FACTA UNIVERSITATIS (NIŠ)
SER.: ELEC. ENERG. vol. 21, no. 1, April 2008, 23-36

Using XML Web Services as a Platform for Remote Access and Control of Embedded Systems

Danilo J. Oklobdžija and Branislav T. Jevtović

Abstract: This paper has pointed out the need for new technologies for integrating embedded devices in heterogeneous distributed networks, and outlines the perspectives opened by applaying service-oriented paradigm for realizing interaction with embedded devices. As a foundation for interaction with embedded devices, XML and Web services have been used. Strong integration power of XML and Web services make presented framework appropriate for unique approach to all resources of embedded device regardless of applied technology. Developed Web services based middleware, provides application designers with a high level of semantic abstraction, hiding the complexity of the applied technologies. This makes presented framework suitable for use in many domains, independent of embedded devices physical realization or network characteristics. Two ways to achieve such integration have been presented in this paper: device level for modern embedded devices, and using service oriented gateway for "legacy" embedded systems.

Keywords: Embedded system, XML, Web service, middleware, service oriented architecture.

1 Introduction

At the end of last century, August 30, 1999, Neil Gross's article [1] provides a vision for the networked embedded systems, i.e. Sensor Web concept: in 21st century. "In the next century, planet Earth will don an electronic skin. It will use the Internet as a scaffold to support and transmit sensations. This skin is already being stitched together. It consists of millions of embedded electronic measuring devices: thermostats, pressure gauges, pollution detectors, cameras, microphones,

Manuscript received on December 9, 2007.

D. Oklobdžija is with Secondary technical school - Niš (e-mail: odanilo@sbb.co.yu). B. Jevtović is with Business School of Professional Studies - Blace, (e-mail: banej@vpskp.edu.yu).

glucose sensors, ECGs, electroencephalographs." This vision starts to become a reality nowadays with embedded Internet integration, so-called M2M (machine to machine) interaction, for remote monitoring and diagnostic, remote maintenance, distance learning, etc.

Despite the high level of integration achieved in B2B (business to business) interactions, the same level of integration is not obtained with embedded systems. One of the major problems of integrating embedded systems is the fact that embedded system does not cover just one concept or class of devices. Instead, embedded systems mean a whole range of devices from very small low power solutions up to large multiprocessor systems. These systems need to take physical inputs, react in real-time, and produce output at the right moment. Since embedded systems are usually highly customized to perform limited set of tasks, when integrating embedded devices into distributed networked systems, the designer must consider constraints on the physical environment, power and memory consumption, code size, etc.

The purpose of our research activities is the development of such framework which radically simplifies the integrating embedded devices in distributed networks and Web applications. The framework should contain a middleware that allows embedded devices to be integrated in heterogeneous distributed systems, by using open, standardized, programming languages and vendor neutral technologies and protocols.

2 Software Engineering for Networked Embedded Systems Applications

For the development of application software for embedded systems, the use of a component based framework is desirable. The components of the framework should provide the functionality of single embedded device, as well as functionality for integrating embedded systems in heterogeneous distributed networks and Internet. According to presented requirements, components are organized into three layer middleware.

The ***sensor/actuator*** layer provides essential basic local functions of the embedded device, has full access to the hardware and is able to access the functions of embedded real-time operating system directly. ***Node or device layer***, contains all application specific tasks, and functions of the applications to build up and maintain the network. ***Network layer*** describes the main tasks and required services of the entire network, without assigning any tasks or services to individual nodes. It represents an interface to the client/administrator to evaluate the network results.

Fig. 1 shows the logical view on a embedded systems network application. The

devices can be contacted only through services of the middleware layers. The Distributed Middleware coordinates the cooperation of the services within the network. It is logically located in the network layer but it exists physically in the nodes. The Administrator/Client terminal is an external entity to configure the network and evaluate the results.

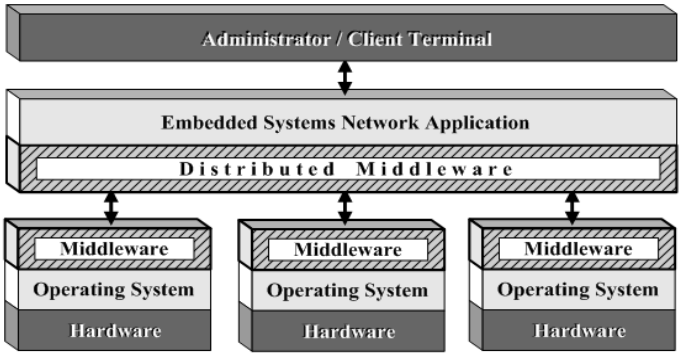


Fig. 1. Logical structure of embedded systems network

3 Middleware for Networked Embedded Systems Requirements

The term middleware refers to the software layer between operating system and embedded device application on one hand, and the distributed application on the other. The primary objective of the middleware layer is to hide the complexity of the network environment by isolating the application from protocol handling, memory management, and network functionality [2]. Middleware provides application designers with a higher level of abstraction, hiding the complexity introduced by distribution. In other words, distribution becomes transparent.

It is a difficult task to define all requirements that need to be taken into consideration by the middleware developers, due to diverse heterogeneous networked environments in which embedded systems can be found. Short overview of basic non-functional requirements of middleware for networked embedded systems is shown in [3]. On the other hand, a number of different functional requirements must be addressed in order to enable communication between the application and the network itself. Basic requirements common for all embedded systems are well represented in [4]. Table 1 shows overviews mentioned above and adapted by the authors, and presents functional and non-functional requirements of the middleware in a concise form.

Table 1. Overview of basic non-functional and functional requirements of middleware

	Requirement	Description
Non-functional	Heterogeneity	Components written in different programming languages, running on different operating systems, executing on different hardware platforms, should be able to communicate using a middleware platform.
	Openness	The capability to extend and modify the system, for example, with respect to changed functional requirements.
	Scalability	The ability of the system to accommodate a higher load in the future.
	Security	Security mechanisms, such as authentication, authorization, and accounting functions enables intelligent control access to computer and network resources.
	Performance	Performance can constitute a requirement of the middleware in various situations. For example, a middleware used for a real-time distributed application should export functions with a minimal execution time, whereas a memory-limited device would require optimization of the memory usage.
	Adaptability	Changes in applications and users requirements or changes within the network, may require the presence of adaptation mechanisms within the middleware.
	Feasibility	Constraints of available resources may limit the feasibility of performing certain tasks or offering certain services in a given system/network environment.
Functional	Addressing	This is the foundation for device networking. In the case of IP-based networking, the addressing capacity is provided by the IP protocol.
	Discovery	Once addressing is established, devices need to be discovered. A discovery protocol enables a device to advertise its services on the network.
	Description	To learn more about the device each device has additional ”metadata” for device’s description. The device metadata may include information like manufacturer name, version, serial number, etc.
	Control	To invoke an action of service a control message has to be send to the network endpoint for that service.
	Real Time	It is the task of the middleware to monitor and maintain time and deadlines in order to maintain these kinds of guarantees.
	Synchronization	Precise synchronization among using different sensors/actuators is required.
	Eventing	Devices may communicate through asynchronous eventing, usually implemented by a ”publish-subscribe” mechanism, through which a service exposes events corresponding to internal state changes.
	Presentation	Finally, a device may expose a presentation interface. Typically this is HTML page or XML document, which enables a user to view the device’s status.

4 Related works

A survey of Middleware for Networked Embedded Systems [3], part of RUNES project, present architectures, basic characteristics and list of the functional and non-functional requirements for embedded networked systems, and middlewares developed for these different systems. Table 2 shows names, concise indication of the main purpose, and the type of embedded system for middlewares presented in RUNES project. Authors consider three types of embedded systems and appropriate middlewares for these systems. ***Mobile Systems*** can be further subdivided into Nomadic systems and Ad Hock systems. Nomadic systems are those containing a core fixed network and some mobile nodes, Cellular networks are typical examples of systems with this topology. Ad Hock systems have a very decentralized structure where no fixed core exists and where each node may be mobile. In both cases the type of network links is wireless. ***Embedded Systems*** are systems assuming that the computing components are embedded into some other purpose built device. These systems are most of the time not mobile and very often not networked in large scale, i.e. only few devices are connected. The type of the connection is often wired. ***Sensor Systems*** are often composed by large numbers of possibly tiny devices that have the sole task of monitoring some conditions within an environment and report back to a central server. The most common sensor networks are usually not mobile but the sensors are connected through wireless links.

A detailed explanation of architecture, structure, characteristics and requirements addressed by each of the presented middleware is given in RUNES project. ”Table 5.1 Assessment of overviewed middleware platforms” ([3] pp. 70-72) show the list of functional and non functional requirements for each presented middleware.

Analyzing characteristics of presented middleware for networked embedded systems is beyond the scope of this paper and only base technologies for service-oriented middleware will be considered. Now a days, several service-oriented middlewares can be identified as basic platform for integrating embedded systems into heterogeneous distributed networks:

- The **OSGi** is Java based specification that serves as a common architecture for service providers, service developers and software equipment vendors who want to deploy, develop, and manage services [5].
- **HAVi** was developed for and is still restricted to the home domain, in particular IEEE1394 networks [6]. It offers plug-and-play capability as well as Quality-of-Service (QoS) support.
- **JINI** was developed by Sun Microsystems for spontaneous networking of services and resources based on the Java technology [7]. Services/devices

are registered and maintained at a centralized meta service called Look-up Service.

- **UPnP** is a simple, easy-to-use service oriented architecture for small networks [8]. It supports ad-hock networking of devices and interaction of services by defining their announcement, discovery and usage. Programming languages and transmission media are not assumed.
- The **Web service** architecture provides a set of modular protocol building blocks that can be composed in varying ways to create protocols specific to particular applications [9]. They address networks of any size and provide a set of specifications for service discovery, service description, security, policy and others.
- **DPWS** is a profile identifying a core set of Web services that enables dynamic discovery of, and event capabilities for Web services [10]. DPWS supports discovery and interoperability of Web services beyond local networks.

Table 2 shows an excerpt from comparing technologies for device integration [11].

Table 2. Comparing evaluated technologies for device integration

	OSGi	**HAVi**	**JINI**	**UPnP**	**WS**	**DPWS**
Plug and play	-	X	X	X	-	X
Device support	X	X	X	X	-	X
Programming language independent	-	X	-	X	X	X
Network media independent	-	-	X	X	X	X
Large scalability	X	-	X	-	X	X
Security	X	X	X	-	X	X
High market acceptance	X	-	X	X	X	X

5 XML Web Services based framework for embedded systems

Even though each integration solution for embedded devices is unique, and targeted for specific usage domains, they all have the basic idea of providing service-oriented architecture which meets the current set of business requirements. XML Web Services based framework for embedded systems, shown in this part of the paper, enables a unique approach to all resources of an embedded device, whether the device is a ”legacy” embedded system, or a new generation embedded system.

For ”legacy” embedded devices service oriented concept, from hardware point of view, is realized by using a PC (general purpose machine) as an access point for network of an appropriate embedded device type. From software point of view PC

transfer different types of data between different network types, and therefore can be treat as a gateway. For new embedded devices with more powerful hardware and software resources, service oriented concept can be realized within the embedded device.

Regardless of the way that the concept is realized, the interaction between a user (Internet browser, application or network device) and the embedded device is done by exchanging XML messages i.e. by using ”semantic” commands. The structure of each command, as well as relationships among different commands, is defined by XML schema. The XML document, which represents command sets, has to be in accordance with the XML schema, i.e. a validation of the document is required. Validation of semantic commands and their transformation into a byte stream is done by an appropriate Web service, which represents a set of methods used for solving a problem and is based on widely accepted standards like SOAP (Simple Object Access Protocol) and WSDL (Web Services Description Language).

Fig. 2 shows logical steps in using ”semantic driver” framework. The integration goal is achieved by integrating Web service as an abstraction layer between hardware and low-level binary controls on one hand, and client for XML meta-language on the other. In this special case, the client wants to acquire information about temperature and humidity in the area of interest. After the client has discoved proxy for appropriate Web service, via standardized Web service discover procedure, the client generate request to the proxy (1). The Web service (i.e. proxy), validates ”semantic command” and communicates with the distributed nodes (embedded devices) using a low level binary proprietary protocol (2). The addresed device, using appropriate sensor protocol (e.g. I2C for Sensirion SHTxx humidity & temperature sensor) reads temperature and humidity and send results to the proxy (3). The proxy translates the information into standardized SOAP protocol and sends them back to the client (4). For addresing embedded devices both address-centric protocol and data-centric protocol can be used, according to the type of the applied network. Web service then has to convert addressing used in XML document into a physical address.

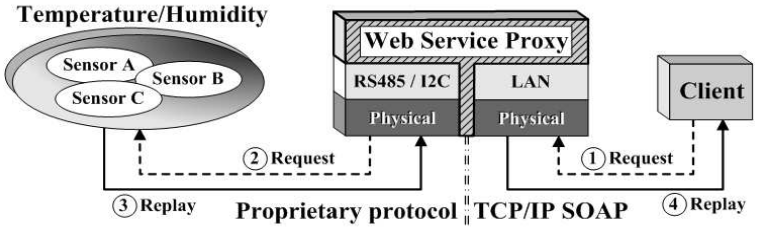


Fig. 2. ”Semantic driver” - Logical overview

As we can see Web service framework for embedded systems is realized as distributed service oriented solution and has three basic layers.

Client application is responsible for authorization, authentication and enables uploading high level command file. Interface with Web service is realized by adding reference to Web service proxy class. Client application can be realized as Active Server Page, Java Server Page or as Windows form application. ***Web service*** is a level of indirection between client and application logic [12]. All business rules for validating and translating XML semantic commands to low level binary commands are realized as Web service methods. As we can see, Web service has dual role: as a programming abstraction and as an infrastructure for integration. As programming abstraction the Web service hides low level details of the hardware and binary command set, which enable semantic networking even with ”dumb” or ”legacy” devices. As integration infrastructure, the Web service provides a comprehensive platform for developing and running complex distributed systems in a heterogeneous environment. ***Embedded software*** deals with the hardware and low level commands. It receives low level binary commands and takes appropriate actions, like read sensors or drive actuators.

6 Case Study Implementations

As proof-of-concept, for using Web services at embedded systems domain, two different applications, LDM (Long Distance Measurement) module and RoboArm, will be presented in this section. In both applications the Microsoft Visual Studio.Net 2005 (C#) and Altova XMLSpy 2006 were used, as developing tools at PC platform. For development of the embedded software, in LDM application, the embedded Web server ”WebCat” and real-time operating system Ethernut/Nut OS were used. In RoboArm module SDCC (Small Device C Compiler) for a SAB 80535 microprocessor was used.

In both applications, address-centric approach was used. Embedded devices were connected in star or bus network topology, where a distinct path from source to sink and end-to-end routing is required.

6.1 Case study 1 - LDM

Within a project for simulation and guidance of a bucket wheel excavator (BWE) on open pit minig, a hardware-software module for remote measurement, signalization and guidance LDM was developed [13]. The LDM module enables controlling the BWE, simulation of malfunctions, resetting an alarm, measuring, and signaling. Fig. 3 shows the hardware configuration of the BWE training center, realized

with the LDM module. In this case, the embedded Web server *"WebCat"* is used as a client. It sends information about the state of its peripherals to a remote server, where this information is translated and stored in a database. The WebCat generates appropriate XML documents only when the state of some sensor changes. Workstations (PC_0, ... , PC_N) can access only the information that is stored in the database, while special purpose workstation is used for 3D visualization. The dynamics of the real system is completely defined by the dominant time constant T_{rt} of the transfer function (T_{rt} = 1.25[s]) [14, 15]. That is why time discretization of $\Delta t = 100[ms]$ ($\Delta t < T_{rt}/10$) is applied in the developed and realized model. The realized model is designated for working in the Intranet network of a surface mine. After a large number of measurements in the real system, in different working conditions and activities, it has been found that latency ranges from 1 to 12[ms], and that the maximum jitter is 92[ms]. At peak traffic the maximum number of lost packages is 3 out of 10,000 sent packages. The model has logic of holding the previous state. This logic is applied in the case of package losing. In the case of package loss, the effect of the "hold previous state" logic is equivalent to the noise of the measurement, which is in this specific case (3/10000) practically negligible. These conditions in the network do not compromise the accurate and reliable operation of the developed model.

The hardware of control desk was realized with Ethernut 1.3f compatible platform. The outputs from the control stick are connected to the A/D converter, which evaluates movements along x and y axis. Switches, buttons and LEDs, that indicate states, malfunctions, alarms and signalization, are connected to the digital input/output ports. On the other hand, by using the integrated Ethernet connector, a connection to the access point (AP) is established, providing access to the server via the wireless Internet (router).

The software of LDM module is based on service oriented architecture. It consists of a client part, a service oriented middleware, and a database. The embedded client scans its peripherals (control stick, buttons and switches), and if some change occurs, generates output signals (LEDs) and sends a message to the server in order to overwrite its state in a database. Logically, the embedded client is part of the distributed middleware (Fig. 1.), but physically it exists in the embedded Web server. The following code snips illustrate the conctete implementation in embedded Web server "WebCat". The part of a middleware software, installed it the general purpose server, accepts XML document, which the embedded server send. In addition it uses Web service to validate the document and interacts with the rest of the system, i.e. the database in a concrete case. The server part of middleware modules is realized for Microsoft Windows XP Professional platform.

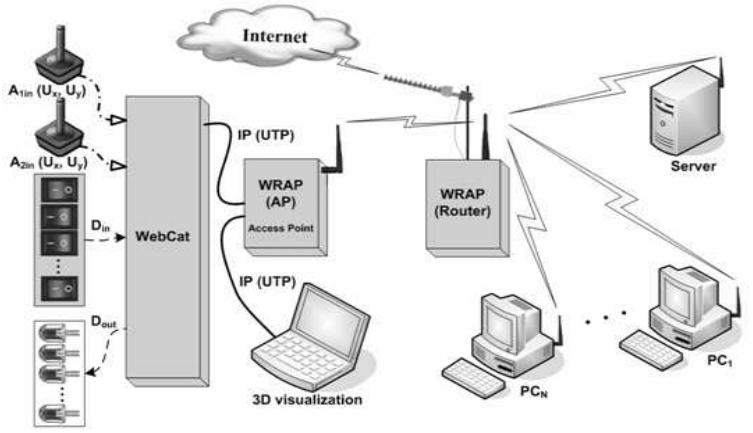


Fig. 3. LDM module - a global scheme

6.2 Case study 2 - RoboArm

Web Laboratory for Robotics - RoboWebLab [16], is a typical example of using service oriented concept for interacting ”legacy” embedded devices. The module of RoboArm [17] is in charge of interacting with mechanical arm with five degrees of freedom: Base - Right / Left 350 degrees, Shoulder 120 degrees, Elbow 135 degrees, Wrist rotate - CW & CCW 340 degrees, and Gripper - Open & Close 50 mm. Sensors from PC mouse have been added to the original mechanical arm, in order to enable gathering of information about mentioned movements. RoboWebLab components are shown on Fig. 4.

The base function of RoboArm module is to enable *semantic* control, local or via Internet, with the mechanical arm as embedded device. This goal is achieved by integrating Web service as an abstraction layer between hardware and low level binary controls on one hand, and special purpose XML meta-language on the other. The main software components are:

- Client application which correspondents to presentation layer in N-layers architecture is responsible for user authorization, authentication and enables uploading of high level ”semantic” command file. In the concrete realization Internet Explorer and ASP ArmAsp.aspx are used. Interface with the Web service is realized by adding reference to the Web service proxy class.
- Web service has the integrating - middleware role between client and application logic. Methods in Web service are responsible for two main middleware functions: validating uploaded XML documents and translating ”semantic” commands from logical level (XML document) to a corresponding bit stream

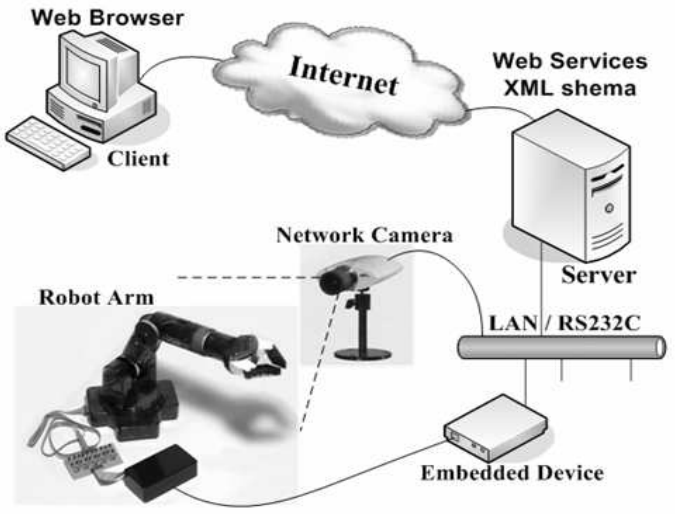


Fig. 4. RoboWebLab components

for direct interaction with hardware. The code snip for validating all semantic commands, using a corresponding XML scheme, is shown in Fig. 5. As

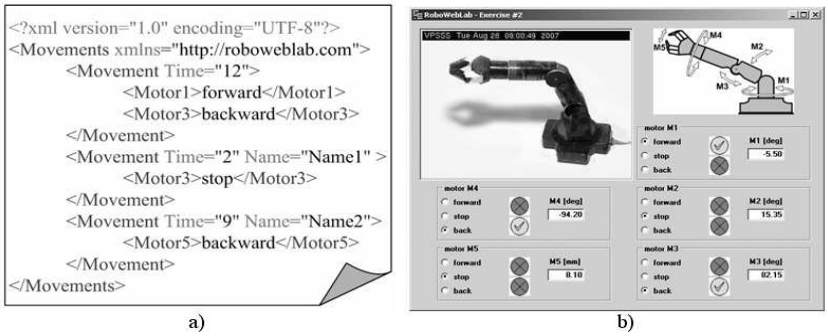
```
[WebMethod]
public bool IsValid(string strXmlFilePath){
      // Shema path
      m_strXmlShamaPath = Server.MapPath(".");
      m_strXmlShamaPath += "\\Files.Sheme\\CommandShema.xsd";
      // Define method for validation cheking
      XmlReader reader = null;
      XmlReaderSettings settings = new XmlReaderSettings();
      settings.ValidationEventHandler +=new
      ValidationEventHandler(this.ValidationEventHandler);
      // Define validation type ans shema
      settings.ValidationType = ValidationType.Shema;
      settings.Shemas.Add(null, XmlReader.Create(m_strXmlShemaPath));
      reader = XmlReader.Create(strXmlFilePath, settings);
      // Validation
      m_bRez = true;
      while (reader.Read()) { }
      return m_bRez;
}
private void ValidationEventHandler(object sender,
ValidationEventArgs args) {
      m_bRez = false;
}
```

Fig. 5. Code for IsValid Web Method.

we can see, Web service has dual role: as a programming abstraction and as an infrastructure for integration. As a programming abstraction Web service hides low level details of hardware and binary command set, which enable semantic networking even with ”dumb” or ”legacy” devices. As an integration infrastructure web service provides a comprehensive platform for developing and running complex distributed systems in a heterogeneous environment.

- Embedded software for immediate hardware control by using low level commands. Microcontroller (SAB 80C535) receives low level binary commands and takes appropriate actions, e.g. reads sensors or drives corresponding motors.

Both open-loop and close-loop control exercises are supported in RoboWebLab. In the open-loop exercise, software module receives an XML command file from a client and returns data about movement direction and reached position. Software for close-loop control receives information about desired position, realizes that request by close-loop, and returns data about performed task. For above mentioned exercises appropriate clients applications, RoboWebLab - Exercise #2 and RoboWebLab - Exercise #3 have been developed. RoboWebLab - Exercise #1 is reserved for developing ”semantic” commands i.e. appropriate XML files. An example of a “semantic” XML command file and user interfaces for RoboWebLab - Exercise #2 are shown on Fig. 6.



a) b)

Fig. 6. RoboWebLab - Exercise #2: a) XML command file, b) User interface

7 Conclusion

The convergence between computing and networking, fuelled by advances in semiconductor and transmission technologies, is revolutionizing the way communications are organized in the domain of embedded devices.

The presented framework has been developed to ease development, integration, maintenance and life cycle management of embedded systems in heterogeneous environment. In order to provide vendor, platform and programming language independent and flexible framework, XML and Web services have been used as new service oriented middleware. Migration from present day architectures to an entirely different architecture cannot happen instantly. Given the large installed base of industrial device networks, coexistence of existing and new technologies, the migration paths must allow gradual replacement. Two ways to achieve such integration have been presented in this paper. For modern, embedded devices, with powerful microprocessor unit and enough memory space, service orientation can be applied at the device level, while ”legacy” embedded systems use PC computer as gateway for appropriate embedded device type. Due to using standardized Web service technology, software components developed for presented framework can be used directly, or with small adaptations, with other service oriented frameworks like MSRS - Microsoft Robotic Studio [18].

Implementation of second generation of WS-* specifications [19], WS - Security, through appropriate SOAP headers is the main path of further framework improvements. Also using AJAX [20] technology in the client layer can rapidly increase potentials of presented framework. Developing new, user friendly and efficient methods and models for integrating embedded devices in heterogeneous distributed environment, is constant authors orientation.

References

[1] N. Gross, “The Earth Will Don an Electronic Skin,” *Business Week*, 1999. [Online]. Available: http://www.businessweek.com/1999/99_35/b3644024.htm

[2] K. Geihs, “Middleware Challenges Ahead,” *IEEE Computer*, vol. 16, pp. 24–31, 2001. [Online]. Available: http://www-di.inf.puc-rio.br/˜rcerq//semGSD/sugestoes/r6024.pdf

[3] C. Mascolo, “D5.1 Survey of Middleware for Networked Embedded Systems,” *Project Reconfigurable Ubiquitous Networked Embedded Systems*, vol. IST 004536 RUNES, 2005. [Online]. Available: http://www.ist-runes.org/docs/deliverables/D5_01.pdf

[4] F. Jammes and H. Smit, “Service-Oriented Architectures for Devices - the SIRENA View,” *Science & Technology, Schneider Electric*, 2006. [Online]. Available: http://www.sirena-itea.org/Sirena/Documents/All/SortedDocumentsList.htm

[5] *OSGi Service Platform Release 4 CORE*, OSGi Alliance, 2005. [Online]. Available: http://www.osgi.org/

[6] J. Teirikangas, *HAVi: Home Audio Video Interoperability*, Helsinki University of Technology, 2001. [Online]. Available: http://www.havi.org/home.html

[7] *Jini Architecture Specification Version 1.2*, Sun Microsystems, 2001. [Online]. Available: http://www.sun.com/software/jini/

[8] *UPnP Device Architecture v.1.0.1*, UPnP Forum, 2003. [Online]. Available: http://www.upnp.org/

[9] *Web Services Architecture*, W3C, 2007. [Online]. Available: http://www.w3.org /TR/ws-arch/

[10] *A Technical Introduction to the Devices Profile for Web Services*, Microsoft, 2004. [Online]. Available: http://msdn2.microsoft.com/en-us/library/ms996400.aspx

[11] H. Bohn, A. Bobrk, and F. Golatoeski, *SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains*, Institute for ME and CE, University of Rostock, 18051 Rostock Germany, 2006. [Online]. Available: http://www.sirena-itea.org/Sirena/Documents/All/SortedDocumentsList.htm

[12] T. Erl, "Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services," *New Jersey: Prentice Hall PTR*, 2004.

[13] B. Jevtović, D. Oklobdžija, and J. Ristić, "Remote Control Using an XML Documents for Data Exchange via Internet," *XIII Exhibition and Conference YU INFO 2007*, 2007.

[14] B. Jevtović and M. Mataušek, "Modeling of Bucket Wheel Excavator: Theory and Experimental Results," *Facta Universitatis, Ser.: Working and Living Environmental Protection*, vol. 2, no. 4, pp. 335–343, 2004.

[15] ——, "A Predictive-Adaptive Hierarchical Control System of Bucket-Wheel Excavator: Theory and Experimental Results," *Facta Universitatis, Ser.: Electronics and Energetics*, vol. 18, no. 3, pp. 493–503, 2005. [Online]. Available: http://factaee.elfak.ni.ac.yu/fu2k53/jevtovic.pdf

[16] B. Jevtović, D. Oklobdžija, and J. Oklobdžija, "Service Oriented Middleware Applied to the Web Laboratory for Robotics - RoboWebLab," *Internacional Scientific Conference UNITECH 07*, vol. 1, Ref s5-p138, pp. 432–437, 2007.

[17] B. Jevtović, D. Oklobdžija, J. Ristic, and J. Oklobdžija, "Web Services as Middleware for Interacting with Embedded Devices," *VI. Exhibition and Conference INFOTEH JAHORINA 2007*, vol. 6, Ref E-I-7, pp. 290–294, 2007.

[18] *Microsoft Robotics Studio*, Microsoft, 2007. [Online]. Available: http://msdn2.microsoft.com/en-us/robotics/default.aspx

[19] *Acknowledged Member Submissions to W3C*, W3C, 2007. [Online]. Available: http://www.w3.org/SUBMISSION/

[20] *AJAX Tutorial*, W3C, 2007. [Online]. Available: http://www.w3schools.com/ajax /default.asp