On the Use of XML for Data Access in Networks of Embedded Devices

Mitko Shopov, Ivan Stankov, Nikolay Kakanakov, Grisha Spasov
Technical University of Sofia, branch Plovdiv, Plovdiv, Bulgaria

***Abstract**: The paper deals with embedded networking and distributed embedded systems, which is a key area of research over the last years. It presents a custom protocol for data access in embedded devices – CNDEP (Controller Network Data Extracting Protocol). The presented work is based on the presumption that all data exchanged should be encoded in universal, platform independent format – XML. The XML-based version of the protocol CNDEP should provide interoperation between embedded devices, considered as components. The underlying communication is UDP/IP/Ethernet, which ensures good integration of embedded devices with other information systems. The protocol CNDEP is designed as a part of Multi-tier Client/Server System for Distributed Measurement and Control. It is used for data transfer in data producing tier. The transfer delays for the protocol are measured and compared to the non-XML implementation for evaluation of the effectiveness and efficiency of the realization.*

***Keywords:** Embedded Networking, Distributed Embedded Systems, XML, UDP.*

1. INTRODUCTION

Advances in IT technologies, have resulted in deployment of a wide range of products and applications with computing and network communication capabilities, other than the casual PCs. Devices with small size, such as mobile phones, PDAs, embedded controllers, etc. are continuously introduced in everyday life, offering new services to the user. Moreover, not detached have left embedded systems, which size is becoming smaller and smaller, whilst their computing capabilities are growing and in such way do the capabilities of the networks supporting their communication feature. This enables the programmers of interconnected embedded devices to build more powerful distributed applications.

The discussion in the current paper is about using XML standard for interaction over embedded systems. The use of XML is currently required because of service oriented character of discussed deployment. Data description language is used for data messages exchange as well as for controlling functionality of system realization. Ubiquitous usage of XML forces the programmer to create lightweight XML parsers, which to be used by the embedded systems on various platforms. Thus, having the XML scheme the parser could extract the data and exchange it to the application using it as quick as possible saving most of embedded systems resources. In the world of embedded systems everything is small and optimized for saving computing power and limited memory resources. Everything is working in acceptable speed, but on count of performance and functionality – traditional acceptance. Increasing the embedded systems capabilities and performance recently, has led applications to stronger solutions using more and more resources. In service oriented architecture the final data is likely to be produced from many sources that left hidden from end consumer. In embedded world aforementioned feature are decisive for current usage of XML standard in communication realization.

2. LIGHTWEIGHT XML PROCESSING

In the world of XML many approaches are tested and many parsers are designed and created [13] that include many features like: speed and size optimizations (Aelfred); small memory footprint and an intuitive operation (Electric XML); non-validating parser (ESPX); an event based and a tree based parsing (eXML); parsing external Document Type Definition (eXpart), implemented in C, sets a high standard for reliability, robustness and correctness. FastXML is a prototype of component software for processing XML and Extensible Stylesheet Language Transformations (XSLT) that is compatible with MSXML3 and optimized for speed. FXP: functional XML parser is a validating XML parser written in Standard ML. HXML – a Haskell-based non-validating XML parser that employs lazy evaluation, where an evaluation is not performed unless and until its result is needed, to keep memory requirements low. IBM's XML for Java EA2 is a Java-based validating parser that supports SAX 2. Maximum Performance and Minimum Size XML Parser (MXP1) is a Java-based non-validating pull parser that implements the Common Application Programming Interface (API) for XML Pull Parsing specification. NanoXML is a very small (5KB) XML parser for Java. Serving XML provides a language for building XML pipelines, and an extendible Java framework for defining the elements of the language. Xerces C++ is a validating parser written in a portable subset of C++ that conforms to DOM 1.0 and SAX 1.0. Additionally, early implementations are included for DOM Level 2 and SAX version 2.0. Xerces-J is the Java open source parser made available by the Apache XML Project. XML Engine contains REALbasic classes offering over 100 methods for parsing, rendering, analyzing, creating and altering XML. XMLBooster analyzes incoming XML messages, and then generates a custom XML parser for the programming language of the host environment. C, C++, COBOL, Delphi and Java are currently supported. XMLtp is a tiny XML parser/processor written in Java. It is meant for use where a small footprint XML processor is needed by applications that store their internal data in XML format. XT is a Java-based XSLT implementation that uses the XP parser. XT can be used with other SAX-compliant parsers. MinML (a minimal XML parser) is a robust parser for a useful subset of XML which takes less than 8Kb of code space on an embedded system and can parse substantial XML documents using less than 64Kb of heap space. MinML supports the SAX1 interface which means that it can be used in a large range of existing Java programs. SAX1 interface allows MinML aware programs to minimize the storage costs of processing the character data in an XML document [12].

3. THE USE OF XML FOR DATA ACCESS IN NETWORK OF EMBEDDED DEVICES

3.1. Controller Networks Data Extracting Protocol (CNDEP)

CNDEP is an asymmetric protocol used in Client/Server systems to "extract" data from microcontrollers working in a LAN. It works on the application layer of the TCP/IP stack and uses UDP as a transport protocol. The client sends a short request to the server and expects a short reply back. Current implementation of the protocol is supposed to work in Local Area Network and to be integrated in multi-tier systems for home and office automation. Benefits of using UDP are: short message exchange; fast communication; good reusability of communication channel; service of devices in sensible time [3, 4].

3.2. XML-based version of CNDEP (xCNDEP)

The XML-based version of the CNDEP protocol is called xCNDEP. The protocol xCNDEP provides an XML schema for defining the observational characteristics of a measurement controller. A measurement controller is a controller with attached sensors and, or actuators for interacting with the environment. There are a great variety of possible configurations for the measurement controller. The main purpose of xCNDEP is to provide: general information about controllers in support to data discovery; support the processing and analyses of measured data; quality characteristics (e.g. accuracy, precision); information about physical properties measured (e.g. temperature, humidity, etc.); and response characteristics (e.g. temporal response, metrics, etc.).

Because xCNDEP is supposed to provide the ability to integrate disparate types of measurement controllers, with different hardware resources, it should be flexible and adaptable. The exchanged data will be with different levels of details depending on the capabilities of the measurement controller – a light-weighted version for controllers with limited resources or minimal time requirements and on the opposite side a full version for extremely capable controllers.

4. EVALUATION

As xCNDEP aims to provide a universal representation of exchanged measurement data it is important to test the implementation of the protocol on the embedded device (the possible bottleneck) and to ensure it is scalable and performing reasonable. This includes an evaluation of characteristics like packet's size, packet distribution, and protocol delay. The experimental data collected during the test-bed will be used for addressing the leaks in the implementation and estimation of the delay the protocol introduces in the communication.

4.1. Test-bed architecture

A test-bed experiments configuration typically consists of experimental, monitoring and simulation-stimulation subsystems. The experimental subsystem is the part which characteristics should be fetched. Thus, the simulation-stimulation subsystem provides parameterized inputs to the experimental subsystem and the monitoring subsystem collects the outputs [2, 8].

For executing the experiments an experimental network is built. It consists of several embedded devices with temperature and humidity sensors and a monitoring station – desktop PC, interconnected in switched LAN. The architecture of the network for the test-bed is presented on figure 1.

The configuration of the system-under-test for the test-bed experiments is shown on figure 2. It is built to address the main requirements of the test-bed experiments, including: network delay times, TCP/IP stack delay times and xCNDEP delay times. Experimental data is collected on a data capturing computer.

The data capturing computer acts as an xCNDEP client. It executes requests to controllers under test and collects their responses. Thus, it provides platform for simulation-stimulation subsystem and monitoring subsystem. A custom build software tool is used for these purposes. This tool allows setting different experimental scenarios. The collected information from the experiments is stored in XML files. The time intervals are measured using "HighPerformanceTimer", provided by the Windows framework. It has accuracy of about 0.001ms.

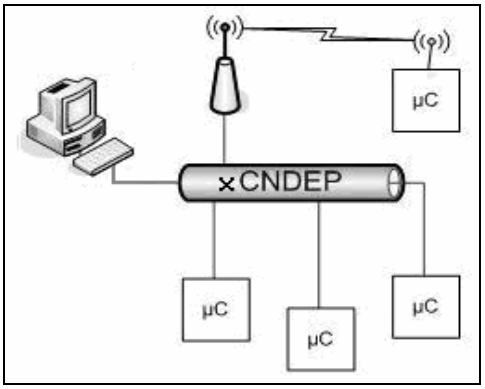

Fig. 1: Experimental network.

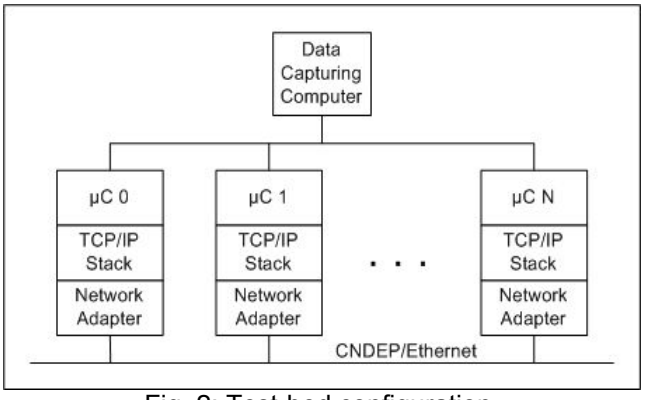

Fig. 2: Test-bed configuration.

Target embedded platforms included in the experiments are DS TINI from Dallas Semiconductors [10] and IPC@Chip SC12 from Beck IPC [9]. On them runs the server side of xCNDEP. For sensing the environment SHT71 intelligent sensor [11], capable of measure temperature and humidity, is used. The DS TINI platform uses Java virtual machine for running applications [7] and the IPC@Chip runs custom RTOS and is programmed in C [1]. In this way the calculated delay times for the protocol are dependable on the platform architecture. Thus, the results can be used for evaluation of the platform themselves.

4.2. Results

For the evaluation of the protocol a series of experiments are carried out. Three commands are used: 'Test', 'Get Temp', and 'Get Hum'. Controller responds immediately upon receiving a 'Test' command. On 'Get Temp' and 'Get Hum' commands the controller first contact the sensor for measured temperature and humidity respectively, before it sends back a response. This adds an additional latency to the command execution delay. All experiments are executed 101 times in three minutes intervals, for providing good statistical results for approximation [3, 8]. The results are presented in table 1.

Tab. 1: Experimental results for two controllers executing three types of commands.

Controller	Command	Request size [byte]	Response size [byte]	AVG Delay [ms]	σ [ms]
IPC@CHIP	Test	234	200	12,847	0,121
	Get Temp	241	286	103,025	0,130
	Get Hum	238	286	141,691	0,139
DS TINI	Test	234	200	132,666	0,342
	Get Temp	241	286	103,025	0,130
	Get Hum	238	286	141,691	0,139

The size of request and response are the sizes of the CNDEP commands. They should not be mixed up with the UDP packets' size. The last may be even equal for both versions of CNDEP because of the minimum required size for UDP packets and possible extra padding appended.

Comparing these results with the results from [5] of the evaluation of non-XML version of CNDEP protocol leads to the graphical representation given on figure 4 for IPC@Chip controller and on figure 5 – for DS TINI controller.

For the IPC@Chip embedded platform, the results show that there is almost no dependence between the time for processing a command by the controller and the size

of the command. This can be explained with features of the platform. Application developed in ANSI C, RTOS, effective I/O operations, optimized string operations. This makes IPC@Chip and similar controllers a potentially good solution for adoption of XML version of CNDEP (Figure 4).

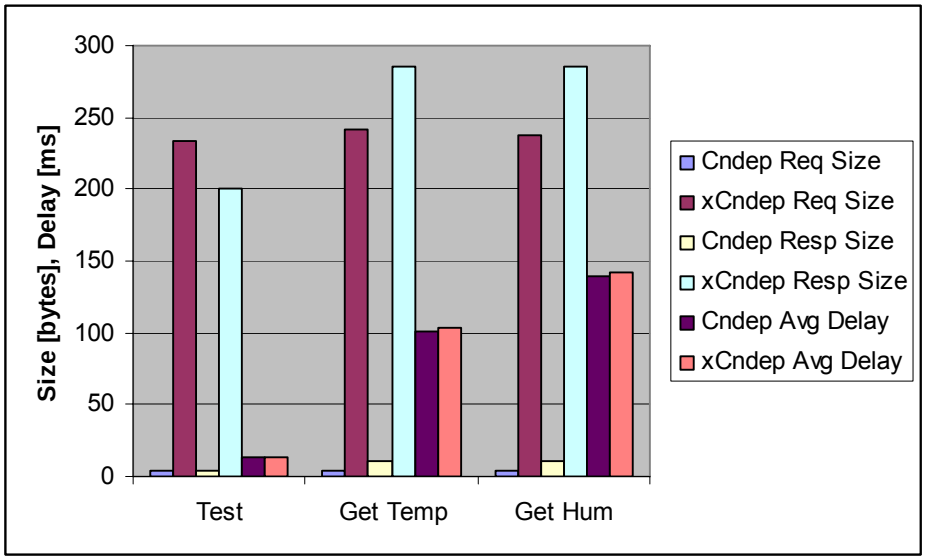

Fig. 4: CNDEP vs. xCNDEP – IPC@Chip controller.

In contrast with IPC@Chip, the DS TINI embedded platform shows less effective performance. There is a notable disruption in the work of the controller, expressed with increasing the time for processing of commands. One possible explanation of these results could be the special features of the platform. The application runs in the Java virtual machine (JVM) environment and every time a request is received a context switching between JVM and Tini native interface is required, which is a time consuming operation (Figure 5).

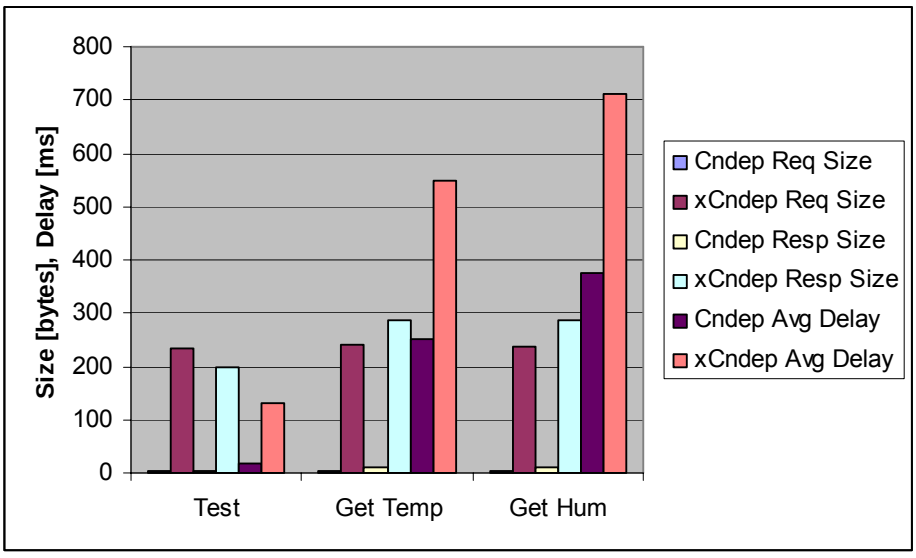

Fig. 5: CNDEP vs. xCNDEP – DS TINI controller.

5. CONCLUSIONS AND FUTURE WORK

XML standards are already widely adopted in high level business and industrial applications. However, its uses for data access in networks of embedded devices is still

expanding and gaining rapid support. The XML-based version of the CNDEP protocol has been used for example how the use of an XML will affect the performance of embedded devices. The protocol CNDEP is designed as a part of Multi-tier Client/Server System for Distributed Measurement and Control. It is used for data transfer in data producing tier [6]. By introducing XML an increased interoperability and data portability is achieved. Also, since the XML is one of the technical foundations for the Web services, the evaluation of xCNDEP protocol shows the possibilities for future deployment of Web services directly on embedded devices.

The evaluation of xCNDEP on two different platforms shows contradictory results. The IPC@Chip platform shows no significant increase in the processing time. Future work for it includes how the increased size of XML encoded messages influence the memory footprint. The reasons for the increased processing time on DS Tini platform should be further analyzed. Future work includes adding new features and optimizations to the xCNDEP implementation to increase its performance.

6. ACKNOWLEDGMENTS

The presented work is supported by National Science Fund of Bulgaria project – "ВУ-966/2005", entitled "Web Services and Data Integration in Distributed Automation and Information Systems in Internet Environment", under the contract "ВУ-МИ-108/2005".

7. REFERENCES

[1] Barr, M. (1999), *Programming embedded systems in C and C++*, O'Reilly, ISBN: 1-56592-354-5.

[2] Fortier, P., G. Desrochers (1990), *Modeling and Analysis of Local Area Networks*, CRC Press.

[3] Holzmann, G. (1991), *Design and Validation of Computer Protocols*, Prentice Hall, ISBN: 0-13-539925-4.

[4] Kakanakov, N., I. Stankov, M. Shopov, and G. Spasov (2006), *Controller Network Data Extracting Protocol – design and implementation*, Proc. CompSysTech'06, Veliko Tarnovo, pp. IIIA-14.1 – IIIA-14.6.

[5] Kakanakov, N., M. Shopov (2002), *Evaluating Controller Network Data Extracting Protocol for Embedded Devices,* Proc. Electronics'06, Sozopol, (in press).

[6] Kakanakov, N., M. Shopov, G. Spasov (2006), *A New Web-based Multi-tier Model for Distributed Automation Systems*, Information Technology and Control (J.), (in press).

[7] Loomis, D., (2001), *TINI(TM) Specification and Developer's Guide*, Addison-Wesley Professional, ISBN: 0-20-12218-6.

[8] Stallings, W. (2002), *High-Speed Networks and Internets*, 2nd Ed, Prentice Hall, ISBN: 0-13-032221-0.

[9] http://www.beck-ipc.com/ipc/ - IPC@Chip Homepage [July 2006].

[10] http://www.maxim-ic.com/products/tini/ - DS TINI Homepage [July 2006].

[11] http://www.sensirion.com/, SHT71 Homepage, [July 2006].

[12] http://www.wilson.co.uk/embedded/emb1.htm

[13] http://www.xml.com/pub/rg/XML_Parsers