Scalable Computing: Practice and Experience
Volume 10, Number 3, pp. 265–275. http://www.scpe.org
ISSN 1895-1767

DEPLOYMENT OF EMBEDDED WEB SERVICES IN REAL-TIME SYSTEMS

GUIDO MORITZ, STEFFEN PRUETER, DIRK TIMMERMANN*AND FRANK GOLATOWSKI†

Abstract. Service-oriented architectures (SOA) become more and more important in networked embedded systems. The main advantages of Service-oriented architectures are a higher abstraction level and interoperability of devices. In this area, Web services have become an important standard for communication between devices. However, this upcoming technology is only available on devices with sufficient resources. Therefore, embedded devices are often excluded from the deployment of Web services due to a lack of computing power, insufficient memory space and limited bandwidth. Furthermore, embedded devices often require real-time capabilities for communication and process control. This paper presents a new table driven approach to handle real-time capable Web services communication, on embedded hardware through the Devices Profile for Web Services.

Key words: service-oriented device architecture, devices profile for Web services, Web service for devices

1. Introduction. High research efforts are made to develop cross domain communication middleware basing on architectural concepts like REST (Representational state transfer) and Service-oriented Device Architectures (SODA) [25] and on technologies like UPnP (Universal Plug and Play), JINI, and DPWS (Devices Profile for Web Services). While UPnP, DLNA and related technologies are established in networked home and small office environments, DPWS is widely used in the automation industry at device level [26] and it has been shown that they are also applicable for Enterprise integration [24, 27]. Barisic et al. [33] outline the potential of SOA to become a key factor in embedded software development. Embedded development process can be improved significantly if the SOA paradigm is used in each development stage. However, to make this happen it is necessary to establish the grounding for deeply embedded systems and real-time system

Besides the advantages of SODA, additional resources are required to host a necessary software stack. There are SODA toolkits available for resource-constrained devices like UPnP stacks [28] or DPWS toolkits [8, 9]. However, additional effort is necessary for deployment on deeply embedded devices and especially for embedded real-time systems. Deeply embedded devices are small microcontrollers with only a few kB of memory and RAM (e.g. MSP430, ARM7). These devices cannot be applied with comprehensive operating systems. But they are essential because as they combine price, low power properties, size and build-in hardware modules.

This work presents a new approach, which can be applied to deeply embedded devices and serve real-time and specification compliant DPWS communication.

2. Services in Device Controlling Systems. The World Wide Web Consortium (W3C) specifies the Web services standard [13]. UPnP is a popular specification in the home domain. Due to the lack of security mechanisms and the missing service proxy it is limited to small networks (see [2]). Furthermore, UPnP based communication scales not with the arising high number of future coming wireless smart cooperating objects due to its usage of Simple Service Discovery Protocol (SSDP) for device discovery at run-time. Web services are already widely used in large networks and the internet for business processes and server-to-server communication mainly. This client-to-server interaction uses SOAP [12] for the transport layer and Extensible Markup Language (XML) for the data representation [1, 15]. On the other hand, the Web services protocols need much computing power and memory, in order to enable a device-to-device communication with more constraint resources as servers. Therefore, a consortium lead by Microsoft has defined the Devices Profile for Web services (DPWS) [4]. DPWS uses several Web services protocols, while keeping aspect of resource constraint devices. In comparison to standard Web services, DPWS is able to discover devices at run time dynamically based on WS-Discovery, WS-MetadataExchange and WS-Transfer, without a global service registry (UDDI). The included WS-Eventing [6] specification also enables clients to subscribe for events on a device to get notified by state changes. Thus, pull messaging is avoided in favor of push messaging, which is a significant advantage for resource constraint devices and networks. DPWS is integrated in Microsofts operating systems Windows Vista and Windows 7 and furthermore in miscellaneous frameworks like e.g. .net Micro Framework. Additionally, open source stacks are available [8]. In August 2008 a technical committee (TC) at OASIS was formed for the "Web Services Discovery and Web Services Devices Profile" (WS-DD) [5]. WS-DD defines a lightweight subset of the Web Services protocol suite that makes it easy to find, share, and control devices on a network. The work of this

*University of Rostock, Rostock 18051, Germany ({guido.moritz, steffen.prueter, dirk.timmermann}@uni-rostock.de)
†Center for Life Science and Automation, Rostock 18119, Germany, (frank.golatowski@celisca.de)

TC is based on the former DPWS, WS-Discovery and SOAP-over-UDP specification. In July 2009 version 1.1 of named specifications were published by the OASIS WS-DD TC. For many companies, this is the reason for developing new interfaces for their products based on these protocols.

Using service gateways is the predominating approach to bridge between different communication layers or between embedded systems and enterprise systems. Our approach aims at lowering the efforts for integration and interaction and to set on standards instead of re-develop new protocols. Buckl et al. [32] assume a data centric processing model is used in embedded systems. Authors differentiate between embedded Services (called as eServices) and Web Services. Both worlds are integrated by a service gateway. In [34] Web Services on Universal Networks (WSUN) is described. The presented SOA based platform (environment) which is composed of broker, registry and universal adaptor and is able to bridge between different SOA technologies, like Jini and DPWS. This work concentrates on interoperability between different subnets and uses DPWS to integrate devices. Some work has been done to integrate DPWS into OSGi. While UPnP support has already been standardized within OSGi, today some initial work and proposals have been done to extend OSGi with DPWS [35, 36]. Fiehe et al. [36] describe a distributed architecture which uses DPWS to extend OSGi and make OSGi a distributed system. This approach is similar to actual work on distributed OSGi inside OSGi initiative. However, there still exists the lack to integrate DPWS capable device into OSGi.

Less work has been done to bring DPWS on deeply embedded systems and sensor networks and especially real-time systems. With the new approach, presented in this paper, Web services become also available on deeply embedded devices. Both, deeply embedded devices and devices that are more powerful will be enabled to communicate and interact with each other. This substitutes the application layer proxies.

Through linking the devices to a higher level of communication, devices no longer rely on specific transfer technologies like Ethernet. All devices in an infrastructure are connected via services. This services based architecture is already used in upper layers. Services based communication becomes available on lower layers nearest to the physical tier. This allows a higher abstraction level of process structures. The first step to allow this is the creation of a SODA framework that fulfills the requirements of deeply embedded devices.

3. Requirements for a light weight SODA. High-level communication on resource constrained embedded devices can result in an overall performance degradation. In a previous paper [7] Prueter et. al presented different challenges which have to be met in order to realize DPWS communication with real-time characteristics.

Firstly, as a basis an underlying real-time operating system must exist, ensuring the scheduling of the different tasks in the right order and in specific time slots. Secondly, the physical network has to provide real-time characteristics. The major challenge in DPWS with respect to the underlying network, is the binding of DPWS and SOAP. SOAP is bound to the Hypertext Transfer Protocol (HTTP) for transmission. HTTP is bound to the Transmission Control Protocol (TCP) [10] (see Figure 3.1). The TCP-standard includes non-deterministic parts concerning a resend algorithm in case of an error. Furthermore, the Medium Access Control (MAC) to the physical tier has to grant access to the data channel for predictable time slots. For example, Ethernet cannot fulfill this requirement.

As shown in Figure 3.1, it is possible to use SOAP-over-UDP. But in accordance to the DPWS specification, a device must support at least one HTTP interface [4].

In [7] Prueter et al. Xenomai [11] is used as operating system and RTNet [14] to grant network access with real-time characteristics. RTNet relies on the User Datagram Protocol (UDP) instead of TCP and uses Time Division Multiple Access (TDMA) for Medium Access Control (MAC). The usage of UDP demands SOAP-over-UDP at the same time. At least two interfaces have to be implemented: A non real-time, DPWS compliant HTTP/TCP interface and a real-time UDP interface. The disadvantage of using a special network stack including a special MAC, also implies building up a separate network. In this network, all participating notes have to conform to the MAC and used protocols.

For deeply embedded devices, various real-time operating systems exist. FreeRTOS [21] is a mini real-time kernel for miscellaneous hardware platforms like ARM7, ARM9, AVR, x86, MSP430 etc. Unfortunately, no useful real-time network stack and operating system combination is currently available for these kinds of deeply embedded devices. Therefore, this paper concentrates on the possibilities to provide real-time characteristics in the upper layers being on the top of TCP/IP.

The binding of DPWS and TCP through HTTP causes different challenges in granting real-time characteristic for DPWS communication and is still an ongoing work in our research group. It is not possible to reach deterministic characteristics without specific real-time operating systems and network stacks. A real-time

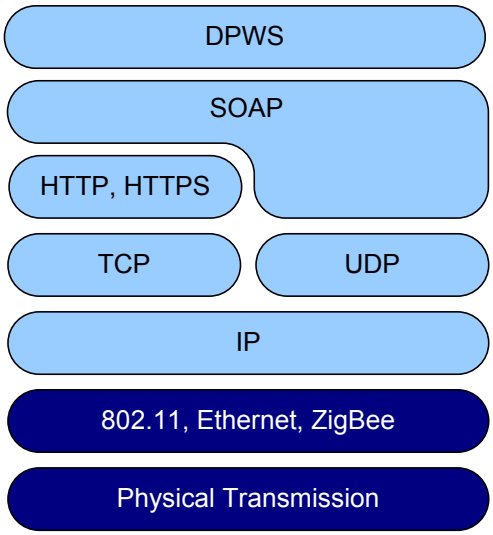

FIG. 3.1. *DPWS protocol stack*

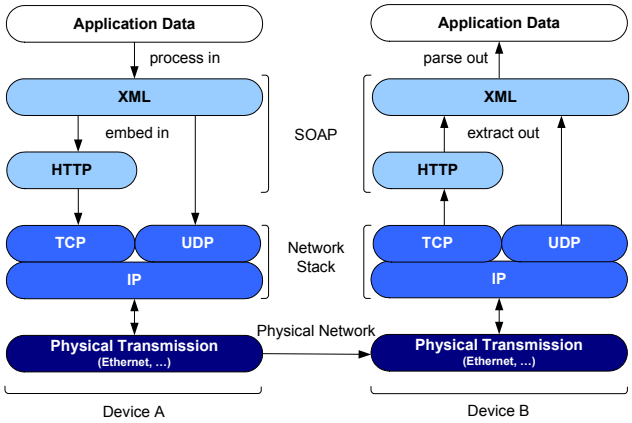

FIG. 3.2. *Modules to be implemented*

operating system grants access to peripheries for predictable time slots and execution of tasks in the right order. The arising high level communication may not interfere with the real-time process controlling. The underlying real-time operating system takes care of correct thread management and correct scheduling of the real-time and non real-time tasks. Tasks on the controller, competing with the communication, are prioritized by the operating system.

In order to provide Web services on microcontrollers, different challenges have to be met. Figure 3.2 shows the particular parts, which have to be realized.

3.1. Network Stack. The network stack, responsible for the right addressing and the way of exchanging data, is the first module, which have to be realized and meet the resource requirements. Dunkels has developed uIP and lwIP, two standard compliant TCP/IP stacks for 8 Bit controller architectures ([15, 16, 17]). uIP fulfills all minimum requirements for TCP/IP data transmissions. The major focuses are minimal code size, memory and computing power usage on the controller, without losing standard conformance. lwIP also fulfills non mandatory features of TCP/IP. Both implementations are designed to run on 8-bit architectures with and without an operating system. The differences between both stacks are shown in the following Table 3.1.

DPWS bases on WS-Discovery for automatic discovery of devices and is based on IP Multicast. Multicast applications use the connectionless and unreliable User Datagram Protocol (UDP) in order to achieve multicast communications. uIP is able to send UDP Multicast messages, but is not able to join multicast groups and receive multicast messages [17]. In contrast to uIP, the lwIP implementation supports all necessary UDP and Multicast features. The above mentioned FreeRTOS can use the lwIP stack for networking. This combines

TABLE 3.1
uIP vs. lwIP

Feature	uIP	lwIP
IP and TCP checksums	X	X
IP fragment reassembly		X
IP options		X
Multiple Interfaces		X
UDP		X
Multiple TCP connections	X	X
Variable TCP MSS	X	X
RTT estimation	X	X
TCP flow control	X	X
Sliding TCP window		X
TCP congestion control	Not needed	X
Out-of-sequence TCP data		X
TCP urgent data	X	X
Data buffered for rexmit		X

the advantages of a compatible, lightweight network stack and the usage of an embedded real-time operating system.

3.2. SOAP. Upon the network stack, HTTP communication protocol is used for transport of unicast messages. The payload is embedded in XML structures and sent via HTTP. All messages utilize the POST method of HTTP for SOAP envelope delivery. Most addressing information in the HTTP header is redundant because they are included in the SOAP message itself with a higher service abstraction. Significant HTTP header information is the Content-Length filed to identify ending of messages in the TCP data stream. Because DPWS requires a small part of the HTTP functionality only, it is not necessary to implement a full functional HTTP stack.

In contrast, the XML processing and parsing draws more attention. On deeply embedded devices, with only few kB of memory, the code size and the RAM usage have to be reduced. The WS-Discovery and WS-Metadata messages exceed the Maximum Transmission Unit (MTU) of most network technologies, including Ethernet. This supports the decision for lwIP in favour of uIP. The uIP implementation only uses one single global buffer for sending and receiving messages. The application first has to process the message in the buffer, before it can be overwritten [17]. In case of a complete XML message, the whole file has to be available before a useful parsing can be processed. Additional, computing power is restricted to resource constrained devices. With respect to the overall performance of the communication task, it is difficult to work through and parse the whole message as a nested XML file. Therefore, our research group has developed and implemented a new approach to handle HTTP and XML analysis. This new approach is described in the next section.

4. New Table Driven Approach. A complete implementation of SODA for deeply embedded systems, like wireless sensor network nodes with limited processing power and memory, is a significant challenge. All modules that are mentioned in section III like network stack, SOAP, HTTP and DPWS have to be implemented.

Due to dedicated characteristics and functionalities of sensors and actors in resource constraint environments, most of the exchanged information are discovery messages for the lose coupling of the devices during run-time and basic service invocations with non-complex data types. We have analyzed different setups with DPWS compliant implementations to identify which parts of DPWS could be omitted or adopted to reduce necessary resources. In most scenarios, only few types of messages have to be processed. After discovery and metadata exchange, the devices and their addresses are known and the services can be invoked. Only a few parts change within the exchanged messages. Major parts of the messages stay unchanged. Every time a service is called, almost the same message has to be parsed and almost the same message has to be build.

With all exchanged messages from the analysis of different scenarios, tables can be generated. The tables contain all appropriated incoming and outgoing messages. The new implemented table driven approach is able to response every request by referring to these tables.

This new table driven implementation is not based on SOAP and HTTP. Instead, we are using an approach basing on a simple string comparison of incoming messages in this new implementation. The SOAP-Envelope

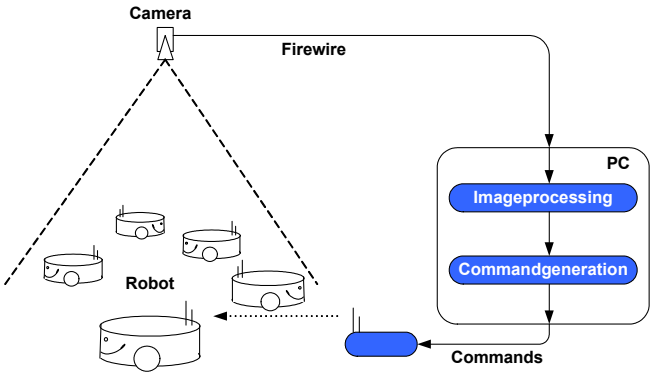

Fig. 5.1. *Mobile Robot Scenario*

containing the service invocations are simple structured and there is no need for complex parsing of messaging including e.g. heavy weight XML namespace support. The messages are interpreted as simple text messages and not as SOAP Envelopes being embedded in HTTP. The relevance of the received strings from HTTP and SOAP protocols are unknown for the table driven device. Certainly, the table driven device can analyze the incoming requests and filter required information. The device is able to send specific response with the correctly adapted dynamic changing sections. The overhead for parsing and building always the same message is reduced by this approach. Thereby memory usage and computation time are decreased in comparison to a traditional implementation.

With respect to a real-time capable communication, the treatment of the messages as strings and not as specific protocols is significant. The parsing as a string is independent of the depth of the nesting of XML structures and defined by the length of the SOAP-Envelope only. The necessary time, to parse the message as a string, is predictable. XML Schema, which is required by DPWS, cannot fulfill these requirements by default.

5. Mobile Robot Scenario. We verified our solution in a real world scenario. An external PC and an overhead camera control a team of five autonomous robots. The robots are coordinated via DPWS interfaces. The robots receive commands from a central server. The commands have to be executed in predicted timeslots to prevent collisions and enable accurate movement of the robots. The whole setup is shown in Figure 5.1.

The team behavior of the robots is controlled by a central server which uses one or more cameras mounted above the ground. Image processing software on the PC extracts the position of all robots in the field. On the PC even the commands for the robots are calculated. These commands consist of global coordinates of the robot positions and the target positions. These commands are sent with a high transmission rate to the robots. The robots use global coordinates to update their own local and imprecise coordinate tracking. The robots need this global updates in regular periods, otherwise a correct controlling cannot be granted. These real-time requirements for controlling the robots with a parallel running communication system make the robot scenario an ideal test bench for our implementations.

5.1. Robot Hardware. To control the robots we use two controller boards alternatively: an embedded Linux board and an ARM7 controller board. The embedded Linux board is the Cool Mote Master (CMM) from LiPPERT. It is equipped with an AMD Alchemy AU 1550 processor [19]. This board is designed as a gateway node for sensor networks. The CMM is already equipped with an 802.15.4 compliant transceiver. We have extended the board with additional Bluetooth and Wi-Fi (IEEE 802.11) interfaces [20]. Thereby, the board has three different radio technologies for networking beside Ethernet.

The ARM7 board is a SAM7-EX256 evaluation board from Olimex [23]. This board is applied with an Atmel ARM7 controller with 256 kB memory and 64 kB RAM. The board already provides an Ethernet interface, which was used for testing. The controller is running with a clock rate of 55MHz. It is possible to schedule the lwIP stack and the implemented table driven device in different prioritized tasks with the help of FreeRTOS.

The implementations are evaluated on a standard PC and on these boards. An overview of used hardware is provided in Table 5.1. The network devices are configured in a way, that all of them can handle IP traffic.

6. Implementation. Our research group has implemented the WS4D-gSOAP toolkit [8]. This is a software solution, which includes a DPWS stack and software tools for creating of own Web services based on

TABLE 5.1
Used hardware for testing the new table driven approach

	x86 PC	CMM	SAM-7
CPU	Intel Pentium 4	Alchemy AU 1550	Atmel ARM7
Clock Rate	3,4 GHz	500 MHz	55 MHz
ROM	500 GB	512 MB	256 kB
RAM	1024 MB	256 MB	64 kB
Operating System	Linux (2.6.24/Ubuntu)	Linux (2.6.17/Debian)	FreeRTOS 5.0
Network interfaces	Ethernet	Ethernet, 802.11g, 802.15.4, Bluetooth	Ethernet

DPWS. This toolkit uses gSOAP [22] for SOAP functionalities and extends gSOAP with an implementation of the DPWS specification. This traditional implementation will be used as benchmark for the new table driven approach.

In the first step a service is created with the existing WS4D toolkit that provides all necessary commands for the robots in our mobile robot scenario. The external PC calls a hosted service on the robots. The service is called every time when new commands have to be send to the robots. The new commands are embedded in the request. The service answers with a response, including a performance parameter of the robot.

In the second step, the exchanged messages are analyzed according to the DPWS specification. All possible outgoing and incoming messages for the mobile robot scenario are generated. In the third step, a completely new DPWS device is implemented. The structures and contents of the possible messages are deposited in the new implemented device as strings. This device does not support any dynamic SOAP or HTTP functionalities. The new table driven approach does not parse the whole incoming message as XML file. Every received message is analyzed with an elementary string compare. Thereby the type of the message is figured out. If the message type is known, the device answers with the related message. The answer is already deposited in the implemented device as a string also. In the answer, only parts required by the DPWS specification and the payload are changed. With respect to available resources of the target hardware platform, the implementation can be optimized concerning the Flash and RAM memory usage. On the one hand, the generated tables can be loaded in RAM at start time of the binary. This requires more available main memory, but can fasten the data access during run time. On the other hand, the generated tables can also be stored in non-volatile memory like Flash. This reduces footprint of the binary but may cause higher execution times during service invocation and message processing. To meet real-time requirements, the choice between both options correlates to real-time features of the underlying management of volatile and non-volatile memory access.

During the implementation of the table driven device, we have taken care that system functions are not called in critical sections. For example, the main memory management is provided by the task itself. The task allocates a pool of main memory when it is started and then organizes the main memory itself. Furthermore, the different threads for the network stack and the threads handling the messages are analyzed to be scheduled in the right order and with correct priorities.

6.1. Message Exchange. Figure 6.1 gives an overview of exchanged messages in the mobile robot scenario. When starting the device, it announces itself with a Hello SOAP Envelope. Within this message only the MessageID and the transport specific address, are dynamically and has to be adapted. Furthermore, the MessageNumber and the InstanceID has to be correct.

When a client was not started, as the device announces itself with a Hello, the client asks with a Probe for available devices. The answer is a Probe Match, where the RelatesTo has to fit to the MessageID of the Probe and the MessageID has to be dynamic. Here, also the MessageNumber and the InstanceID has to be incremented.

When the devices and their addresses are known, the client will ask for the hosted services on the device in the next step. Therefore, a GetMetadata Device is send to the hosting service, which is at least a hosted service that announces representative the available hosted services. The GetMetadata message is the first one that is sent via HTTP. Within the HTTP header, the Content-Length header field, the length of the message, and the IP address has to be adopted. The address only has to be changed, if it was not known at compile time. This applies to all IP addresses in the scenario. In the GetMetadata Device message SOAP-Envelope, the To XML tag has to match to the address of the device, detected through the Probe. The device answers with a GetMetadata Device Response message. In this message the RelatesTo has to match the MessageID of the GetMetadata Device.

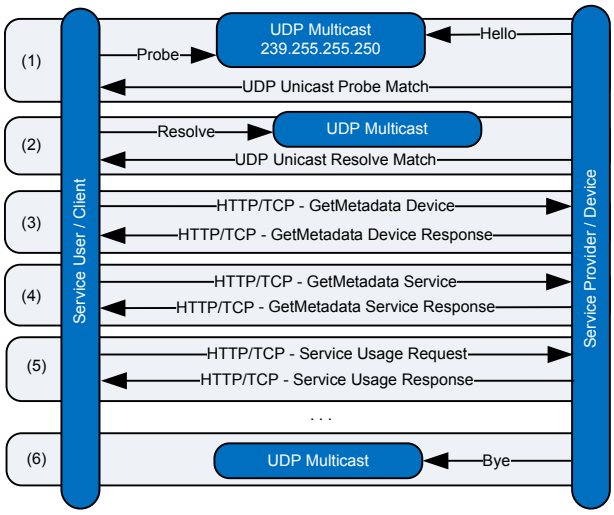

Fig. 6.1. *Message Exchange*

When the client knows available hosted services, the specific hosted service, that the client is looking for, is asked for the usage interface with a GetMetadata Service. The GetMetadata Service Response refers to the GetMetadata Service through the RelatesTo XML tag.

After the metadata exchange is complete, the client knows how to interact with the specific service and the service usage starts. The client invokes the service with a message, where To and MessageID has to be correct. In the Service Usage Request, the coordinates of the mobile robot scenario are integrated. The service answers with the Service Usage Response. Therein, the reference to the request is given through the RelatesTo tag. In our special mobile robot scenario, the response also contains the ProcessingTime tag. In this section, the service informs the service user about the time, the application needs to process the new coordinates and is a performance parameter for the mobile robot.

An overview about the dynamic parts of the different messages is given in Table 6.1. The overall size for the exchanged messages is 12.839 Bytes. The overall number of Bytes that can change is 588. Only 4.6% of the overall exchanged bytes are dynamic in the mobile robot scenario.

6.2. Devices Footprint. The memory optimized WS4D toolkit implementation of the DPWS device needs 360 kB of disk space when compiled for Linux on a x86 architecture. The table driven device implementation has a 16 kB footprint when compiled for a standard x86 PC running with Linux. Both versions do not contain networking stacks in these x86 implementations. Both implementations for an x86 PC running with Linux are using the BSD Socket API and corresponding network stacks included in Linux to handle the network traffic. The same implementation of the new table driven approach ported to the SAM7-EX256 board running with FreeRTOS 5.0 has a 13 kB footprint without network stack and interface drivers. As network stack the independent lwIP stack in Version 1.3 is applied to the board. Therefore, the stack was ported to FreeRTOS 5.0.

The required disk space for the different parts on the SAM7 board is shown in Table 6.2. The overall memory being used on the board, including FreeRTOS, lwIP and the device needs 146 kB.

The heap and stack usages of both implementations are given in Table 6.3. The maximum stack and heap usage of the table driven approach is much lower, because exchanged messages and their sizes are known at compile-time and no non-required memory has to be allocated during run-time. Due to the soft resource requirements of the used hardware platforms, the implementation of the table driven approach is still not full optimized concerning heap and sack usage. It depends on the specific scenario, if the message tables are kept into RAM during run-time or are loaded separately into RAM from non-volatile memory like flash on demand. Keeping the contents of the tables in flash reduces heap and stack usage, but may cause a gain in responds time due to higher access time to flash compared to RAM. For highly energy constrained devices with flash memory for data storage on external hardware modules, swapping the tables to flash can have an influence on the overall power consumption also. The flash hardware component can be switched of while keeping the tables without consuming any energy, but have to be switched on every time when access to the tables is required. Thus, depending on the specific scenario, a hybrid solution for table storage might be optimal. Often required tables

TABLE 6.1
Overview Exchanged Messages

Message Type	Changing parts	Dynamic Bytes
Hello	MessageID	36
	XAddrs (IP)	max. 17
	AppSequence MessageNumber	approx. 2
	AppSequence InstanceId	10
Probe	MessageID	36
Probe Match	MessageID	36
	RelatesTo	36
	AppSequence MessageNumber	approx. 2
	AppSequence InstanceId	10
GetMetada Device	HTTP Content-Length	max. 5
	HTTP Host	175, max. c.f. [10]
	MessageID	36
	To	36
GetMetadata Device Response	HTTP Content-Length	max. 5
	RelatesTo	36
	Address	175, max. c.f. [10]
GetMetadata Service	HTTP Content-Length	max. 5
	HTTP Host	175, max. c.f. [10]
	MessageID	36
	To	175, max. c.f. [10]
GetMetadata Service Response	HTTP Content-Length	max. 5
	RelatesTo	36
Service Invocation Request	HTTP Content-Length	max. 5
	HTTP Host	175, max. c.f. [10]
	MessageID	36
	To	36
	Payload	16
Service Invocation Response	HTTP Content-Length	max. 5
	RelatesTo	36
	Payload	3

TABLE 6.2
Footprint of SAM7 Implementation with FreeRTOS and lwIP

Module	Footprint
static DPWS device	13 kB
lwIP 1.3	77 kB
FreeRTOS including Debug Tasks	56 kB

and content are kept into RAM with low access times and no additional energy consumption, while infrequent used tables are kept into flash to reduce heap and stack usage while run-time.

6.3. Time Responds. Also some timing measurements have been done in order to have an objective comparison for the new static approach. Therefore, the round trip time was measured that is required from sending the message to receiving the response on the client side. Through this method the overall performance and the maximum number of service invocations per second can be determined which can be served.

These measurements are done for a standard x86 PC and the SAM7 board. On both devices a 100 MB/s Ethernet interface is applied, which has been used for the measurements. On the SAM7 boards, an independent thread simulated an additional CPU load. This CPU load thread was scheduled with different priorities. As requesting client a standard PC (2x3,5 GHz with 1 GB RAM) was used in all cases.

The following Table 6.3 shows the times measured for the different implementations of DPWS server/device. The values are the average over 1000 requests, send back-to-back.

TABLE 6.3
Round Trip Time and Memory Usage

Device	Round Trip Time	Requests per sec	Max Heap Usage	Max Stack usage
x86 PC WS4D toolkit	1,05 ms	952	112,76 Bytes	97,64 Bytes
x86 PC Table Driven DPWS	0,9 ms	1111	8,928 Bytes	15,324 Bytes
SAM7-EX256 Table Driven DPWS	18,6 ms	53	N.A.	N.A.
SAM7-EX256 Table Driven DPWS	18,6 ms	53	N.A.	N.A.
SAM7-EX256 Table Driven DPWS	30,2 ms	33	N.A.	N.A.

On the PC, the new implemented approach provides a faster overall processing. The time responds on the real-time operating system of the SAM7 board, depend on given priorities for the different competing tasks. As long as the CPU load task has a lower priority than the DPWS and the lwIP tasks, no effect to the average times could be measured.

7. Message structure optimizations. All messages in DPWS make use of XML for data representation. The application of XML in DPWS and Web Services has multiple advantages considering independency of programming language, operating system, communication channel, data representation, and character set. Certainly, XML implies a message overhead. Hence, this subsection describes several concepts for optimized data encodings of SOAP messages.

Fast Web Services [30] 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. Sandoz et al. developed a solution to convert Web Services specific XML Schema into ASN. The proposed approach reduces the size of the XML data by more than factor four and "performance is nearly 10 times that of XML literal. In other words, Fast Web Services will perform better as the size of the content increases." 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 [29] 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.

In comparison to Efficient XML and Fast Web Services, A-SOAP [31] describes concepts for XML encoding, which can be easily implemented in hardware and thereby much more energy efficient then in software. Additionally, this approach provides real-time parsing characteristics.

A-SOAP (Adaptive SOAP) uses hash functions, to encapsulate complex XML structures. Constantly recurrent XML structures are represented by hash values (see example in Figure 7.1). 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 which 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.

A-SOAP is a proper addition to the table driven approach. The contents of the generated tables can be attached to hash values for identification. Not the contents have to be transmitted but only the dedicated hash values. On the one hand, this reduces parsing efforts as hash values can be parsed in hardware and in software fast and easily. On the other hand, the integration of the A-SOAP approach in toolkits for the implementation of the table driven approach reduces footprints and memory consumption significantly. If possible communication partners are already known at compile-time and thus all clients invoking the table driven device comply the DPWS extended with A-SOAP functionalities, the tables might be omitted completely in the binaries. Only

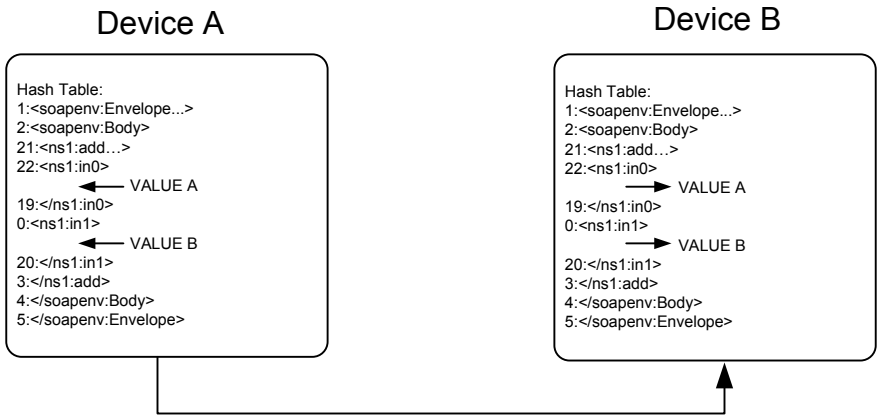

FIG. 7.1. *A-SOAP*

the associated hash values have to be included. This results in real-time DPWS based communication even for highly resource constrained platforms.

The disadvantage of A-SAOP is that it recently was granted as patent. Hence, there is no proposal for a data compression to apply DPWS in WSNW.

8. Conclusion. The new table driven approach allows the usage of Web services on deeply embedded devices. Furthermore, the implemented services can grant real-time capabilities. Thus, the deeply embedded devices can be integrated in enterprise service structures. The created service interfaces can be reused in different application. The connectivity between such large numbers of embedded devices normally needs proxy concepts with static structures. Now, these proxies are no longer required. The devices can be directly accessed by a high level process logic. Furthermore, the validation and certification become cheaper because of the slim implementation and reusability of the interfaces.

The measurements show that the binary size of a device can be reduced by the factor of more than 20. At the same time, the time responds can be improved. Heap and stack usages do not depend on specific dependent values but on specific message exchange patterns of dedicated scenarios Through the implementation in different threads, the time responds of the new implemented static approach is independent from other competing tasks. However, this assumes an underlying real-time operating system.

Further optimization of the footprint and dynamic memory usage are a main focuses for the future work. Future work will also research on a completely specification compliant implementation including optimized message structuring for real-time parsing.

REFERENCES

[1] W. DOSTAL, M. JECKLE, I. MELZER, AND B. ZENGLER, *Serviceorientierte Architekturen mit Web Services*, Elsevier, (2005).
[2] ELMAR ZEEB, ANDREAS BOBEK, HENDRIK BOHN, FRANK GOLATOWSKI, *Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services*, 2nd International IEEE Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2007), (2007), Niagara Falls, Ontario, Canada,pages 956–963.
[3] MARCO SGROI, ADAM WOLISZ, ALBERTO SANGIOVANNI-VINCENTELLI AND JAN M. RABAEY, *A Service-Based Universal Application Interface for Ad-hoc Wireless Sensor Networks (Draft)*, Unpublished article, (2003).
[4] MICROSOFT CORPORATION, *DPWS Specification*, Technical Report, http://specs.xmlsoap.org/ws/2006/02/devprof, (2006).
[5] OASIS, *Web Services Discovery and Web Services Devices Profile (WS-DD) TC*, Technical Report, http://www.oasisopen.org/committees/ws-dd/, (2009).
[6] WORLD WIDE WEB CONSORTIUM, *Web Services Eventing (WS-Eventing) Submission*, Technical report, http://www.w3.org/Submission/WS-Eventing/, (2006).
[7] STEFFEN PRUETER, GUIDO MORITZ, ELMAR ZEEB, RALF SALOMON, FRANK GOLATOWSKI, DIRK TIMMERMANN, *Applicability of Web Service Technologies to Reach Real Time Capabilities*, 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC), (2008), Orlando, Florida, USA, pages 229-233.
[8] UNIVERSITY OF ROSTOCK, *DPWS-Stack WS4D*, Technical report, http://ws4d.org, (2009).
[9] SCHNEIDER ELECTRIC, *SOA4D Forge*, Technical report, https://forge.soa4d.org/, (2009).
[10] INTERNET ENGINEERING TASK FORCE, *Hypertext Transfer Protocol—HTTP/1.1 [RFC 2616]*, Technical report, http://tools.ietf.org/html/rfc2616, (1999).
[11] PHILIPPE GERUM, *Xenomai - Implementing a RTOS emulation framework on GNU/Linux*, Whitepaper, (2004).

[12] World Wide Web Consortium, *Simple Object Access Protocol Specification*, Technical report, http://www.w3.org/TR/soap/, (2008).
[13] World Wide Web Consortium, *Web Service Architecture Specification*, Technical report, http://www.w3.org/TR/ws-arch/, (2007).
[14] Kiszka, J. and Wagner, B. and Zhang, Y. and Broenink, J.F., *RTnet - A flexible Hard Real-Time Networking Framework*, 10th IEEE International Conference on Emerging Technologies and Factory Automation Volume 1, (2005) Catania, Italy, page 8.
[15] Adam Dunkels, *Full TCP/IP for 8-Bit Architectures*, International Conference On Mobile Systems, Applications And Services, (2003), San Francisco, California, pages 85-98.
[16] Adam Dunkels, *uIP*, Technical report, http://www.sics.se/~adam/uip/, (2007).
[17] Adam Dunkels, *lwIP—A Lightweight TCP/IP stack*, Technical report, http://savannah.nongnu.org/projects/lwip/, (2008).
[18] Adam Dunkels, *The uIP Embedded TCP/IP Stack*, The uIP 1.0 Reference Manual, Technical report, (2006).
[19] LiPPERT: PC solutions for rugged industrial applications, *Cool Mote Master Board*, Technical report, http://www.lippert-at.com/, (2008).
[20] Thorsten Schulz, *Evaluierung verschiedener Prozessorloesungen fuer RoboCup Roboter*, Technical report, (2006).
[21] FreeRTOS— The Standard Solution For Small Embedded Systems, http://www.freertos.org, (2008).
[22] Robert A. van Engelen, Kyle A. Gallivany, *The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks*, 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID), (2002) Washington, DC, USA, page 128.
[23] Olimex, SAM7-EX256 Evaluation Board, Technical report, http://www.olimex.com/dev, (2008).
[24] S. Karnouskos, O. Baecker, L. Moreira, S. de Souza, P. Spiess, *Integration of SOA-ready networked embedded devices in enterprise systems via a cross-layered web service infrastructure*, Emerging Technologies and Factory Automation (ETFA), (2007) Patras, Greece, pages 293-300.
[25] Scott de Deugd, Randy Carroll, Kevin E. Kelly, Bill Millett, and Jeffrey Ricker, *SODA: Service-Oriented Device Architecture*, IEEE Pervasive Computing, (2006), vol. 5, no. 3, pages 94–C3.
[26] H. Bohn, A. Bobek, and F. Golatowski, *SIRENA - Service Infrastructure for Realtime Embedded Networked Devices: A service oriented framework for different domains*, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06), (2006), Washington, DC, USA, page 43.
[27] Elmar Zeeb, Steffen Prueter, Frank Golatowski, Frank Berger, *A context aware service-oriented maintenance system for the B2B sector*, 3rd International IEEE Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2008), (2008) Ginowan, Okinawa, Japan, pages 1381–1386.
[28] Intel, *UPnP - Technology Overview*, Technical report, http://www.intel.com/cd/ids/developer/asmo-na/eng/downloads/upnp/overview/, (2008).
[29] W3C, *Efficient XML Interchange Working Group*, Technical report, http://www.w3.org/XML/EXI/, (2009).
[30] Sandoz, P., Pericas-Geertsen, S., Kawaguchi, K., Hadley, M., Pelegri-Llopart, E., *Fast Web Services*, Article, (2003).
[31] Rosu, M.-C., *A-SOAP: Adaptive SOAP Message Processing and Compression*, IEEE International Conference on Web Services (ICWS2007), (2007) Salt Lake City, Utah, USA, pp.200-207.
[32] Christian Buckl, Stephan Sommer, Andreas Scholz, Alois Knoll, Alfons Kemper, Joerg Heuer, Anton Schmitt, *Services to the Field: An Approach for Resource Constrained Sensor/Actor Networks*, International Conference on Advanced Information Networking and Applications Workshops (AINAW2009), (2009) pp. 476–481.
[33] Daniel Barisic, Martin Krogmann, Guido Stromberg, Peter Schramm, *Making Embedded Software Development More Efficient with SOA*, 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW2007), (2007) vol. 1, pp. 941–946.
[34] Hyung-Jun Yim, Il-Jin Oh, Yun-Young Hwang, Kyu-Chul Lee, Kangchan Lee, and Seungyun Lee, *Design of DPWS Adaptor for Interoperability between Web Services and DPWS in Web Services on Universal Networks*, International Conference on Convergence Information Technology (ICCIT 2007), (2007) pp. 1032–1039.
[35] André Bottaro, Anne Gérodolle, Sylvain Marié, Stéphane Seyvoz, Eric Simon, *RFP 86 DPWS Discovery Base Driver*, OSGi Alliance, (2007).
[36] A. Bottaro and A. Gérodolle, *Home SOA: facing protocol heterogeneity in pervasive applications*, 5th international Conference on Pervasive Services (ICPS2008), (2008) ACM, New York, NY, pp. 73–80.
[37] Christoph Fiehe, Anna Litvina, Ingo Lueck, Oliver Dohndorf, Jens Kattwinkel, Franz-Josef Stewing, Jan Krueger, Heiko Krumm, *Location-Transparent Integration of Distributed OSGi Frameworks and Web Services*, International Conference on Advanced Information Networking and Applications Workshops, (2009) pp. 464–469.

Edited by: Janusz Zalewski
Received: September 30, 2009
Accepted: October 19, 2009