Using XML Web Services for Embedded Systems Interoperability. World's Smallest Web 2.0 Server Demo.

Risto Serg
Tallinn University of Technology
risto.serg@dcc.ttu.ee

Johannes Helander
Microsoft Research
Redmond
jvh@microsoft.com

Abstract

If the embedded sensors and devices could directly work together and with other computing devices, they would add value to each other, and enable new consumer applications. Present requirements for cyber-physical systems are usually too high for implementing them on single, non-networked units. Using service oriented architecture is one of the solutions to achieve interoperability and possible future scaling of the system. In our demo we show that a limited subset of XML Web Service protocols can be implemented in very limited environment. Surprisingly we found out that limited XML Web Services implementation introduces only minimal overhead.

1 Introduction

Most embedded systems are seldom used alone. Systems that communicate between each other are much more common in real world ubiquitous computing applications, such as the car comfort equipment, sensor and actuator networks, smart houses etc.

Service oriented systems architecture is gaining popularity among system designers as it can help to reduce development and maintenance costs. Communication between various system components can be arranged using either some custom, possibly optimized and application specific protocols or applying some well defined protocol for example based on XML Web Services [6]. One could think that implementing Web Services on small embedded systems causes huge unnecessary overhead. This can be true in case of implementing the whole SOAP protocol support, including the WS-*

specifications, but this is not necessary in most cases. A lot of small embedded systems provide just a very limited set of services and therefore an approach where only a subset of the protocols, explicitly needed by the application, is implemented in the device. Using a few specific techniques we show that XML Web Services can be implemented on a tiny platform – an 8-bit AVR microcontroller with minimal overhead. The sample application provides two sample services – addition and subtraction using less than 128 bytes on RAM and less than 4 kilobytes of ROM. The XML messages the system receives as service requests and sends out as replies are all several hundreds of bytes long. The whole system doesn't need any hardware or software interrupts to work, making it possible to target even simpler hardware platforms.

2 Techniques used

2.1 Minimal real-time scheduling and off-line analysis

One of the techniques used in the system design separates functional and temporal behaviors of the system allowing us to analyze both aspects separately [6][7]. The functional part is developed using the C language and object oriented methodology. The temporal behavior is described in a separate "program" utilizing XML syntax. Temporal behavior can be analyzed off-line with corresponding tools that can suggest optimal schedules (see partiture in [6]) [9][8][1]. In the extreme case all the scheduling structures will be constants and can be saved into ROM memory. On small microcontrollers ROM is often a more plentiful and cheaper resource than RAM.

2.2 Futures

The concept of futures was originally introduced in MultiLisp [4]. Futures expressed possible concurrency in programs by lazy evaluation – results of the function call aren't calculated immediately, but instead when the result is needed [2][3]. In our demo application we use a minimalistic XML message parser, which assigns the target service (function, task) into a predefined scheduling slot. The actual service function is executed later, followed by reply message generation. Some details about theoretical aspects of scheduling futures can be found in related works [5].

3 Demo application – Worlds smallest Web2.0 server

Usage of the techniques explained above helped us to develop a system, which we believe is the world's smallest web server, considering required memory footprint. The system is implemented on an 8-bit AVR microcontroller board – the AVR Butterfly. The Atmel AVR family microcontroller AtMega169 used in our project has a total 16 kibibytes of FLASH ROM and 1024 bytes of RAM. Due to the constraints set by memory sizes it's advisable to use ROM instead of RAM where possible.

The sample application

1. listens for XML messages currently on serial line
2. parses the message
3. in case of match includes requested service code into the schedule
4. executes scheduled code
5. sends back XML message containing result of the service requested and

6. displays the arguments and result on board LCD screen

Breakdown of the memory requirements for the partly optimized code are presented in Table 1:

RAM static	RAM stack	RAM total	ROM
33 bytes	53 bytes	< 128 bytes	3982 bytes

Table 1: Memory requirements of the demo application

The static RAM includes all system and application variables but not the stack. Runtime usage of stack is 53 bytes and could be decreased some by further optimization of the code. Total RAM usage of the system including stack is below 128 bytes. Messages that have to be parsed for detecting service requests and replies are all several hundred bytes long. Total ROM usage is 3982 bytes including LCD and serial line drivers and bootloader.

A photograph of the demo device – an Atmel AVR Butterfly board is presented in Figure 1. The device is battery powered and it can be seen that the LCD screen shows the arguments and the result of the last service request – in this case subtraction 49 - 7 = 42.

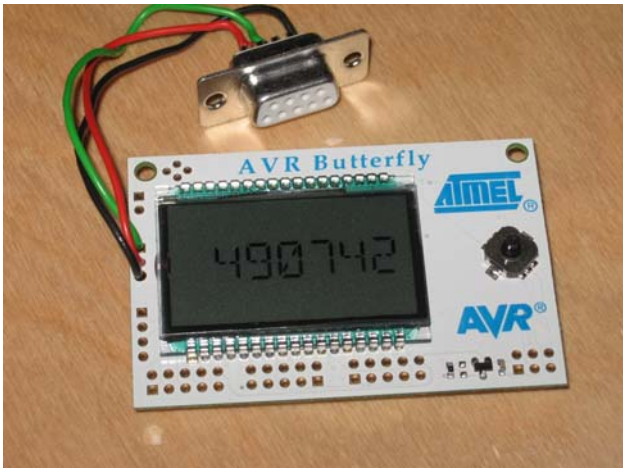

Figure 1: Demo device

4 Conclusion

Using XML Web Services as the communication layer for embedded systems facilitates standardization of communication and helps increase interoperability between different types of communicating systems. According to Metcalfe's law the value of the system is larger than the sum of its components. Limited scale SOAP implementation introduces only insignificant overhead and simplifies several ubiquitous computing scenarios. Memory constraints can often be dealt with by using offline analysis tools and fixing code and data as much as possible so as to put it into the ROM memory area.

5 References

[1] Farcas, E., Farcas, C., Pree, W., and Templ J., Transparent distribution of real-time components based on logical execution time, SIGPLAN Notices, Volume 40 Issue 7, pp. 31-39, 2005.

[2] Forin, A. (1990) “Futures” in book “Topics in Advanced Language Implementation” Peter Lee ed., MIT Press, Cambridge MA. ISBN: 0-262-12151-4

[3] Friedman, D., and Wise, D., Technical Report TR44: CONS should not Evaluate its Arguments (Jan 1976), 26 pages plus appendix [I. S. Michaelson and R. Milner (eds.), Automata, Languages and Programming, Edinburgh University Press, Edinburgh (1976), 256—284]

[4] Halstead, R., Multilisp: A Language for Concurrent Symbolic Computation ACM Transactions on Programming Languages and Systems, Vol 7, No. 4, October 1985

[5] Helander J. et al. Adapting Futures: Scalability for Real-World Computing, Proceedings of the Real-Time Systems Symposium, 2007.Tucson AZ, USA, December 2007, pp.105-118

[6] Helander J., Deeply Embedded XML Communication: Towards an Interoperable and Seamless World, Proceedings of the 5th ACM international conference on Embedded software, Jersey City, NJ, September 2005.

[7] Henzinger, T., Kirsch, C., and Matic, S., Schedule carrying code. In Proceedings of EMSOFT, Philadelphia, USA, October 2003. LNCS 2855, pp. 241--256, Springer, 2003.

[8] Liu, C., Layland, J., Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, Journal of the ACM, New York, USA, 1973.

[9] Mõtus, L. and Rodd, M. “Timing Analysis of Real-time Software”, Elsevier Science/Pergamon, 1994, 212pp