Proceedings of the International Multiconference on Computer Science and Information Technology, pp. 695 – 701

ISBN 978-83-60810-14-9
ISSN 1896-7094

Real-Time Service-oriented Communication Protocols on Resource Constrained Devices

Guido Moritz, Steffen Prüter, Dirk Timmermann
University of Rostock, Rostock 18051, Germany {guido.moritz, steffen.prueter, dirk.timmermann}@uni-rostock.de

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

Abstract - **Service-oriented architectures (SOA) become more and more important in connecting devices with each other. 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 and memory. 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.**

I. Introduction

THE usage of a standardized device-centric SOA is a possible way to fulfill interoperability requirements in future networked embedded systems. Technologies like UPnP (Universal Plug and Play), DPWS (Devices Profile for Web Services), REST (Representational state transfer) and Web services are used to realize a so called SODA (Service-oriented device architectures) [23]. 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 [24] and it has been shown that they are also applicable for Enterprise integration [22, 26].

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 or DPWS toolkits [WS4D, SOA4D]. However, additional effort is necessary for deeply embedded devices and for embedded real-time systems especially. Deeply embedded devices are small microcontrollers with only a few kB of memory and RAM (e.g. MSP430, ARM7). These devices cannot be applied to huge operating systems. But they are essential because as they combine price, low power properties, size and build-in hardware modules.

In this Paper a new approach is presented, which can be applied to deeply embedded devices and serve real-time and specification compliant DPWS requests.

II. Web services in Device Controlling Systems

The World Wide Web Consortium (W3C) specifies the Web services standard [11]. 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]). Web services are more important when using larger networks. This client-to-server interaction uses SOAP [10] for the transport layer and Extensible Markup Language (XML) for the data representation [1, 13]. On the other hand, the Web services protocols need much computing power and memory, in order to enable a device-to-device communication. Therefore, a consortium lead by Microsoft has defined DPWS [4]. DPWS uses several Web services protocols, while keeping aspect of resource constrained devices. In comparison to standard Web services, DPWS is able to discover devices at run time dynamically, without a global service registry (UDDI). The included WS-Eventing [5] specification also enables clients to subscribe for events on a device. The device sends a notification to the client, whenever an event occurs. State changes not have to be polled by the client. DPWS is integrated in the Microsoft operating system Vista. For many companies, this is the reason for developing new interfaces for their products based on these protocols.

In specific scenarios, communication proxies are necessary because of the low memory and computing power of deeply embedded devices. With the new implemented approach, 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 communication 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 ensemble 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.

III. 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 [6] we have presented different challenges which have to be met in order to realize DPWS communication with real-time characteristics.

This work was partly supported by the ITEA LOMS project.

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) [8] (see Figure 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 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].

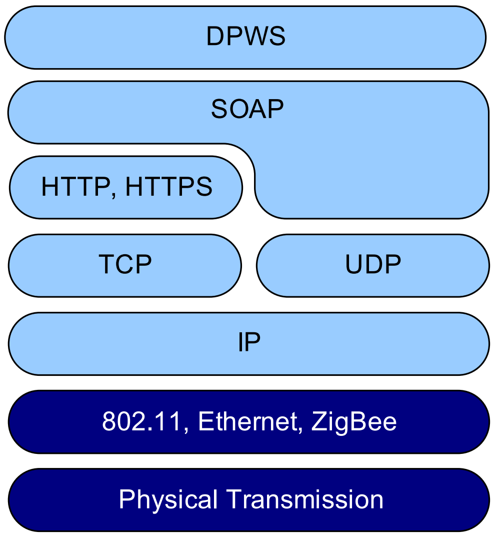

Figure 1. DPWS protocol stack

In Prüter et al. Xenomai [9] is used as operating system and RTNet [12] is used 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[19] 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 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 2 shows the single parts, which have to be realized.

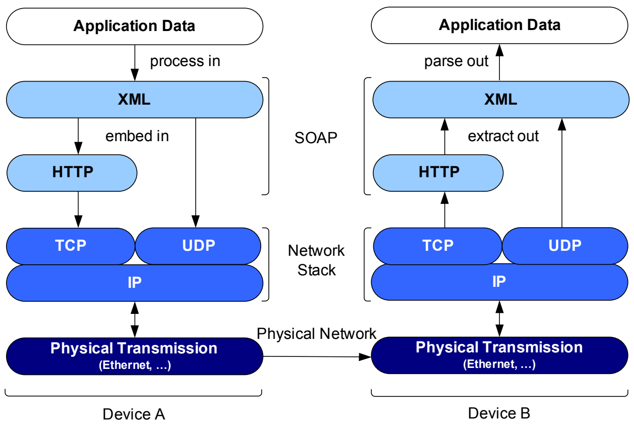

Figure 2. Modules to be implemented

A. 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. Dunkels has developed uIP and lwIP, two standard compliant TCP/IP stacks for 8 Bit controller architectures ([13, 14, 15]). 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.

DPWS utilizes 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 [15]. 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 the advantages of a compatible, lightweight network stack and the usage of an embedded real-time operating system.

Table 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
TCP options	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

B. SOAP

Upon the network stack, HTTP communication protocol is used. The payload is embedded in XML structures and sent via HTTP.

Because DPWS requires a small part of the HTTP functionality only, it is not necessary to implement a full functional HTTP stack. All DPWS messages are using the POST method of HTTP for delivering.

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 [15]. 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.

IV. 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.

We have analyzed different setups with DPWS compliant implementations to identify which parts of DPWS could be omitted 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 are generate. The tables contain all appropriated incoming and outgoing messages. The new implemented table driven approach is able to answer every request with the right answer by referring to these tables. In section the dynamic changing parts of the different messages are shown.

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 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. 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 useful. The parsing as a string is independent of the depth of the nesting of XML structures. The necessary time, to parse the message as a string, is predictable. XML Schema, which is required by DPWS, cannot fulfill these requirements.

V. Mobile Robots as Test Bench

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 3.

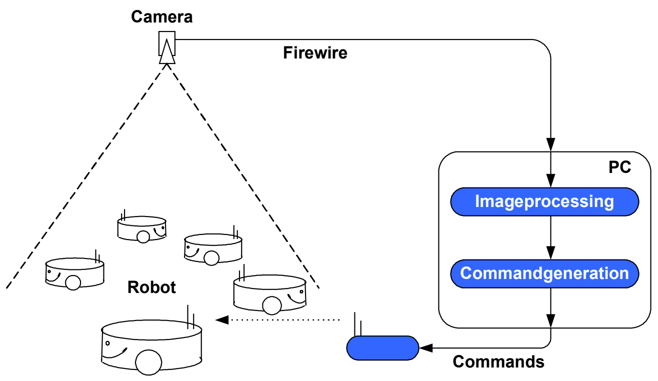

Figure 3. Robot Scenario

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.

A. 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 [17]. 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 [18]. Thereby, the board has three different radio technologies for networking beside Ethernet.

The ARM7 board is a SAM7-EX256 evaluation board from Olimex . 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 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 2 . The network devices are configured in a way, that all of them can handle IP traffic.

Table 2.
Used hardware for testing the new table driven approach

	PC	CMM	SAM-7
CPU	Intel Pentium4	AlchemyAU 1550	Atmel ARM7
Clock	3,4GHz	500MHz	55MHz
ROM	500 GB	512MB	256 kB
RAM	1024 MB	256MB	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

VI. Implementation

Our research group has implemented the WS4D-gSOAP toolkit[7]. This is a software solution, which includes a DPWS stack and software tools for creating of own Web services based on DPWS. This toolkit uses gSOAP [20] 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 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 right 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.

During the implementation of the table driven device, we have taken care that system functions are not called in critical sections. For example, the memory management is provided by the task itself. The task allocates a pool of memory when it is started and then organizes the 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.

A.Message Exchange

Figure 4 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 wsa:MessageID and the wsd:XAddrs, the IP 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 wsa:RelatesTo has to fit to the wsa:MessageID of the *Probe* and the wsa: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 service that announces the available hosted services. The *GetMetadata* message is the first one that is sent via HTTP. Within the HTTP header, the content length, 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, the wsa:To entry 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 wsa:RelatesTo has to match the wsa:MessageID of *the GetMetadata Device*.

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 wsa:RelatesTo section.

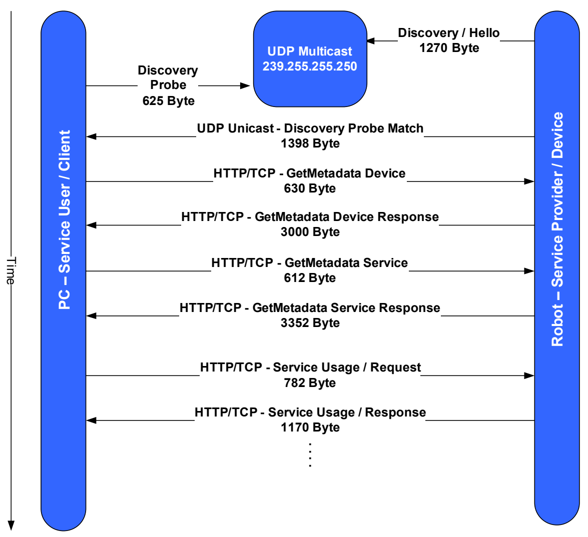
PC – Service User / Client
Robot – Service Provider / Device
Time
UDP Multicast
239.255.255.250
Discovery / Hello
1270 Byte
Discovery
Probe
625 Byte
UDP Unicast - Discovery Probe Match
1398 Byte
HTTP/TCP - GetMetadata Device
630 Byte
HTTP/TCP - GetMetadata Device Response
3000 Byte
HTTP/TCP - GetMetadata Service
612 Byte
HTTP/TCP - GetMetadata Service Response
3352 Byte
HTTP/TCP - Service Usage / Request
782 Byte
HTTP/TCP - Service Usage / Response
1170 Byte

Figure 4. Message Exchange[1]

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 wsa:To and wsa: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 wsa:RelatesTo section. In our special mobile robot scenario, the response also contains the mrs:ProcessingTime section. 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 3 .

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.

VII. Measurements/Comparison

In this section, the performances of the original WS4D toolkit and the new table driven approach are compared.

[1] number of Bytes may vary because of different IP addresses and payload e.g.

Table 3. Overview Exchanged Messages

MessageType	**Changingparts**	**Dynamic Bytes**
Hello	wsa:MessageID	36
	wsd:XAddrs(IP)	max.17[2,3]
	wsd:AppSequenceMessageNumber	approx.2
	wsd:AppSequence InstanceId	10
Probe	wsa:MessageID	36
ProbeMatch	wsa:MessageID	36
	wsa:RelatesTo	36
	wsd:AppSequenceMessageNumber	approx.2
	wsd:AppSequence InstanceId	10
GetMetada Device	HTTP content-length	max.5
	HTTP host	max.17[2,3]
	wsa:MessageID	36
	wsa:To	36
GetMetadataDeviceResponse	HTTP content-length	max.5
	wsa:RelatesTo	36
	wsa:Address	max.17[2,3]
GetMetadata Service	HTTP content-length	max.5
	HTTP host	max.17[2,3]
	wsa:MessageID	36
	wsa:To	max.17[2,3]
GetMetadata Service Response	HTTP content-length	max.5
	wsa:RelatesTo	36
Service Usage Request	HTTP content-length	max.5
	HTTP host	max.17[2,3]
	wsa:MessageID	36
	wsa:To	max.17[2,3]
	mrs:Position[4]	16
Service Usage Response	HTTP content-length	max.5
	wsa:RelatesTo	36
	mrs:ProcessingTime[4]	3

A.Devices Sizes

The WS4D toolkit implementation of the DPWS device needs 794 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. The table driven device does not contain the independent lwIP stack in this x86 implementation. Both implementations for a x86 PC running with Linux are using the BSD Socket API 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 interface. 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.

[2] depends on IP Address and Port number
[3] if not known at compile time
[42] depends on IP Address and Port number
[3] if not known at compile time
Payload

The required disk space for the different parts on the SAM7 board is shown in Table 4. The overall memory being used on the board, including FreeRTOS, lwIP and the device needs 146 kB.

Table 4.
Footprint of SAM7 Implementation with FreerRTOS and lwIP

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

B. Time Responds

Also some timing measurements have been done in order to have an objective comparison for the new static approach. Therefore, the overall 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 requests per second can be determined which can be served.

These measurements are done for a standard 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 5 shows the times measured for the different implementations of DPWS server/device. The values are the average over 1000 requests, send directly successive.

Table 5.
Respond time and processed requests per second

Device System	Response time	Requests per sec
Standard PC WS4D toolkit	1,05 ms	952
Standard PC Table Driven DPWS	0,9 ms	1111
SAM7-EX256 Table Driven DPWS	18,6 ms	53
SAM7-EX256 Table Driven DPWS CPU load thread lower priority then lwIP and DPWS tasks	18,6 ms	53
SAM7-EX256 Table Driven DPWS CPU load thread same priority like lwIP and DPWS tasks	30,2 ms	33

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.

VIII. 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 size of a device can be reduced by the factor of about 50. At the same time, the time responds can be improved. 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.

The 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. Therefore, the specification has to be analyzed in detail and all possible messages, even for error cases, have to be discovered and integrated into the static device.

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," in *2nd International IEEE Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2007)*, pages 956-963, Niagara Falls, Ontario, Canada, May 21-23, 2007.

[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*, November 2003.

[4] Microsoft Corporation, *DPWS Specification*, Technical Report, Microsoft Corporation, http://specs.xmlsoap.org/ws/2006/02/devprof, 2006.

[5] World Wide Web Consortium, *Web Services Eventing (WS-Eventing) Submission*, Technical report, http://www.w3.org/Submission/WS-Eventing/, March 2006.

[6] Steffen Prüter, 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),* Orlando, Florida, USA, pages 229-233, May 2008.

[7] University of Rostock. *DPWS-Stack WS4D*, Technical Report, University of Rostock, http://ws4d.org, 2007.

[8] Internet Engineering Task Force, *Hypertext Transfer Protocol--HTTP/1.1*, Technical Report, http://tools.ietf.org/html/rfc2616, 1999.

[9] Philippe Gerum, “Xenomai - Implementing a RTOS emulation framework on GNU/Linux,” Whitepaper, 2004.

[10] World Wide Web Consortium, *Simple Object Access Protocol Specification,* Technical report, World Wide Web Consortium, http://www.w3.org/TR/soap/, 2008.

[11] World Wide Web Consortium, *Web Service Architecture Specification*, Technical report, World Wide Web Consortium, http://www.w3.org/TR/ws-arch/, 2007.

[12] 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*, Catania, Italy, page 8, September 2005.

[13] Adam Dunkels, “Full TCP/IP for 8-Bit Architectures ,” *International Conference On Mobile Systems, Applications And Services* , San Francisco, California, pages 85-98, 2003.

[14] Adam Dunkels, *uIP* , Technical report, http://www.sics.se/~adam/uip/index.php, 2007.

[15] Adam Dunkels, *lwIP - A Lightweight TCP/IP stack* , Technical Report, http://savannah.nongnu.org/projects/lwip/, 2008.

[16] Adam Dunkels, *The uIP Embedded TCP/IP Stack*, The uIP 1.0 Reference Manual, Technical report, 2006.

[17] LiPPERT: PC solutions for rugged industrial applications, *Cool Mote Master Board*, Technical report, http://www.lippert-at.com/, 2008.

[18] Thorsten Schulz, *Evaluierung verschiedener Prozessorlösungen für RoboCup Roboter* , Technical report, 2006 .

[19] *FreeRTOS - The Standard Solution For Small Embedded Systems*, http://www.freertos.org, 2008.

[20] 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)* , Washington, DC, USA, page 128, May 2002.

[21] Olimex, *SAM7-EX256 Evaluation Board*, Technical report, http://www.olimex.com/dev, 2008.

[22] 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 & Factory Automation (ETFA)*, Patras, Greece, pages 293-300, Sept. 2007.

[23] Scott de Deugd, Randy Carroll, Kevin E. Kelly, Bill Millett, and Jeffrey Ricker, "SODA: Service-Oriented Device Architecture," *IEEE Pervasive Computing* , vol. 5, no. 3, pages 94-C3, 2006.

[24] 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)* , page 43, Washington, DC, USA, 2006.

[25] Elmar Zeeb, Steffen Prueter, Frank Golatow ski, 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)*, Ginowan, Okinawa, Japan, pages 1381-1386, March 26, 2008.

[26] Intel, *UPnP - Technology Overview* , Technical report, http://www.intel.com/cd/ids/developer/asmona/eng/downloads/upnp/overview/index.htm , 2008.