Lessons Learned Implementing the ISO/IEEE11073 Standard into Wearable Personal Devices

Miguel Martínez-Espronceda, *Student Member, IEEE,* Ignacio Martínez, Luis Serrano, *Senior Member, IEEE,* Santiago Led, Jesús Daniel Trigo, Asier Marzo, Javier Escayola, Gilberto Barrón and José García*, Member, IEEE*

***Abstract*— In this paper several approaches to implement the ISO/IEEE11073 interoperability standard into personal health devices using low-voltage low-power (LV-LP) microcontrollers are proposed. The patterns methodology previously suggested by the authors is followed in all of the approaches which include an 1-process implementation using non-blocking functions, multiprocess within an operating system, an specific multithread framework, and a code compression based interpreter. Finally, a qualitative comparison between the different approaches is provided.**

I. INTRODUCTION

The fast development of Information and Communication Technologies applied to the Health System is fostering the appearing of new technological solutions and services centred in the patient. The so-called patient empowerment [1] enables the patient to manage its health and, furthermore, improves its quality of life. Indeed, these solutions contribute to the reduction of waiting queues and consequently to the improvement of public health services. From the company point of view, these new services suppose a new market that can lead to a new economical growing motor [2]. Nevertheless these patient centred services should be implanted in a consensual and ordered way [3]. As a response some standards are being evolved or even created from scratch to cover the new scenarios. In the case of health information standards one important field of development in the last years is being the interoperability standard for medical device communications ISO/IEEE11073 (X73) [4], [5]. This standard has recently evolved from scenarios of Point of Care (PoC) and Intensive Care Unit (ICU) (X73PoC) [6] to Personal Health environments (X73PHD) [7]. A double aim must be overcome in this evolution: on one hand the incorporation of new wireless (such as Bluetooth [8] and ZigBee [9]) and wired technologies (such as USB); on the other hand the reduction of X73 complexity in order to decrease processor usage and memory needs. The resulting standard defines point to point communications between the so-called agents and managers. Agents, usually wearable, are the sensors, sources where measurements are taken. Managers corresponds to gateways or sinks that gather the measurements.

Due to the new in-home healthcare paradigm using low-voltage low-power (LV-LP) personal devices, it is common to have constrains in design of hardware and software. For example in most common System-on-Chip (SoC) devices used in ZigBee applications, the Random Access Memory (RAM) for the application, shared with the ZigBee stack, is less than a few Kilobytes. Depending on the X73 specialisation implemented, the size of incoming and outgoing Application Protocol Data Units (APDUs) can require as much as 63KB and 8KB, respectively. In these conditions it is hard for developers to implement X73 into embedded devices [10], [11]. In order to optimise the use of resources in agents, and having in mind to reduce power consumption, a methodology to implement X73-conforming agents was proposed and showed in [12]. Using it, the processing of X73 results very efficient in terms of memory and processor. As an example, the generation of an outgoing X73 APDU using some patterns and a few variables in RAM is shown in Fig. 1. A proof-of-concept of implementation was made in INTENSA project [13]. There are different approaches to implement the standard according to the patterns methodology. Each one of these alternatives can be better in one situations than in others depending on several factors such as memory space, cpu resources, type of Operating System (OS), communication technology stack, etc. The proposal of new implementation alternatives to the development of LV-LP agents is fundamental.

The paper is organised as follows. Section II introduces the modular architecture employed to, then, propose four implementation approaches which include a 1-process implementation using non-blocking functions, a multiprocess system within an OS, an specific multithread implementation, and a pseudocode-based environment. In Section III a discussion about other possible approaches as well as a qualitative comparison of Section II approaches are given. Conclusions are drawn in Section IV.

This research work has been partially supported by projects TIN-2009-08414 and TIN-2008-00933/TSI from Comisión Interministerial de Ciencia y Tecnología (CICYT) and European Regional Development Fund (ERDF), TSI-020302-2009-89 and TSI-020302-2009-7/Plan Avanza I+D from Ministerio de Industria, Turismo y Comercio, and FPI grant to M.Martínez-Espronceda (Res.1342/2006 Public University of Navarre).

M.Martínez-Espronceda, L. Serrano, S. Led, and A. Marzo are with the Electrical and Electronics Engineering Dept., Public University of Navarre (UPNA), Campus de Arrosadía, s/n 31006 Pamplona, Spain (corresponding e-mail: miguel.martinezdeespronceda@unavarra.es).

I. Martínez, J.D. Trigo, J. Escayola, and J. García are with the Communications Technologies Group (GTC), Aragón Institute of Engineering Research (I3A), University of Zaragoza (UZ), c/María de Luna, 1 50018 Zaragoza, Spain.

G. Barrón is with the Coordination of Electronic and Communication Engineering, Autonomy University of Nuevo León, University Avenue s/n, University City, San Nicolás de Los Garza, NL66450, Mexico.

978-1-4244-6561-3/10/$26.00 ©2010 IEEE

II. METHODOLOGY

The architecture follows the division of conceptual blocks proposed by X73PHD (see Fig. 2). It includes the transport layer, the exchange layer, and the application layer. The transport layer corresponds to the communication technology stack, such as Bluetooth (using the Health Device Profile – HDP), USB (using the Personal Healthcare Device Class – PHDC) or ZigBee (using the Health Care Profile – HCP). Internally the exchange layer is divided into the patterns library, which contains the content of the patterns to be used in a X73 communication, and the kernel X73 that contains the algorithms to generate the messages extracted from the patterns. The kernel X73 and the pattern library can be implemented in different ways. In the remaining of this section some of them are described.

A. 1-process implementation using non-blocking functions

In this solution the APDUs are passed from and to the transport layer in one piece to the exchange layer. The system is programmed using non-blocking functions and callbacks. APDUs are stored and passed between functions in RAM buffers. Depending on the device's configuration, the size of the transmission buffer for one APDU can reach as much as 63KB while the reception buffer can require as much as 8KB (using a specialisation, the maximum size appears in the specific X73-104xx document). When the exchange layer receives an X73 APDU, it processes the APDU using the patterns methodology. The exchange layer analyses the APDUs using patterns and returns the result to the application layer when the code encounters that the APDU matches a sequence of pattern with an valid APDU. At the same time, depending on the APDU received, the exchange layer changes the state of the Finite State Machine (FSM), informs to the application layer, and generates other APDU to transmit in response when it is appropriate. The response APDU generated in this way is buffered in the transmission queue of the transport layer. The entire process can be seen in Fig. 3.

Most of the X73 agents can be implemented following the ideas given here if they are featured with reasonable RAM

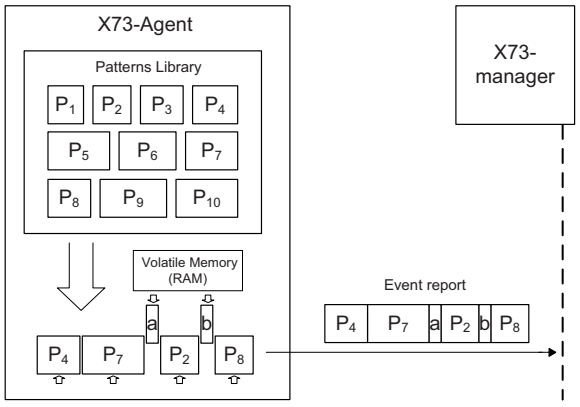

Fig. 1. Concept of pattern

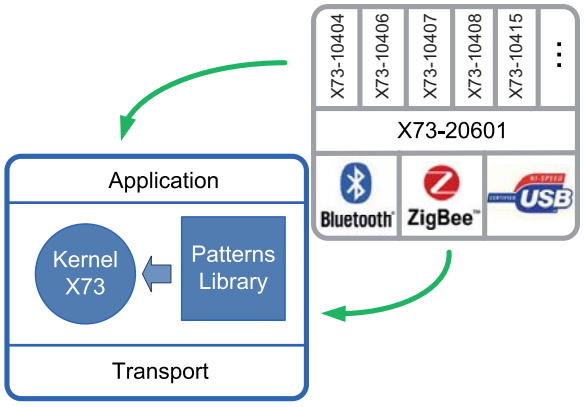

Fig. 2. Conceptual building blocks

capacity or if its functionality is very reduced due to the fact that in these cases the APDU sizes are limited. The limitation in buffer sizes has to be taken into account by the device designer as the possibility to run out of memory exists. This technique was used in [14].

B. Multiprocess within an OS

In this case, the exchange layer is divided into several processes. Each transmission channel has two processes: the writer, intended for formatting and transmission of APDUs, and the reader, intended for reception and matching of APDUs (see Fig. 4). Each time the reader process a complete APDU, it updates the state of the FSM, optionally sends a command to any of the writers (e.g. transmit a response to the received message), and generates an event to alert the application layer (e.g. sending a command like 'STATUS_OPERATING' when X73 association and configuration procedures have been concluded).

The advantage of this strategy with respect to the 1-process is that the incoming or outgoing APDUs are processed part

```
const byte_t pttrn_assoc_req[] = {
    0xE2 0x00  // APDU CHOICE Type (AarqApdu)
    0x00 0x32  // CHOICE.length = 50
    0x80 0x00 0x00 0x00  // assoc-version
    ...
}
const byte_t pttrn_assoc_rsp_0[] = {
    0x00, 0x2C              //   CHOICE.length = 44
}
...
matchAssocResp(APDU_buff * apdu) {
    x73_memcmp(apdu->buff,pttrn_assoc_rsp_0,
                sizeof(pttrn_assoc_rsp_0));
    ...
    x73_recv16(result);
    x73_ready(ST_RECV_ASSOC_RESP, result);
    return result;
}
```

Fig. 3. 1-process implementation

by part thus reducing the buffering and the needs of RAM memory (only a small buffer for transport layer needs). The drawbacks are the possible increase of the footprint of the overall program (because of the OS needs) and the power consumption (because possibly the OS has background activities). Depending on the implementation of the OS, these can be specially crucial in idle devices (e.g. weigh scale is used a few times a day and the rest of the time is slept).

The X73 transport specification imposes that changes due to an APDU must be transactional (it is an all or nothing). For this reason, readers and writers must be programmed in a specific way. In the case of readers, they must commit all the operations and changes in status variables when they have finished matching the complete APDU. In the case of writers, they must not carry out any operation that affects to other readers or writers.

C. Multithread implementation

The drawbacks of the previous solution (OS based one) are solved in the next strategy (see Fig. 5). It consist on a threaded application, with the peculiarity that the threading algorithm is optimum for implementing the X73PHD. This threading algorithm is implemented in a specific threading environment which contains basic code to enable basic services as interprocess communication as well as access to application and transport layers.

Traditionally two types of multitasking have been prevalent: preemptive multitasking (used in modern OS) and collaborative multitasking (used in batch-based systems). This strategy is based on collaborative multitasking where each task owns the processor until it decides to give it up to other tasks. This briefly consist on an abstraction layer that implements threading. This scheme is chosen because of two reasons: first one is that the overhead due to context switching is lower with this alternative; second one is that tasks require short time-slots and therefore there is no need for the OS to remove any task from the processor (indeed, it is expected that a task does not leave the processor until it has finished its work). Moreover, high responsiveness and preemption were not needed but low footprint and memory usage are desirable.

This approach uses the same separation of tasks as the approach based on OS. There are reader tasks, which match and process incoming APDUs from the channel, and writer tasks, which build APDUs and send them to the channel. The difference is that the scheduling algorithm is of non-preemptive round robin type and gives priority to the readers.

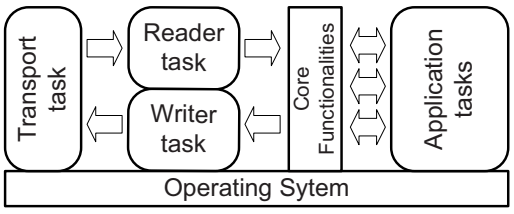

Fig. 4. Multiprocess within an Operating System

The threaded environment can be easily integrated in a real OS or with the transport stack facilities.

D. Pseudocode-based implementation

Some improvements can be done with respect to the footprint and memory usage. Previous proposed solutions make use of a general purpose language and, after compilation, executed in machine language. These strategies are very optimised in terms of processor efficiency. However, there is a way to reduce the footprint that the algorithm employs. The idea is to use a set of high level instructions (pseudocode) that represent blocking operations. Each blocking operation is encoded as a instruction in the pseudocode. In this way it is possible to save the status of the program written in pseudocode when it is executing a blocking instruction, and resume the program later, thus emulating multithreading. The architecture used is similar to the multithreads strategy. Again, readers play the role of matching and processing incoming messages while writers the role of building and sending outgoing messages. The tasks are implemented partially in pseudocode and executed by the pseudocode ad-hoc Virtual Machine (VM) (See Fig. 6). The VM, as it is very specialised, takes up a few lines of native code.

There is a tradeoff between size and speed; for that reason, some parts of the tasks are coded natively (that is compiled to machine code). In this way, the exchange layer is scattered between the pseudocode and the native code. The high level instructions codified in pseudocode include stream management (pattern and variable transmission as well as reception and matching of buffers), and FSM state management. The code implemented in a native way include loop encoding, jumps, math operations, access to variables, and calls to external functions, among others.

In order to reduce the programming complexity an assembler must be used to generate the pseudocode. As the pseudocode is specific of this implementation, the assembler shall be developed ad hoc (see Fig. 6).

III. DISCUSSION

A qualitative comparison between the different approaches proposed in this paper is given in Table I. The four ap-

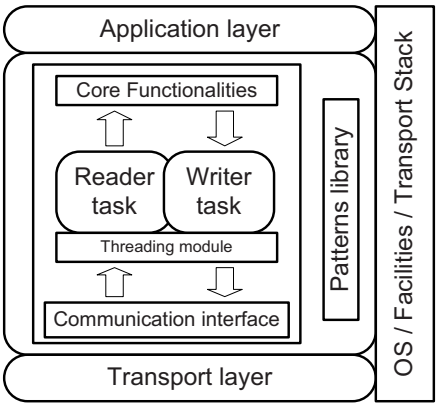

Fig. 5. Multithread implementation

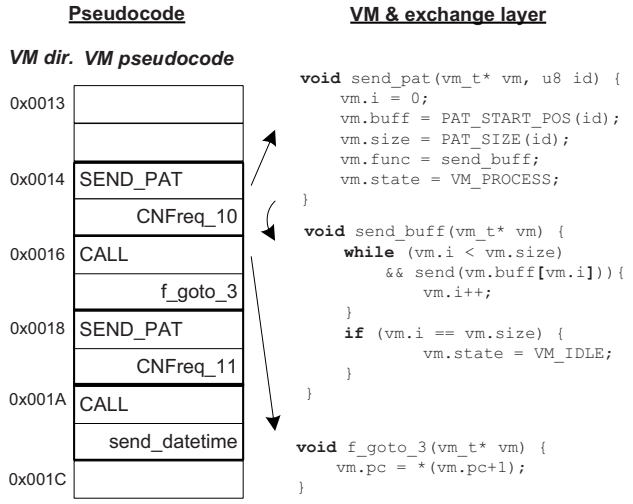

Fig. 6. Pseudocode-based implementation example

proaches appear in the same order that they have been exposed during the paper. In the table, more crosses means that the implementation consumes more resources (memory, processor, development time or power consumption). The values of this table are the result of some implementation experiences but, as the authors do not have quantitative results yet, the results have to be taken as illustrative.

Table I shows the differences between the possible implementations. As it is shown, the approach that requires less memory is the pseudocode-based approach. Nevertheless this approach is the most expensive with respect to development time. Processor usage and power consumption is inferior in both multithread and 1-process. This last approach, 1-process, is the easiest technique to develop. The most versatile between the four approaches seems to be the multithread. Some studies are being done by our group to reduce development time of this solution.

The alternatives exposed here have been chosen as the most feasible from a collection that includes both techniques that have been studied from projects that the group has gained access and individual proposals from the members of the group. Although these techniques cover most common applications, there can be other different approaches in special situations to deal with implementations in LV-LP microcontroller-based platforms that can be more suitable. Hence, each case must be studied thoroughly by the device developer.

TABLE I
COMPARISON BETWEEN THE DIFFERENT APPROACHES

Implementation technic	Memory needs	Processor needs	Time to develop	Power consumption
A. 1-process	++++	+	+	+
B. OS-based	+++	++	++	++
C. Multithread	++	+	+++	+
D. Pseudocode	+	++	++++	++

IV. CONCLUSION

Four different approaches to implement the X73 standard in LV-LP systems based on microcontrollers using the patterns methodology are presented in this paper. A detailed description of the different approaches has been given by showing the techniques used in every implementation. The discussion shows the differences between them in a qualitative way and remarks that there are compromises between memory needs, processor usage, development time and power consumption, which indicates that the suitableness of each one for a specific device depends on the features of this device.

REFERENCES

[1] J. L. Monteagudo and O. Moreno, "ehealth for patient empowerment in europe," World Wide Web electronic publication, 2009. [Online]. Available: http://ec.europa.eu/information_society/newsroom/cf/itemdetail.cfm?item_id=3448.Lastaccess:04/10

[2] R. Carroll, R. Cnossen, M. Schnell, and D. Simons, "Continua: An interoperable personal healthcare ecosystem," *IEEE Pervasive Computing*, vol. 6, no. 4, pp. 90–94, 2007.

[3] F. Wartena, J. Muskens, and L. Schmitt, "Continua: The impact of a personal telehealth ecosystem," in *Proceedings - International Conference on eHealth, Telemedicine, and Social Medicine, eTELEMED*, 2009, pp. 13–18.

[4] M. Galarraga, L. Serrano, I. Martinez, P. D. Toledo, and M. Reynolds, "Telemonitoring systems interoperability challenge: An updated review of the applicability of iso/ieee 11073 standards for interoperability in telemonitoring," in *Annual Int Conf IEEE EMB - Proceedings*, 2007, pp. 6161–6165.

[5] M. Clarke, D. Bogia, K. Hassing, L. Steubesand, T. Chan, and D. Ayyagari, "Developing a standard for personal health devices based on 11073," in *Annual Int Conf IEEE Eng in Medicine and Biology (EMBC) - Proceedings*, 2007, pp. 6174–6176.

[6] N. Lutter, T. Norgall, J. Mell, C. Weigan, and J. Schuettler, "Point of care: New connectivity standards and novel technologies in intensive care," *International Journal of Intensive Care*, vol. 12, no. 4, pp. 175–185, 2005.

[7] L. Schmitt, T. Falck, F. Wartena, and D. Simons, "Novel iso/ieee 11073 standards for personal telehealth systems interoperability," in *2007 Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability - HCMDSS-MD PnP '07*. IEEE, 25-27 June 2007 2008, pp. 146–8.

[8] W. S. Wang, "Bluetooth: A new era of connectivity," *IEEE Microwave Magazine*, vol. 3, no. 3, pp. 38–42, 2002.

[9] P. Kinney, "Zigbee technology: Wireless control that simply works," in *Communications Design Conference*, 2003.

[10] S. Warren, J. Lebak, and J. Yao, "Lessons learned from applying interoperability and information exchange standards to a wearable point-of-care system," in *1st Transdisciplinary Conference on Distributed Diagnosis and Home Healthcare*, vol. 2006, 2006, pp. 101–104.

[11] M. Martínez-Espronceda, L. Serrano, I. Martínez, J. Escayola, S. Led, J. Trigo, and J. García, "Implementing iso/ieee 11073: Proposal of two different strategic approaches," in *Annual Int Conf IEEE Eng in Medicine and Biology (EMBC) - Proceedings*, 2008, pp. 1805–1808.

[12] M. Martínez-Espronceda, I. Martínez, J. Escayola, L. Serrano, J. Trigo, S. Led, and J. García, *Standard-Based Homecare Challenge: Advances of ISO/IEEE11073 for u-Health*. Handbook of Digital Homecare, 2009, pp. 179–202, doi:10.1007/978-3-642-01387-4.

[13] M. Martínez-Espronceda, I. Martínez, S. Led, J. D. Trigo, I. Oss, J. Escayola, L. Serrano, J. García, and A. García, "Intensa: Heart failure patient's follow-up system using the iso/ieee11073 standard," in *Final Program and Abstract Book - 9th International Conference on Information Technology and Applications in Biomedicine, ITAB*, 2009.

[14] "X73 agent and manager implementation source-code," World Wide Web electronic publication, 2009, last visit: July 2010. [Online]. Available: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MEDICALUSB