WS4D: Toolkits for networked embedded systems based on the Devices Profile for Web Services

Elmar Zeeb, Guido Moritz, Dirk Timmermann
Institute of Applied Microelectronics
and Computer Engineering
University of Rostock
18057 Rostock, Germany
{elmar.zeeb, guido.moritz, dirk.timmermann}
@uni-rostock.de

Frank Golatowski
Center for Life Science and Automation
18119 Rostock, Germany
frank.golatowski@celisca.de

***Abstract*— As the application of the Internet Protocol (IP) is not longer restricted to the internet and computer networks, future IP-based application scenarios require an enormous diversity of heterogeneous platforms and systems. Thereby emerging communication architectures, concepts, technologies and protocols must be capable of handling thousands of devices and communication endpoints on the one hand and be flexible and extensible enough on the other hand, to provide cross domain interoperability independent of platform specific constraints. The Devices Profile for Web Services (DPWS) is such a cross domain technology. This paper provides an overview of DPWS and existing DPWS implementations and toolkits with special focus on the Web Service for Devices (WS4D) initiative. Therefore, features and capabilities of DPWS are described in detail by referring to the open source WS4D implementations. The target platforms are ranging from resource rich server platforms down to highly resource constrained embedded devices.**

I. Introduction

Future application scenarios for networked embedded devices are characterized by a tremendous heterogeneity of cooperating objects. According to Bell's Law [1], every ten years a new class of computers is emerging. With the ongoing technical advances in integrated circuits, the class of the embedded systems can provide more resources. A new device class is formed by Wireless Sensor Networks (WSNs). Early developments in the domain of WSNs were designed as isolated applications. Latest developments are heading towards the application of Internet Protocol (IP) for WSNs also [2], [3]. A similar tendency exists in the domains of embedded systems such as factory automation. Because of arising pervasiveness of IP the usage of matured transport layer protocols like TCP and UDP in all classes of devices make sense, and a former gap is filled. Thereby seamless connectivity of device centric applications with higher valued services is possible also and application scenarios of networked embedded systems are not longer isolated systems.

The increasing complexity of device networks consisting of up to thousands of devices and the availability of IP are demanding new technologies for interoperability. Service-oriented Architectures (SOAs) are often used to improve flexibility and reusability of components in complex distributed applications. This is achieved by modeling functional blocks as independent services. Web services [4] are capable of implementing a SOA and have achieved the highest market penetration. But the Web services technology lacks certain features for device-centric applications like ad-hoc device discovery, device description and eventing channels. Thus Devices Profile for Web Services (DPWS) was developed to enable secure Web Service (WS) capabilities on resource-constraint devices. DPWS is a base technology for device communication that can be easily composed with and extended by other specifications and technologies. DPWS has an architectural concept that is similar but different to the Web Services Architecture (WSA) to fit better into device scenarios. But differences are as small as possible to offer the interoperability to directly integrated DPWS devices into WSA-based enterprise systems.

This paper introduces the (WS4D) initiative, their developments and common goals. WS4D has brought diverse DPWS stacks and toolkits for a variety of system, ranging from resource rich server platforms down to highly constraint wireless sensor nodes with tens of KB RAM and ROM. The remainder of this paper is organized as follows. Section II introduces into basics of the DPWS and outlines the development of current OASIS WS-DD specifications. In section III and section IV related work and the WS4D initiative are described briefly. Section V forms the main part of this paper and provides a detailed overview about existing WS4D implementations and toolkits. The focus of this paper lies in the description of the WS4D-gSOAP and the WS4D-uDPWS toolkit. This is the first publication of the WS4D-uDPWS toolkit, the latest toolkit of WS4D. Section V-E gives a brief comparison of all WS4D toolkits and section VI concludes the results and provides a summary.

II. Devices Profile for Web Services

The probably most popular implementation of the SOA are World Wide Web Consortium (W3C) Web services. But the Web services technology lacks certain features for device-centric applications like ad-hoc device discovery, device description and eventing channels. Thus, DPWS was defined.

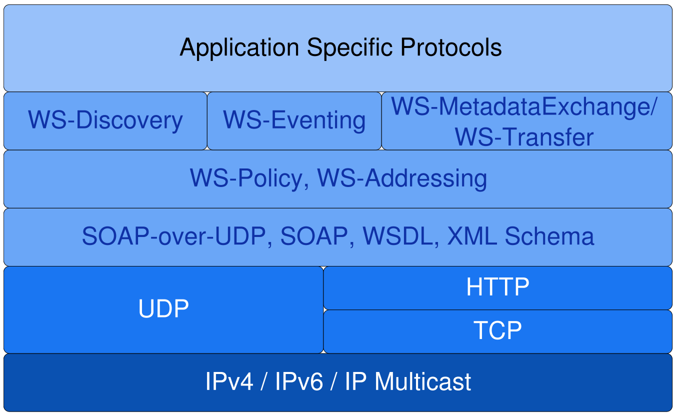

Fig. 1. The Devices Profile for Web Services protocol stack

It uses specific Web services protocols and restricts their usage because of resource limitations in embedded systems. Furthermore, DPWS includes enhancements to fit into device centric applications and offer required functionalities like e.g. the former mentioned discovery and eventing capabilities. So DPWS enables the usage of Web services based technologies to implement device centric SOAs and thus offers the same modular and clearly defined software architectures in device networks ([5], [6]).

DPWS is based on well known protocols and Web service specifications (see Figure 1). It uses similar messaging mechanisms like the WSA [4] with restrictions to complexity and message size ([7], [8], [9]). On top of the low level communication basics it uses Extensible Markup Language (XML), SOAP and XML-Schema for data and information exchange. DPWS specifies mechanisms for ad-hoc device discovery are based on WS-Discovery. The device and service descriptions are based on WS-MetadataExchange and WS-Transfer. Publish-subscribe mechanisms for push messaging in contrast of pull communication pattern are achieved by using WS-Eventing.

In general, DPWS is very similar to Universal Plug and Play (UPnP) while the main difference is the direct alignment of DPWS to the latest Web Service specifications. In UPnP the references to extneral specifications are not updated to latest versions.

DPWS defines a client role that uses the features described in the following paragraphs and a device role that implements these features.

A. DPWS Features

Similar to UPnP **addressing**, the foundation of DPWS is Internet Protocol version 4 (IPv4)/Internet Protocol version 6 (IPv6) addressing. DPWS assumes that a device has obtained a valid IP address. DPWS refers to the Dynamic Host Configuration Protocol (DHCP) and IPv6 auto configuration as mechanisms to obtain valid IP addresses.

The **discovery** mechanisms enables devices to announce their availability in the local network with IP multicast messages. Clients can listen for this messages or send messages to search for devices in the network. All discovery messages can contain device type and device scope information. A device type is a unique identifier that classifies a device and is normally defined at design time. For example printer device or scanner device are device types. The meaning of device types should be specified at design time in separate specifications comparable to a device control protocol (DCP) in UPnP. A device can support (implement) several device types. In contrast to device types, a device scope is a classification that can be configured at runtime. Room 1227 would be an example for a scope to identify all devices that are located in room 1227.

An important feature that was introduced with version 1.1 of DPWS and WS-Discovery is the discovery proxy. This is an alternative mode to the regular mode for discovery based on multicast messages. When clients detect a discovery proxy in the network they switch to managed mode an send discovery requests to the discovery proxy. The discovery proxy reduces multicast messages in the network and can integrate other discovery and directory services into DPWS and WS-Discovery.

The **description** mechanism of DPWS enables the dynamic description of device metadata such as hosted services, device information, model information or service description. This metadata is associated with a metadata version number that is distributed in discovery messages: Thereby clients can track changes of the device description. The interface to retrieve the device description is based on WS-Metadataexchange. As specified in WS-Metadataexchange, metadata is partitioned into sections. DPWS defines which endpoints of a device should provide at least which metadata sections. Custom metadata sections can be specified to extend the device description data for custom applications.

To **control** devices DPWS offer services with operations and **events**. Operations are the same as SOAP Web Services operations, whereby the services hosted on a DPWS device are regular SOAP Web Services. Events are controlled with WS-Eventing and represented as inverse SOAP Web Service operations. This means that client and service exchange roles and the message exchange is triggered by the device. To subscribe for an event, a client can send a subscribe message to the service endpoint. The subscribe message contains the requested delivery mode and event filter. With the delivery mode mechanism a client can negotiate a suitable delivery mechanism. DPWS defines the delivery push mode that sends the events to an endpoint specified in the delivery mode. Further application specific delivery modes can be defined. A dedicated event filter specifies which events the device should send. DPWS defines the action filter. SOAP and WS-Addressing define actions (identifiers) for operations. Thus event filters can be applied and mapped directly to the corresponding inverse operations. Like the flexibility to define own delivery modes, also own event filters can be used to meet applications needs. To transport binary data and other data formats than XML, DPWS includes an **attachment** mechanism to attach arbitrary data to SOAP messages.

The **presentation** is an alternative way to retrieve information about a device and control a device. In DPWS the presentation is announced as part of the device description. It is announced as an HTTP IRI and can be used with a regular web browser.

The DPWS specification defines the concept of **security** profiles. One such security profile is defined in the specification. Devices can support this security profile or others not specified in DPWS. As security requirements heavily depend on application scenarios, the security profile targets easy and lightweight implementation than wide coverage of possible security requirements. Thus the security profile defined in DPWS consists of two main features: secured discovery and secure channel. The discovery multicast messages are secured with WS-Discovery compact signatures. These signatures are similar to XML signatures as defined in WS-Security with less overhead. The secure channel provides encryption on transport layer based on SSL/TLS. Via the secure channel DPWS provides authentication of clients and services, message integrity and confidentiality. A device indicates the support for the security with an HTTPS scheme IRI for its transport address.

A big advantage of DPWS is the **extensible and composable nature** of its specification and Web services in general. Web services mostly define interfaces and technology independent mechanisms. So an implementer is free to implement these interfaces in a way to meet the requirements of a specific application scenario but can still provide interoperability with other implementations. It is up to the implementer to implement DPWS in a specific way. At several places in the specification as for example in the discovery part DPWS provides several options for implementers. In the case of discovery DPWS defines a generic discovery mechanism to discover devices in a subnet that offers high dynamics but may produce high network traffic. However if DPWS clients can discover devices by other means like static configuration or a more static discovery scheme they are still within the scope of the DPWS specification.

How certain features of DPWS are implemented causes different properties of the resulting system concerning performance, scalability, extendibility, interoperability, etc. Hence an implementer should further profile the features of DPWS to meet application scenario specific requirements. This could be done in a DCP specification where the semantic of a DPWS device type is described. Such specifications are application scenario specific and can combine further Web service specifications.

B. DPWS Versions

The DPWS version 1.0 was initially published in May 2004 at http://schemas.xmlsoap.org/ and was submitted for standardization to OASIS in July 2008. Within Web Services Discovery and Web Services Devices Profile (WS-DD) Technical Committee (TC) DPWS 1.1 was approved as OASIS Standard together with WS-Discovery 1.1 and SOAP-over-UDP 1.1 on June 30, 2009. During the standardization process of DPWS, the interoperability of several DPWS implementations (including the WS4D toolkits) was tested. At the moment the WS-DD TC is working on DPWS 1.2 and also the 1.2 versions of WS-Discovery and SOAP-over-UDP.

DPWS versions are not downwards compatible. Each DPWS version has a distinct XML namespace. Devices or clients supporting several versions of DPWS must implement several namespaces (and thus versions) of DPWS.

The main changes between DPWS version 1.0 and 1.1 is the update of the referenced WS-Addressing specification to version 1.0, definition of the discovery proxy in WS-Discovery and clean ups in the security feature of DPWS.

As DPWS version 1.2 is still work in progress not all changes are known yet. The biggest change known at the moment is the update of the referenced specifications WS-Eventing, WS-Transfer and WS-Metadataexchange that are standardized in the Web Services Resource Access (WS-RA) Working Group at the W3C.

III. Related work

Beside Service-oriented Architectures (SOAs), Resource-oriented Architectures (ROA) like described in the REST design style [10] are a proper approach to provide interoperability in heterogeneous deployments. Both provide basic functionalities to meet requirements not only of single application scenarios but to be applied as platform independent cross domain technologies. The underlying architectures are most remarkable difference between both, but while based on partly same protocols and technologies, discrepancies are preprogrammed. Most RESTful deployments base on HTTP as application protocol directly in contrast of an application layer transport protocol like in the SOAP Web services.

While Web services based SOAs do not define semantic meanings of the used methods of a service, RESTful deployments restrict the used methods to a simple CRUD style (Create, Request, Update, Delete). These restrictions concerning methods and about stateless design of the server in RESTful deployments may lead to more lightweight implementations, independent of the used protocols. But DPWS must not be used to model a SOA and can also be used to realize a RESTful application. Hence DPWS and RESTful style are not a contradiction. The OASIS WS-DD technical committee is working in close cooperation with the Web Services Resource Access Working Group (WSRA)[1] at the W3C. WS-RA is heading towards specifications for resource oriented Web services, represented in XML and manipulated with SOAP based protocols. DPWS for example uses WS-Transfer of WS-RA to exchange device metadata. Hence the metadata is modeled as a resource and requested with a lightweight well defined operations.

Furthermore, Schneider Electric (F) – project coordinator of the SIRENA project – has developed the first implementation of a DPWS stack for embedded devices in the SIRENA project [6]. The SIRENA follow-up project – Service Oriented Device and Delivery Architecture (SODA) has brought

[1] http://www.w3.org/2008/11/ws-ra-charter.html (2010)

forward a toolkit around the Schneider stack to improve manageability, orchestration and security [11]. In the current active OSAMI [12] where several WS4D partners are involved, DPWS is the core technology for Remote OSGi [13] and for sensor device integration.

During standardization process in OASIS WS-DD, the different mentioned implementations of DPWS are tested for interoperability. The next section will present the available implementations of the WS4D initiative, their specific characteristics and application scenarios.

IV. The WS4D Initiative

The *Web Services for Devices (WS4D)* [14] initiative was established by academic and industrial partners in the middle of 2006 for several reasons. Before forming the WS4D community, all partners were actively involved in the award-winning European R&D project SIRENA, which ended in spring-time 2006. One of the main challenges in SIRENA was to establish a software infrastructure in which networked embedded devices can be integrated. The devices should be self-contained and able to communicate with each other. As one result, some prototypes implementing earlier versions of DPWS were developed.

The WS4D initiative can be considered as a non-profit follow-up project to preserve and extend the SIRENA results and also to maintain the collaboration between the partners. Since the founding of WS4D, the partners were involved in miscellaneous pursuing international projects like LOMS, OSAMi and uServices. But the main scope of WS4D is still participating in further development processes:

1) Creating a community arround DPWS and the WS4D toolkits,
2) improving available toolkits by providing them as *open source* software to animate the community to contribute,
3) Developing *test suites*, and
4) Promoting *standardization* process, e.g. delivery of standardized devices and basic services such as management services.

By putting these items into action, WS4D creates interoperable solutions. In a nutshell, the overall objective of WS4D is ensuring interoperability between various implementations of the Devices Profile on different platforms and different programming languages by the help of an active and open community. Currently the partners of WS4D are also members in OASIS WS-DD and thus are actively pushing the standardization process of DPWS.

V. Toolkits

In the following sections the four WS4D Web services development toolkits are introduced: WS4D-gSOAP, WS4D-uDPWS, WS4D-Axis2 and WS4D-JavaME. These toolkits are published under open source licencees. More detailed Information is available on the WS4D website [14].

The WS4D-gSOAP, WS4D-uDPWS and WS4D-Axis2 stacks are maintained by the University of Rostock and the WS4D-JavaME stack is maintained by the University of Dortmund and Materna Information & Communications [15].

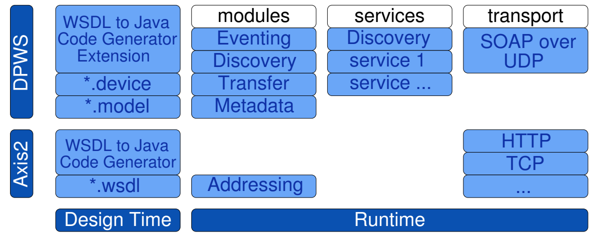

Fig. 2. Axis2 Architecture and DPWS Extensions

A. WS4D-Axis2

The WS4D-Axis2 DPWS stack is based on Axis2 being the successor of Axis – a Java-based SOAP processor for Web services [16]. As with all projects of the Apache group Axis2 is also available as an open source software with an Apache licence.

Axis2 supports SOAP 1.1 as well as SOAP 1.2, offers several transport connectors such as HTTP, TCP, JMS and SMTP and comes with a WS-Addressing enabling module. So by default asynchronous message exchange is supported.

Axis2 is a modular built SOAP stack in which implementations of additional Web services specifications can be easily plugged in via modules.

By using Axis2 we can bridge the two worlds of embedded devices and application development at the enterprise level without abandoning existing solutions and customs: Axis2 runs on J2SE, an Eclipse plug-in is available for code creation, and further lots of plug-ins implementing other WS specifications such as WS-Security, WS-ReliableMessaging, WS-Coordination and WS-AtomicTransaction are available or on the way.

Since the engine is Java-based several platforms (Windows, Linux etc.) are targeted. Services are deployable in standalone mode or under servlet container (default) such as Tomcat.

WS4D-Axis2 is a stack solution on top of Axis2 as depicted in Fig. 2 for writing primarily clients for controlling DPWS enabled devices, e.g. for management tasks. WS4D-Axis2 comes with several modules (plug-ins) which are independent of each other: SOAP over UDP, Discovery, Eventing and DPWS. It offers three main APIs for searching for, subscribing to and controlling devices.

Since WS4D-Axis2 runs on J2SE it is not targeted for embedded devices, but it can be used for device management, discovery proxies and other rich client implementations.

B. WS4D-JMEDS

WS4D-JMEDS stands for WS4D Java Multi Edition DPWS Stack and is targeted at for embedded devices with low amount of memory and restricted computing power as well as for computing systems without such limitations. The modularity of JMEDS allows the usage of different modules and ensures high adaptability. JMEDS supports different Java Editions, including Java ME CLDC [17]. Additionally, it defines

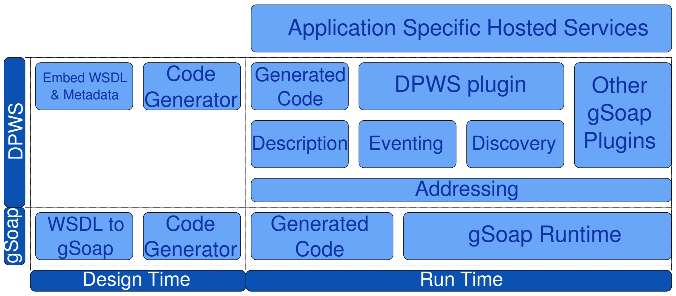

Fig. 3. WS4D-gSOAP toolkit

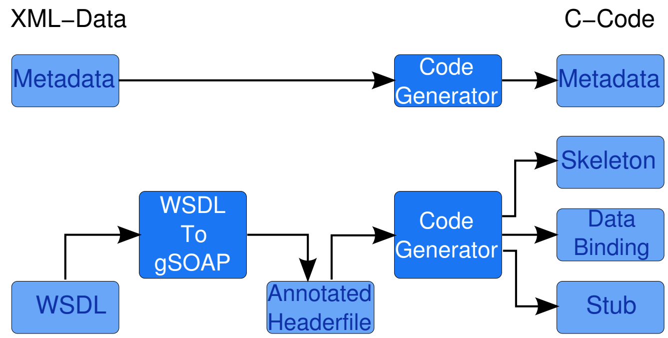

Fig. 4. WS4D-gSOAP code generation

an environment (*Connected Device Configuration*) for more capable devices like set-top boxes or other high-end embedded devices. The *WS4D-JMEDS* stack is based on the *Connected Limited Device Configuration*, the smallest subset of JavaME configurations and can thus be used on all JavaME platforms and even on Java 2 Standard editions using platform dependent toolkits.

C. WS4D-gSOAP

WS4D-gSOAP is an extension of the well known gSOAP Web services toolkit, a toolkit for building SOAP-based Web services with C/C++ developed by Robert A. van Engelen [18]. It is designed to develop small footprint and high throughput Web services. The toolkit consists of a development and a runtime environment.

gSOAP offers code generation tools for implementing Web services. gSOAP has defined its own service description language that is based on C syntax. This description is contained in special gSOAP files that are similar to C header files with annotations. To complete the Web services design flow the toolkit also includes a tool to translate WSDL files into gSOAP files.

The second part of the development environment is the gSOAP code generator. It generates XML schema to C data binding as well as stub and skeleton code for a specific gSOAP service description. The XML schema to C data binding creates a mapping from every type of the used XML schema definitions to a C type structure and generates functions for marshalling and demarshalling. The skeleton and stub code generator finally maps WSDL operations to C functions.

The runtime part of gSOAP consists of the generated code and the gSOAP runtime. The gSOAP runtime consists of functions for the service developer and functions used in the generated code.

As shown in Fig. 3 the WS4D-gSOAP toolkit uses gSOAP's plug-in mechanism to implement WS-Addressing, WS-Discovery, WS-MetadataExchange / WS-Transfer and WS-Eventing on top of gSOAP. Further features of WS4D-gSOAP are described in the following paragraphs. The features described in this section are part of the WS4D-gSOAP 0.8 release.

WS4D-gSOAP uses a similar workflow as gSOAP for **code generation** (see Fig. 4). To create a DPWS device a developer has to specify a WSDL description of the services on a device and the device's metadata. The WSDL files are used for code generation in gSOAP's typical way as described in the last paragraph. The device metadata is used to generate code for service setup and assignment of model metadata and device characteristics. With the resulting code a developer can concentrate just on the implementation of the functionality of the services hosted by a device.

WS4D-gSOAP supports **IPv4** by default. **IPv6 support** can by activated by enabling the a cmake option. This mode is still experimental and not yet finished. The **DPWS security support** is also implemented partially. At the moment the secure channel mechanism based on SSL/TLS and all related features is implemented with the help of gSOAP. The discovery compact signatures are still missing.

WS4D-gSOAP supports four different **endpoint types** that can be switched at compile time: device, client, hosted and peer. With the device role an endpoint implements the device side of the specification. The client role is used to create code for a Web service client, respectively. The peer role has to be used when both client and device are about to be integrated in one application. The hosted endpoint type is later described in the hosting service paragraph.

Currently, WS4D-gSOAP supports two **DPWS versions** that can be switched at compile:

- 2006_02 (default): This is the DPWS version 1.0.
- 2009_01: This is the official OASIS DPWS version 1.1.

WS4D-gSOAP and gSOAP **support several operating systems** with POSIX like APIs and BSD Socket like network API. Besides the network API gSOAP and WS4D-gSOAP has only few dependencies on operating system APIs. Thus WS4D-gSOAP offers multi-platform support such as the Linux x86, Windows-native, Windows-cygwin and embedded Linux (FOX Board [19] and Nokia Maemo/Meego [20]) platforms. To develop devices a typical GNU software development toolchain can be used. Developers preferring integrated development environments can use Visual Studio 8.0 on Windows or Eclipse on other platforms.

In addition WS4D-gSOAP supports **cross compiling** for systems with POSIX and BSD Socket like APIs with the help of cmake and the toolchain file mechanism of cmake. WS4D-gSOAP already supports several embedded systems with toolchain that are part of the source distribution. With

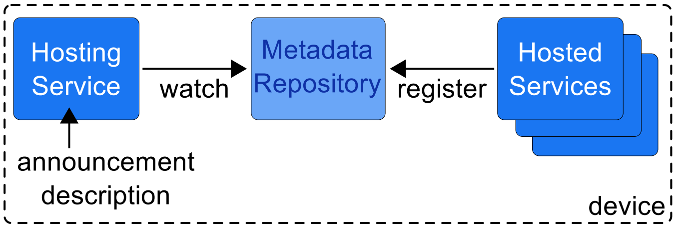

Fig. 5. WS4D-gSOAP hosting service

TABLE I
ROM USAGE OF UDPWS IN BYTE

	ROM	ROM Source Code (.text)	ROM Generated Code (.text)	ROM (.data)
System	3992	3992	-	-
Contiki	30418	30280	-	138
Service Modules	509	148	355	6
Device Modules	690	26	660	4
uDPWS Core	10275	8344	1771	160
Total	**45884**	**42790**	**2786**	**308**

this mechanism further cross compilers can be added.

For systems that do not offer POSIX and BSD Socket like APIs, WS4D-gSOAP has a **alternative API mode** to implement wrapper code to integrate other network and system APIs. This can can be activated with cmake(ALT_IO_MODE). Currently WS4D-gSOAP supports the following ALT_IO_MODEs:

- none (default): no alternative mode, uses the POSIX and BSD Socket like API.
- lwip: uses the lwIP [2] TCP/IP network stack for networking.

The **hosting service** bases on the separation of the hosting service from the hosted services on a device. Figure 5 illustrates the services on a device and how they can be separated from the hosting service. With this separation the DPWS specific discovery and description part can be split into a separate process that is called hosting service. Hosting and Hosted services share a metadata repository where hosted services put their metadata and the hosting services retrieves the information to announce in the network, This approach improves modularity and offers deployment of services during the runtime of a device. Furthermore a device can be distributed by announcing services running on different physical units.

The **life cycle manager** is a hosted service based on the hosting service described above to manage the services hosted on a device. The life-cycle-manager is able to install, start, stop, update and remove services during run-time with a Web service interface.

D. WS4D-uDPWS

The latest implementation of DPWS is called WS4D-uDPWS and is focusing wireless sensor nodes as target platforms. These platforms are characterized by enormous resource constraints like memory, computing power and bandwidth. Beside the approach implemented in WS4D-uDPWS, different other approaches have been proposed for DPWS in 6LoWPAN networks [3]. In [21] and [22], a mapping of DPWS to generic WSN protocols is described. Both approaches use intermediate devices or proxies for the mapping. uDPWS is capable of using DPWS in 6LoWPAN networks directly without the need of intermediate proxies.

uDPWS is currently in the state of a prototype to enable DPWS device and service provider functionalities in 6LoWPAN wireless sensor networks. To not start from the scratch we used the Open Source operating system Contiki [23] release 2.3, which already includes 6LoWPAN, TCP and UDP network stack [24]. The uDPWS implementation is written in C and compliant with current WS-DD specifications in most points. Not implemented features of the WS-DD specifications are related to security mechanisms due to missing SSL libraries that fit the requirements. As hardware platforms Crossbow TelosB[3] and Atmel Raven 2.4GHz Evaluation and Starter Kit[4] are currently supported.

Sensor nodes with very limited resources are not expected to provide complex services and methods. The integration of this class of devices in device-centric SOAs and seamless integration resource richer systems is crucial. Analyzing the SOAP header fields WS-Addressing:To, WS-Addressing:Action and WS-Addressing:MessageID is sufficient for most application scenarios. Other vendor or deployment specific header fields might also be analyzed, but are out of scope if this paper. Unknown header fields or not supported values are replied with the corresponding SOAP faults.

As the outgoing messages of the DPWS node are related to the capabilities of the node, most parts of these messages are already known at development time and are static. Only minimal changes of the messages are required at run time. Hence, a lookup table based approach for response generation is used (c.f. [25]). The required static messages are included at compile time with a code generator.

The ROM usage of the uDPWS stack including the generated code and the related underlying Contiki operating system is shown in table V-D. The service modules include all implementations of service specific (i.e. Hosted Services) invocations. The device modules include all device specific (i.e. Hosting Service) service invocations. Based on the parsed SOAP header fields, the SOAP body is processed by the corresponding module. This allows tailored implementations by omitting modules at design time which are not required at run time. The modules are responsible for processing the SOAP body, performing required actions on the sensor nodes and generating the corresponding response if required. For the SOAP Body processing by the corresponding module, a lightweight XML parsing library is included in uDPWS. For event delivery a different processing flow is required, because the device has to initiate a connection actively and

[2] http://savannah.nongnu.org/projects/lwip/

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

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

TABLE II
FEATURE COMPARISON OF WS4D-* DPWS TOOLKITS

Feature	WS4D-Axis2	WS4D-gSOAP	WS4D-J2ME	WS4D-uDPWS
DPWS Version support	V1	V1 & V1.1	V1 & V1.1	V1.1
IPv6 support	-	partial	Yes	IPv6 only
target platform	server	embedded	embedded	deeply embedded
DPWS security support	-	secure channel	secure channel	-
discovery proxy support	-	-	yes	-
presentation support	-	-	yes	-
MTOM support	yes	yes	yes	-
eventing support	-	yes	yes	-
SOAP-over-UDP service binding support	yes	yes	yes	-

is not waiting for incoming requests to process. A dedicated software module in uDPWS, the event manager, is responsible for managing the subscriptions. Services indicating an event to be delivered register at the event manager and hand over a callback function. The event manager queues all events to be delivered. For each event delivery the callback can be used to generate the corresponding SOAP body content. Hence the SOAP body content generation is in responsibility of the service and not the event manager. The event manager queues the generated event delivery message for sending in the processing loop of uDPWS.

E. Comparison

Table II gives an brief feature comparison of all WS4D DPWS toolkits.

F. Other DPWS implementations

Besides the open source DPWS implementations of WS4D there are several other DPWS implementations available.

The Service Oriented Architecture for Devices (SOA4D) is a similar project as WS4D. The mission is to foster an ecosystem for the development of service-oriented software applications based on DPWS. As the SOA4D community was initiated on 2007 by Schneider Electric it is also originated in the ITEA SIRENA project. The SOA4D.org is a open source software development web site, supporting and providing open source Web services toolkits:

- DPWS Core: provides an embeddable C Web Services stack, compliant with the Device Profile for Web Services (DPWS) specification.
- DPWS4J Core: provides a Java Web Services stack for the J2ME CDC platform, compliant with the Device Profile for Web Services (DPWS) specification.

Microsoft as one of the authors of the DPWS specification ships its latest Windows versions (Vista and 7) with an implementation of the DPWS protocol stack, which is called *WSDAPI*. This API is part of the new PNP-X subsystem which allows locally installed devices and such attached through the network (e.g. by UPnP or DPWS) to be accessed and used in a uniform way [26].

Additionaly Microsoft provides DPWS implementations for the Microsoft .NET Micro Framework.

VI. CONCLUSION AND FUTURE WORK

In this paper we introduced the WS4D initiative, i.e. its purpose, the tools which are currently in progress and upcoming challenges. The main objective is building up an open community which practically applies the Devices Profile for Web Services to attain interoperability.

We will further foster the standardization process. One of the next tasks will be a proposal for device templates. Such template system could standardize certain device types and provide similar advantages (e.g. easy code generation) like device templates in the UPnP technology.

ACKNOWLEDGMENT

This work has been funded by German Federal Ministry of Education and Research (BMBF) under reference number 01IS08003 in the context of the European ITEA2 OASMI project.

REFERENCES

[1] G. Bell, "Bell's law for the birth and death of computer classes," *Commun. ACM*, vol. 51, no. 1, pp. 86–94, 2008.

[2] J. W. Hui and D. E. Culler, "Ip is dead, long live ip for wireless sensor networks," in *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*. New York, NY, USA: ACM, 2008, pp. 15–28.

[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 Architecture*, World Wide Web Consortium (W3C), 2004.

[5] H. Bohn, A. Bobek, and F. Golatowski, "SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains," in *International Conference on Networking (ICN)*, 2006.

[6] F. Jammes, A. Mensch, and H. Smit, "Service-oriented device communications using the devices profile for web services," in *3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC05) at the 6th International Middleware Conference*, 2005.

[7] *Devices Profile for Web Services Version 1*, Microsoft, Intel, Ricoh, Lexmark, 2006, http://schemas.xmlsoap.org/ws/2006/02/devprof/.

[8] *Devices Profile for Web Services Version 1.1*, OASIS, 2009, http://docs.oasis-open.org/ws-dd/dpws/1.1/os/.

[9] E. Zeeb, A. Bobek, H. Bohn, and F. Golatowski, "Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services," in *2nd International IEEE Workshop on SOCNE 07*, 2007.

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

[11] SODA consortium, *SODA - Technical Framework Description*, 2007, http://www.soda-itea.org/Documents/AllDocuments/.

[12] OSAMI-Commons consortium, *OSAMI - Open Source Ambient Intelligence Commons for an Open and Sustainable Internet*, 2010, http://www.osami-commons.org/.

[13] C. Fiehe, A. Litvina, I. Lück, O. Dohndorf, J. Kattwinkel, F.-J. Stewing, J. Krüger, and H. Krumm, "Location-transparent integration of distributed osgi frameworks and web services," in *Proceedings of the IEEE 23rd International Conference on Advanced Information Networking and Applications (AINA 2009)*. Bradford, UK: IEEE Computer Society, 2009, pp. 464–469.

[14] WS4D: Web Services for Devices, http://www.ws4d.org, 2010.
[15] Materna Information & Communications, http://www.materna.com, 2010.
[16] Apache Axis2 / Java, http://ws.apache.org/axis2/, 2010.
[17] Sun Microsystems, *Java 2 Micro Edition*, 2007, http://java.sun.com/javame/index.jsp.
[18] R. A. van Engelen, *gSOAP*, 2007, http://www.cs.fsu.edu/~engelen/soap.html.
[19] Acme Systems, http://www.acmesystems.it, 2010.
[20] Maemo: Development Platform for Nokia Internet Tablet Products, http://www.maemo.org, 2010.
[21] J. Leguay, M. Lopez-Ramos, K. Jean-Marie, and V. Conan, "An efficient service oriented architecture for heterogeneous and dynamic wireless sensor networks," oct. 2008, pp. 740 –747.
[22] I. Samaras, J. Gialelis, G. Hassapis, and V. Akpan, "Utilizing semantic web services in factory automation towards integrating resource constrained devices into enterprise information systems," sept. 2009, pp. 1 –8.
[23] A. Dunkels, B. Grnvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, Nov. 2004. [Online]. Available: http://www.sics.se/~adam/dunkels04contiki.pdf
[24] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels, "Making sensor networks ipv6 ready," in *Proceedings of the Sixth ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008), poster session*, Raleigh, North Carolina, USA, Nov. 2008, best poster award. [Online]. Available: http://www.sics.se/~adam/durvy08making.pdf
[25] G. Moritz, S. Pruter, D. Timmermann, and F. Golatowski, "Web services on deeply embedded devices with real-time processing," in *Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA2008)*, sept. 2008, pp. 432 –435.
[26] Microsoft Rally, http://www.microsoft.com/rally, 2010.