# Devices Profile for Web Services in Wireless Sensor Networks: Adaptations and Enhancements

Guido Moritz[1], Elmar Zeeb[1], Steffen Prüter[1], Frank Golatowski[1], Dirk Timmermann[1], Regina Stoll[3]

[1]Institute of Applied Microelectronics and Computer Engineering, University of Rostock
[3]Institute of Preventive Medicine, University of Rostock
[1,3]18055 Rostock, Germany
{guido.moritz, elmar.zeeb, steffen.prueter, frank.golatowski, dirk.timmermann, regina.stoll }
@uni-rostock.de

## Abstract

*For Service-oriented Architectures, Web Services are claimed as state of the art to connect business execution layers as well as networking devices. Additionally, the deployment of Wireless Sensor Networks became applicable over the last years. The usage of application layer gateways and proxy concepts allow the integration of these sensor networks into real world scenarios and existing networks that make use of Web Services. This paper presents a new approach to adapt and enhance the Devices Profile for Web Services to be applied in Wireless Sensor Networks directly. Thus, seamless connectivity between business layers, device level networks, and Wireless Sensor Networks are possible.*

## 1. Introduction

The research on Wireless Sensor Networks (WSN) for the last years allows the application of WSN in manufacturing environments. There are several industry standards and solutions by major hardware vendors available. Still a lot of issues concerning reliability, energy consumption, etc., are ongoing research topics, but researches and engineers can start to design applications for their specific environments that are based on WSN.

Most solutions for WSN come equipped with their own architectural concepts which raise the problem of possible incompatibility of computer network and the WSN. Often gateway concepts are used to overcome this problem. But this is not the best solution on the long term. Other research fields and industrial domains, like factory automation, are heading for universal architecture concepts based on internet technologies that are more mature and better understood.

Thus, application of middleware concepts that are widespread in other application domains and may be used as cross domain technologies are required. This would help the researchers and engineers to design new applications based on WSN and concentrate on the application, not the integration.

The Devices Profile for Web Services (DPWS) [1] is such a cross domain technology for inter machine communication and base on Web Services. DPWS employs similar messaging mechanisms as the Web Services Architecture (WSA) [19], with restrictions to complexity and message size ([1, 20]). It features secure message exchange, dynamic discovery, and description of devices based on WS-Discovery, WS-MetadataExchange and WS-Transfer. Furthermore, it provides a publish-subscribe eventing mechanisms based on WS-Eventing.

In August 2008 a technical committee (TC) at OASIS was formed for the "Web Services Discovery and Web Services Devices Profile" (WS-DD) [11]. WS-DD defines a lightweight subset of the Web Services protocol suite that will make it easy to find, share, and control devices on a network. The work of this TC is based on the former DPWS, WS-Discovery and SOAP-over-UDP specification.

Even if DPWS is the best candidate to integrate WSN in existing infrastructures, it cannot be applied to WSN without research efforts, because it addresses softer resource constraints as required in WSN. But DPWS provides a minimal set of constraints for applications in resource constrained devices. So this paper describes an approach that further restricts DPWS for WSNs, but keeps it still interoperable with DPWS. The proposed approach is a recommendation and might be refined in future work.

## 2. Related Work

TinyDB [24] and Cougar [25] are the best known middleware for data acquisitioning in WSNs. Both approaches inject SQL like queries directly into the sensor network. The formulated query can be performed once or in defined intervals. While TinyDB strongly depends on TinyOS [23], only sensors running TinyOS are supported by TinyDB. Like TinyDB, Cougar also performs a clustering process and data

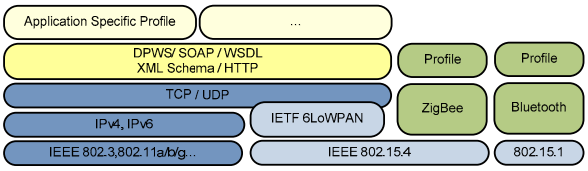


**Fig. 1. 6LoWPAN, ZigBee and Bluetooth Stack**

aggregation within the cluster but adds an additional graphical user interface (GUI), which allows formulating queries. Both approaches strongly depend on the used sensor network hardware as well as the software. An integration of WSN applied with TinyDB or Cougar results in vulnerable gateway and proxy concepts.

Different research groups are working on enhancements to enable IP, TCP, and UDP based traffic in WSN. The adaptation of existing network concepts and technologies is one approach to handle the rising number of interacting nodes inside a WSN. Further, TCP/IP and UDP/IP based communication enables the WSN to interact with external networks including the internet.

### 2.1. IP in Wireless Sensor Networks

With respect to the power consumption, the IEEE 802.15.4 radio communication standard [1] for wireless personal area networks is gaining in importance. In accordance to the IPv6 specification, the Internet Engineering Task Force (IETF) has established the 6LoWPAN working group [3]. The focus of 6LoWPAN is to compress IPv6 headers to be sent on top of 802.15-based technologies, especially 802.15.4 [5]. 6LoWPAN establishes the basis for TCP and UDP data transmissions in WSN. The main advantage of 6LoWPAN over nanoIP [4] and related protocols is the compliance to a regular computer network protocol (in this case IPv6). Other proprietary protocols like ZigBee and Bluetooth cannot offer this level of compliance. 6LoWPAN is the first choice for network interconnection between sensor networks and computer networks without application level gateways. To support the communication between internal (6LoWPAN) and external (IPv6) networks, a compliant router is necessary only and no proxy concepts as used in other architectures.

With the availability of TCP/IP and UDP/IP in sensor networks, middleware concepts like DPWS can be applied (see Fig. 1). Thereby, a bridge is build between WSN and existing computer networks with one comprehensive architecture. Further verifications concerning the power consumption of DPWS on top of 6LoWPAN and 802.15.4 are out of scope of this paper. Nevertheless, these analyses are part of ongoing research work. This paper focuses on necessary adaptations and enhancements of the DPWS specification, to be applied in WSN in general.

### 2.2. Implementing DPWS for Wireless Sensor Nodes

Wireless sensor nodes are equipped with small microcontrollers with a few KB of RAM and ROM only without conventional operating system. These microcontrollers are necessary because of the low costs, the low power and the small size constraints in this application area. Web Services were originally designed for conventional server and PC platforms, but a single node in a sensor network cannot provide such complex services. Typically, these nodes are dedicated for a single function like measuring one or a small set of physical phenomena. So these simple functions could also be modeled as services and provided with a Web Service based technology like DPWS. In [7, 8] a new approach for the implementation has been presented that can provide DPWS functionalities on wireless sensor nodes also. These papers describe deeply embedded devices which have similar hardware constraints like wireless sensor nodes concerning memory, data transmission range and rate, computing power, and energy consumption. Basing on this prove of concept implementation, this paper focuses on necessary adaptations of DPWS in general to allow a more energy efficient deployment of DPWS and to ensure an improved overall lifetime of the WSN.

### 2.3. Discovery Phase of DPWS

Additionally to the memory usage of the implemented software stacks, the number of transmitted messages has to be reduced to a minimum. Discovery scenarios in DPWS can vary depending on the application scenario and how static or generic this scenario is designed. Thus, this section describes an exemplary worst case discovery phase with the maximum of possible messages exchanged. In the following sections, a formal approach is presented to reduce the number of messages.

DPWS differentiates between two service classes: Hosting and Hosted Services. Hosted Services are services on a device which can be invoked by a service consumer, which is called client. The device is the service provider and hosts the Hosted Services. Hosting Services announce the Hosted Services during the discovery phase. When starting the device, it announces itself and the Hosted Services with a Hello message through an UDP multicast message. When a client missed a Hello message, the client can explore the network with a Probe message for available devices. The answers are unicast Probe Match messages. In both cases, the device address included in the Hello or Probe Match is given in a network independent form, wherefore a Universally Unique IDentifier (UUID) is mandatory in DPWS. It is optional to include a transport specific form also.

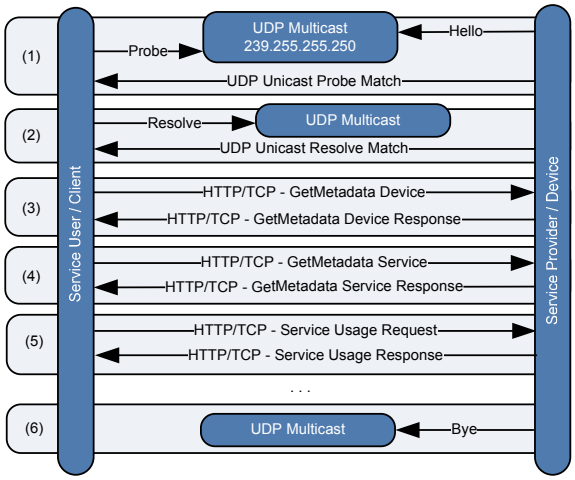


**Fig. 2. Discovery Phase of DPWS**

Transport specific addresses, like HTTP and IP addresses, can change when a device enters a new network. Network independent UUID addresses do not change. If a device does not include a transport specific address in Hello or Probe Match messages, the network independent address can be resolved by a UDP multicast Resolve message. The response is a Resolve Match send unicast through UDP. After finding the transport specific address of a device, the Hosted Services on the device can be queried. Therefore, a GetMetadata Device is send unicast to the Hosting Service. The device responds with a GetMetadata Device Response message. The response provides several sections that describe the device (manufacturer, firmware version, serial number, etc.) and sections with relationship data. These relationship data contain information about the Hosted Services and include the service type, service id, and the endpoint reference (transport address) of the Hosted Services. When the client knows the available Hosted Services and their addresses, the client can ask a Hosted Service for its interface description, with a GetMetadata Service message. The service may respond with a GetMetadata Service Response message, including the WSDL. The Web Service Description Language (WSDL) describes the supported operations and the data structures used in the service operations. When a device is shutting down it may announce this by sending a Bye message multicast through UDP. To get an overview, Fig. 2 illustrates the described discovery of DPWS with the typical phases.

### 2.4. Templates for DPWS Devices

At the moment, DPWS offers no trivial way to directly discover services available on the network. In a generic scenario, as described in Fig. 2, where a client does not know the services hosted on a device, a client always has to discover a device first and then discover its Hosted Services, with the help of the metadata provided in the description of a device.

Bobek et al. [9] describe device and service templates for DPWS in a similar way to UPNP templates. The goal is to describe the service and device types defined by DPWS at development time in a formal language and to shorten the discovery phase. The templates are divided into device and service templates, as DPWS differentiates between service and device types. Device templates describe the mandatory and optional services that must be implemented by devices offering a specific device type. Service templates describe a service that is related to a specific service type. Furthermore, device templates can include other device templates and build a hierarchical structure to enable extensibility. Bobek et al. relates the types transmitted during the discovery phase directly to a type defined by a device template. A service type that is part of the device description is related to a service template. The metadata, that is currently only available at runtime, can be formally defined at development time. With device and service templates, more static scenarios can be defined where a client already knows specific services and their endpoints when the client finds a device on the network. A typical example would be a printer device. This exemplary device type offers a printer service that is always hosted on port 80. The language proposed by Bobek et al. has some inconsistencies with the DPWS specification that are discussed and corrected in the next section.

## 3. DPWS Enhancements for Wireless Sensor Networks

DPWS allows the definition of application specific profiles (see Fig. 1). All enhancements, which are made in this section, are summarized in the new defined device type. Additional to the DPWS device type, a new device type for DPWS in WSN is introduced. The boxes in the sub sections are recommendations for requirements for the specification of the new WSN DPWS profile.

The major goal of all restrictions and enhancements is the minimization of exchanged messages inside the WSN and the reduction of memory usage of DPWS implementations. The presented adaptations result in one discovery message only instead of the previously presented worst case scenario.

### 3.1. Templates for Devices Profile for Web Services

Section 2.4 describes a language proposed by Bobek et al. to define service and device templates for DPWS. If sensor network nodes are modeled as DPWS devices (service providers), a sensor network is simply a heap of services. To implement more intelligent WSN, which require interaction inside WSN, sensor nodes must be modeled as peers that are DPWS devices and clients at the same time. Device templates can be used to create tailor-made DPWS clients for a specific scenario with smaller memory usage. At the same time,

the template concept reduces the message exchanges on the network required to bind a client to a service also.

The device and service template concept proposed by Bobek et al. bases on a misunderstanding of the DPWS type model. WSDL port types are the recommended way to describe Web Service interfaces in WSDL documents. The service template concept relates DPWS service types to WSDL port types. As DPWS service types are directly related to WSDL port types, the service template concept is redundant. This direct relation is not clearly stated in the specification, but will be described in the non-normative documents that are published along with the DPWS 1.1 specification by OASISI WS-DD.

Therefore, the template concept has to be revised in general. As depicted in Fig. 3, the template system is reduced to device templates only. A device template describes exactly one device type. Further device types may be included, to enable extensibility. The Hosted Service element may define a URL template for the endpoint reference of a Hosted Service. In contrast to the proposal by Bobek et al. this proposal refers directly to WSDL port types, as service types do in DPWS. To be in line with the DPWS and WS-Discovery [17] specification, all types are lists of qualified names as specified in WS-Discovery.

### 3.2. Discovery Considerations

The following subsections will address considerations concerning the discovery phase of DPWS and refer to the pattern described in section 2.3. The numbers in the parentheses refer to the sequences illustrated in Fig. 2.

As mentioned before, the discovery phase in DPWS can vary in number and length of exchanged messages between the device and the client. Step (4) is optional in general. The step (3) can be superseded by the usage of device templates.

#### 3.2.1. Include Transport Specific Address in Hello and Probe Match

In step (1), the device announces itself by using a network independent address, which has to be resolved in step (2) into a transport specific address. In accordance to the DPWS specification [1], embedding of a transport specific address in phase (1) is not mandatory. Thus, for the new WSN device type the optional fields for the transport specific addresses are now mandatory to avoid Resolve messages.

> A DEVICE MUST include the transport specific addresses in its Hello and Probe Match messages.

#### 3.2.2. Include device type in Hello and Probe Match

In sections 2.4 and 3.1, the template concept is introduced. A specific class of devices and their related

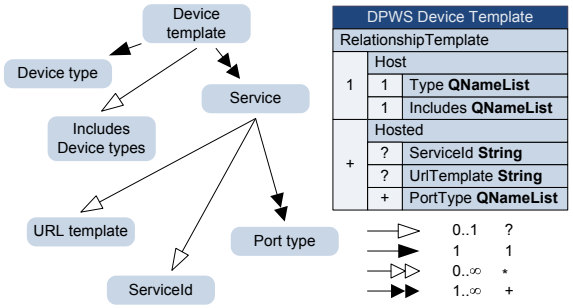


**Fig. 3. Structure of a device template**

services are identified through a device type, which is transmitted during the discovery. In the current version of the DPWS specification, it is not mandatory to include the type field in the Hello and Probe Match messages. To avoid additional Probe and ProbeMatches message exchanges, including the type is mandatory.

> A DEVICE MUST include all device types in Hello and Probe Match messages.

#### 3.2.3. XML Namespaces

SOAP and thus DPWS makes use of XML namespaces. XML Schema definitions, WSDL definitions, and DPWS device types must be assigned to these XML namespaces to avoid collisions in XML documents. In most cases HTTP scheme URIs are used to declare XML namespaces. For devices with many device types, the Probe Match messages may contain a lot of namespace declarations. These declarations have a typical length of 50 or even more characters. In some of the analyzed scenarios, half of the message size is consumed by the namespace declarations. Hence, namespaces should be kept small or be short UUIDs, if applicable.

> XML namespaces SHOULD have less than 30 characters.

#### 3.2.4. Providing WSDL file for Clients

Providing the WSDL during the discovery phase is optional [11]. These WSDL files have a size of several kB in most analyzed scenarios. The expensive and memory consuming storage of these WSDL files on the device and on the client node is not applicable for WSN. Furthermore, the exchange itself consumes a high bandwidth. To skip the exchange of these WSDL files for every service (see step (4) Fig. 2), devices implementing the new WSN device type should not provide the WSDL at runtime. The device types and provided services should already be known at development time.

> A SERVICE SHOULD NOT provide the WSDL file for CLIENTS.

3.2.5. Discovery Proxy in Wireless Sensor Networks

DPWS allows the automatic discovery of devices in the network without invoking a central service registry. Nevertheless, DPWS allows the optional usage of the Discovery Proxy (DP) as service registry. A DP cannot be restricted by the same energy constraints like regular device nodes, because of the increased traffic. Therefore, it should be connected to mains power systems.

If a Discovery Proxy is applied, it advises the clients to suppress multicast discovery messages and send these messages unicast to the DP instead. Therefore, the DP responds to a Probe or Resolve message with a unicast Hello message to the client. The DP sends Probe Matches and Resolve Matches representative for the devices it knows. Hence, the major difference between networks with and without DP is the multicast suppression for clients. The behavior of the devices is unmodified. They still answer to Probe and Resolve message arrived via multicast. In networks with many changes in the device structure, where many Probes and Resolves appear, the DP can disburden the WSN. Certainly, the usage of a DP is controversial concerning the energy consumption and overall lifetime in WSN.

On the one hand, the centralized service proxy requires routing of all discovery messages to this central node. Depending on the network topology (e.g. mesh, star and tree), nodes which are logical (and in wide networks also physical) close to the DP have to route and forward almost all discovery specific messages to and from the DP. Nodes close to the DP require additional active transmitting time and receiving time, which causes significant higher power consumption. Thus, the usage of a DP highly depends on the network topology and the expected scenario. In a mesh network with single hop to the service registry, the DP should be used in favor to unload the device nodes. In larger networks with more complex topologies, the usage of a DP may not be applicable. Furthermore, several DPs can build hierarchical structures also. The effect of a huge number of DPs distributed in a WSN cannot be expected and researched through direct simulations only and is out of scope of this paper.

On the other hand, the absence of a DP causes all Probe and Resolve messages to flood the whole network and affect all nodes. Discovery messages cause all nodes in a WSN to change into active state for receiving. These messages can also come from outside of the WSN. The WSN is flooded from outside by external devices announcing themselves through Hello messages and external clients searching for devices with Probe message. Therefore, a DP can be located on the router node, which connects the WSN with the backbone. Certainly, this approach is only applicable in WSN with only one router to the external network. The DP can respond directly to external Probes and Resolves, because its knowledge about internal devices, so the external multicast messages do not flood the WSN.

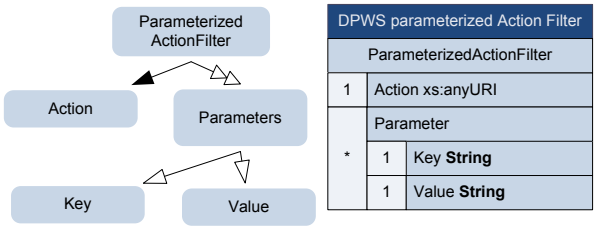


**Fig. 4. Eventing with key value filter**

### 3.3. Eventing Considerations

The DPWS specification also includes eventing concepts. A client can subscribe to specific events of a service, instead of permanently polling a service. Events are mapped directly to service operations. For example, if a node measures the temperature and offers a service method which provides the measured value, the client has to subscribe to this method and gets a notification every time the measured temperature changes. The porting of the event concept to WSN has two major problems.

The first problem is the insufficient filter concept. A client cannot define parameters for the events. If a method represents more than one temperature value, the subscribed client gets an event every time any temperature changes, even if the client only want to observe one temperature value or a specific change like exceeding a certain threshold. It is necessary to include parameters in the service subscription. This can be done by expanding the existing action filter system with a key value filter, as depicted in Fig. 4.

A HOSTED SERVICE MUST at least support Event Filtering indicated by "http://www.ws4d.org/wsn0901/ParameterizedActionFilter".

The second problem is the delivery mode of the notifications. DPWS defines a push delivery mode, where a device (Event Source) establishes a dedicated connection to all subscribed clients (Event Sinks) every time an event occurs. The device has to send the notification to every subscriber individually. Hence, the delivery mode should use a multicast communication scheme. The Event Source only has to send one UDP multicast message and all subscribed clients can receive this message by listening to the dedicated multicast group. The effort for the complete network depends on the mapping of multicast communication scheme to the link layer. In worst case scenarios, the multicast message causes all nodes in the network to change into active mode to receive the

notification and analyze it on the IP layer. This can be avoided, if the routers in the network only forward multicast messages to the nodes that have subscribed to this multicast group. Otherwise, the energy saving of the Event Source causes higher energy consumption for all other nodes

A HOSTED SERVICE SHOULD at least support Multicast Delivery Mode indicated by "http://www.ws4d.org/wsn0901/DeliveryModes/Multicast".

At the moment, there is no proposal how this delivery mode can be used in the subscription mechanism of WS-Eventing [18]. In the case of multicast delivery, Event Sink and Event Source must be able to negotiate a multicast group.

### 3.4. Messaging Considerations

For messaging in general and additionally to the previous enhancements, communication scheme and data representation have to be focused also. Depending of the radio technology and implementation, an active receiver consumes nearly the same or even more energy than a sender. Thus, wireless sensor nodes turn off the radio and go in an energy saving mode as often as possible, to reduce energy consumption significantly. A node only wakes up within defined time slots for receiving messages. Therefore, in large multihop WSNs long delays for message exchanges are expected. Thus, only asynchronous message exchanges should be used to avoid the power consuming active state of the node while waiting for the response.

DEVICES and CLIENTS MUST support asynchronous messaging.

All messages in DPWS make use of XML for data representation. The introduction of XML in DPWS has multiple advantages considering independency of programming language, operating system, communication channel, data representation, and character set. Certainly, XML implies a message overhead. Thus, XML based protocols are often considered as too expensive for WSN. Hence, this subsection describes several concepts for optimized data encodings of SOAP messages.

Fast Web Services [13] are using ASN to compress the XML files into a resource optimized binary representation and to overcome performance issues for complex string operations in message processing. The proposed approach of Sandoz et al. reduces the size of the XML data by more than factor four and "performance is nearly 10 times that of XML literal." Nevertheless, this approach leads to isolated applications and is incompatible with DPWS devices and clients that do not support the ASN data representation.

The Efficient XML Interchange Working Group [12] develops an encoding format for XML, "that allows efficient interchange of the XML Information Set and allows effective processor implementations". The main focus is high data compression even of big and deep structures completely compliant to XML. Analyses have shown that the binary documents can be up to 90% smaller than the original XML document [16].

In comparison to Efficient XML and Fast Web Services, A SOAP [15] describes concepts for XML, which can be easily implemented in hardware and thereby much more energy efficient then in software. A SOAP (Adaptive SOAP) uses hash functions, to encapsulate complex XML structures. Constantly recurrent XML structures are represented by hash values. For the transmission only changing parts of the XML files are transmitted as proper XML tags. All tags known by sender and receiver are transmitted by using hash values. Especially A SOAP can be integrated in DPWS and assure compliance to clients and devices that not support A SOAP. An endpoint that cannot understand a SOAP message, responds with a SOAP Fault message. In the case when an endpoint is not A SOAP enabled, the overhead is one additional request and one additional SOAP Fault. The sender then has to retransmit the message as a compliant XML message. Certainly, this generic A SOAP support detection mechanism is completely DPWS compliant. The disadvantage of A-SAOP is that it recently was granted as patent. Hence, there is no compliant proposal for a data compression to apply DPWS in WSN.

## 4. Estimated Savings

Due to the presented improved discovery process, fewer messages occur during a discovery phase. In section 2.3 we presented a worst case scenario where 9 messages are required before service usage. Three of these messages are send via multicast, 6 of them via unicast. Our presented adaptations and enhancements reduce the required messages to one multicast message only before service usage can start.

A device still announces its presence and its leaving from the network with a *Hello* and a *Bye* message. *Probe* and *Probe Match* messages are only required if the client was not online in the time of the *Hello* of the device. *Resolve* and *Resolve Match* messages are not longer required (cf. section 3.2.2). Also the *GetMetadata Device* messages (cf. section 3.1) and the

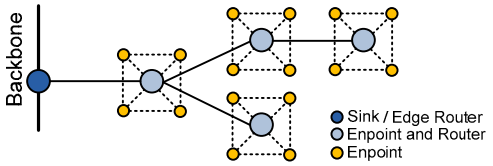


**Fig. 5. Exemplary WSN for estimations**

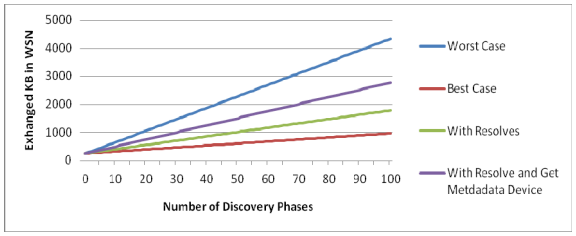


**Fig. 6. Dynamic Scenario with Exchanges over Discovery Phases**

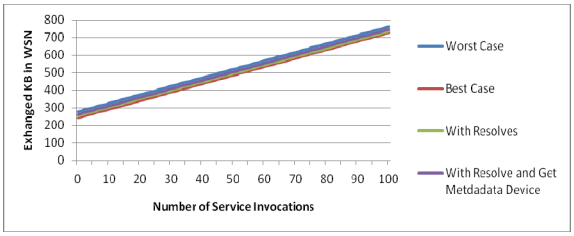


**Fig. 7. Static Scenario with Exchanges over Service Usages**

*GetMetadata Service* messages (cf. section 3.2.4) can be omitted. In this section, the paper presents the estimated savings of our enhancements on an exemplary scenario.

The simulations refer to an in-house WSN deployment as depicted in Fig. 5. The WSN consists of one sink, 4 routers and 16 endpoint nodes. The routers are also endpoint nodes but can provide routing functionalities as well. All nodes and routers are deployed as DPWS devices. Every cell, which consists of one router and 4 nodes, is deployed in one room. All nodes of one room are in direct communication range of all other nodes in this room. But, only the routers can send data to the routers in the next room. Our focus is to calculate the energy costs for the whole WSN over a long period.

The costs depend on the number of exchanged messages, the required number of connections (hops) for message delivery, and the sizes of the messages. To determine the message sizes for a DPWS scenario, we implemented an exemplary climate control device with our WS4D-gSOAP toolkit [22], which can request simulated temperature measurements and adjust the temperature. We analyzed the scenario and measured the message sizes as presented in Table 1.

To determine the number of connections (hops), a difference has to be made between multicast and unicast communication. Multicast is equivalent to broadcast in our scenario. Every multicast delivery consists of 5 connections, independent of the source

**Table 1. Message Size**

| Message Type | Size in Byte |
|---|---|
| Hello | 1.337 |
| Bye | 1.093 |
| Probe | 697 |
| Probe Match | 1.425 |
| Resolve | 864 |
| Resolve Match | 1.446 |
| Get Metadata Device Request | 603 |
| Get Metadata Device Response | 3.030 |
| Get Metadata Service Request | 584 |
| Get Metadata Service Response | 5.217 |
| Service Request | 595 |
| Service Response | 1.187 |

which may be a sensor node or be injected from an external network. For unicast connections over a long period, a statistical average can be computed that assumes a uniform distribution of sources and destinations of requests. This factor includes the number of existing hop counts in the network and the fact that multi hop communication is more expensive then a one hop connection. This average value is 2.86 hops for our exemplary WSN.

With these values, the overall costs for the WSN can be computed. *Hello* and *Bye* messages always have to be sent, which we call static quota. The second addend is the service invocation itself which depends directly on the number of service usages. This is the dynamic quota. The discovery quota is the third addend and depends on the required number and type of discovery processes after a *Hello* of the device. The type depends on the implementation of our enhancements and may include or not include *Probes*, *Resolves*, *Get Metadata Devices* and *Get Metadata Services* requests and their responses. In the best case implementation, no further discovery phase – expect the *Hello* and the *Probe* messages - is required and thus only service invocations appear. In the worst case scenario, all discovery messages are required.

For clarity, this paper presents two different estimations[1]. The first one assumes that every time a service is invoked, a discovery phase is required. This conforms to the most possible - dynamic scenario - with permanent changing clients (Fig. 6). The second estimation assumes that clients are permanent and a discovery phase is required one time only by the client. After this discovery phase, the service is invoked multiple times by one static client (Fig. 7). This is the most possible - static scenario -. The Figures 6 and 7 present the estimated overall costs for the WSN with increasing number of discovery phases or with increasing number of service invocations. For both estimations of our scenarios, Fig. 8 presents the percentages of the estimated savings for the static and for the dynamic scenario. The results show that the proposed adaptations and enhancements have a deep

[1] The estimations omit headers and footers on network layer and below, as these additional message parts vary depending on the basing technology. The estimations omit retransmissions due to errors also.

saving impact on dynamic scenarios. In more static scenarios without the need for a high number of discovery phases, the savings are minor significant.

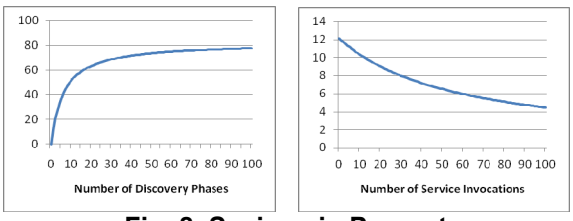


**Fig. 8. Savings in Percent**
**Left: dynamic scenario Right: static scenario**

## 5. Conclusion

This paper describes adaptations and enhancements for DPWS. With the presented modifications it is applicable to deploy DPWS in WSN and thereby allow the seamless connection of WSN and computer networks with one overarching technology. Therefore, a template concept was introduced that allows an optimization of client size and message incidence. Advancements for the discovery process are achieved by adding additional information to the Hello and Probe Match messages, limiting the size of namespaces, introducing eventing filter concepts, and defining a new delivery mode for event notifications. Furthermore, advancements by adding Device Proxies to the WSN are discussed. The requirements for asynchronous messaging and effective compressed XML processing are highlighted and possible solutions are outlined. Additionally, the estimated message savings are presented. Nevertheless, not all problems are solved. The lack of consistent security mechanisms and the energy costs for processing large XML messages are still an ongoing research issue.

### References


[1] Microsoft, Intel, Ricoh, Lexmark, *Device Profile for Web Services Specification*, http://specs.xmlsoap.org/ws/2006/02/devprof, 2006.

[3] IETF, *IPv6 over Low power WPAN (6lowpan)*, Technical report, http://tools.ietf.org/wg/6lowpan/, 2008.

[4] Shelby, Z.; et. al, *"NanoIP: the zen of embedded networking"*, In: IEEE International Conference on Communications (ICC2003), vol.2, pp. 1218-1222, Anchorage, 2003.

[5] IETF Network Working Group, *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*, RFC 4944, http://tools.ietf.org/html/rfc4944, 2008.

[7] Moritz, G.; Prüter, S.; Timmermann, D.; Golatowski, F., *"Real-Time Service-oriented Communication Protocols on Resource Constrained Devices"*, In: International Multiconference on Computer Science and Information Technology (IMCSIT2008), vol. 3, pp. 695 – 701, Proceedings on, , Wisla, 2008.

[8] Moritz, G., et. al, "Web services on deeply embedded devices with real-time processing", *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA2008)*, pp.432-435, Proccedings on, Hamburg, 2008.

[9] Bobek, A., et. al, *"Device and service templates for the Devices Profile for Web Services"*, In: 6th IEEE International Conference on Industrial Informatics (INDIN2008), pp.797-801. Daejeon, 2008.

[10] *IEEE: GUIDELINES FOR 64-BIT GLOBAL IDENTIFIER (EUI-64) REGISTRATION AUTHORITY*, http://standards.ieee.org/regauth/oui/tutorials/EUI64.html, 2007.

[11] OASIS Web Services Discovery and Web Services Devices Profile (WS-DD) TC, www.oasis-open.org/committees/ws-dd/, (2009)

[13] Sandoz, P., et. al, *"Fast Web Services"*, Article, 2003.

[14] VTD-XML, The Future of XML Processing, http://vtd-xml.sourceforge.net/, Technical report, 2008.

[15] Rosu, M.-C., *"A-SOAP: Adaptive SOAP Message Processing and Compression"*, IEEE International Conference on Web Services (ICWS2007), vol., no., pp.200-207, 2007.

[16] Robin Berjon, R., Kangasharju, J., *"Analysis of the EXI Measurements"*, W3C Public Working Group, 2006.

[17] Microsoft Corporation Inc., *"Web Services Dynamic Discovery (WS-Discovery)"*, http://schemas.xmlsoap.org/ws/2005/04/discovery/, 2005.

[18] W3C, *Web Services Eventing (WS-Eventing)*, http://www.w3.org/Submission/WS-Eventing/, 2006.

[19] World Wide Web Consortium (W3C), Web Services Architecture, 2004.

[20] E. Zeeb, et. al, *"Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services"*, In 2nd International IEEE Workshop on SOCNE 07, 2007.

[21] W3C: Namespaces in XML 1.0 (Second Edition). http://www.w3.org/TR/xml-names/, 2006.

[22] WS4D: Web Services for Devices, http://www.ws4d.org, 2009.

[23] P. Levis, et. al, TinyOS: An Operating System for Sensor Networks. *Ambient Intelligence*, Springer, pages 115–148. 2005.

[24] S. R. Madden, et. al., TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst., 30(1):122–173, 2005.

[25] Y. Yao and J. Gehrke, The cougar approach to in-network query processing in sensor networks. SIGMOD Rec., 31(3):9–18, 2002.