Applicability of Web Service Technologies to Reach Real Time Capabilities

Steffen Prüter, Guido Moritz, Elmar Zeeb,
Ralf Salomon, Dirk Timmermann
University of Rostock,
Rostock 18051, Germany
{steffen.prueter, guido.moritz, elmar.zeeb,
ralf.salomon, dirk.timmermann}@uni-rostock.de

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

Abstract

Currently the developing process for enterprise applications is improved by the Service Oriented Architecture (SOA) paradigms. With SOAs the creation of modular and clearly defined software architectures at a high grade of interoperability and reusability is possible. For resource-constraint networked devices the Devices Profile for Web Services (DPWS) specification adapt the SOA paradigms to create a framework for interoperable and standardized communication between embedded devices. This paper explains how special parts of DPWS can be adapted to provide real-time capabilities. So this paper shows how developers of real-time software can use the advantages of Web services and which adaptations are necessary to create Web services with real-time capabilities.

1 Introduction

Today the cross linking between equipment is a major requirement for a wide range of new products. In many cases distributed systems with several independent parties and well-defined interfaces for all components are required. The usage of Web services (WS), to dynamically connect known and unknown devices with each other, is one answer in the network area. The World Wide Web Consortium (W3C)[14] defines Web services as a client-to-server interaction with the Extensible Markup Language (XML) via SOAP [13, 1]. But for device-to-device communication, especially for resource-constraint devices, the WS-Protocols often need too much resources and computing power. Thus, the Devices Profile for Web Services DPWS)[7] was defined by Microsoft, which uses some specific WS-Protocols (see Section 5) but also restricts the usage of Web services to keep aspects of the limitations in embedded systems. Since Microsoft has published DPWS in their new operating system Vista, many companies develop new interfaces for their products based on this protocol.

Our research group has deployed the WS4D-gSOAP toolkit [10]. This is a DPWS solution, which includes a DPWS toolkit and software tools for the creation of own Web services. The usage of DPWS enables a device-centric SOA on the level of devices and embedded systems. This has several advantages, such as providing plug-and-play capability, fault-tolerant services, and standardized interfaces. Furthermore, in the automation and in industrial domains also real-time constraints have to be met. This paper represents ongoing work, which aims at adding real-time capabilities to DPWS, running on networked embedded devices.

2 Network Interface

The network interface is the basis for providing real-time communication of DPWS services. This Section discusses the problem of existing environments, like Ethernet and 802.11, and will discuss the possibility of adding real-time features in this layer.

Two real-time requirements are important in this environment. First, the system must ensure that a data packet arrives at the receiver within a given time. Second, an availability of 100% must be guaranteed. Different projects and papers [6, 3] try to handle real-time communication in the area of wired and wireless network interfaces. But these techniques are not adaptable for real-time services. There are some proposals to for a real-time IEEE 802.11 standard but no wireless real-time communication with more than two parties is established today.

Because of the fact, that no stable wireless real-time communication exists, wired networks are the alternative. The Real-Time Transport Protocol (RTP) uses UDP/IP and was developed for transmitting media streams. The problem why even these RTP and all also other Ethernet based protocols cannot be used for real-time is the Media Access Control (MAC). The MAC uses the CSMA/CD (Carrier Sense Multiple Access/Collision Detection) method to organize the data flow. If a station wants to send data, it checks whether the cable is clear for sending. If yes it starts to send its data. Otherwise it waits for a random time and checks the cable again. Before it starts sending, the station also waits a random time to avoid that two stations are waiting the same amount of time for a free medium. When high network traffic occurs and after a given number of attempts for sending, the upper layer is informed about an error and the transmission on the MAC layer is aborted. With this media access method it cannot be ensured if and when a station can send.

This problem was solved in 2001 by the University of Hannover which developed RTNet [5]. This project developed an Open Source real-time network stack for Xenomai and RTAI [9]. But RTNet only supports UDP connections with real-time characteristics. The limitations of the TCP specification do not allow real-time at all. In case of an error in transmission of the data, resending the data is not deterministic. Due to the fact that with RTNet the important steps to create real-time TCP connections exist, our research group has decided to create an own real-time network.

The fundamental idea of a time slot algorithm for the MAC was used to create static timeslots for knots. Before starting the transfer, the network card driver checks, if it is allowed to send the data. Every network hardware driver in the Linux Kernel 2.6 has a specific function in its API which is used to start transmitting the data [4]. This function (hard_start_xmit) gets as handover the data, which is to be given on the cable. Within this function a blocking algorithm was implemented. If the knot is not allowed to send the data, the driver changes to sleep-mode and waits for the next allowed slot. Of course, this is not conforming to the TCP standard. But in our scenario with a deterministic network, it was an easy way to ensure real-time characteristics for TCP connections. The tests have shown that this simple algorithm yields good results in keeping the deadlines of the slots for all knots.

To ensure that the created algorithms work as deliberated, two problems need to be solved. The first one is that Linux has its own packet scheduler for Quality of Service features. This can be solved by switching this feature off before compiling the kernel. The second is that the Linux network stack must be changed to meet real-time conditions. So the usage of operating systems with real-time capabilities is necessary and will be described in the next Section.

3 Operating System

In order to provide real-time capabilities for DPWS devices the existence of a real-time operating system is an essential requirement. These special types of operating systems can handle all tasks and processes and assure defined execution times and deadlines. This can only be achieved by real-time scheduling algorithms. In addition to these scheduling algorithms, real-time Operating Systems (OSs) have a much finer granularity of timing and a special interrupt-handling. The incoming interrupts must be reorganized according to the attached processes with priorities and deadlines.

This paper focuses on the open-source Xenomai project [2]. Xenomai uses the Adaptive Domain Environment for Operating System's (Adeos) nanokernel [15]. So that Multiple OSs can run on one system in different domains. Furthermore, Xenomai handles the thread scheduling also for DPWS threads. This is a major facilitation for developers, who can use priorities to control the processing sequences of threads and processes. In the implementations of the DPWS toolkit described later, this is necessary to separate real-time and non real-time threads easily. A further important point is Xenomai's Real-Time-Driver Model (RTDM) API. This API must be used to ensure access to hardware with real-time characteristics. The RTNet described earlier is based on Xenomai and must be accessed by using the RTDM. After the setup of a system that can handle real-time message exchange with TCP connections, the real-time parsing of the messages is the next step to provide real-time DPWS.

4 SOAP and XML Schema

The WS4D-gSOAP toolkit, deployed by our research group, uses SOAP messages for the communication between devices. The toolkit is based on the well-known gSOAP Web services toolkit. A toolkit for building SOAP-based Web services with C/C++, developed by Robert A. van Engelen [11]. The toolkit creates a mapping from every type of the used XML schema definitions to a C type structure and generates functions for the marshalling and demarshalling, which allows real-time capabilities. In the paper [12], Robert A. van Engelen describes that gSOAP has a high throughput and low delays. This enables the creation of real-time applications using gSOAP. But also at this point, the worst case and contradiction between DPWS and real-time capabilities arise. So two major problems needs to be solved here.

4.1 HTTP Declared In DPWS Specification

The first problem is that the communication defined in the DPWS specification requires at least one HTTP service interface, which is a TCP Stream. Because of none deterministic parts this is not feasible for real-time applications. To solve this problem, it is necessary to create a service interface, which is separated from the HTTP part and based on UDP. The gSOAP toolkit already supports SOAP over UDP. So this problem is solved. But the adaptation of the DPWS toolkit is required to support both service interfaces and to separate them from each other. Thus the HTTP service interface cannot conflict with the real-time UDP service interface.

4.2 Freely User Defined Service Interfaces

The second problem is that the DPWS specification enables developers to create services with their own freely defined service interfaces. Every structure and complexity for service interface definitions is allowed to support the wide range of possible applications and devices. To declare real-time capabilities, it is necessary to predict the computing time for parsing the SOAP messages. Without the knowledge about the messages, it is not possible to predict this time. Therefore it is not feasible to declare real-time capabilities for DPWS without restriction or incompatibility with the DPWS specification. This problem can be solved with the declaration of restrictions for real-time DPWS. The wide range of unknown service interfaces is only interesting during the discovery time of services. After the services have announced themselves, have discovered the communication partner, as well as the used service interface, the time of handling the communication can be predicted and it only depends on the known service interface structure. Because of this, the new declaration for DPWS with real-time capabilities can only support real-time communication after the service interfaces are discovered, but no limitations for the used service interface specifications are necessary. The service must be separated in different tasks, which is an implementation task of the DPWS toolkit and will be discussed in the next Section.

5 Device Profile For Web Services

This Section describes the structure of the WS4D-gSOAP toolkit and gives an introduction to the used Web service specifications in DPWS. These descriptions of the Web service standards are necessary in order to understand the following discussions in this paper. For further details, the interested reader is referred to the literature [14, 7].

5.1 Specification

DPWS was developed to enable secure Web service capabilities on resource-constraint devices. It features secure exchange of messages with Web services, dynamic discovery, description of Web services, and subscribing to and receiving of events from a Web service. DPWS can be used for inter-machine communication. However, the latter requires the devices to feature peer functionality as well as a specific DPWS client implementation, in order to use the corresponding services hosted on other devices. It employs similar messaging mechanisms as the Web Services Architecture with restrictions to complexity and message size. DPWS specifies further mechanisms for ad-hoc device discovery, device and service description, and eventing. Devices can advertise their services to the network and clients can probe a network for specific devices. Finally, the DPWS also contains a publish-subscribe mechanism (WS-Eventing) for services acting as an event source and sending events to subscribed clients [16].

5.2 Implementation

The Implementation of the WS4D toolkit is based on the open-source gSOAP toolkit [11]. The different WS-specifications are realized in separate modules. So the isolation and reusability of each module is ensured. At the design time the service developer only needs a file, which describes the service functions and data structures in XML metadata. With these files, the code generator creates prototype C source files for clients and servers. The generated client and server executables have the DPWS and gSOAP runtime included and work stand-alone. The different services are combined in one device. Every device consists of a hosting service module, which is responsible for the announcement of all hosted services. Furthermore, a variable number of hosted services that represent the different functions, provided by this device. One XML description file exists for one device.

5.3 Universal Hosting Service

In a device structure, the hosting service, the XML description, and the hosted services are combined in one executable module with sequential processing. Our group has developed a DPWS toolkit with separated hosted and hosting service. In this structure, only one universal hosting service is required, which can announce any kind of hosted services as well as the hosted services running on other systems in the network. This is possible, because the device description is no longer statically implemented inside the device. The universal hosting service has an external device description in the XML format. In this file, the device describes itself and all included hosted services within the device. Other hosted services can add their own XML description in a fixed place, e.g. an XML folder under the universal hosting service. When the file is added, the new services will be announced, and clients can start interacting with the hosted service.

The main advantage of the separation of hosted and hosting services in single executables is the shifting of the scheduling of different services in the field of the underlying operating system. It is not necessary to implement priority-based scheduling or Quality of Service (QoS) features in the DPWS toolkit directly. These tasks are performed by Xenomai. Therefore the existing DPWS toolkit is ported into a Xenomai real-time process. On this account, a Universal Hosting Service is implemented. With the separation of the hosted and the hosting services, it is possible to use different networks for one service. The discovery and metadata exchange can be realized with non real-time characteristics in a standard Ethernet. The usage of the service and the eventing is the part, which can ensure real-time. *Of course, developer of service interfaces must keep in mind that larger interfaces with a big amount of data need more processing and transmission time. This time depends on the size and complexity of the service interfaces.* But with this solution, the processing times can be calculated when the target platform and the interface is known.

6 Robot Scenario

This Section describes the robot environment for testing the implemented real-time DPWS solutions. To control the robots as fast as possible, the data transfer has to be performed within defined and deterministic deadlines. This is necessary to keep the latency of the control loop as small as possible. This affects the local positioning systems of the robots, which needs global coordinates in given time intervals to adjust the errors of the local positing system, caused by slip and friction [8]. For evaluating the real-time DPWS solution, a real-time DPWS service has been integrated in an existing Small Size League controlling infrastructure. The RoboCup server and the embedded systems use Xenomai. All processes used in the tests run with the highest real-time priority. For the communication between the robots and the robot server, a DPWS service interface was defined. The service interface can communicate via Ethernet. The created service interface consists of the position and the commands for the robots summering up to about 125 Bytes of payload data. The data is transferred from the robot server outside the field to the robots on the field. The measured results will be analyzed in the next Section.

7 Measurements

This Section shows the measured data for different tests inside the described scenario. The measured values are the time in μs between the start of the packet at the RoboCup server and the time when the echoed packet is received again at the RoboCup server. The measurement of the time for sending the packet only to the robot was not feasible because no time synchronization between the systems was possible. Also with the usage of the Network Time Protocol (NTP), the time synchronization was not accurate. The time drift between two synchronizations was larger than 2 ms, so this solution has been discarded. This problem is caused by the inaccurate time generation based on the embedded device CPU. So the time for a single data transfer is about less than half of the shown times in the following figures.

The measurements in Figure 1 show the time for the packets after the installation of the real-time OS and the usage of the real-time network. The variance of an unloaded system is 33 μs with an average value of about 2000 μs. In the time between packet 400 and 800, also the CPU load was increased to maximum. As a result, the variance is slightly increased to 49 μs with an average value of 2070 μs. The maximum time in this measurement was about 2300 μs. This means that the usage of a real-time operating system is a good alternative to reduce or nearly eliminate the influence of the CPU load while using Web services.

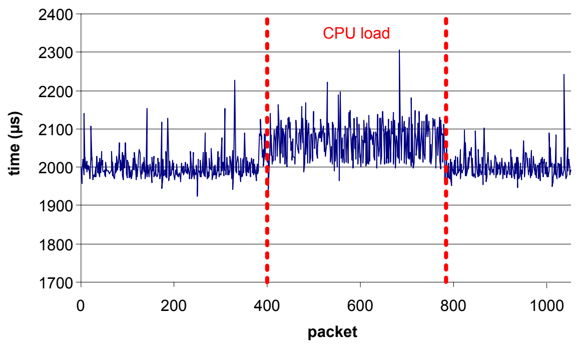

Figure 1. Ethernet packet time with real-time network, real-time OS, and CPU load

In Figure 2, the measurement shows the differences between an unloaded system and respectively network load. The packets from start to 400 are equal to the measurements before. Between packet 400 and 900, network traffic from the embedded system to the RoboCup server was added. Thus, the variance is increased to 530 μs with an average value of 2440 μs. While the packets from 1100 to 1600 have been processed, network traffic from the robot server to the embedded system was added. It seams that the limited resources of the embedded system is the bottleneck of the Web service processing time. The variance is about 3100 μs with an average value of 28200 μs. The maximum time in this measurement was about 29600 μs. Because of this, a deadline of 30 ms for the given service interface and the used hardware can be declared.

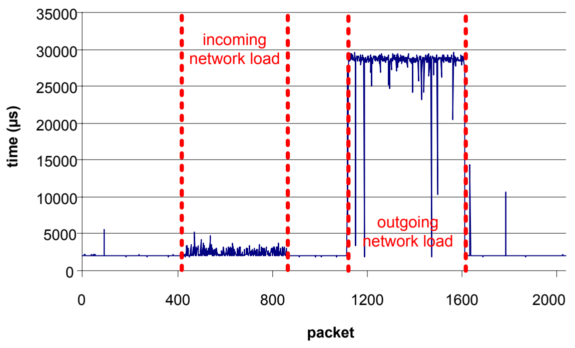

Figure 2. Ethernet packet time with real-time network, real-time OS, and network load

8 Conclusions & Future Work

This paper shows that DPWS with real-time capabilities is not possible in the complete functionality. But the realization of real-time data transfers with the developing of a universal hosting service and separated real-time service interfaces is possible. The problems with SOA specifications and the size limitation of schemata descriptions to provide real-time capabilities are evaluated. The necessary steps for the creation of real-time DPWS based on an example scenario were discussed. After that, the implementation of a possible real-time DPWS interface was shown and the measured data was analyzed. In a closed network, the created service interface had real-time capabilities. Different measurements have show that for known services, a maximum time for processing and transferring the data can be declared. This ensures real-time capabilities for the given scenario. Future interesting research topics are Life Cycle Management, SOAP over UDP, and standard real-time wireless communication.

Acknowledgments

This work has been achieved in the European ITEA project LOMS (www.loms-itea.org) and has been funded by the German Federal Ministry of Education and Research under contract number 01ISF11H.

References

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

[2] GNA people. Xenomai, Real-Time Development Framework. Technical report, GNA people, https://gna.org/projects/xenomai, 2007.

[3] P. Hollczek and B. Vogel-Heuser. *Echtzeitaspekte bei der Koordinierung autonomer Systeme*. Springer, 2005. ISBN 978-3-540-29594-5.

[4] A. R. Jonathan Corbet and G. Kroah-Hartman. *Linux Device Drivers*. O'Reilly Media, 2005. ISBN 978-059-600590-0.

[5] J. Kiszka, B. Wagner, Y. Zhang, and J. Broenink. RTnet - A Flexible Hard Real-Time Networking Framework. In *10th IEEE Internation Conference on Emerging Technologies and Factory Automation*, 2005.

[6] H. Kopetz and G. Grünsteidl. TTP - A Time Triggered Protocol for Fault-Tolerant Real-Time Systems. In *23. International Symposium on Fault-Tolerant Computing*, 1993.

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

[8] S. Prüter, R. Salomon, and H. Burchhardt. *Evolutionary Design of a Control Architecture for Soccer-Playing Robots*. Springer, 2007. ISBN: 978-3-540-72695-1.

[9] Uni Hannover. RTNET. Technical report, Uni Hannover, http://www.rts.uni-hannover.de/rtnet/, 2007.

[10] University of Rostock. DPWS-Stack WS4D. Technical report, University of Rostock, http://ws4d.org, 2007.

[11] R. van Engelen, K. Gallivan, and G. Cybenko. gSOAP Toolkit. Technical report, Florida State University, 2007.

[12] R. A. van Engelen and K. A. Gallivan. The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, 2002.

[13] World Wide Web Consortium. Simple Object Access Protocol Specification. Technical report, World Wide Web Consortium, http://www.w3.org/TR/soap/, 2007.

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

[15] K. Yaghmour. Adaptive Domain Environment for Operating Systems. In *Opersys. Inc.*, 2001.

[16] E. Zeeb, A. Bobek, H. Bohn, S. Prüter, A. Pohl, H. Krumm, I. Lück, F. Golatowski, and D. Timmermann. WS4D: SOA-Toolkits making embedded systems ready for Web Services. In *3rd International Conference on Open Source Systems*, Limerick, Ireland, 2007.