Abstract—Web Service technologies are state of the art in current enterprise distributed applications. To integrate highly resource-constrained devices, such as sensor nodes, into enterprise applications via Web Services, gateways are commonly used. Our aim was to investigate the intermediate, homogeneous communication between enterprise applications and highly resource-constrained devices. We therefore present the prototype implementation of an embedded enterprise compatible W3C Web Service solution. Furthermore, we give an evaluation of memory requirements and a timing analysis for a representative platform with 48 KiB of ROM, 10 KiB of RAM and IEEE 802.15.4 radio. As the prototype implementation requires only 8.5 KiB of ROM and 3 KiB of RAM, our results claim that enterprise Web Services, despite of prevailing assumptions, are feasible in this context.

Keywords—Web Services, Wireless sensor networks, resource-constrained devices, DPWS, 6LoWPAN

I. INTRODUCTION

Networked embedded systems play an essential role in a variety of application domains such as automotive, industrial automation and avionics. They consist of spatially and functionally distributed and interconnected embedded nodes. Their primary use is to monitor and control certain systems and environments [1].

In the field of wireless communication systems and networked embedded systems, wireless sensor networks (WSNs) have become a main topic of research activities. Despite the amount of research which has been done, aiming at the integration of WSNs to enterprise systems, they are still not widely used in industry [2]. Partly, this is due to missing uniform, general-purpose service interfaces, which are difficult to design for a large diversity of WSNs [3]. However, recent investigations on IP-based communication for WSNs promise to bring interoperability on the network layer of communication systems. The 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) working group has defined mechanisms that allow efficient IPv6-based communication over IEEE 802.15.4 based networks (IETF RFC 4944). The use of gateways and protocol translation, which is common in classical wireless sensor network architectures to connect WSNs to the Internet [3], can be avoided this way. Nevertheless, IP does not automatically enable syntactic interoperability at higher protocol levels. As a result, many existing sensor network setups, like in [4], use specialised binary protocols and require gateways. To overcome gateway architectures, we demonstrate how to bring together networked embedded systems with the well known and widely-used W3C Web Service standards (WS-*), which are based on the SOAP protocol. We illustrate that W3C’s main statement about Web Services [5] also applies to networked embedded systems:

Web Services provide a standard means of inter-operating between different software applications, running on a variety of platforms and/or frameworks.

In this paper we describe a novel standards compliant Web Services implementation for networked embedded systems. The intent is to eliminate the need for gateways for their integration to modern IT systems. We assume, that this will increase to use smart objects in future applications, thus, fostering the paradigm of pervasive computing. The presented prototype implementation is based on the WS-subset Devices Profile for Web Services (DPWS), an OASIS standard [6]. While DPWS was not specifically developed for highly resource-constrained devices, we show that it is applicable. In addition we state that constraints need to be considered.

II. RELATED WORK

The use of Web technologies in sensor networks has been explored in research work over the past few years. Within this field, basic standards of the Internet like the Internet Protocol (IP), and enterprise SOA solutions, such as W3C Web Service protocols, are seen as important technologies for the integration of highly resource-constrained devices (in the following device) to enterprise systems. Web Services refer to common techniques to develop interoperable distributed applications. Web Service protocols have already been considered for WSN gateway architectures in [3, chapter 3]. This leads to the conclusion, that IP can be used to achieve network layer interoperability. We presume that Web Services are an enabling technology for interoperability at upper network layers. An initial Web Service implementation for WSNs was given by the SOCRADTES project, where DPWS was used on gateways, while WSN devices have been considered as not being powerful enough.
The architectures to integrate devices in network systems, as proposed by previous works, mainly differ in terms of network protocols and gateway functionalities. While some of the proposed architectures require gateways to introduce Web technologies for device interfaces, other approaches aim to embed Web technologies directly on devices. Previous experiences have demonstrated that an implementation of Web application protocols directly on devices in general is feasible.

The feasibility of using RESTful Web Services in an IP-based multi-hop low-power sensor network and an evaluation of performance and power consumption was shown in [7]. The aim of this work was to allow the direct interaction between Web Service-based IT systems and IP-based sensor networks. The results claim that Web Service requests can be completed below one second and with a low power consumption.

In [8] an architecture to integrate real-world embedded devices to the Web with the help of a device mashup application was presented. Contrary to [7], the proposed architecture lacks support of IP connectivity for the focused class of devices. Instead, “Smart Gateways” were used to offer the devices functionality through a RESTful API to the Web.

Another REST based architecture was described in [9]. Even though the sensor nodes in this architecture run an embedded IP stack, a gateway is still required to apply a transparent compression of JSON object data. Thus, compressed JSON is used for proxy-to-device communication, while standard JSON is used for proxy to Web communication. Nonetheless, it can be presumed that JSON compression could be omitted in favor of a non-gateway architecture.

Similar to our work, [10] analysed the use of W3C Web Services with XML for data exchange on highly resource-constrained devices. The authors argued that XML Web Services are desirable for use in embedded systems. Despite the use of XML Web Services on devices, the authors describe that the widely-used SOAP protocol is not supported by devices themselves. For this purpose an additional “controller” gateway must be used.

As a conclusion, previous studies have indicated that, for implementing Web Services directly on devices, RESTful approaches or gateway architectures are required. However, little attempt has been made to implement and evaluate enterprise, SOAP-based Web Services on highly resource-constrained devices. The purpose of our investigation was to determine if an implementation of enterprise Web Services is possible and if it is feasible to run them directly on highly resource-constrained hardware.

To begin with, we describe the implementation of a W3C Web Service implementation for resource-constrained devices.

### III. Prototype Implementation

In this section we present the uDPWS prototype\(^1\) which is an implementation of the Devices Profile for Web Services (DPWS) for highly resource-constrained devices. First, this section gives an overview about resource-constrained devices. Second, the concept and architecture of uDPWS is briefly described.

DPWS defines a subset of the WS-* specifications suitable for “resource-constrained endpoints” [6]. Even though there is no information about performance prerequisites given in the DPWS standard, it is obvious that devices with around 10 KiB of RAM have been out of focus. DPWS proposes a Web Service based architecture for networked embedded devices and tries to foster interoperability between small embedded devices and fully Web Service-capable computers. The DPWS standard states that a DPWS-Device is represented through a Web Service which is called the Hosting Service. The Hosting Service is found by DPWS-Clients in a network using WS-Discovery. Furthermore, the Hosting Service provides the device description through WS-MetadataExchange. The device offers access to its functionality through Hosted Services. For publish-subscribe communication DPWS uses WS-Eventing.

uDPWS has a modular structure, which is shown in Figure 1. The uDPWS-Core consists of four modules, which are highlighted grey: the uDPWS-Process-, HTTP-, DPWS- and Sending-Machine module. The uDPWS-Process handles the distribution of incoming connections and messages. It is the only module that interacts with the operating system (OS) Contiki. As a result, porting to different operating systems, such as TinyOS, can be accomplished by adopting the uDPWS-Process module. Contiki comes with an embedded TCP/IP stack and an implementation of 6LoWPAN which provides standardised IPv6 based communication over IEEE 802.15.4. After a new connection is established, the HTTP module buffers request messages. As soon as the complete message is buffered, it is forwarded to the DPWS module. If the message is dedicated to one of the Hosted Services, the DPWS module forwards the message content to the corresponding service implementation. Otherwise, if the message is a DPWS message (e.g., WS-MetadataExchange), the DPWS module itself handles the message. The DPWS module is the main module of the uDPWS-Core. Finally, response messages are created using the Sending-Machine module in a very resource-saving manner: As most parts of the XML messages are constant (e.g., XML namespaces) and are already stored in ROM, this module avoids a separate response message buffer by concatenating memory references of XML message fragments to a list. Therefore, only dynamic message fragments (e.g., measurement values) need to be buffered in RAM.

In uDPWS, sensors and actuators are modeled as DPWS-Devices. Basically, three steps are required to create a new

\(^1\)uDPWS is open source and available at http://www.ws4d.org/udpws/
IV. EVALUATION AND DISCUSSION OF RESULTS

We have tested our embedded Web Service stack uDPWS on a representative hardware platform. A performance evaluation of the prototype implementation is presented here. As an early evaluation we concentrated on memory usage and timing analysis for typical message exchange patterns. Therefore, in contrast to characteristic requirements of WSNs, an analysis of lifetime and scalability was out of scope.

A. Experimental Setup

We chose Crossbow’s Telos B as hardware reference platform. The TelosB development board is based on an MSP 430 microcontroller and is equipped with an IEEE 802.15.4 compatible radio chip, light and temperature sensors, and a USB interface. The MSP 430 operates at 8 MHz, has 10 KiB of internal RAM and 48 KiB Flash memory.

We examined a simple DPWS-Device consisting of a DPWS Hosting Service and one Hosted Service which provides the single method `GetTemp` to receive temperature measurements. The source code was compiled with mspgcc 3.2.3.

Obtained timing measurements are based on a single-hop scenario, which is illustrated in Figure 2. On the right side the TelosB running the DPWS-Device is connected via 6LoWPAN to a Host PC (3 GHz, 2 GB RAM). Therefore, another TelosB is used, which acts as a IPv6/6LoWPAN-Bridge. It is connected via USB to the Host PC on the left. Hence, the Host PC is able to emulate a local Ethernet device to communicate with the Sensor Node.

Table I

<table>
<thead>
<tr>
<th>PARTITION</th>
<th>ROM</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>System &amp; OS</td>
<td>34410</td>
<td>74.99</td>
</tr>
<tr>
<td>uDPWS-Core</td>
<td>10275</td>
<td>22.39</td>
</tr>
<tr>
<td>Services &amp; Application</td>
<td>1199</td>
<td>2.62</td>
</tr>
<tr>
<td><strong>∑</strong></td>
<td><strong>45884</strong></td>
<td><strong>100.00</strong></td>
</tr>
</tbody>
</table>

B. Memory Usage

This evaluation presents the memory usage for both ROM and RAM memory of the TelosB hardware reference platform. This was accomplished by evaluating the linker section sizes.

We separated the memory usage into three partitions: (1) System & OS, (2) uDPWS-Core and (3) Services & Application. The first part of the System & OS partition includes the bootloader, the interrupt vector table and the standard library (libgcc). The second part contains the OS Contiki, including the TCP/IP and 6LoWPAN implementation. Contiki, as well as the TCP/IP implementation, has been configured in a way that memory consumption was reduced to a minimum. The uDPWS-Core partition holds the four main modules as shown in Figure 1. Finally, the Services & Application partition includes the implementation of the DPWS-Device, Web Services and the application, which performs temperature measurements by accessing the hardware.

Table I gives an overview about the ROM and RAM usage. Beside that, it should be noted that with 55 % (5651 B) of the ROM usage, the DPWS module is by far the dominant part of the uDPWS-Core. Further, 27 % (2786 B) of the uDPWS-Core ROM are constant XML fragments of Web Service messages.

The investigation of the RAM usage has shown that message buffers require most of the available memory. Contiki requires an IPv6 buffer with at least 1280 B, given by the IPv6 Minimum Transmission Unit (MTU). Additionally, the uDPWS-Core needs a SOAP buffer, which was configured to a size of 2000 B. One should notice that the program stack was not taken into account. However, it can be assumed that the program stack does not exceed 1 KiB. Since Contiki and uDPWS-Core do not allow dynamic memory allocation, there is no heap to be taken into account.

In summary, the complete prototype implementation requires 93 % of ROM and 69 % of RAM of the hardware reference platform Telos B.

C. Roundtrip Times

We analysed all types of request-response message patterns which are supported by DPWS and the message patterns.
pattern of the Hosted Service method GetTemp. The typical message exchange at TCP-level is shown in Figure 3. The roundtrip time is the sum of the duration of the following five steps: (1) connection establishment, (2) request transmission, (3) processing, (4) response transmission, and (5) connection closing. The roundtrip time measurements can be depicted in Table II.

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Request Size [B]</th>
<th>Response Size [B]</th>
<th>Roundtrip Time [ms]</th>
<th>Processing Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectProbe</td>
<td>764</td>
<td>1232</td>
<td>470.80</td>
<td>12.82</td>
</tr>
<tr>
<td>GetMetadata Device</td>
<td>651</td>
<td>2268</td>
<td>617.81</td>
<td>11.62</td>
</tr>
<tr>
<td>GetMetadata Service</td>
<td>667</td>
<td>831</td>
<td>358.31</td>
<td>8.91</td>
</tr>
<tr>
<td>GetTemp</td>
<td>650</td>
<td>751</td>
<td>343.37</td>
<td>8.94</td>
</tr>
<tr>
<td>Probe (UDP)</td>
<td>668</td>
<td>1139</td>
<td>325.32</td>
<td>6.09</td>
</tr>
<tr>
<td>Resolve (UDP)</td>
<td>814</td>
<td>1033</td>
<td>328.37</td>
<td>6.60</td>
</tr>
</tbody>
</table>

Figure 3. TCP transmission diagram

As a conclusion, processing takes less than 3% of the overall roundtrip times. This points out, that message processing is not the bottleneck for XML Web Services on highly resource-constrained devices. Instead, to reduce roundtrip times in this context, message sizes must be reduced dramatically. Further investigations have shown that especially the connection establishment and closing of TCP has a deep impact on roundtrip times. This leads to an almost constant overhead of 66 ms for each TCP connection. In contrast, the minimum roundtrip time of an empty UDP message was 24 ms.

V. CONCLUSION AND OUTLOOK

In this paper we presented a prototype implementation of the DPWS standard for highly resource-constrained devices. It emerged that the limited resources are still sufficient to process XML data. However, the increased transmission times are significantly noticeable. In contrast to that, we pointed out that data processing time can be neglected in this context. We conclude that it is necessary to put more effort into reducing message sizes to allow an effective use of XML/SOAP based Web Services on highly resource-constrained devices. This could be achieved by using UDP instead of TCP. Furthermore, we expect a significant reduction of message sizes from current research activities concerning the compression of XML data [11].

ACKNOWLEDGMENT

This work has been achieved in the European ITEA project uSERVICE and has been funded by the German Federal Ministry of Education and Research under contract number 01—S0902F.

REFERENCES