2011 Eighth IEEE International Conference on Mobile Ad-Hoc and Sensor Systems

A Lightweight SOAP over CoAP Transport Binding for Resource Constraint Networks

Guido Moritz. Frank Golatowski, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18055 Rostock
Email: {guido.moritz,frank.golatowski,dirk.timmermann}@uni-rostock.de

***Abstract*—A huge momentum towards IP enabled Wireless Sensor Networks (WSN) appeared through the emerging 6LoWPAN protocols (i.e. IPv6 over Low power Wireless Personal Area Networks). By using existing cross domain open standards in contrast of proprietary solutions, technologies and deployments are not tailored too tight for specific applications. Nevertheless, 6LoWPAN is only the first step. Still efforts on higher layers on top of 6LoWPAN are an urgent need to provide the seamless connectivity and interaction of highly resource constrained devices with higher valued applications and services. This paper describes a new approach to bind SOAP to the emerging Constrained Application Protocol (CoAP) protocol. Thereby, CoAP provides a lightweight but reliable transport binding for SOAP. Compared to the widespread TCP based HTTP binding, round trip times can be reduced by 43% in an exemplary scenario because of the omitted expensive bidirectional TCP handshake mechanisms. Combined with dedicated XML compressors like EXI, SOAP based protocols become also applicable in WSNs.**

***Keywords*-SOAP, DPWS, CoAP, 6LoWPAN, Wireless Sensor Networks, SOA**

I. Introduction

Since decades, wireless sensor networks (WSN) have their own methodologies and technologies. Protocols are designed for isolated applications, proper only for dedicated scenarios [1]. With the appearance of the "Internet of Things" vision, a new way of thinking has appeared. The vision is to (re-)use Internet technologies in device networks and WSNs for a seamless integration into Internet applications. One of the main enablers 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, taking the tight resources into account [2], [3]. But the Internet of Things is more than just IP communication. It also means creation, deployment and management of distributed applications. It means "things" sharing their resources, data and knowledge. Therefore, proper application layer protocols on top of IP are necessary. These protocols should be open and should have well defined interfaces. Thereby, new applications can be formed by composing devices and higher valued services in an easy way. The compositions may result in applications unintended by the device vendors, but become possible because of the open interfaces.

The goal of finding suitable application layer protocols can be achieved by again (re-)using and optimizing well-established Internet technologies. Having a look at the Internet shows, that currently the two main architectural styles are common: 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, a short discussion is given in the related work section. This paper investigates the application of general Web services by means of SOAP Web services (WS) in WSNs. SOAP Web services are proven to be platform and programming language independent, scalable and secure even in heterogeneous environments due to decades of usage in the World Wide Web. But for WSNs and thus target platforms with only tens of kB RAM and ROM, current Web service protocols and technologies need adaptations and enhancements as described in this paper.

As a suitable WS-subset candidate for device communication the Devices Profile for Web Services (DPWS) is applied, which is currently an OASIS standard [4]. DPWS has an architectural concept that is similar to the Web Service Architecture (WSA), but enhanced to fit better into device scenarios. The main difference is the multicast and unicast service discovery with WS-Discovery that does not require any central service registry such as UDDI. But the service usage of services on devices is similar to the service usage in WSA, whereby DPWS devices can be directly integrated into WSA based enterprise systems [5], [6]. Additionally, DPWS also supports dynamic device and service description, eventing (i.e. publish/subscribe patterns) and security.

While neither DPWS nor SOAP have been specifically developed for highly resource constrained devices, this paper presents a new binding of SOAP to the arising IETF Constrained Application Protocol (CoAP) [7]. The major advantages of the CoAP binding are avoiding TCP connections (as used in the existing SOAP-over-HTTP binding) and providing reliable messaging compared to standard UDP (as used in the existing SOAP-over-UDP binding). By combining the SOAP-over-CoAP binding with XML compressors like the Efficient XML Interchange format (EXI) [8], [9], SOAP WS become also applicable in WSNs.

978-0-7695-4469-4/11 $26.00 © 2011 IEEE
DOI 10.1109/MASS.2011.101

Section II presents the general architecture and how the developed SOAP-over-CoAP binding allows integration of SOAP enabled WSNs in higher valued networks and services. Section III describes the binding in detail and section IV provides details about the implementation of the CoAP binding. Before concluding the results in section VI, section V discusses the approach presented herein in the context of architectural styles.

As dedicated scenario a DPWS enabled air conditioner device is used in this paper. The air conditioner device is the default example implementation of all DPWS stacks of the Web Services for Devices initiative (WS4D)[1]. The implemented air conditioner example is capable of presenting all features of DPWS in detail. Beside discovery functionalities, it provides simple one way requests like for example setting a new target temperature or two way request response message exchanges like for example requesting the current and the target temperature. Additionally, the air conditioner also implements eventing, to get notified if the temperature changes. In the context of the SOAP-over-CoAP binding, the air conditioner example requires all possible DPWS message exchange patterns.

II. Network Topology and Architecture

Existing solutions to integrate WSNs in higher valued distributed networking environments are often based on heavyweight application layer gateways. Such gateways are deployed at the edge of WSNs to connect proprietary protocols of the WSN with e.g. Internet protocols running in the backbone. Thus, gateways are responsible to map the functionalities and mechanisms of the WSN protocols to the external protocols and vice versa. Each change in the WSN may require changes in the gateway as well. This leads to a very inflexible deployment and lowers reusability of implementations over different applications and domains.

In IP enabled WSNs (i.e. 6LoWPANs), these heavyweight gateways can be replaced by simple IP routers. To increase scalability, existing IP networks are often supplemented by proxies. Proxy servers are also working on application layer, but with a completely different motivation than gateways. Proxies can be used e.g. to supplement data while passing through the proxy or change data representation of the payload. But proxy servers need not necessarily to be aware of the payload semantics. This lightens proxies compared to gateways significantly. For example compliant utf-8 XML, Fast Infoset and EXI are all capable of representing XML Infoset [10] based documents like SOAP envelopes. But re-encoding one format into another one by a proxy is stateless and can be performed transparent and requires no understanding of the specific document meaning.

Proxy servers can be placed anywhere in the routing path between the communicating endpoints. Thus, proxies

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

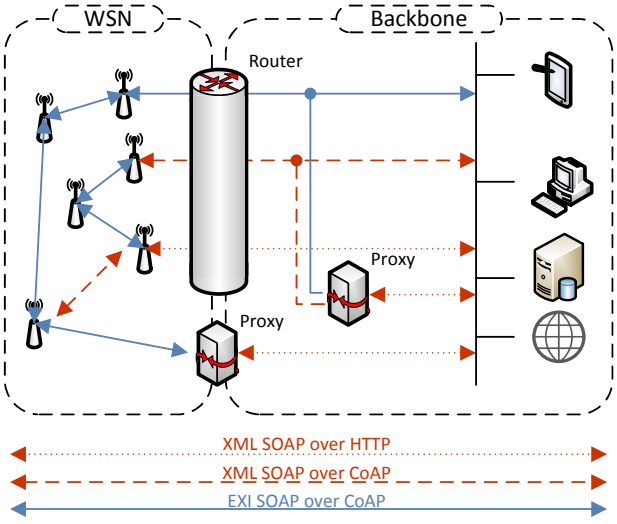

Figure 1. Messaging scenarios

are more flexible and can be placed at the edge of the WSN, inside the WSN or in the external backbone network. 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.

In [9], the feasibility of XML compressors for lightweight SOAP deployments 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. Figure 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 figure 1. By using such proxies, existing external implementations are not required

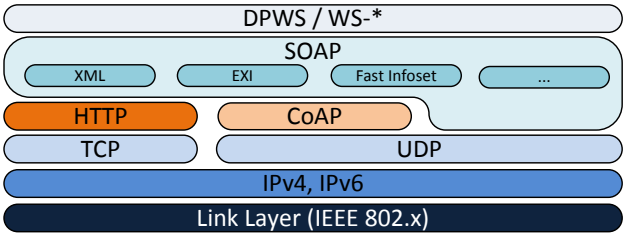

Figure 2. WS-* Stack with CoAP Binding

to be changed, but still fit into the architecture.

III. SOAP-over-CoAP Binding

Most existing deployments of SOAP are using the existing SOAP-over-HTTP [11] binding for transport of SOAP envelopes. But SOAP is not bound to HTTP exclusively. For example OASIS has defined the SOAP-over-UDP [12] binding. The intention of SOAP-over-UDP is to send SOAP envelopes over IP multicast.

Both the HTTP and the UDP binding are not optimal to be used in highly resource constrained networks like 6LoWPANs. HTTP is bound to TCP. But the overall performance of TCP in 6LoWPAN networks is questionable due to inappropriate congestion mechanisms and expensive handshakes for establishing and closing connections. UDP datagram messaging instead is lightweight at the expense of transport reliability.

The IETF Constrained RESTful Environments (CoRE) working group[2] is developing a protocol suite around the CoAP (i.e. Constrained Application Protocol) protocol [7]. Even if not identically, CoAP is leaned to HTTP. In contrast to HTTP, CoAP is based on UDP. Due to the unreliable nature of UDP, CoAP separates between the request/response layer (similar to HTTP) and the messaging layer. The messaging layer realizes transport reliability on application layer without the need for heavyweight handshakes.

Additional to the usage of reliable UDP instead of TCP, the advantage of the CoAP binding is a more compact message format, because the CoAP header is encoded binary and no string values are used like in HTTP. The messaging layer in combination with UDP provides duplicate detection mechanisms and multicast support for CoAP and thus also for the SOAP binding. Figure 2 shows the protocol stack with the three feasible SOAP bindings.

Because CoAP is leaned to HTTP, CoAP defines a detailed mapping procedure of CoAP to HTTP in dedicated proxies. This CoAP/HTTP proxies can also be used to map the CoAP to the HTTP binding and additionally to re-encode optimized EXI data representation to compliant XML and vice versa (c.f. section II).

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

A. Usage of CoAP

Most of the required protocol mechanisms are already embedded in SOAP Web services through the appropriate usage of corresponding WS-* specifications. Similar to the existing SOAP-over-HTTP binding, the CoAP binding is not intended to fully exploit the features of CoAP. Among these not required features for this binding are certain discovery and eventing (c.f. publish/subscribe) features of CoAP that are already covered by e.g. WS-Discovery and WS-Eventing.

Nevertheless, the SOAP-over-CoAP binding makes appropriate 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).

Like the existing HTPP binding, all SOAP/CoAP messages must use the POST method of CoAP. Since CoAP defines a very narrow meaning for methods and the corresponding response codes, the meaning of the existing response codes defined in [7] must be changed. In the response, success should be indicated by the response code '2.04 Changed'. This meaning of the response code is only valid for this SOAP-over-CoAP binding.

B. Messaging Patterns

DPWS deployments support in general two different message exchange patterns (MEP). A one way MEP consists only of one SOAP message for a request without a SOAP envelope in the response. The two way MEP contains a SOAP envelope in both request and response. Thus, the SOAP-over-CoAP binding must cope with these SOAP request/response patterns additionally to the CoAP request/response patterns. In a one-way MEP the CoAP request carries a SOAP envelope and the response consists only of the CoAP response header. In a two-way MEP the CoAP response header is additionally supplemented by a SOAP response envelope.

In addition to the CoAP request/response layer, CoAP defines the messaging layer to decouple request/response and transport. The messaging layer differentiates between confirmable (CON) and non-confirmable (NON) messages. CON messages must be acknowledged, while NON only consist of a single message without acknowledgement. If using a CON message for a request, the response can either be embedded synchronously in the acknowledged or send in an additional further message exchange. This asynchronous additional message exchange to transmit the response to the origin sender of the request can be again either CON or NON.

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.

To match CON messages and acknowledgements and for duplicate detection, CoAP uses message ids which are

unique for each message exchange. To match requests and responses instead, the *CoAP Token Option* is used. *WS-Addressing* defines the *message id* property to identify and match SOAP requests/responses in time and space and thus provides the same mechanism like the *CoAP Token Option*. The *CoAP Token Option* may be much more compact by providing equal functionality like the *WS-Addressing message id*. In case of using the CoAP binding, the *WS-Addressing message id* property can be empty and must contain a value only if the *CoAP Token Option* is not present or not sufficient. The intention of this adaptation is to reduce message size. The *WS-Addressing message id* 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.

C. Serialization

Both EXI and standard XML are based on XML-Infoset [10] and thereby interoperable. Hence, the SOAP-over-CoAP binding only requires the capability to represent XML-Infoset documents. This includes utf-8 XML for interoperability reasons with existing SOAP implementations and compressed XML like EXI.

D. Endpoint Identification

WS-Addressing defines an *anonymous* URI that can appear in the *address* property of an endpoint reference. If the *reply endpoint* property of a SOAP request transmitted over CoAP has an *WS-Addressing 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 send to an endpoint different from the origin requester.

Implementations of the CoAP binding must be aware of that difference. It might be required to send the acknowledgment of a CoAP CON message to the origin requester and use an additional CoAP message exchange to send the response to the third party endpoint indicated in the *WS-Addressing reply endpoint*.

IV. Implementation

The existing WS4D-uDPWS[3] [13] implementation of a compliant DPWS stack for sensor node class devices has proven that SOAP WS can also run in highly resource constrained environments. WS4D-uDPWS is implemented on top of the operating system Contiki and running for example on TelosB modes. These TelosB modes have a MSP430 as main processor with only 48kB ROM and 8kB RAM. WS4D-uDPWS supports utf-8 XML message processing and features the existing SOAP-over-UDP and SOAP-over-HTTP bindings.

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

Table I
ROM usage of the different bindings in Byte

	(1) HTTP Binding	(2) CoAP Binding	(3) HTTP and CoAP Binding
System	1600	1302	1600
Contiki	26688	26526	26688
WS4D-uDPWS	12805	15039	16815
Sum	41093	42867	45103

To prove the approach described in this paper, the new SOAP-over-CoAP has been integrated into the existing WS4D-uDPWS stack. For the CoAP part of the implementation, an existing stack has been used that is already integrated into Contiki [14] and compliant with the latest drafts of the CoAP specifications.

Table I presents the ROM footprint of the implementations. Compared in table I is the WS4D-uDPWS containing (1) the SOAP-over-HTTP binding only (2) the SOAP-over-CoAP binding only and (3) the SOAP-over-HTTP and SOAP-over-CoAP bindings together. The SOAP-over-UDP binding is mandatory according to the DPWS specification and thus included always.

Omitting TCP completely in version (2) has no major influence on the ROM size. The major differences between the three versions are in the WS4D-uDPWS part. These differences occur due to the usage of the existing CoAP stack. This CoAP stack has a ROM size of 3036 Byte. Because only a very limited subset of the functionalities of this CoAP stack is required by the SOAP-over-CoAP binding, future footprint optimizations will focus on adapting the CoAP stack.

A. Performance Measurements

For the performance measurements, the CoAP binding is additionally implemented in the existing WS4D-gSOAP toolkit. WS4D-gSOAP is a DPWS toolkit for embedded systems and is based on the well-known gSOAP stack [15]. For the CoAP binding of the WS4D-gSOAP toolkit, the libcoap[4] implementation was used and integrated in gSOAP.

The implemented scenario of the air conditioner always uses confirmable messaging and includes the response payload in the CoAP acknowledgements to allow comparable performance results.

From the air conditioner example realized to evaluate the binding, a two way MEP has been selected to be presented in this paper (i.e. setting the target temperature and returning status information in the response). Table II presents the results of the round trip measurements. All messages of this two way MEP are under the minimum MTU of IPv6 (i.e. 1280 Byte), whereby IP layer fragmentation is avoided. As server and client, desktop computers where used connected over a laboratory Ethernet IPv6 backbone. A standard ICMP echo (i.e. ping) of 56 Bytes between the endpoints takes on an average 0.502 ms, measured over 30.000 runs.

[4]http://sourceforge.net/projects/libcoap/

The HTTP keep alive mechanism causes only one TCP handshake for connection setup and sending further application data and TCP acknowledgements in one message instead of separating them. For completeness, HTTP is presented with and without keep alive. The sizes in the table are presented as follows

- The rows differentiate between request and response messages.
- *No. of Messages* refer to the number of messages on transport layer and thus also include TCP handshakes (e.g. SYN, SYN-ACK, FIN...).
- *Tran. Header Size* refers to the headers on transport layer of TCP or UDP.
- *App. Header Size* refers to the headers on application layer of HTTP or CoAP (UDP has no additional header).
- *SOAP Size* refers to the SOAP envelope.
- The SOAP envelopes have been encoded in EXI to present the overall performance of the CoAP binding.

For the EXI encoding, bit aligned schema informed mode is applied. As schema files only the basic schema files as published along with DPWS and the related WS-* specifications are used. Including also application specific schema information (i.e. schemas for the air conditioner example) would further reduce resulting message sizes.

This differences in the payload occur mainly through the usage of the *WS-Addressing message id* mechanism. This mechanism is only required in the HTTP binding with keep alive and the UDP bindings. For the HTTP binding without keep-alive, this feature is not required, because the endpoints have an direct TCP end-to-end connection used only for a single request/response. For the CoAP binding, this feature is provided in a much more compact format by the *CoAP Token option* and the *CoAP Message ID*.

While the message size still can be reduced significantly for all messages, the HTTP binding suffers from the additional required HTTP header and the TCP handshake overhead. UDP instead requires no further header, but requires to use the *WS-Addressing message id* mechanism. These additional HTTP and *WS-Addressing message id* headers causes comparable large message sizes especially in the EXI encoding.

To sum up, the CoAP binding combined with the EXI encoding results in the most efficient overall solution to transport SOAP envelopes in resource constrained networks.

Table II
PERFORMANCE OF SOAP-OVER-COAP

	HTTP	HTTP keep alive	UDP	CoAP
No. of Messages	10	after TCP handshake: 2	2	2
Tran. Header Size Request+Request	200	after TCP handshake: 20	16	16
App. Header Size Request	208	213	0	4
App. Header Size Response	136	122	0	6
SOAP Size Request	819	794	919	843
SOAP Size Response	752	848	845	841
SOAP+App. Header Size Request	1027	1007	919	847
SOAP+App. Header Size Response	888	970	845	775
Round Trip in ms	2.25	1.35	1.23	1.24
Round Trip in %	100	59.9	54.6	55.2
SOAP Size EXI Request	146	196	201	150
SOAP Size EXI Response	130	181	186	135
SOAP+App. Header Size EXI Request	354	409	201	154
SOAP+App. Header Size EXI Response	266	303	186	141

V. RELATED WORK

The trend of (re-)using Internet technologies in device centric domains and in latest developments also in WSNs is obvious. For classical Internet applications two prevailing architectural styles exist. Firstly, the Representational State Transfer (REST) style, which is a result of the dissertation of Roy Fielding [16]. Fielding describes how the current World Wide Web communicates and on what design principals it is built on. The current web is mainly browsers requesting resources of servers (i.e. websites) and is focused on human to machine (H2M) interaction. Secondly, the Service-oriented Architectures (SOA). In SOAs, complex functional blocks are encapsulated as much more abstract services, where existing services or service implementations can be reused over different applications. SOAs are often used to model and realize complex business logics. A good discussion on REST and SOA is provided by Pautasso [17]. Pautasso argues that comparing SOA and REST is like comparing apples and oranges: REST is an architectural style for the web and SOAs instead can be used to realize a middleware of interoperable standards.

Essentially, SOA and REST are two different architectural styles having different advantages and disadvantages. It depends on the application which is the right architectural decision. Hence, this paper is beyond the SOA vs. REST discussion. The paper concentrates on protocol adaptations required for existing protocols to meet the resource constraints of WSNs.

The most known RESTful protocol is HTTP. HTTP uses a certain number of standardized methods (e.g. GET, PUT, POST...) to deal with resources like for example websites. For SOAs commonly SOAP in conjunction with the WS-* specifications is used. The large number of interoperable WS-* specifications and their extensions causes a high flexibility, but also a steep learning curve compared to HTTP. But it should be mentioned that SOAP based protocols not necessarily lead to a SOA, as well as HTTP not necessarily leads to a RESTful architecture. For example the W3C Web Services Resource Access[5] working group specifies SOAP based resource oriented Web services. Thus, the use of DPWS, which is used as a WS-subset in this paper, not forces the user to build a SOA. Although it was originally intended to build service-oriented device networks, it can be also used to build RESTful applications. The CoAP protocol

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

used in this paper instead is designed as RESTful protocol, but can also be used as transport mechanism for SOAP envelopes as described herein.

VI. Conclusion

The SOAP based WS-* framework provides a huge number of cross domain protocol features. But the basic characteristics like data representation and transport mechanisms of SOAP Web services imply an overhead which prevents them to be applied in WSNs. Studies in [9] have shown that the data representation of SOAP can be tailored for endpoint capabilities. It can be based on compliant utf-8 XML or binary encoded and compressed, while the different formats can be re-encoded stateless and transparent.

But using existing solutions also for the transport of SOAP documents in these networks is questionable. The CoAP protocol developed by the IETF CoRE working group is leaned to HTTP, but designed to fit better in 6LoWPAN limitations. This paper describes in detail an approach of a SOAP-over-CoAP binding for transport of SOAP messages in resource constrained environments. Combined with the mentioned binary encodings like for example EXI, SOAP Web services can also be deployed in bandwidth limited environments like WSNs.

Both EXI and CoAP are designed to map seamless and stateless to XML and HTTP. Existing implementations, based on compliant XML SOAP-over-HTTP, deployed on resource rich devices and used in higher valued applications not have to be changed. Dedicated proxies are responsible for the transparent mapping, whereby WSNs can be integrated in these higher valued applications and services with one comprehensive technology.

Acknowledgment

The authors would like to thank the student Benjamin Beichler for the tremendous contribution of the CoAP binding implementation for WS4D-uDPWS.

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,” *Proceedings of the 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, http://www.oasis-open.org/committees/ws-dd/, 2009.

[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 *Web Services, 2009. ICWS 2009. IEEE International Conference on*, 2009, pp. 968 –975.

[6] D. Savio and S. Karnouskos, “Web-service enabled wireless sensors in soa environments,” in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, 2008, pp. 952 –958.

[7] 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

[8] *Efficient XML Interchange (EXI) Format 1.0 (W3C Recommendation*, W3C, 2011.

[9] G. Moritz, F. Golatowski, D. Timmermann, and R. Stoll, “Encoding and compression for the devices profile for web services,” in *Proceedings of 5th International IEEE Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE2010)*, april 2010, to appear.

[10] R. Tobin and J. Cowan, “XML information set (second edition),” W3C, W3C Recommendation, Feb. 2004, http://www.w3.org/TR/2004/REC-xml-infoset-20040204.

[11] J.-J. Moreau, M. Gudgin, M. Hadley, N. Mendelsohn, Y. Lafon, A. Karmarkar, and H. F. Nielsen, “SOAP version 1.2 part 2: Adjuncts (second edition),” W3C, W3C Recommendation, Apr. 2007, http://www.w3.org/TR/2007/REC-soap12-part2-20070427/.

[12] R. Jeyaraman, “Soap-over-udp version 1.1,” OASIS WS-DD TC, 2009.

[13] C. Lerche, N. Laum, G. Moritz, E. Zeeb, F. Golatowski, and D. Timmermann, “udpws - the devices profile for web services for resource-constrained devices,” in *8th European Conference on Wireless Sensor Networks*, ser. EWSN 2011. Springer, 2011.

[14] M. Kovatsch, S. Duquennoy, and A. Dunkels, “A low-power coap for contiki,” in *Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2011)*, Valencia, Spain, Oct. 2011.

[15] E. Zeeb, G. Moritz, D. Timmermann, and F. Golatowski, “Ws4d: Toolkits for networked embedded systems based on the devices profile for web services,” in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, 2010, pp. 1 –8.

[16] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, 2000, chair-Taylor, Richard N.

[17] C. Pautasso, O. Zimmermann, and F. Leymann, “Restful web services vs. ”big”’ web services: making the right architectural decision,” in *Proceeding 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