicated in the *reply endpoint*.

V. Data Representation and Encoding

Concerning overhead, the W3C Web services specifications suffer from the usage of SOAP for messaging and thus XML utf-8 based data representation. While utf-8 encoded XML is often used in favor of platform independency, nodes in WSNs are not expected to provide complex services with high data rate. Hence, the overhead due to the data representation is a critical aspect for WSN applications. Consider for instance the comparatively small payload of a single sensor reading.

The W3C Efficient XML Interchange (EXI) Working Group has developed an encoding that allows efficient interchange of XML Information Set documents [31]. A major design decision in EXI is the usage of XML schema information for the encoding. Therefore, both endpoints must use the same set of schema files to generate the EXI grammar. With such a schema at hand, each string or value used (e.g., tag and attribute names or values which are already known through the schemas) is then encoded binary as an EXI event. The receiver is capable of interpreting the EXI event code correctly, only if the same schema information is used. Hence, all other values that are not known by the grammar are carried inline as strings in the resulting EXI stream. The endpoints may use either EXI internal or external mechanisms (e.g., prior defined) to determine the used schema set. However, highly resource constrained devices are not capable of parsing and processing dynamically changing schema sets during runtime. Hence, the EXI grammar must be generated and integrated in the nodes at compile time, as described in [32].

Based on the results in [27], this paper presents an approach to use further device and service specific XML schema information for the EXI encoding. Even if there are more candidates for the required message compression and encoding, such as Fast Infoset (FI) [33], EXI results in best compression rates (cf. [27]).

The EXI format is designed to map seamless and stateless to conventional utf-8 XML documents. Hence, encoding and decoding can be performed transparently and *on the fly* between the communication endpoints. That is to say, the endpoints do not require any knowledge of the encoding and decoding along the communication path. Therefore, combined with the above described SOAP-over-CoAP binding, a seamless and transparent integration of WSNs (using optimized EXI encoded SOAP-over-CoAP messaging) into traditional XML encoded SOAP-over-HTTP deployments is possible. The mapping can for instance be performed on dedicated proxies located at the edge routers that connect the 6LoWPAN with the backbone network.

A. XML Schema Set Adaptations

Optimizing the used XML schemas for the knowledge-based schema informed encoding mode of EXI has significant impact on the resulting message sizes. In particular, this means for DPWS to use *xs:restriction* enumerations for tag and attribute values containing mainly well-known URIs.

Using the enumerations avoids carrying full strings inline and requires only transmitting the corresponding binary EXI grammar code. The required URI enumerations for DPWS occur in the attribute and tag values of, for example, *WS-Addressing:Action*, *WS-Addressing:To*, *WS-Eventing:Mode*, *WS-Eventing:Dialect*, and *WS-MetadataExchange:MetadataSection:Dialect*.

B. DPWS and Application Specific Adaptations and Enhancements

Independent of the described general approach, further DPWS and application specific adaptations and enhancements are possible to reduce message sizes.

DPWS uses WS-Addressing to decouple transport and network specific layers as well as information from the addressing logic in the application layer. The *WS-Addressing:MessageID* and *WS-Addressing:RelatesTo* tags are used to distinguish messages during asynchronous message exchange and are by default of type *xs:anyURI*. *MessageID* and *RelatesTo* are optimistically random and must always be carried inline, because the dynamic changes cannot be described in the XML schemas. Highly resource constrained sensor nodes though are not expected to handle a huge number of requests and responses at the same time due to missing memory for buffering. Thus, a much more efficient data type like *xs:long* can be used for these purposes. If used in conjunction with the previously described SOAP-over-CoAP binding, the mechanism might be replaced by a CoAP internal mechanism and might thus not be required at all.

Depending on the scenario, deployment, network topology and dynamics of the network, further application specific information might be additionally included in the schema set. In particular, this concerns, for example:

- WS-Addressing:Address:
  Due to the usage of WS-Addressing, DPWS devices have this transport and network layer independent address, which might be constant.
- WS-Discovery:XAddrs:
  The transport specific network address. Depending on the mechanism to obtain the network address, this value might be constant.
- WS-Discovery:MetadataVersion:
  Indicates changes in Metadata (e.g., changes in network addresses, hosted services…).
- WS-Discovery:Types:
  Device types implemented in this DPWS device.
- WS-Discovery:Scopes:
  Scopes of the DPWS device.
- DPWS:ThisDevice and ThisModel
  Metadata of the DPWS device.
- WS-Eventing:Filter and Dialect:
  Possible Filter types and dialects.

C. Bootstrapping Message Encoding

The common problem of all general adaptations described above and those adaptations required by specific applications is the need for the identical schema set at both endpoints. In dynamic, heterogenous and cross domain environments, more generic schema sets may be necessary at the price of increased message size and overhead. However, static scenarios may exploit the benefits of dedicated adapted schema files.

Nevertheless, nodes may be capable to distinguish different schema sets at runtime (which were defined and included at compile time). For example, a node may use a generic set, containing only basic DPWS schemas, for discovery purposes. During the discovery, the node advises the other endpoint via internal mechanisms of DPWS to switch to an extended schema set for service invocations. The different required schemas may be provided either by the nodes directly or the node points to external resources to download the additional schemas.

The high degree of flexibility of EXI is the basis for the wide range of applications and scenarios for this optimized encoding. It depends on the specific scenario though, which solution satisfies the needs best and which schemas and adaptations are most appropriate. It should ideally be the responsibility of vendors or domain specific standardization bodies to define dedicated profiles (including schemas) to guarantee compatibility.

VI. Implementations

To evaluate the feasibility of the approach described herein, compliant implementations of a DPWS stack and an EXI parser for highly resource-constrained sensor nodes have been developed. The DPWS stack is called uDPWS and is available as Open Source. uDPWS is for instance running on TelosB modes with 48 kB ROM and 8 kB RAM, supports utf-8 XML message processing and features the SOAP-over-UDP, SOAP-over-CoAP and SOAP-over-HTTP bindings. The EXI parser is called uEXI and is a standalone EXI stream parser. uEXI will be included in uDPWS in future versions. In the remainder of this section, implementation details will be given and a thorough analysis of the performance is presented.

A. Implementation of uDPWS and SOAP-Over-CoAP

Not to start from the scratch, for the implementation of uDPWS, the Open Source operating system Contiki [34] was used. Contiki already includes 6LoWPAN, TCP and, UDP network stacks [35]. Implementation details of uDPWS are available on the project website.[4] The uDPWS implementation is compliant with current WS-DD specifications [4] in most points. Not implemented features of the WS-DD specifications are, e.g., related to security mechanisms due to missing SSL libraries that would not fit the platform requirements. uDPWS was compared to and tested with other existing WS4D DPWS stacks[5] that have passed the OASIS WS-DD interoperability scenarios.

For the SOAP-over-CoAP binding, the existing CoAP stack implementation described in [36] was used. For testing uDPWS and the CoAP binding, SOAP-over-CoAP was additionally implemented in the existing WS4D-gSOAP toolkit [37], which is a DPWS toolkit for embedded systems and is based on the well-known gSOAP stack [38].

Sensor nodes with very tight resource constraints are not expected to provide complex services and methods. Hence, analyzing only a subset of available SOAP header fields is sufficient. Other vendor or deployment specific header fields can similarly be analyzed. The uDPWS processing flow is depicted in Fig. 3. Because the outgoing messages of the DPWS node are related to the capabilities and functionalities of the node (i.e., the physical device), most parts of these messages are already known at design time and are static over multiple service invocations. Only a few changes of the SOAP messages are required at runtime. Therefore, the generation of responses is based on a

[4]http://code.google.com/p/udpws/

[5]Web Services for Devices (WS4D), http://www.ws4d.org

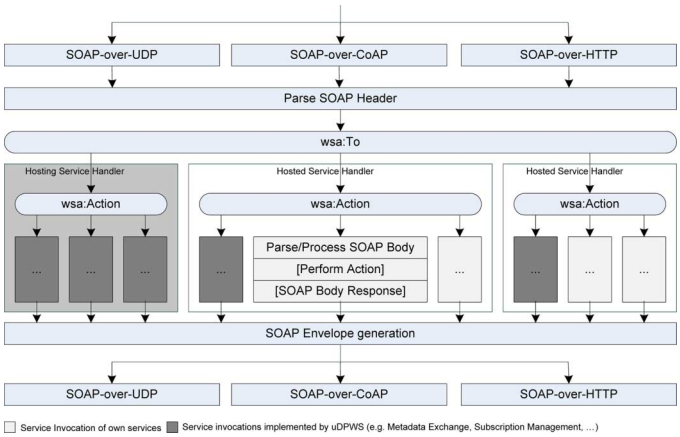

Fig. 3. uDPWS processing flow.

TABLE I
ROM USAGE OF UDPWS IN BYTE

	HTTP binding	CoAP binding	CoAP and HTTP binding
System	1602	1603	1572
Contiki	26394	23254	26394
Service Modules	1978	2002	2467
Device Modules	686	686	686
uDPWS Core	10206	12458	14216
Total	**40866**	**39702**	**45335**

lookup table (cf. [39]). The required fragments for the static response messages are generated at design time by a dedicated code generator and are then included at compile time.

Based on the parsed SOAP header fields, the SOAP body is processed by the corresponding module. The modules are responsible for processing the SOAP body, performing required actions, and generating the corresponding response if required. The modularity allows tailoring specific implementations by omitting modules at design time (which are not required at runtime).

The chosen hardware platforms are the Crossbow TelosB[6] and the Atmel Raven 2.4GHz Evaluation and Starter Kits.[7] However, in order to discuss the results in detail, only the results of the TelosB platform are presented here because the TelosB platform results in lower footprint and better performance.

The ROM usage of the uDPWS stack and the related underlying Contiki operating system is shown in Table I. The table shows three different configurations: 1) uDPWS width HTTP binding; 2) with CoAP binding, and 3) with both bindings. The UDP binding is always included, because it is required for discovery purposes. The service modules include all implementations of application-specific invocations. The overall footprint is lowest for the configuration with only the CoAP binding. The reason is that TCP is not required at all and thus can be removed completely at compile time. Nevertheless, the uDPWS ROM usage is larger than in the HTTP only binding. This is based on the used CoAP stack, which was not optimized further, but used from the shelf. Because SOAP-over-CoAP uses only a very minor subset of CoAP, optimizations are possible here.

[6]http://www.xbow.com/

[7]http://www.atmel.com/

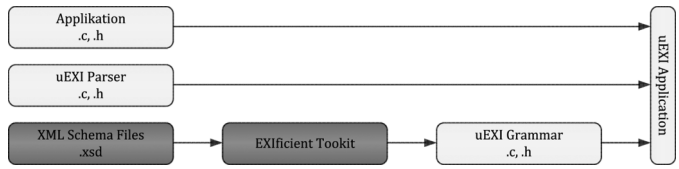

Fig. 4. uEXI.

TABLE II
ROM FOOTPRINT OF UEXI

Included Schemas	ROM usage in Byte
none (uEXI parser only)	1.088
+SOAP, XML, XML-Schema, XML Namespace	4.372
+WS-Addressing	7.002
+WS-Eventing	10.588
+WS-MetadataEchange	11.672
+WS-Discovery	15.246
+DPWS	17.692
+AirConditioner (Application)	18.000

B. Implementation of uEXI

The uEXI implementation is a lightweight EXI parser that satisfies the needs of sensor nodes with highly constrained resources. As uDPWS, uEXI is implemented in plain C and can be deployed on platforms like the TelosB. However, in contrast to uDPWS that is bound to Contiki, uEXI is not bound to a specific operating system. The parser is capable of parsing bit and byte aligned schema-informed and schema-less EXI streams both in strict and nonstrict mode.

The uEXI implementation consists of the parser itself and the specific EXI grammar that is generated from the XML schema files. The parser implements general functions as for data type-specific parsing functions (strings, qualified names, integers, unsigned integers, Booleans, and dates), as well as functions for reading the EXI event codes depending on context and coding mode. The EXI grammar is generated during the compile time using the existing EXIficient[8] implementation. The grammar is extracted from EXIficient by a newly developed code generator that produces the required C code. The C code is then compiled into the resulting EXI parser (see Fig. 4). The grammar is realized as a finite-state machine, while the EXI events define the transitions between the states. Callback functions can be registered in uEXI for each state, representing the occurrence of a specific tag, attribute name or their values.

Table II shows the resulting footprint of the uEXI implementation, compiled for MSP430 (i.e., TelosB) class motes with a gcc-based compiler. The first column refers to the included XML schemas, while each row includes the former schemas and the one of the current row. The right column refers to the resulting binary size of the parser and grammar. The RAM usage of uEXI highly depends on the message structure and on the number of inline strings, which have to be learned during the message parsing. It is important to notice that the footprint in Table II refers to the complete grammar, which results from the used schema files of all related WS-* specifications. For dedicated applications and deployments, most of the grammar en-

[8]http://exificient.sourceforge.net/

TABLE III
EXI ENCODED MESSAGES OF THE AIR CONDITIONER SCENARIO

Message Type	XML	Basic Bit aligned	Basic Byte aligned	Extended Bit aligned	Extended Byte aligned
Hello	1051	146	179	34	68
Probe	626	17	36	17	36
Probe Match	1046	83	119	38	74
Dirceted Probe	501	14	28	14	28
Directed Probe Match	1022	83	117	38	72
Resolve	649	63	84	18	39
Resolve Match	892	145	176	34	65
Bye	713	73	97	27	52
Get Device Metadata	420	61	78	14	28
Get Device Metadata Response	2040	391	464	50	128
Invoke 1-Way	370	74	87	10	21
Invoke 2-Way	492	88	106	14	29
Invoke 2-Way Response	542	91	114	16	36
Event Subscribe	815	112	134	44	69
Event Subscribe Response	745	123	146	64	87
Event Delivery	538	87	110	16	36
Event Unsubscribe	538	62	78	62	78
Event Unsubscribe Response	374	10	21	10	21
Average	743.0	95.7	120.8	28.9	53.7
Minimum	370.0	10.0	21.0	10.0	21.0
Maximum	2040.0	391.0	464.0	64.0	128.0

tries are not required and can be removed, which will cause a significant decrease in terms of ROM usage.

Although uEXI is currently operating as a standalone parser on the nodes and is not integrated in uDPWS, Table II clearly shows that EXI is well suited to operate on highly resource-constrained devices. It depends on the dedicated scenario and used hardware platform, which configuration of uEXI should be used, because the number and complexity of the required schemas have on the one hand a huge impact on the resulting footprint, but on the other hand also a huge impact on the resulting message size.

VII. EVALUATION OF RESULTS

The above described implementations have been used to evaluate the results and measure the resulting performance.

A. Message Encoding

In Section V, an encoding based on EXI is described. The results of the encoding are presented in Table III and are based on two different sets of XML schemas. The generic *basic* XML schema set consist of schemas published along with the core DPWS specification and related WS-* specifications. For the basic EXI grammar, the required XML schema documents were only optimized as discussed above in order to include tag and attribute values in the grammar that are already known by DPWS and related specifications. Furthermore, an *extended* XML schema set is used with all possible adaptations in the XML schema files as discussed in Section V. This also includes application-specific enhancements. Both the basic and the extended set are encoded in EXI bit-aligned and byte-aligned mode.

The results presented in Table III refer to the most compact encodings of the messages within reference scenario of the air conditioner. The message sizes can vary depending on the dynamics of the dedicated deployment. Thus, resulting message sizes of other deployments might be between the presented values of the extended and the basic set. For example, if the *WS-Addressing:Address* is not expected to be a stable URN, it may not be included in the extended set or included but not used for encoding.

If the presented SOAP envelopes are transported over the proposed SOAP-over-CoAP binding, around about 4–20 bytes are required additionally for the CoAP headers. The lower layers below the application layer (i.e., TCP, UDP, IP) are the same for all application layer protocols, and thus out of scope for the evaluations within this paper.

The average message size within the scenario using the basic schema set with EXI bit-aligned mode is 12.3% (min. 2.7%, max. 20%) compared to the initial XML utf-8 encoding. In EXI byte aligned mode, using the basic schema set, the messages are on average 39,8% (min. 17,6%, max. 111,8%) bigger than in EXI bit aligned mode. By using the proposed, extended schema

set (see columns five and six), message sizes are further reduced significantly. This concerns specifically those messages with payloads that are static during the life cycle of a device (e.g., Get Device Metadata Response). The messages using the extended schema set are on average reduced down to 4% (min. 2.5%, max. 11.5%) with the EXI bit-aligned mode, respectively, down to 7.2% (min. 5.6%, max. 14.5%) with the EXI byte-aligned mode. These relations are given with respect to the initial XML utf-8 encoding (second column of Table III). The maximum message size using the extended schema set is 64 byte in bit-aligned mode and 128 byte in byte-aligned mode, whereas the maximum of 128 byte in the byte-aligned mode is required for the *Get Device Metadata Device* message. In comparison, the second largest message size in the same mode is only 87 byte.

The compressed mode of EXI, where the generic DEFLATE algorithm is applied on the resulting EXI document, is not presented in these evaluations. This compression mode implies usage of the byte-aligned mode of EXI, because DEFLATE exploits from redundancies in the byte stream of the data. But the used messages are structured too simple for many redundancies, whereby the compressed mode results in non-optimal message sizes.

In summary, it is possible to drastically reduce message sizes within the realized scenario in order to meet the bandwidth limitations of WSNs. However, the XML schema files must be adapted carefully to achieve the best tradeoff between implementation footprint, resulting message sizes, and application-specific optimizations. It is important to notice that the encoded messages used for the evaluations are still compliant SOAP envelopes that can be re-encoded in compliant utf-8 XML. Hence, existing implementations do not have to be changed and the re-encoding can be performed in dedicated stateless and transparent proxies.

B. Performance Measurements

The overall performance of a WSN running 6LoWPAN is not only related to the used application layer protocol for data exchange, but also on the necessary efforts for control and management. This concerns, for example, the used Neighbor Discovery mechanisms for the IPv6 address configuration, the used routing protocol, the link layer, and the specific network topology. These mechanisms of the lower layers are important, for instance, for the expected overall lifetime of the WSN. Since such mechanisms are the same for different application layer protocols and since this paper concentrates on the SOAP Web services on the application layer, the lower layers are out of scope for the evaluations.

For the measurements with uDPWS, a Linux desktop computer was used as DPWS client. As server/device, uDPWS was used running on the mentioned TelosB sensor nodes. The desktop computer was also used as border router using the Contiki Serial Line IP (SLIP) interface. All presented time values in this section refer to a single-hop scenario with direct communication between the border router and the sensor node running uDPWS. In multihop scenarios, control and management communication increases significantly, which reduces the available bandwidth. Hence, the single-hop scenario gives closer insights into the data communication itself.

TABLE IV
SAMPLE ROUNDTRIP AND PROCESSING TIME UDPWS

Message Type	Request	Response	Roundtrip	Processing
Probe HTTP	721 byte	1217 byte	420.3 ms	12.8ms
Probe CoAP	653 byte	1133 byte	311.2 ms	11.7ms
Probe UDP	645 byte	1101 byte	305.5 ms	11.2ms
Service Invocation 2-Way HTTP	623 byte	729 byte	313.2 ms	8.9ms
Service Invocation 2-Way CoAP	555 byte	646 byte	216.3 ms	8.1ms

1) uDPWS Round Trip Measurements: Table IV presents the round trip times between the client and the server of two exemplary messages within the air conditioner scenario. The rows are separated for the usage of the SOAP-over-HTTP, SOAP-over-CoAP, and SOAP-over-UDP bindings. Table IV also contains the processing time that the uDPWS stack requires to process a request and to generate the corresponding response on the node. Despite the usage of compliant XML messaging and the associated complex XML parsing, the processing time itself is negligible compared to the transmission time. The high transmission times originate from the verbose SOAP-over-HTTP binding and the corresponding overhead of the utf-8 XML encoding.

The message sizes here differ from the values presented in Table III because the EXI encoding was changed based on the different messages to conform with the optimizations described in this paper. More precisely, the sizes in Table IV refer to values that conform with the WS4D stacks used as clients, and these do not feature EXI and the presented optimized EXI encoding format.

2) Impact of EXI: The uEXI parser is not integrated in uDPWS yet. uEXI is currently a standalone parser for WSN nodes and running on platforms like the TelosB. However, the previous figures have shown that the footprints of the implementations meet the requirements of nodes with only tens of kB RAM and ROM. To estimate the general impact of optimized EXI encoding and of UDP in favor of TCP, presenting separated measurements of one scenario is avoided in this subsection. Instead, Fig. 5 presents the results of an echo test.

For the echo test, the desktop PC was used as client that generated random data for the transmission to the sensor node. The node responded with exactly the same data. Fig. 5 also includes the theoretical transmission time of 2.4GHz IEEE 802.15.4 running with 250KBit/s. The resulting data rate for UDP and TCP is nearly equal, but TCP exhibits outliers with every multiple of 1280 bytes (cf. MTU of IPv6). The offset of the resulting linear approximation is caused by the TCP handshake during the initialization of a connection. The difference to the theoretical data rate of IEEE 802.15.4 can be explained by the header sizes of the lower layer that are not considered, the device drivers (e.g., SLIP) and the design of the used hardware platform. More information of theoretical throughput even in multihop scenarios is available in [40].

Moreover, Fig. 6 depicts the ratio of the measured data rate of UDP and TCP. This ratio is independent of any specific delays of driver design. Particularly, for small data sizes, UDP requires less than half of the transmission time compared to TCP since no handshake is required and since the transport layer uses

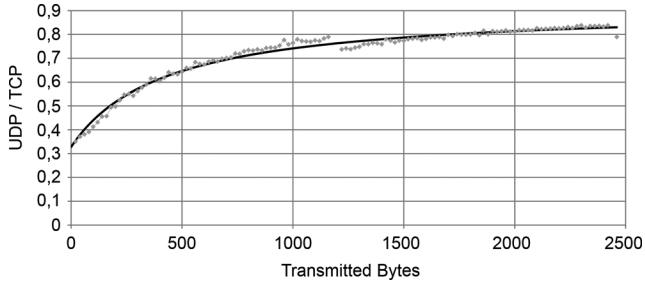

Fig. 6. Ratio UDP/TCP.

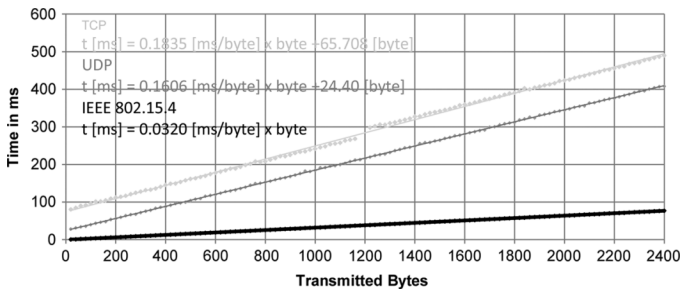

Fig. 5. Comparison UDP/TCP.

smaller headers. The TCP window size of the sensor node running Contiki was set to 1280 byte. Smaller window sizes have considerable influence on performance, because Contiki uses a single buffer and every frame must be acknowledged before the next frame can be transmitted.

With respect to the above presented compression rates for EXI, assuming on an average less than 100 bytes to be transmitted is a proper assumption. Based on the echo test with a current state-of-the-art operating system for WSNs, the impact of using EXI instead of native XML can be easily estimated.

VIII. Conclusion

WSNs are the outmost benchmark for the scalability and efficiency of protocols. With the rise of IP for WSNs (i.e., 6LoWPAN), a common layer for all classes of devices, including higher valued services as found in the Internet, is available. While DPWS has been shown applicable as a cross domain protocol, it is still restricted to embedded devices. In contrast to embedded devices though, WSN platforms are much more resource-constrained. However, DPWS already features important protocol mechanisms such as scalable discovery, security and eventing. Thereby, DPWS is a qualified candidate to be (re-)used as application layer protocol also in 6LoWPAN networks.

While neither DPWS nor SOAP have been specifically developed for WSNs, this paper shows that they are applicable. Even though several enhancements were presented for WSNs, the proposed approach described in this paper does not require any changes of existing implementations on resource-rich devices. Thereby, new applications become possible that directly incorporate WSNs within existing networks. These new applications are using one and the same comprehensive technology, whereby cross domain reusability is improved.

Acknowledgment

We would like to thank M. Rethfeldt, M. Gotzman, and B. Beichler, who have helped implementing and shapping uDPWS and the uEXI parser.

References

[1] B. Raman and K. Chebrolu, "Censor networks: A critique of sensor networks from a systems perspective," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 75–78, 2008.

[2] J. Hui and D. Culler, "IPv6 in low-power wireless networks," *Proc. IEEE*, vol. 98, no. 11, pp. 1865–1878, 2010.

[3] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 packets over IEEE 802.15.4 networks," , RFC 4944 (Proposed Standard), Internet Engineering Task Force, Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4944.txt

[4] *Web Services Discovery and Web Services Devices Profile (WS-DD)*, OASIS Open, 2009. [Online]. Available: http://www.oasis-open.org/committees/ws-dd

[5] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. Souza, and V. Trifa, "Soa-based integration of the Internet of things in enterprise services," in *Proc. IEEE Int. Conf. Web Services, ICWS'09*, 2009, pp. 968–975.

[6] D. Savio and S. Karnouskos, "Web-service enabled wireless sensors in SOA environments," in *Proc. IEEE Int. Conf. Emerging Technol. Factory Autom., ETFA'08*, 2008, pp. 952–958.

[7] H. Bohn, F. Golatowski, and D. Timmermann, "Dynamic device and service discovery extensions for WS-BPEL," in *Int. Conf. Service Syst. Service Manage.*, 2008, pp. 1–6.

[8] G. Candido, A. Colombo, J. Barata, and F. Jammes, "Service-oriented infrastructure to support the deployment of evolvable production systems," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 759–767, Nov. 2011.

[9] T. Cucinotta, A. Mancina, G. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A real-time service-oriented architecture for industrial automation," *IEEE Trans. Ind. Informat.*, vol. 5, no. 3, pp. 267–277, Aug. 2009.

[10] N. Freris, H. Kowshik, and P. Kumar, "Fundamentals of large sensor networks: Connectivity, capacity, clocks, and computation," *Proc. IEEE*, vol. 98, no. 11, pp. 1828–1846, 2010.

[11] G. Anastasi, M. Conti, and M. D. Francesco, "A comprehensive analysis of the MAC unreliability problem in IEEE 802.15.4 wireless sensor networks," *IEEE Trans. Ind. Informat.*, vol. 7, no. 1, pp. 52–65, Feb. 2011.

[12] Z. Hanzalek and P. Jurcik, "Energy efficient scheduling for cluster-tree wireless sensor networks with time-bounded data flows: Application to IEEE 802.15.4/Zigbee," *IEEE Trans. Ind. Informat.*, vol. 6, no. 3, pp. 438–450, Aug. 2010.

[13] A. Koubaa, R. Severino, M. Alves, and E. Tovar, "Improving quality-of-service in wireless sensor networks by mitigating hidden-node collisions," *IEEE Trans. Ind. Informat.*, vol. 5, no. 3, pp. 299–313, Aug. 2009.

[14] L. L. Bello and E. Toscano, "Coexistence issues of multiple co-located IEEE 802.15.4/Zigbee networks running on adjacent radio channels in industrial environments," *IEEE Trans. Ind. Informat.*, vol. 5, no. 2, pp. 157–167, May 2009.

[15] T. Sauter and M. Lobashov, "End-to-end communication architecture for smart grids," *IEEE Trans. Ind. Electron.*, vol. 58, no. 4, pp. 1218–1228, Apr. 2011.

[16] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based Internet of things: Discovery, query, selection, and on-demand provisioning of web services," *IEEE Trans. Services Comput.*, vol. 3, no. 3, pp. 223–235, 2010.

[17] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, CA, 2000, Chair-Richard N. Taylor.

[18] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. big web services: Making the right architectural decision," in *Proc. 17th Int. Conf. World Wide Web, WWW'08*, New York, 2008, pp. 805–814. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367606

[19] D. Yazar and A. Dunkels, "Efficient application integration in IP-based sensor networks," in *Proc. 1st ACM Workshop on Embedded Sensing Syst. Energy-Efficiency in Buildings, BuildSys'09*, New York, 2009, pp. 43–48. [Online]. Available: http://doi.acm.org/10.1145/1810279.1810289

[20] D. Guinard, "Towards the web of things: Web mashups for embedded devices," in *Proc. WWW'09*, 2009.
[21] X. Jiang, S. Dawson-Haggerty, and D. Culler, "SMAP: Simple monitoring and actuation profile," in *Proc. 9th ACM/IEEE Int. Conf. Inform. Process. Sensor Netw., IPSN'10*, New York, 2010, pp. 374–375. [Online]. Available: http://doi.acm.org/10.1145/1791212.1791261
[22] D. Crockford, "The Application/json Media Type for Javascript Object Notation (JSON)," RFC 4627 (Informational), Internet Engineering Task Force Jul. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4627.txt
[23] J. Helander, "Deeply embedded xml communication: Towards an interoperable and seamless world," in *Proc. 5th ACM Int. Conf. Embedded Softw., EMSOFT'05*, New York, 2005, pp. 62–67. [Online]. Available: http://doi.acm.org/10.1145/1086228.1086241
[24] T. Sauter and M. Lobashov, "How to access factory floor information using internet technologies and gateways," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 699–712, Nov. 2011.
[25] Z. Shelby, "Embedded web services," *IEEE Wireless Commun.*, vol. 17, no. 6, pp. 52–57, Dec. 2010.
[26] R. Tobin and J. Cowan, "XML information set," W3C, W3C Recommendation2nd ed. Feb. 2004. [Online]. Available: http://www.w3.org/TR/2004/REC-xml-infoset-20040204
[27] G. Moritz, D. Timmermann, R. Stoll, and F. Golatowski, "Encoding and compression for the devices profile for web services," in *Proc. IEEE 24th Int. Conf. Advanced Inform. Networking Appl. Workshops (WAINA)*, Apr. 2010, pp. 514–519.
[28] J.-J. Moreau, M. Gudgin, M. Hadley, N. Mendelsohn, Y. Lafon, A. Karmarkar, and H. F. Nielsen, "SOAP version 1.2 part 2: Adjuncts, W3C, W3C Recommendation2nd ed. Apr. 2007. [Online]. Available: http://www.w3.org/TR/2007/REC-soap12-part2-20070427
[29] R. Jeyaraman, "Soap-Over-UDP," Version 1.1, OASIS WS-DD TC, 2009.
[30] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, Constrained application protocol (COAP), IETF Draft, 2011. [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-coap-06
[31] *Efficient XML Interchange (EXI) Format 1.0 (W3C Candidate Recommendation)*, W3C 2009.
[32] S. Kaebisch, D. Peintner, J. Heuer, and H. Kosch, "Efficient and flexible XML-based data-exchange in microcontroller-based sensor actor networks," in *Proc. 24th Int. Conf. Advanced Inform. Networking Appl. Workshops (WAINA)*, 2010, pp. 508–513.
[33] *Recommendation X.891—Generic Applications of ASN.1—Fast Infoset,* International Telecommunication Union (ITU), 2005.
[34] A. Dunkels, B. Grnvall, and T. Voigt, "Contiki—A lightweight and flexible operating system for tiny networked sensors," in *Proc. 1st IEEE Workshop Embedded Networked Sensors (Emnets-I)*, Tampa, FL, Nov. 2004. [Online]. Available: http://www.sics.se/~adam/dunkels04contiki.pdf
[35] 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 *Proc. 6th ACM Conf. Networked Embedded Sensor Syst. (ACM SenSys'08), poster session*, Raleigh, NC, Nov. 2008. [Online]. Available: http://www.sics.se/~adam/durvy08making.pdf
[36] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power COAP for Contiki," in *Proc. 8th IEEE Int. Conf. Mobile Ad-Hoc Sensor Syst.*, Oct. 2011, vol. 1, pp. 855–860.
[37] E. Zeeb, G. Moritz, D. Timmermann, and F. Golatowski, "Ws4d: toolkits for networked embedded systems based on the devices profile for web services," in *Proc. 39th Int. Conf. Parallel Process. Workshops (ICPPW'10)*, 2010, pp. 1–8.
[38] R. A. V. Engelen, "A framework for service-oriented computing with C and C++ web service components," in *ACM Trans. Internet Technol.*, May 2008, vol. 8, pp. 12:1–12:25. [Online]. Available: http://doi.acm.org/10.1145/1361186.1361188
[39] G. Moritz, S. Pruter, D. Timmermann, and F. Golatowski, "Web services on deeply embedded devices with real-time processing," in *Proc. 13th IEEE Int. Conf. Emerging Technol. Factory Autom. (ETFA'08)*, Sep. 2008, pp. 432–435.
[40] F. Oesterlind and A. Dunkels, "Approaching the maximum 802.15.4 multi-hop throughput," in *Proc. 5th ACM Workshop on Embedded Networked Sensors (HotEmNets'08)*, Jun. 2008. [Online]. Available: http://www.sics.se/~adam/osterlind08approaching.pdf

Guido Moritz received the Diploma degree in electrical engineering in 2007 from the University of Rostock, Rostock, Germany. He has been working towards the Dr.-Ing. degree in electrical engineering at the University of Rostock, since 2007.

His research focuses on applicability of web services technologies in resource constrained environments like 6LoWPANs. Part of his research efforts is also the active participation in standardization efforts within the IETF.

Frank Golatowski received the Doctorate degree in computer engineering from Rostock University, Rostock, Germany.

He is a Senior Research Assistant of Electrical Engineering at the University of Rostock. His research interests include networked embedded systems, wireless sensor networks, and device-centric service-oriented architectures. As a Senior Research Assistant he is leading the contributions of the University of Rostock to the Web Services for Devices (WS4D) initiative.

Christian Lerche received the B.Sc. and M.Sc. degrees in information technology from the University of Rostock, Rostock, Germany, in 2007 and 2010, respectively. Currently, he is working towards the Ph.D. degree at the Institute of Applied Microelectronics and Computer Engineering, University of Rostock.

His primary research interests are the Internet of Things and the design of application layer protocols for wireless sensor networks and embedded systems with higher resource constraints.

Dirk Timmermann received the Diploma degree in electrical engineering from the University of Dortmund, Dortmund, Germany, and the Dr.-Ing. degree in electrical engineering from the University of Duisburg, Duisburg, Germany, in 1984 and 1990.

From 1993 to 1994, he was a Professor of Computer Engineering at the University of Paderborn, Paderborn, Germany. Since 2004, he is the Director of the Institute of Microelectronics and Computer Engineering, a Professor with the Department of Computer Science and Electrical Engineering and holds the Chair of Computer Engineering, University of Rostock, Rostock, Germany. His research interests include plug and play architectures, wireless sensor networks, energy aware systems and reliable nanoelectronic systems, and VLSI design.