Web services on Deeply Embedded Devices with Real-Time Processing

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 become more and more important in connecting devices with each other. The main advantages of Service-oriented architectures are higher abstraction level and interoperability of devices. In this field Web services become the most important standard for communication between devices. But this upcoming technology is only available on powerful devices. Embedded hardware is often excluded from the deployment of Web services because of the lack of resources like computing power and memory. In this area also real-time capabilities for process control are required. This paper presents a new approach to handle Web services communication on deeply embedded hardware with the Devices Profile for Web Services Specification.

1. Introduction

Device centric Service-oriented Architectures (SOA) with standardized interface are one method for solving the problem of interoperability in networking systems. The Service-oriented Device Architecture (SODA) [16] is realized through technologies like OSGi, UPnP, DPWS, REST and Web services. Thereby, the Devices Profile for Web services is widely used in automation industry on device level to connect devices with generic interfaces [2].

This client-to-server interaction uses SOAP for transport and Extensible Markup Language (XML) for data representation [10, 1]. The Web services protocols often need much computing power and memory to enable a useful device-to-device communication. Therefore Microsoft defined the Devices Profile for Web Services (DPWS) [7]. DPWS describes a specific profile of Web services, which keeps aspect of the limitations of resource constrained devices. In comparison to standard Web services, DPWS is able to discover devices at run time dynamically through WS-Discovery and includes WS-Eventing as mandatory part.

The usage of DPWS provides a device-centric SOA on the level of devices and embedded systems. This has several advantages, such as providing plug-and-play capability for network devices, fault-tolerant services, standardized interfaces and a higher abstraction of the overall process.

The existing toolkits for providing a SODA software stack for embedded systems are not small enough to be applied deeply embedded devices. Deeply embedded devices are small microcontrollers with only few KB of memory and RAM. On these devices no underlying huge operating system can be provided. These microcontrollers are necessary because of low costs, low power and small size. Additionally real-time constraints have to be met, to control sensors and actors safely.

To enable the creation of such device implementations, this paper represents ongoing work, which aims at adding Web services functionality to low cost microcontrollers, without losing real-time capabilities.

2. Challenges to be met

When enabling Web services on deeply embedded devices, all single parts shown in Figure 1 have to be implemented.

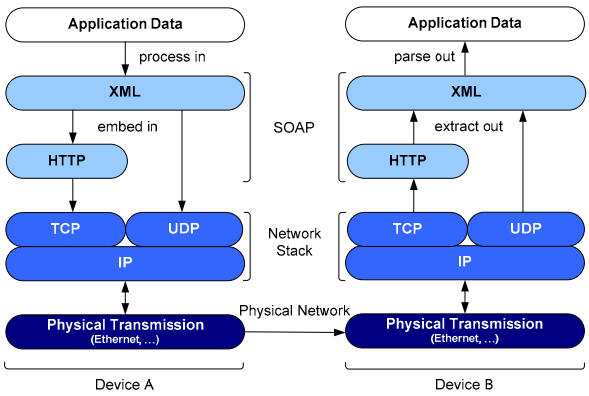

Figure 1. Modules to implement

1-4244-1506-3/08/$25.00 ©2008 IEEE

Authorized licensed use limited to: Universitaetsbibl Rostock. Downloaded on October 17, 2008 at 05:01 from IEEE Xplore. Restrictions apply.

In order to provide real-time characteristics, an underlying real-time operating system must exist. This real-time operating system is responsible for the predictable scheduling of the different tasks for communication and for controlling the hardware and for the interprocess communication. For deeply embedded devices the mini real-time kernel FreeRTOS [14] can fulfil these requirements. FreeRTOS is available for many platforms like e.g. ARM7, ARM9 and MSP430. The real-time operating system hosts all necessary tasks for communication and network stack.

2.1. Network Stack

The network stack is responsible for the right addressing and the way of exchanging data. Dunkels developed uIP and lwIP. These are two full standards compliant TCP/IP stacks for 8-bit Architectures [11, 12, 13]. The uIP implementation fulfils all minimum requirements for TCP/IP data transmission. The major focuses of uIP are minimum code size, and memory and computing power usage on the controller. In contrast the lwIP implementation also fulfils non mandatory features of TCP/IP. Both implementations are designed to run on 8-bit architectures with and without operating system. The differences between both stacks are shown in the following table.

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

Table 1. uIP vs. lwIP

The DPWS specification, which is using WS-Discovery for automatic finding of devices, requires the functionality of IP Multicast. Multicast applications use UDP to achieve Multicast Communication. The uIP implementation is able to send UDP Multicast messages, but is not able to join multicast groups and receive multicast messages [13]. In opposite to uIP, the lwIP implementation supports all necessary UDP and Multicast features. When realizing the required network stack, only lwIP can be applied.

FreeRTOS is able to use the lwIP stack for networking. Therefore, our research group has ported lwIP 1.3 to FreeRTOS 5.0. The existing ports of older lwIP versions to older FreeRTOS versions did not provide all necessary features for UDP multicast. The usage of FreeRTOS and lwIP combines the advantages of a compatible and lightweight network stack and the usage of an embedded real-time operating system.

2.2. SOAP

Upon the network stack, the HTTP communication protocol is used. The application data is embedded in XML structures and sent through HTTP, which then have to be converted.

It is not necessary to implement a full functional HTTP stack, to enable DPWS functionality. Only few HTTP features are used. 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 kilobytes of memory, the code size and the RAM usage have to be reduced on a minimum. With respect to the overall performance of the communication task, it is difficult to work through and parse the whole XML message.

3. New Table Driven Approach

Our research group has implemented the WS4D toolkit [9]. This toolkit includes software tools for the creation of own Web services. Engelens gSOAP [15] is used to implement SOAP functionalities. Furthermore gSOAP is extended to implement the DPWS specification. For deeply embedded devices, an implementation in this traditional way would need too much memory and computing power.

We analyzed different setups and scenarios and generated tables of all exchanged messages. 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. Within the exchanged messages, only few parts change. Major parts of the messages stay unchanged. An overview about the changing parts is given section 4. Every time a service is called, almost the same message has to be parsed and almost the same message has to be build. To save resources on the deeply embedded devices, a new table driven approach was developed. The overhead for parsing and building the same message is reduced by this approach. Thereby memory usage and computation time are decreased, compared to the traditional implementation.

This new table driven implementation is not based on SOAP and HTTP. For the table driven device the relevance of the strings as HTTP and SOAP protocol is unknown. The messages are analysed as simple ASCII strings. All possible incoming messages and all

Authorized licensed use limited to: Universitaetsbibl Rostock. Downloaded on October 17, 2008 at 05:01 from IEEE Xplore. Restrictions apply.

answers for the specific requests are known. The device is able to parse the incoming messages with a simple string compare and send the specific response with the correct adapted dynamic changing sections.

The parsing as a string is independent of the depth of the nesting of the XML structures. The necessary time to parse the message as a string is predictable, because all possible exchanged messages and their sizes are known. Thereby real-time message processing can be performed.

4. Implementation

We build up a simple test scenario. In this scenario a mobile robot has to be controlled by an external server. To control the robot, three single integer numbers have to be transmitted to the robot and one single integer value is returned. With the help of the DPWS specification all possible outgoing and incoming messages for this scenario are generated. Figure 2 gives an overview about the exchanged messages of the scenario.

When starting the device, it announces itself with a *Hello SOAP Envelope*. When a client was not started, as the device announced itself with a *Hello*, the client asks with a *Probe* for available devices. The answer is a *Probe Match.*

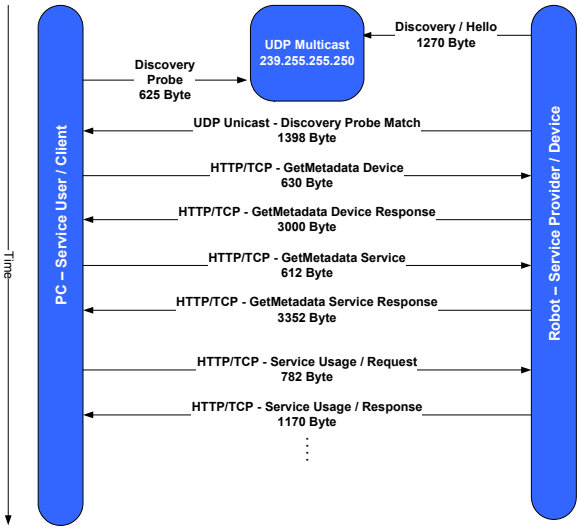

Figure 2. Message Exchange[1]

When the devices and its addresses are known, the client will ask in the next step for the hosted services on the device. Therefore a *GetMetadata Device* is send to the hosting service, which is at least a service that announces the available hosted services. When the client knows the available hosted services, the specific hosted service, that the client is looking for, is asked for the usage interface with a *GetMetadata Service*. After the metadata exchange is complete, the client knows how to interact with the specific service and the service usage starts. An overview about the dynamic parts of the different messages is given in Table 2.

The overall size for the exchanged messages is 12.839 Bytes (see Figure 2). The overall number of Bytes that can change is 588 (see Table 2). Only 4.6% of the overall exchanged bytes are dynamic in this test scenario.

Message Type	Changing parts	Dynamic Bytes
Hello	wsa:MessageID	36
	wsd:XAddrs (IP)	max. 17[2,3]
	wsd:AppSequence MessageNumber	approx. 2
	wsd:AppSequence InstanceId	10
Probe	wsa:MessageID	36
Probe Match	wsa:MessageID	36
	wsa:RelatesTo	36
	wsd:AppSequence MessageNumber	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
GetMetadata Device Response	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

Table 2. Overview Exchanged Messages

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

[2] depends on IP Address and Port number

[3] if not known at compile time

[4] Payload

Authorized licensed use limited to: Universitaetsbibl Rostock. Downloaded on October 17, 2008 at 05:01 from IEEE Xplore. Restrictions apply.

The new table driven approach for the DPWS device we implemented has a footprint of 16 kb. The table driven device does not contain the independent lwIP stack in this implementation. In [11] the size of an AVR compatible compiled stack is given with 21.756 Byte. That increases the overall table driven device size to 45 kB.

5. Future Work and Conclusion

The new table driven approach is still an ongoing research project. To have an objective comparison for the new table driven approach, some timing measurements and more implementation work has to be done. The overall time that is required from sending the message to receiving the response on the clients side has to be measured. Through this method the overall performance and the maximum number of requests per second which can be served can be determined. But even the number of operations that are needed to handle the messages on the devices side should be taken into account.

The shown implementation of the new approach allows the usage of Web services on deeply embedded devices. 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 directly be accessed by higher level process logic.

Future work will research on a complete specification conform implementation. Therefore, the specification has to be analyzed detailed and all possible messages, even for error cases, have to be discovered and integrated in the table driven device.

References

[1] W. Dostal, M. Jeckle, I. Melzer, and B. Zengler, *Serviceorientierte Architekturen mit Web Services*., Elsevier, 2005.

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

[3] 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)*, Ginowan, Okinawa, Japan, March 2008, pages 1381-1386.

[4] 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, May 2008, pages 229-233.

[5] 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)*, Niagara Falls, Ontario, Canada, May 2007, pages 956-963.

[6] Marco Sgroi, Adam Wolisz, Alberto Sangiovanni-Vincentelli, Jan Rabaey, "A Service-Based Universal Application Interface for Ad-hoc Wireless Sensor Networks (Draft)", *Unpublished article*, November 2003.

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

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

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

[10] World Wide Web Consortium, *Simple Object Access Protocol Specification*, Technical Report, http://www.w3.org/TR/soap/, April 2007.

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

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

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

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

[15] 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, May 2002, page 128.

[16] 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, July-September 2006, pp. 94-C3.

Authorized licensed use limited to: Universitaetsbibl Rostock. Downloaded on October 17, 2008 at 05:01 from IEEE Xplore. Restrictions apply.