An implementation of embedded RESTful Web services

Author

Chang, CE, Mohd-Yasin, F, Mustapha, AK

Published

2009

Conference Title

2009 CONFERENCE ON INNOVATIVE TECHNOLOGIES IN INTELLIGENT SYSTEMS AND INDUSTRIAL APPLICATIONS

DOI

10.1109/CITISIA.2009.5224244

An Implementation of Embedded RESTful Web Services

C.E. Chang, F. Mohd-Yasin, and A.K. Mustapha

***Abstract*—Web service is a common Internet application that enables interactions of machines over a network. As to establish ubiquitous Internet, enabling Web services on embedded systems is certainly among the development trend in near future. This paper presents an implementation of REST style or RESTful Web services on embedded system. The prototype had been implemented using a Xilinx Spartan-3E Starter FPGA board for home device control application on a 100Mbps LAN environment.**

I. Introduction

THE Web service is an Internet application that enables interactions between machines. It is widely deployed by, yet not limited to business organizations. Current Web services enable a services provider to publish its available services on the Internet, while the clients may freely search and invoke these services through the Internet. Marching towards the vision of ubiquitous Internet, enabling Web services on embedded system is certainly on the to-do-list of the consumer electronics industry for the near future.

This paper provides brief introduction and discussions towards embedded Web services. It then presents an implementation of embedded RESTful Web services on a Xilinx Spartan-3E Starter FPGA Board for home device control application on a 100Mbps LAN environment.

II. Literature Reviews

Two major trends of attempts for embedded Web applications had been observed during the past decade, namely dynamic HTML Web server and SOAP Web services. Both of them are built on top of HTTP and TCP, the 2 well-known protocols of the Internet.

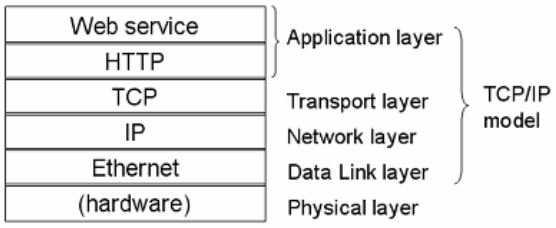

Fig. 1 Protocol stack of Web services (based on TCP/IP model)

Manuscript received 12th June, 2009. This work was supported in part by the Panasonic R&D Centre Malaysia (PRDCM).

C. E. Chang is with the Multimedia University, Cyberjaya, Malaysia (e-mail: childs72@gmail.com).

F. Mohd-Yasin is with the Multimedia University, Cyberjaya, Malaysia (e-mail: faisal.yasin@mmu.edu.my).

A. K. Mustapha was with the Multimedia University, Cyberjaya, Malaysia (e-mail: azhar@alum.mit.edu).

A. HTTP and TCP

HTTP (Hypertext Transfer Protocol) [1] is a protocol designated for request-response communication of machines in server-client mode. A server-client mode refers to a communication mode where server receives request massage from client, processes the request, lastly replies with response message to the client. HTTP message may be in either request or response form. A HTTP message contains of various HTTP headers that contain information of the application, and optional payload data that contain actual information. The payload data may exist in the form of plain text, HTML, picture, XML, etc.

On the other hand, TCP (Transmission Control Protocol) [2] is a reliable, connection-oriented transport protocol. It has been widely adopted by the Internet for over 20 years. TCP receives message from its Application layer to transmit over the network and vice versa. It employs various algorithms and mechanisms to provide 2 main attributes: connection reliability and network performance. In short, it is designed to ensure the data transfer between 2 TCP hosts is done without error at optimized performance.

B. Dynamic HTML Web Server and Related Works

Dynamic HTML (Hypertext Markup Language) Web server had been one of the most common methods of accessing information through the Internet for a good many years. Upon receive request from client, the Web server generates dynamic HTML pages via CGI (Common Gateway Interface) script or ASP (Active Server Pages) to fulfill the request.

Among the examples of embedded HTML Web server implementations, C.N. Coelho *et al.* [3] had connected an uninterrupted power supply (UPS) system to the Internet using an x86 processor. J. Huang *et al.* [4] had implemented Embedded Temperature Web Controller (ETWC) using an 8-bit microcontroller with simplified TCP/IP stack to work as a Web based device controller. T. Lin *et al.* [5] had also implemented Webit using 8-bit microcontroller to attach mini Web server ability to equipments, such as an air-conditioner. Besides, J. Riihijärvi *et al.* [6] had presented the WebChip to host a Web page on an APEX 2K100, a FPGA (Field Programmable Gate Array) board from Altera Corporation.

C. SOAP Web Services and Related Works

Web service differs with traditional Web architecture mainly by its feature of loose coupling between client and server through the use of Extension Markup Language

(XML) format. XML is a very general markup language that had been recognized as "the ASCII of the Web". Transferring data in XML format is much shorter than that in HTML format. Besides, it also allow client to easily perform post-processing towards the desired data, instead of the HTML page, that are received. [7] Current Web services style is dominated by SOAP (Simple Object Access Protocol) style; followed by its rising competitor REST (Representational State Transfer) style.

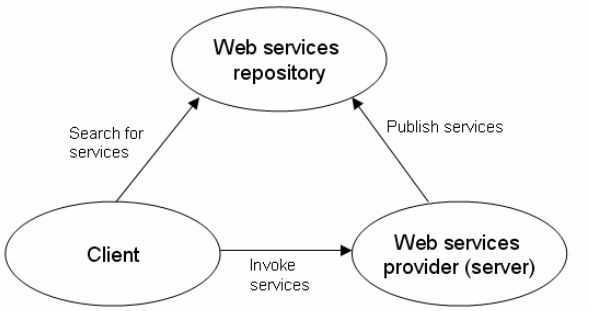

Fig. 2. Web services operations

Fig. 2 shows how Web services work. First, the Web services providers will publish their services and related information (e.g. WSDL) on a Web services repository (e.g. UDDI). The client may search for the services they desired on the repository. Based on the services information acquired from the repository, the client may invoke the services from the service provider or server. The service invocation is a request-response communication, where the request and response messages are wrapped in certain packaging format (e.g. SOAP or REST) and transmitted between client and server.

Embedded Web services implementation are normally done on microcontroller or microprocessor with higher computation power using special programming toolkits. For instance, G. Bucci *et al.* [8] had implemented SOAP/WSDL Web services on a RabbitCore RCM2100 board using ASP.NET. G.B. Machodo *et al.* [9] had also implemented SOAP/WSDL Web services on an ARM-based SHIP board using gSOAP toolkit. O. Almeida *et al.* [10] had implemented SOAP-based Web services on an Atmel EB63 Arm7 board using Microsoft Invisible Computing software platform. Lastly, the only FPGA implementation of SOAP Web services was carried out by S. Cuenca-Asensi *et al.* [11] on a Celoxica RC203E FPGA board.

III. JUSTIFICATIONS

Section II had reviewed that Web services possess 3 advantages over dynamic HTML Web server by the use of XML data format: loose coupling between server and client, shorter message length, and flexibility for post-processing at client side [7].

For both conventional and embedded Web services implementations, SOAP appears to be the main concentration of current trend. SOAP is a well-established standard and supported by majority of vendors and companies in the industry. Meanwhile, REST is a Web service style that is newer but supported by a good number of individual Web developers. Proposed by R.T. Fielding [12], one of those who helped to develop HTTP, REST is highly coupled with HTTP. As both SOAP and REST normally sit on top of HTTP, REST, which extended from HTTP, is thus much plainer and simpler to process than SOAP. It is generally agreed that SOAP is more suitable for applications of higher complexity; while REST fits better for less complicated applications [13].

Besides, REST produces service messages with shorter length than that of SOAP Web services [14]. Table 1 shows the brief differences between SOAP and REST style Web service messages. For service requests, a SOAP service request exists as HTTP payload in SOAP envelope while REST service request message resides in HTTP headers. Hence the operation of request interpretation is fairly simpler in REST because the Web server can directly extract the request message from the HTTP headers instead of parsing the HTTP payload data. A SOAP service response exists as HTTP payload in SOAP envelope. For REST on the other hand, currently, the RESTful Web service has not yet strictly defines the format of the REST response message.

TABLE 1
BRIEF DIFFERENCES BETWEEN SOAP AND REST MESSAGE FORMAT

	SOAP	REST
Request Format	HTTP request headers (usually POST method) with SOAP envelope as payload data; service request resides in SOAP envelope.	HTTP request headers only (usually GET or POST method); service request resides in method and URI fields in HTTP headers.
Response Format	HTTP response headers with SOAP envelope as payload data; service response resides in SOAP envelope.	HTTP response headers with XML message as payload data; service response resides in XML message.

Since RESTful Web service uses shorter message and its overall architecture is significantly simpler compared with SOAP, it requires less processing power and memory compared with SOAP. From the point of view of embedded system design, a RESTful Web service is therefore more suitable option than a SOAP Web service.

IV. IMPLEMENTATION OF RESTFUL WEB SERVICES

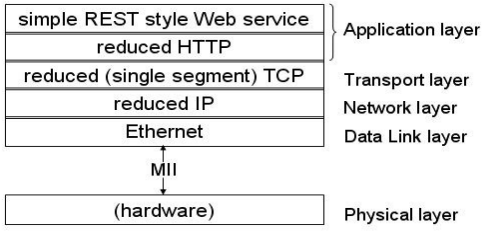

Fig. 3. Protocol stacks of prototype

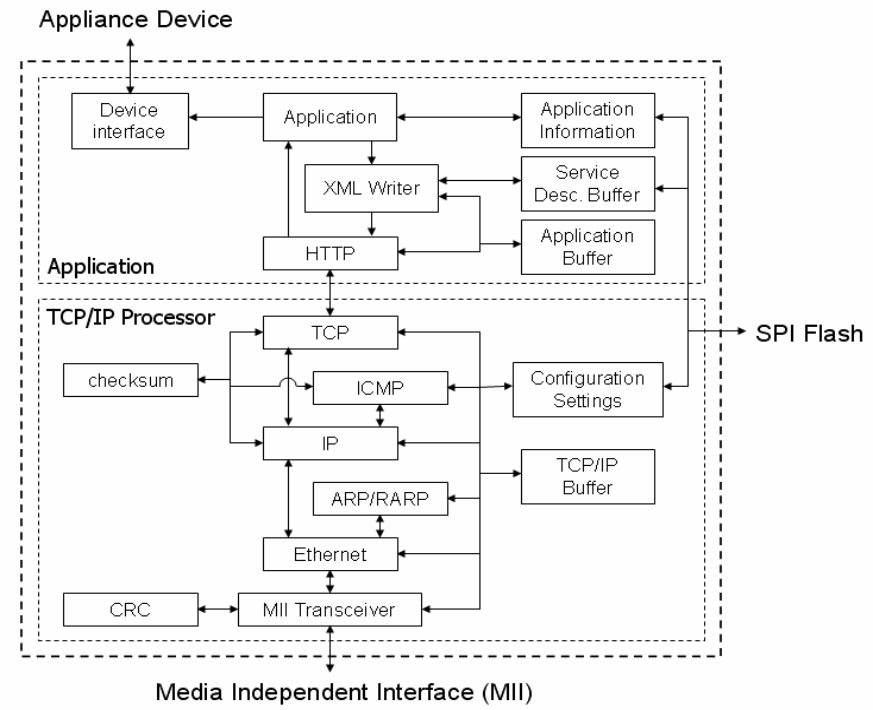

Fig. 4. Block diagram of embedded RESTful Web services

To better illustrate how to embed RESTful Web services, a simple prototype had been designed and implemented. The prototype of this embedded RESTful Web services is targeted to provide home device control application. It had been designed using VHDL (Very High Speed Integrated Circuit Hardware Description Language) and implemented on a FPGA board. The prototype design covers from layer 2 (Data Link layer) to layer 5 (Application layer) of the Web service as shown by Fig. 3. The prototype was designed as a passive server, which only responses to incoming request or communication instead of initiating communication towards any device.

Fig. 4 shows the complete block diagram of the embedded RESTful Web services. There are 3 main external interfaces on the server. The MII (Media Independent Interface) connects the server to the physical layer of 10/100Mbps LAN (Local Area Network). The home appliance device to be controlled is connected to the server through a device interface. Lastly, a SPI (Serial Peripheral Interface) is used to store the non-volatile information, such as the application information, services descriptions and server configuration information, when the server is turned off. Inside, the prototype is divided into 2 parts: the TCP/IP Processor and the Application.

A. TCP/IP Processor

The TCP/IP Processor resembles a reduced network processor for the Web service application. As shown in the diagram, it supports basic protocols of TCP/IP stack: Ethernet, ARP, RARP (layer 2), IP, ICMP (layer 3), and TCP (layer 4). These protocols are each handled by individual module. Error detection mechanisms, such as CRC checking for Ethernet and checksum calculation for IP, ICMP and TCP, are implemented to ensure correctness of data received from LAN.

Due to resource minimization consideration, each protocol module is designed to provide only basic functionality necessary for a passive Web server. The server uses static IP

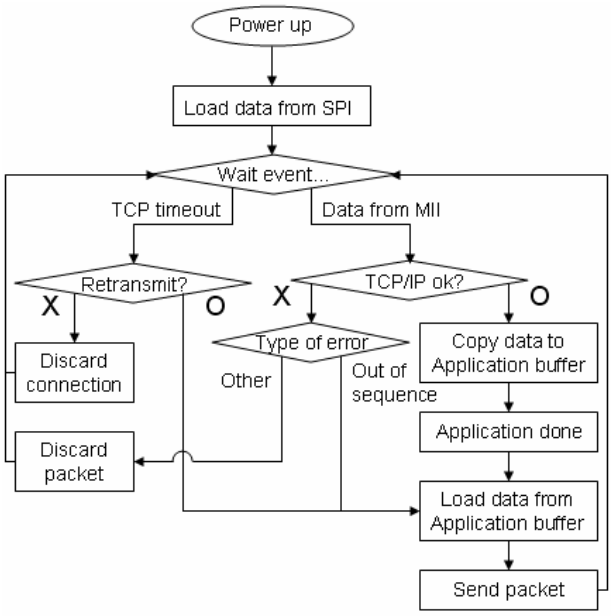

Fig. 5. TCP/IP Processor operation flow

address. The TCP/IP Processor is able to receive and send TCP segment. Aside to TCP handling, it only processes and responses to "ping" (ICMP), ARP and RARP requests. Other messages received with unrecognized protocols or unsupported operations will be discarded at once.

The TCP/IP buffer size is limited to only single packet or 576 bytes of data at IP layer. Only 1 TCP segment can be received or sent at one time. Consequently, some of the conventional TCP features and algorithms, that are important to support TCP performance to send or receive large amount of data (typically more than single TCP segment), are safely omitted in this prototype. These algorithms and features include Nagle Algorithm, Slow Start, Congestion Control, Fast Retransmit, Fast Recovery, window buffer management and multiple simultaneous connections [15]. Separate buffers are employed by TCP/IP Processor and Application to allow TCP segment retransmission to assure the reliability of the connection.

B. Application

The major difference of data type processed by TCP/IP Processor and Application is the former handles protocols headers in binary data format; while the latter handles protocols headers and Web services information in character (ASCII) format. In TCP/IP Processor, 16-bit width buffers and data buses are employed to speed up the process as well as to ease the checksum and CRC calculations (where both are calculated by 16-bit data width basis). In Application, however, 8-bit width buffers and data buses are used to ease the processing of ASCII characters.

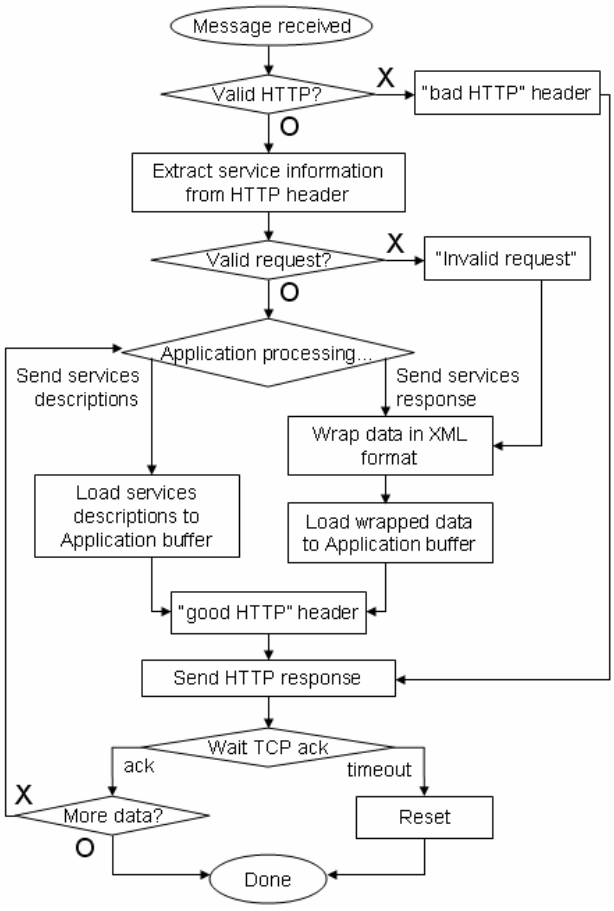

Fig. 6. Application operation flow

Fig. 6 presents the operation flow of the Application. The main modules responsible to handle REST request reception and response generation are the HTTP and XML Writer modules. Upon receives a valid TCP/IP segment, the HTTP module checks for the data validity as a HTTP request. It then extracts the REST Web service information from the HTTP headers and verifies the service request. The core application unit will then process the request. Either a services response message or the services descriptions will be sent back to the client in XML format through the XML Writer module. When the response to be sent is larger than the available buffer size (576 bytes after includes TCP/IP headers, as mentioned earlier), the core application unit will halt at once [16] and the XML Writer will send out the incomplete service response. After TCP/IP Processor receives an acknowledgement that indicates the partial response has been received by the client host, the core application unit shall resume until full response is completely sent to the client. This design that only allow single TCP segment to be sent at a time can degrade the system performance. However, such scenario where large response message size is less likely to happened in this prototype by the use of REST style Web services with very plain XML data format.

Simple responses in XML format are defined for the prototype Application by using few tags and messages as shown in Table 2.

TABLE 2
PREDEFINED TAG NAMES AND ELEMENT NAMES

Tag Name	Element Name	Description
REST	(none)	Contain the entire REST response message.
RET	(retrieved information)	Contain retrieved information as requested by client.
MSG	NO_MATCH	Request failed as targeted information cannot be found.
	FUNC_ERR	Request failed as it has an invalid request format.
	DONE	Request to control appliance is successfully performed.

C. Implementation and Testing

Considering the ease of access towards MII and SPI memory, Spartan-3E Starter Board, an FPGA board product from Xilinx Inc. [17], had been selected as the prototype platform. Spartan-3E Starter Board has both MII and SPI memory onboard. The server is operating with 50MHz clock rate on a 100Mbps LAN.

For the prototype testing, a washing machine simulator from Bytronic International Ltd. [18] was chosen to act as the target home appliance device to be controlled by user

through embedded RESTful Web services. A Java program, Direct Socket Control v6.0 (DSC) [19] had been used to open a HTTP connection from a PC as a Web service client to communicate with the server and invoke the RESTful Web services. Fig. 7 shows a screenshot of the DSC program, running on a PC, acting as client to communicate with the server, operating on the Spartan-3E Starter Board, through 100Mbps LAN. The services messages are noticeably simple and short. Note that at the left side of the screenshot, the symbol "[>]" indicates a message sent by the client to the server; while the symbol "[<]" indicates the vice versa. The screenshot shows a few services request-response communications between the client and the server:

1. Client requested to retrieve Web services descriptions;
2. Server replied with Web services descriptions in XML format;

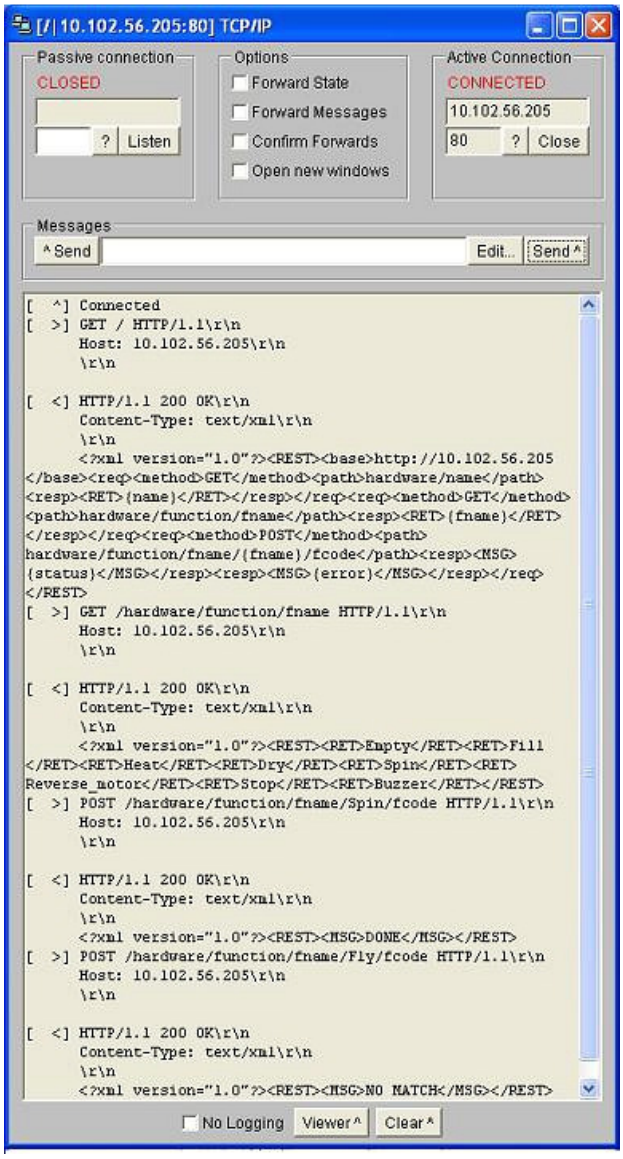

Fig. 7. Screenshot of DSC communicating with embedded Web server

3. Client requested to retrieve information on available function on target home appliance;
4. Server replied with list of available functions;
5. Client requested to execute function "Spin";
6. Server executed function "Spin" and replied to notify that function successfully performed;
7. Client requested to execute an unavailable function "Fly";
8. Server replied to notify that "Fly" function is not available.

The total resource usages of the embedded server on the Spartan-3E Starter Board are 2793 flip-flops (FFs), 7958 4-input look-up tables (LUTs) and 4180 slices. The breakdown of resource used by main modules is shown as table 3 below. Note that this entire prototype covers both TCP/IP Processor (starts from layer 2) and Application (until layer 5). The modules responsible to dispatch request information and to generate response of REST messages (HTTP and XML Writer modules) only consume altogether 371 FFs and 36 MUXs.

TABLE 3
RESOURCE USAGE BY MAIN MODULES

Modules	FF	MUX
TCP/IP Processor:		
1. MII Transceiver	124	-
2. CRC	87	-
3. Ethernet	50	35
4. ARP/RARP	35	35
5. IP	79	71
6. ICMP	45	77
7. TCP	513	64
8. Checksum	40	-
Application:		
9. HTTP	277	36
10. XML Writer	94	-
11. Application	478	41
12. Device Interface	20	-
Miscellaneous:		
13. SPI Interface	67	-

V. CONCLUSIONS

By enabling the Web services on embedded systems, significant power and resource savings can be achieved compared with conventional Web applications implemented on general computing platforms (such as PCs). Such power and resource cut down will show tremendous implication during the arrival of ubiquitous Internet era, which is just around the corner.

This paper proposes to embed the REST style Web services. Compared with dynamic HTML Web server and SOAP Web services, REST style Web services provide simpler processing and shorter message length, thus requires lower resource. Besides, it also provides flexibility on client

side post-processing over the dynamic HTML Web server. A prototype of embedded RESTful Web services had been implemented on FPGA board and tested on 100Mbps LAN environment. As results of the simplicity provided by REST style Web services, resource reductions are achievable throughout the Web server design. Estimated short REST style messages allowed the implementation of single segment TCP module. Meanwhile, the less complicated REST messages format allowed the design of a plain top level Application module.

In a nutshell, RESTful Web service is almost as useful as, yet simpler than the SOAP Web service. Therefore the REST should be promoted as the mainstream of embedded Web services development. Further researches to support the embedded RESTful Web services include to optimize the REST messages processing as well as to standardize the REST Web services for embedded Web applications.

REFERENCES

[1] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyks, L. Masinter, P. Leach, and T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, RFC2616, Jun 1999, http://www.ietf.org/rfc/rfc2616.txt.

[2] J. Postel, Transmission Control Protocol, RFC793, Sept 1981, http://www.ietf.org/rfc/rfc793.txt.

[3] C. N. Coelho Jr., D. C. Silva Jr., W. C. Padrão, C. Á. Rodrigues, J. M. Mata, and A. O. Fernandes, "Reengineering Embedded Systems for the Internet," 15th Triennial World Congress, Barcelona, Spain, 2002.

[4] J. Huang, J. Y. Ou, and Y. Wang, "Embedded Temperature Web Controller Based on IPv4 and IPv6," Proc. 31st Annual Conf. of the IEEE Industrial Electronics Society (IECON'05), 6-10 Nov. 2005.

[5] T. Lin, H. Zhao, J. Wang, G. Han, and J. Wang, "An Embedded Web Server for Equipments," Proc. 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '04), 2004.

[6] J. Riihijäarvi, P. Mäahöonen, M. Saaranen, J. Roivainen, and J. P. Soininen, "Providing network connectivity for small appliances: a functionally minimized embedded web server," *IEEE Communications Magazine*, vol. 39, no. 10, pp. 74–79, Oct 2001.

[7] D. Martin, M. Birbeck, M. Kay, B. Loesgen, J. Pinnock, S. Livingstone, P. Stark, K. Williams, R. Anderson, S. Mohr, D. Baliles, B. Peat, and N. Ozu, *Professional XML*, Birmingham, Eng.: Wrox Press, 2000.

[8] G. Bucci, F. Ciancetta, E. Fiorucci, D. Gallo, and C. Landi, "A Low Cost Embedded Web Services for Measurements on Power System," *IEEE International Conf. on Virtual Environment, Human-Computer Interface, and Measurement Systems* (VECIMS 2005), 18-20 July, 2005.

[9] G. B. Machado, F. Siqueira, R. Mittmann, and C. A. V. Vieira, "Embedded System Integration Using Web Services," *Proc. International Conf. on Networking, International Conf. on Systems and International Conf. on Mobile Communications and Learning Technologies* (ICNICONSMCL '06), 2006.

[10] O. Almeida, J. Helander, H. Nielsen, and N. Khantal, " Connecting Sensors and Robots thought the Internet by Integrating Microsoft Robotics Studio and Embedded Web Services," *IADIS International Conference WWW/Internet 2007*, Vila Real, Portugal, 5-8 Oct, 2007.

[11] S. Cuenca-Asensi, H. Ramos-Morillo, H. Llorens-Martinez, F. Mácia-Pérez, "Reconfigurable Architecture for Embedding Web Services," *4th Southern Conference on Programmable Logic*, Bariloche-Patagonia, Argentina, 26-28 Mar, 2008.

[12] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," PhD Dissertation, Chapter 5, Dept. of Computer Science, Univ. of California, Irvine, 2000.

[13] W. Brogden, "REST vs SOAP – the SOAP story," SearchSOA.com, 28 Nov 2006 (accessed 30 Jan 2008), http://searchsoa.techtarget.com/tip/0,289483,sid26_gci1231889,00.html.

[14] D. Winer, "REST, SOAP and XML-RPC," blog, 4 Jul 2006 (accessed 3 Mar 2008), http://www.therssweblog.com/?guid=20060704042846.

[15] M. Allman, V. Paxson, and W. Stevens, *TCP Congestion Control*, RFC2581, Apr 1999, http://www.ietf.org/rfc/rfc2581.txt.

[16] W. R. Stevens, B. Fenner and A. M. Rudoff, *UNIX Network Programming，Vol. 1: The Socket Network API*, 3rd ed., Addison Wesley Professional, Chapter 2, Nov 2003. [E-book], http://safari.oreilly.com/0131411551.

[17] Xilinx, "Spartan-3E Starter Kit," (accessed 2007), http://www.xilinx.com/products/devkits/HW-SPAR3E-SK-US-G.htm.

[18] Bytronic, "Washing Machine Simulator," (accessed 2007), http://www.bytronic.net/html/wms.html.

[19] M. Schierl, "Direct Socket Control," (accessed 2008), http://home.arcor.de/mschierlm/sm-soft/dsc.htm.

[20] S. Feit, *TCP/IP-Architecture, Protocols & Implementations*, New York: McGraw-Hill, 1993.

[21] J. Bentham, "Miniature Web Server," *Embedded.com*, 15 Mar 2001 (accessed 1 Feb 2008), http://www.embedded.com/story/OEG20010312S0103.

[22] P. Brittenham, "An overview of the Web Services Inspection Language," *IBM*, 1 Jun 2002 (accessed 15 Feb 2008), http://www.ibm.com/developerworks/webservices/library/ws-wsilover/.

[23] D. Chapell, "REST: Another Way of Looking at Web Services," *informIT*, 12 Jul 2002 (accessed 30 Jan 2008), http://www.informit.com/articles/printerfriendly.aspx?p=27645.

[24] J. Canosa, "Introduction to Web Services," *Embedded.com*, 2 Jan 2002 (accessed 2 Oct 2008), http://www.embedded.com/story/OEG20020125S0103.